SIMULATION OF SPREAD SPECTRUM SYSTEMS USING ICSSM

FINAL TECHNICAL REPORT 546-1

by

K.S. Shanmugan

V.S. Frost

C.A. Paul

Prepared for

ROME AIR DEVELOPMENT CENTER

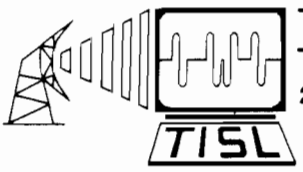
COMMUNICATIONS BRANCH

DCLF Section

Griffiss Air Force Base

Rome, New York

CONTRACT F30602-81-0205



TELECOMMUNICATIONS AND INFORMATION SCIENCES LABORATORY

THE UNIVERSITY OF KANSAS CENTER FOR RESEARCH, INC.

2291 Irving Hill Drive - Campus West Lawrence, Kansas 66045

ABSTRACT

Evaluating the performance of spread-spectrum systems operating in complex jamming environments is an important problem in the design of jam resistant communication links. The presence of nonlinearities, intersymbol interference, non-Gaussian noise, and other degradations make the performance evaluation a tedious and often an impossible analytic task. Computer simulation provides a flexible and accurate method for performance evaluation. This paper describes a simulation approach for computer-aided analysis of spread-spectrum systems operating in the presence of multiple jammers.

Algorithms and software modules that simulate the generic features of a spread-spectrum modem are described in the paper. Features simulated include burst and frame formatting, PN-spreading and despreading, filtering, jamming and synchronizing. Simulated and predicted performance of a direct sequence spread-spectrum system operating in the presence of various jammer are also presented and compared.

Table of Contents

Final Technical Report 546-1

LIST OF FIGURES

LIST OF TABLES

1.0 INTRODUCTION

Assessing the vulnerability of spread-spectrum (SS) systems to complex jamming waveforms is a difficult task. The presence of nonlinearities and intersymbol interference further complicate the analysis. Analytic techniques do not provide sufficient flexibility to assess the vulnerability of different SS systems in common jamming environments. Further it is not feasible to use hardware to evaluate a proposed SS system.

Computer simulation of communication systems has proven to be an effective tool for performance prediction [1-5]. Computer simulation provides a feasible mechanism for vulnerability assessment of SS systems. Simulation of a communications system consists of generating and processing signals in the computer which accurately represent the operation of the desired system. Thus communications simulation software must be structured so signals and signal processing functions can be efficiently represented. Further, because many signal processing functions are common to all SS systems, it is important to allow these signal processing functions to be used repeatedly. These restrictions lead to a software structure built upon functional blocks (modules), each module representing one general signal processing operation, for example, filtering. A SS system is then simulated by connecting the modules together in the desired configuration.

It has been found that modeling the functional blocks of a communications system in the time domain leads to the most efficient simulation. Time domain simulation implies that all signals are represented in the computer by a sequence of time samples. All signal processing functions are then modeled as operations on these sequences.

The mechanics of connecting the functional blocks together to form a complete system model is handled by a supervisory program. The simulation

software described in this report was developed and implemented within the Interactive Communications System Simulator Model (ICSSM) structure.

This report describes the development, implementation and testing of a set of ICSSM modules needed to simulate SS systems. These modules can be configured to model the generic features of a SS system operating in the presence of jammers. To demonstrate the accuracy of the modeling and the resultant simulation a comparison between theory, measurements, and simulation is presented. These results demonstrate that computer simulation of SS systems is an appropriate tool for assessing the vulnerability of SS systems operating in complex jamming environments.

A communications system simulation is developed in several phases. First, the system is decomposed into functional blocks. These blocks are usually identical to those found in a system block diagram. The function of each block is then modeled using a time domain, lowpass equivalent representation. Once adequate models are defined, algorithms (modules) are implemented within the appropriate framework, e.g., ICSSM. When the desired set of modules are implemented, a SS system is configured and the simulation tested. These tests are best conducted through a comparison of a simulated probability of bit errors (P_e) to that predicted by theory (for simple cases) or to a measured P_e.

Section 2 of this report describes the modeling of a SS system for simulation. The basic SS system principles will be reviewed and a generic system block diagram presented and discussed. The specifics of each functional block is discussed along with its time-domain model. Problems associated with the simulation of a frequency-hopped (FH) system in the time domain are presented and our solution discussed. The theoretical basis for each module is also presented in section 2. In section 3 the simulation

modules are described. The main purpose of this section is to explain the operation of each module and show how the modules can be configured to form a SS system. The meaning of module parameters is also contained in section 3.

The application of the SS system simulation is presented in section 4. Example waveforms and spectra are shown to illustrate the operation of various modules. Three noise environments are simulated to validate the performance of a simulation. First, the performance of a SS system in broadband white Gaussian noise is simulated. Theoretical results for this case are easily derived for comparison. The difference between the simulated signal-to-noise ratio (S/N) and the calculated S/N for equal P_e was less than .4 dB. The SS system performance in the presence of tone jamming was also simulated and compared favorably to theory. A noise-loaded FM jamming environment was also considered and the results compared to measurements and theory.

2.0 MODELING SPREAD SPECTRUM SYSTEMS FOR SIMULATIONS

The purpose of this section is to develop a model for a generic spread-spectrum system which is suitable for simulation. As background the basic principles of SS systems will be briefly reviewed and a generic system block diagram presented. The simulation of each function block will then be discussed.

2.1 Spread-Spectrum Systems

Spread-spectrum systems have been in use for many years [6-10]. Detailed discussions of SS systems are to be found in [6,10]. Spread-spectrum systems are currently used for a variety of applications. These include jamming protection, multiple access, multipath protection, and ranging. In this report we are only concerned with the antijamming property of SS systems. SS systems achieve their jamming protection by pseudo-randomly distributing the information to be transmitted over a range of alternatives. These alternatives include time (time hopped, TH, or burst), frequency (frequency hopping, FH) and phase (direct sequence, DS). The pseudo-random pattern used to distribute the information is known only to the receiver and transmitter. Jamming protection is thus achieved because the jammer does not know the pseudo-random pattern and must distribute its limited resources (power) over many alternatives to be effective.

This report will deal with a class of SS systems which uses DS and FH in a burst format. Thus by appropriate specification of parameters many systems of interest can be simulated. Figure 2.1 shows the spread-spectrum signal structure. The burst format provides some jamming protection. Each burst interval is divided into a transmission interval of length T_{on} sec and a quiet interval of length T_{off}. During each transmission interval a direct-sequence

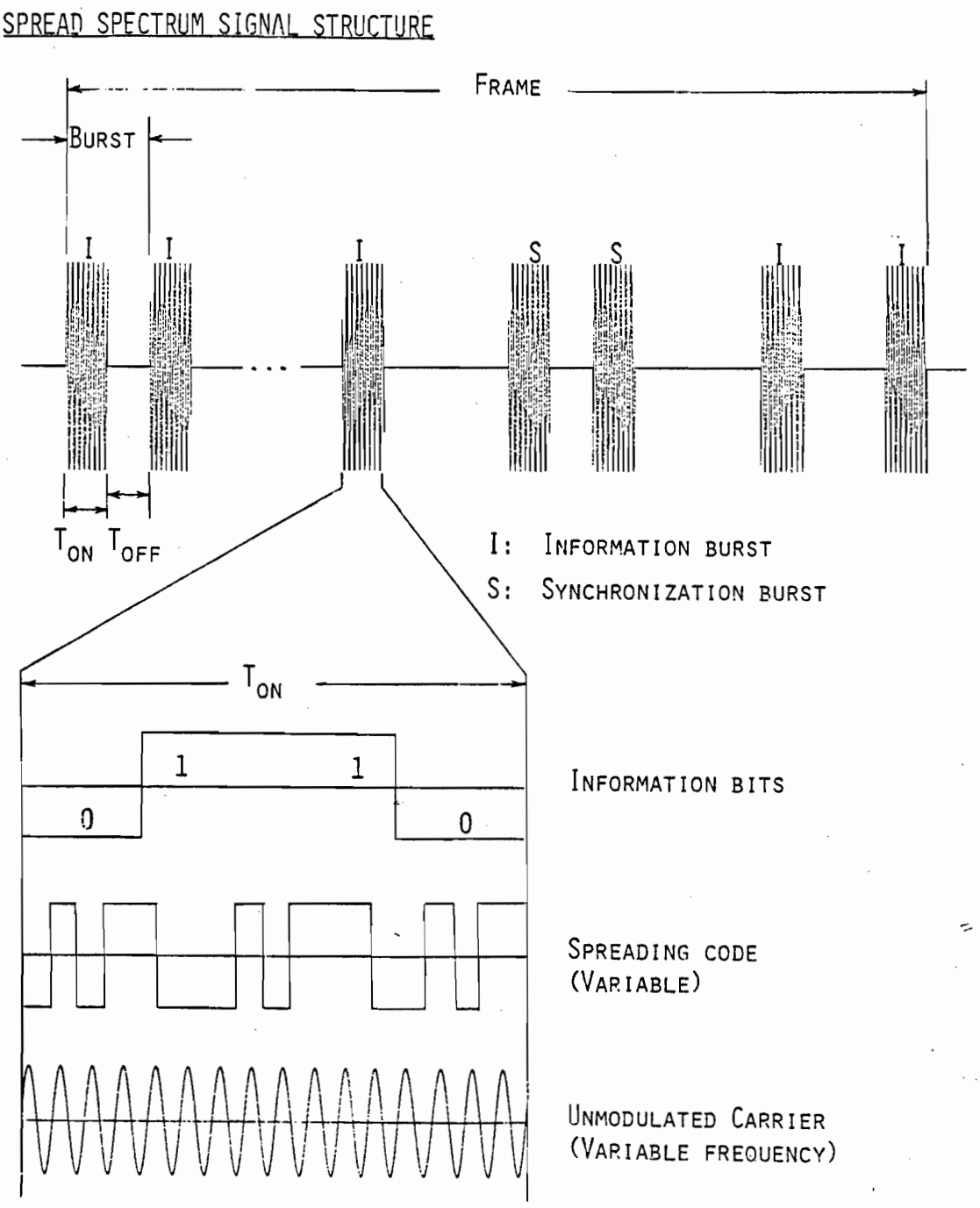


Figure 2.1 Spread Spectrum Signal Structure

spread-spectrum signal containing the information bits is sent. The spreading code can also change pseudo-randomly from burst to burst. Each burst is then transmitted on a different carrier frequency. Bursts are grouped into frames. Each frame contains information and synchronization bursts which use different spreading codes. The synchronization bursts are used for acquiring the direct-sequence code. This format provides a general framework for the modeling and simulation of many different SS systems. For example, by setting the duty cycle to unity and removing the frequency hopping a direct-sequence system can be modeled.

2.2 Spread-Spectrum System Analysis

A block diagram for this SS system, including the channel and jammers, is shown in Figure 2.2. The basic function of each block will be presented here. In the next section the simulation of each functional block will be discussed.

The information source can be voice or data. For voice signals it is common to perform source coding to reduce the bit rate by removing redundancies from the data. Typically, 16 kbps is needed for voice, however, more advanced techniques, e.g., linear predictive coding (LPC), can further reduce this requirement. Channel coding is included to insert controlled redundancies back into the data for error detection and correction. Source and channel coders are not considered further in this report.

The burst formatter combines blocks of data, e.g., 4 bit blocks, from the information source with a PN code sequence. The DS processing gain is then the length of the PN code sequence divided by the data block length. For

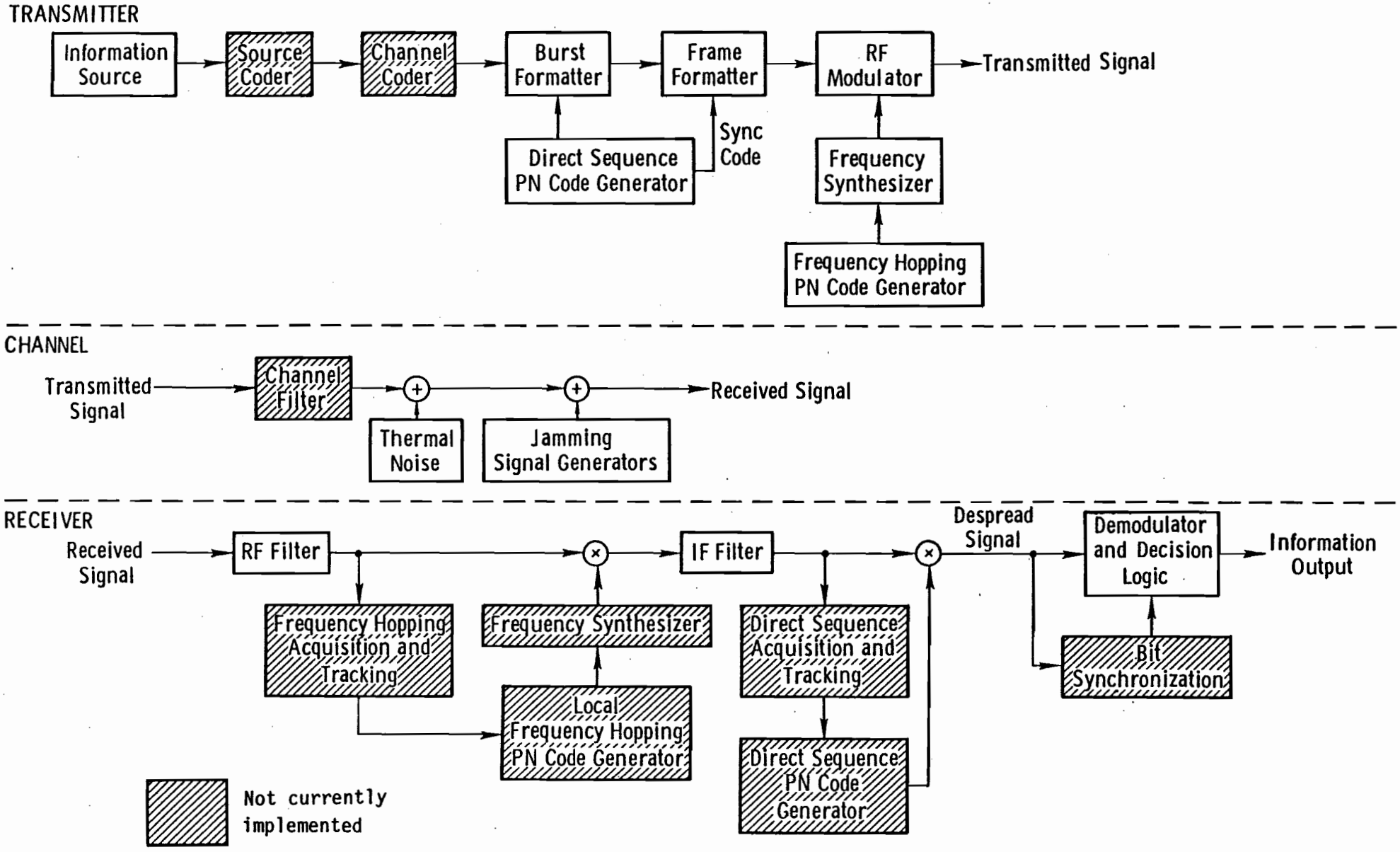


Figure 2.2 Spread Spectrum System Block Diagram

example, with a 4 bit information block and a 127 chip* PN code the processing gain is ≈32. The information bits are combined with the code chips by a direct multiply or a cyclic shift. Gold codes [10] are used as the PN sequences and a different code can be assigned to each burst.

The frame formatter groups a number of information bursts, i.e., bursts containing information, with a number of synchronization (sync) bursts. As shown in Figure 2.1, the sync bursts contain no information bits and are used in the receiver for code acquisition and tracking. Sync bursts are assigned a unique PN code.

The output of the frame formatter is applied to a RF modulator. Common modulation formats include BPSK, QPSK and MSK. The carrier frequency is changed for each burst interval using a separate PN code generator and a frequency synthesizer. The output of the RF modulator is the transmitted signal.

The transmitted signal is affected in the channel by thermal noise, jamming signals and signal fading. The signal fading can be modeled by a time-varying filter. The jamming signals considered here are tone interference, noise-loaded AM and noise-loaded FM.

After the transmitted signal is perturbed by the channel it is supplied to the receiver. The receiver RF amplifier/filter must be wideband to accommodate the full frequency-hopped bandwidth.

The amplified signal is then supplied to a FH acquisition and tracking circuit which aligns the local FH PN code with the transmitted code. Once acquisition has been achieved, the local PN code is used with a frequency

* The term bit will be used in reference to the information source while the term chip will refer to the PN code.

synthesizer to remove the frequency hopping, i.e., the signal is converted to an IF frequency.

After further amplification the direct-sequence PN code is acquired and tracked and used to despread the DS signal. The despread signal is then demodulated and the appropriate decision logic is used to recover the information bit sequence. Notice that AGC has not been included in this model. AGC presents significant problems and its specific implementation varies considerably from system to system.

All of the major components of a SS system have been included in this model. The system represented in this block diagram can be used to model a wide variety of SS systems. A computer simulation of this model provides a tool for assessing the vulnerability of SS systems in the presence of jamming. In the following section a detailed model suitable for implementing a simulation is derived for many of the functional blocks shown in Figure 2.2.

2.3 Functional Block Modeling for Spread-Spectrum System Simulation

The purpose of this section is to develop models for the major functional blocks in a SS system which are suitable for a time-domain computer simulation. All assumptions and approximations contained in each functional block model will be clearly stated. The models described here have been implemented as ICSSM modules; the description of the ICSSM implementation are presented in Chapter 3.

Simulating a communications system in the time domain results in some conditions on the form of the models used for the functional blocks. In a time-domain simulation all signals and filters are modeled using their complex lowpass equivalent representation (see [11,12] for a detailed discussion of complex lowpass equivalents). A bandpass signal can be written as

$$s(t) = \mathrm{Re}\left[\tilde{s}(t)\ e^{j2\pi f_c t}\right] \tag{2.1}$$

where

f_c = carrier frequency.

The term $\tilde{s}(t)$ is the complex lowpass equivalent representation of s(t). A bandpass filter can be represented by

$$h(t) = \mathrm{Re}\left[\tilde{h}(t)\ e^{j2\pi f_c t}\right] \tag{2.2}$$

If s(t) is the input signal to h(t) the output signal is found from

$$s_0(t) = \mathrm{Re}\left[\tilde{s}_0(t)\ e^{j2\pi f_c t}\right]$$

with

$$\tilde{s}_0(t) = \tilde{s}(t) * \tilde{h}(t) \tag{2.3}$$

where

* denotes convolution.

All models to be discussed will be put in the form of equations (2.1)-(2.3). Note when using complex lowpass equivalent representations for signals and filter the carrier term is suppressed.

2.3.1 Information Source Model (INFORM)

The information source is assumed to be digital. The analog-to-digital conversions along with source and channel coding is not currently modeled. The information source generates a sequence of bits with zeros and ones being

equally probable and statistically independent. The output of the information source is not a waveform rather it is a bit sequence.

2.3.2 Pseudo-Noise (PN) Code Generation (GOLDCD)

PN code generators are needed for both the DS and FH components of the system. Maximal length codes [10] are usually used. In the DS component of the system Gold codes [10] are used because of their cross-correlation properties. A Gold code generator is shown in Figure 2.3. This structure allows for a different member of the Gold code family to be used for each burst. This is done by changing the initial condition of the Gold code generator. The output of the PN code generator is a sequence of chips and not a waveform.

2.3.3 Burst Formatter and the Simulation Bandwidth (BURSTF)

The burst formatter model requires the information bit sequence as well as the PN code sequence as input. The parameters for the burst formatter model are (1) the number of chips/bit N_c, i.e., the DS processing gain; (2) the duty cycle; (3) number of bits per burst, i.e., the length of a data block; and (4) the number of time samples per chip. This final variable establishes the simulation bandwidth. For example, let the chip rate be 5 Mcps (Mcps = Million chips per second) and assume 8 time sample/chip, then the sample rate, f_s, is $4\text{x}10^7$ samples/sec and the simulation bandwidth is $f_s/2$ or $2\text{x}10^7$ hz. Between 8 and 16 time samples/chip are needed to adequately simulate a SS system in the time domain.

The burst formatter performs the DS spreading. Each block of information bits is combined with N_c chips. This combination is currently modeled as a multiplication operation. That is, let $\{b_i\}$ denote the data block

GOLD CODE GENERATOR:

Code length = $2^m - 1$

Number of codes = $2^m + 1$

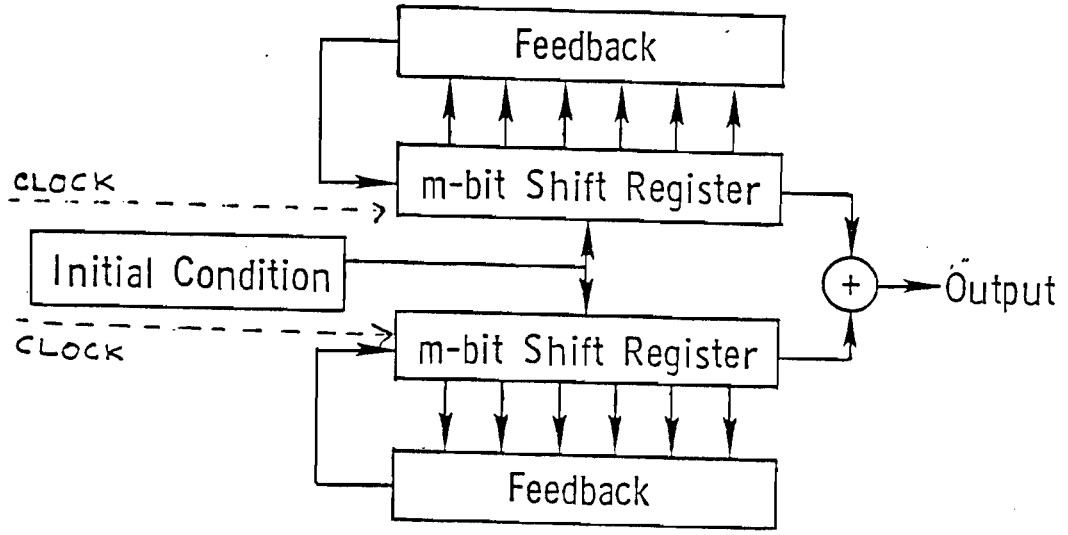


Figure 2.3 Gold Code Generator

and $\{c_j\}$ the code sequences. The sequences $\{b_i\}$ and $\{c_j\}$ are first transformed into biphase sequences $\{b_i'\}$ and $\{c_j'\}$, where

$$b_i' = 1 \quad \text{for} \quad b_i = 1$$

$$b_i' = -1 \quad \text{for} \quad b_i = 0$$

and (2.4)

$$c_i' = 1 \quad \text{for} \quad c_i = 1$$

$$c_i' = -1 \quad \text{for} \quad c_i = 0$$

The biphase sequences are then transformed into sampled time waveforms at the specified sample rate. If d(n) and p(n) represent the sampled time waveforms obtained from the biphase data and code blocks, respectively, then the burst formatter output is

$$s_B(n) = d(n)\ p(n) \qquad 0 < n < N_{on}$$

$$= 0 \qquad N_{on} + 1 < n < N_T \qquad (2.5)$$

where

$N_{on} = T_{on} \cdot f_s$

$N_T = (T_{on} + T_{off}) \cdot f_s$

f_s = sample rate

T_{on} = transmission time

T_{off} = quiet time

n = sample index

The output of the burst formatter is a waveform.

2.3.4 Frame Formatter (FRAMEF)

A frame is composed of a group of information bursts and sync bursts. The input to the frame formatter is $s_B(n)$ and the sync PN code (or codes). The

parameters of the frame formatter are the number of information bursts/frame and the number of sync bursts/frame. Sync bursts are incorporated into the input waveform in the frame formatter. The output of the frame formatter is a sampled waveform.

Because of the constraints of ICSSM the burst and frame formatter do not follow the above description directly, however, the functions performed and the parameters described above are those used in the ICSSM implementation.

2.3.5 RF Modulator (QPSKMD, MSKMOD)

The input to the RF modulator is a time-sampled real waveform $s_B(n)$. For simulation purposes the function of the modulation is to transform the input waveform into the complex lowpass equivalent signal which represents the bandpass transmitted signal. A sequence of symbols at the output of the frame formatter can be found by resampling the signal. This can be represented as

$$v_k = s_B\left(\left[\frac{n}{N_S}\right] = k\right) \tag{2.6}$$

where

$[n/N_s]$ denotes integer part of n/N_s; k=1,2,3,...

N_s = number of time samples/chip.

n = sample index

The sequence v_k has three levels -1, 0 and 1. The output of the RF modulator takes on different forms depending on the desired type of modulation. For BPSK the output waveform is

$$\tilde{s}_{BPSK}(n) = s_B(n) \tag{2.7}$$

For OPSK the sequence of symbols v_k are transmitted in a quadrature format, i.e., the k^{th} symbol is transmitted on the inphase channel and the k^{th} + 1

symbol on the quadrature channel. The complex lowpass representation for this OPSK signal is

$$\tilde{s}_{OPSK}(n) = v_k + j\ v_{k+1} \tag{2.8}$$

where

$$k = \left[n/N_s\right] \text{ for } k \text{ even}$$

To keep the baud rate in OPSK the same as in BPSK, the symbol duration is doubled, i.e., $v_k' = v_k$ for $2N_s$ samples and

$$\tilde{s}_{OPSK}(n) = v_k' + j\ v'_{k+1} \tag{2.9}$$

For MSK the complex lowpass waveform is

$$\tilde{s}_{MSK}(n) = v_k' \sin\left(\frac{n}{2N_s}\pi\right) + j\ v'_{k+1} \sin\left[\left(\frac{n}{2N_s} + v_s\right)\pi\right] \tag{2.10}$$

2.3.6 The Time Domain Simulation of a FH SS System in Jamming

One problem with using complex lowpass representations for signals and filters is simulating frequency hopping. It is the carrier frequency which changes in a FH system but in using the complex lowpass equivalents it is the carrier frequency which is suppressed.

Each burst is transmitted on a different frequency in a frequency hopping system. If the FH system can select from a set of N_H possible frequencies and the jammer distributes its power J over N_J of these frequencies then the probability of a burst being jammed is

$$P_J = \frac{N_J}{N_H} \tag{2.11}$$

To simulate a FH system in the time domain, jammer power is added to a burst with a probability of P_J. In other words, partial-band jamming for a FH system is simulated like pulse jamming on a DS system. Note that the jammer pulse period is random. The average period for the equivalent pulse jammer is $N_H P_j T_F$ where T_F is the frame time (or the hopping epoch). The type of signal added to a jammed burst can be a tone, noise-loaded AM, noise-loaded FM, or any combination.

2.3.7 Channel Modeling

A general model for the channel should contain three components: (1) a direct path, (2) specular multipath, and (3) diffuse multipath [1]. Only the direct path is currently modeled.

2.3.8 Modeling of Thermal Noise and Noise-Loaded AM Jammers (NOISE, CNOISE, AND AMJAMR)

The power spectral density of a noise-loaded AM jammer is shown in Figure 2.4. The time waveform is given by

$$n_l(t) = \mathrm{Re}\left[\left(x_c(t) + j\, x_s(t)\right) e^{j2\pi\Delta f t+\theta}\, e^{j2\pi f_c t}\right] \qquad (2.12)$$

where

$x_c(t)$ = inphase noise component,

$x_s(f)$ = quadrature noise component,

Δf = frequency offset, between the jammer and the SS waveform,

θ = random phase uniformly distributed on $0-2\pi$.

The lowpass equivalent is given by

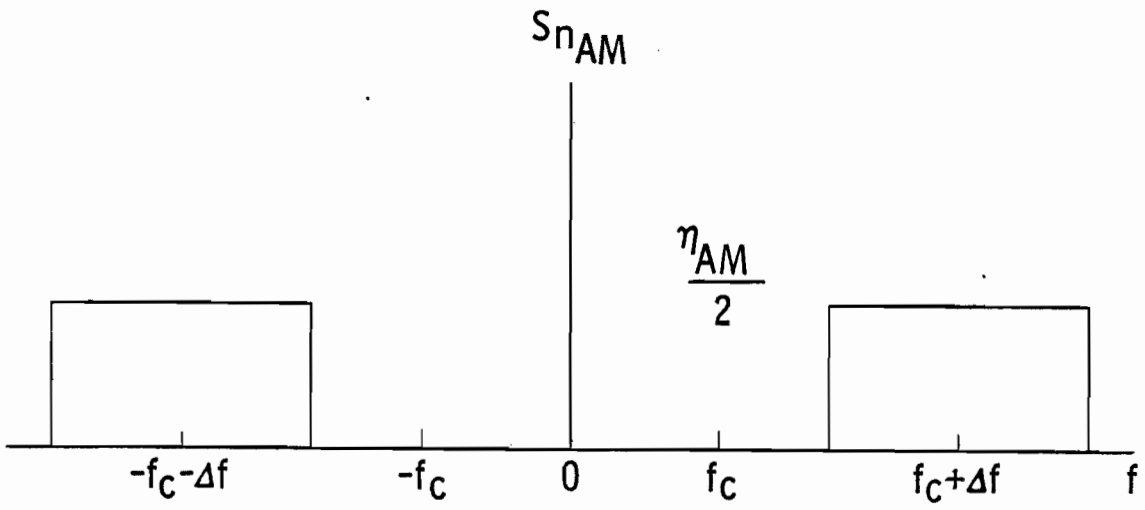


Figure 2.4 Power Spectral Density for a Noise-Loaded Jammer

$$n_{l_{LP}}(t) = \left(x_c(t) + j\, x_s(t)\right) e^{j2\pi\Delta f t+\theta} \qquad (2.13)$$

The generation of a discrete-time noise-loaded AM signal in ICSSM is shown in Figure 2.5. The jammer's power is related to the variance of $x_c(k)$ and $x_s(k)$, i.e., σ^2_{AM} by

$$\sigma^2_{AM} = \frac{J_{AM} f_s}{2\ B_J}$$

where

J_{AM} = desired jammer power

B_J = jammer bandwidth

f_s = simulation sample rate.

The sampled signals $x_c(k)$ and $x_x(k)$ represent broadband white noise as shown in Figure 2.6. The purpose of the lowpass filter is to limit the noise to the desired bandwidth. The multiplication by the complex exponential term models any offset between the SS system center frequency and the jammer's center frequency. (Note: the signals $x_c(k)$ and $x_s(k)$ have a power spectrum as shown in Figure 2.6, and thus by changing σ^2_{AM} those signals can be used to model broadband thermal noise.)

2.3.9 Modeling of Tone Jammers (CWJAMF,CWJAMR)

The complex lowpass representation of a tone (multiple tones are allowed) jammer is

$$n_T(t) = \sum_{i=1}^{L} A_i\, e^{j2\pi\Delta f_i t+\theta_i} \qquad (2.15)$$

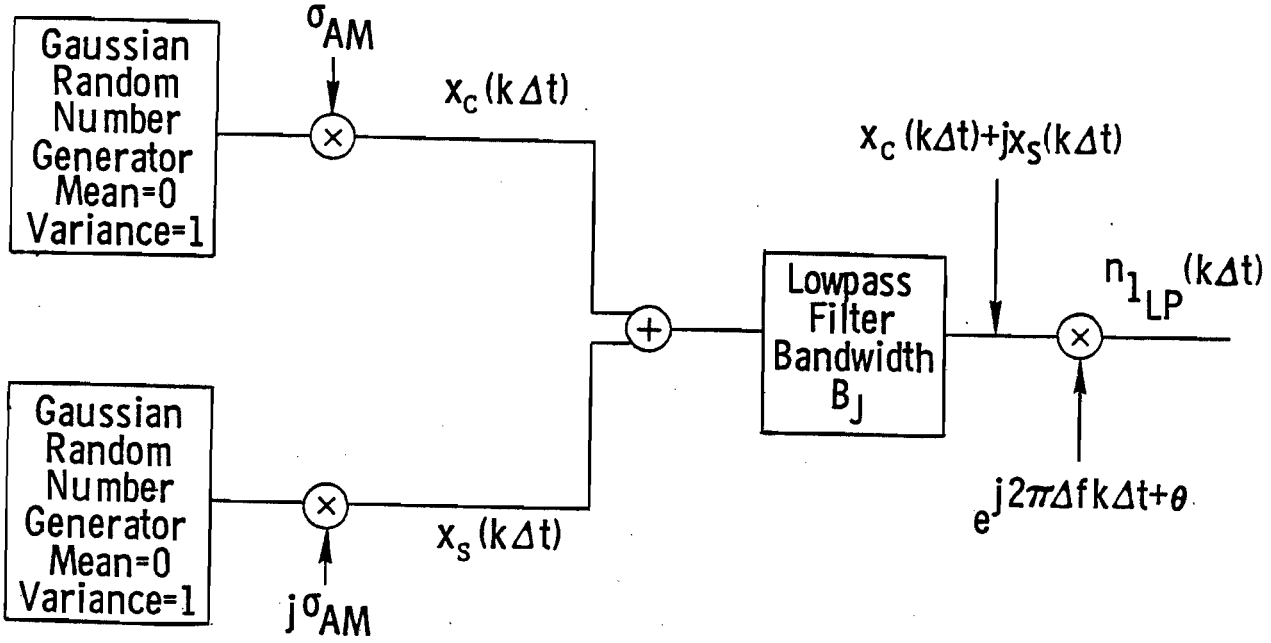


Figure 2.5 Block Diagram for the Generation for a Noise-Loaded Jammer

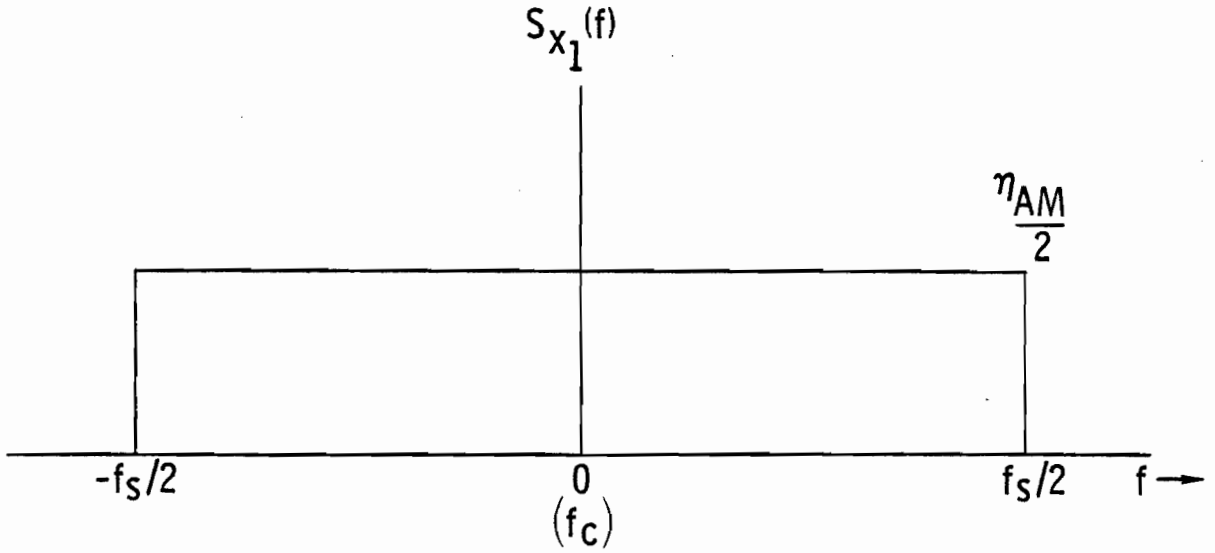


Figure 2.6 Power Spectral Density for Broadband White Noise

where

L = number of tone jammers

Δf_i = frequency offset for the i^{th} jammer

θ_i = random phase offset for the i^{th} jammer.

The total complex jammer power is

$$J_T = \sum_{i=1}^{L} A_i^2 \tag{2.16}$$

For a comb jammer

$$J_T = L\ A^2 \tag{2.17}$$

where

$A = A_i$, for each i.

A theoretical analysis of the effect of multiple tone jammers can be found in [13]. Both Δf_i and θ_i can be chosen randomly. Thus, a tone jammer can appear anywhere in the signal band.

2.3.10 Modeling of a Noise-Loaded FM Jammer (FMJAMR)

The complex lowpass representation for a noise-loaded FM jammer is given by

$$n_{FM}(t) = A\ e^{j2\pi By(t)+\theta+2\pi\Delta ft} \tag{2.18}$$

where

$$y(t) = \int_{-\infty}^{t} n(\tau)\ d\tau \tag{2.19}$$

and

n(t) = bandlimited white Gaussian noise with a bandwidth of $f_s/2$.

B = jammer bandwidth parameter

Clearly the jammer power, J_{FM}, in this case is

$$J_{FM} = A^2 \tag{2.20}$$

The primary parameters of this jammer are its bandwidth and spectral shape. Next we will develop an expression for the autocorrelation of $n_{FM}(t)$ from which the spectral shape can be derived. As specified above, the autocorrelation function for the quadrature components of $n_{FM}(t)$ has a Laplacian shape, as will be shown. The bandwidth of n(t) can be changed to generate signals with Gaussian-shaped spectra.

We will consider the discrete-time expression for the inphase component of $n_{FM}(t)$,

$$n_I(k) = A \cos \left[2 \pi \beta \sum_{i=1}^{k} n(i) + \theta \right] \tag{2.21}$$

where

θ = random phase on $0-2\pi$,

k = sample index.

All bandwidth calculations can be made from this waveform. Now n(i) is a Gaussian random variable with

$$\begin{aligned} &E[n(i)] = 0 \\ &Var[n(i)] = 1 \\ &R_n(i) = \delta(i) \end{aligned} \tag{2.22}$$

where $E[\cdot]$, $Var[\cdot]$ Rn denote expectation, variance, and autocorrelation respectively. The autocorrelation of $n_I(k)$ is defined as

$$R_{n_I}(k,m) = E[n_I(k)\, n_I(m)] \tag{2.23}$$

From the symmetry of the autocorrelation function we can consider only $k \geq 0$ and $k > m$. Now

$$R_{n_I}(k,m) = A^2 \, E\left[\cos\left(2\pi B \sum_{i=1}^{k} n(i) + \theta\right) \cos\left(2\pi B \sum_{i=1}^{m} n(i) + \theta\right)\right]$$

$$= \frac{A^2}{2} E\left[\cos\left(2\pi B\left(\sum_{i=1}^{k} n(i) - \sum_{i=1}^{m} n(i)\right)\right)\right]$$

$$+ \frac{A^2}{2} E\left[\cos\left(2\pi B\left(\sum_{i=1}^{k} n(i) + \sum_{i=1}^{m} n(i)\right) + 2\theta\right)\right] \qquad (2.24)$$

The second term is zero, so

$$R_{n_I}(k,m) = \frac{A^2}{2} E[\cos(2\pi B \, z(k,m))] \qquad (2.25)$$

where

$$z(k,m) = \sum_{i=1}^{k} n(i) - \sum_{i=1}^{m} n(i)$$

It follows from the assumption that $k > m$ that

$$z(k,m) = \sum_{i=m+1}^{k} n(i) \qquad (2.26)$$

also it follows from eq. (2.22) that

$$E[z] = 0$$

and

$$\mathrm{Var}[z] = k - m = \sigma_z^2 \qquad (2.27)$$

where $z=z(k,m)$. The term $z(k,m)$ can be thus modeled as a Gaussian random variable with mean zero and variance $k-m$. Now

$$R_{n_I}(k,m) = \left(\frac{A^2}{2}\right)\left(\frac{1}{2\pi\sigma_z^2}\right) \int_{-\infty}^{\infty} \cos(2\pi B z) \, e^{-z^2/2\sigma_z^2} dz \qquad (2.28)$$

From [14] this integral is found to be

$$R_{n_I}(k,m) = \frac{A^2}{2} e^{-2\pi^2 B^2 (k-m)} \tag{2.29}$$

Note that $R_n(k,m) = R_n(k-m)$ so $n_I(k)$ is a wide-sense stationary process. Again from symmetry we can write

$$R_{n_I}(k) = \frac{A^2}{2} e^{-2\pi^2 B^2 |k|} \tag{2.30}$$

which is a Laplacian shape. The 3, 6 and 9 dB bandwidths are given by $\sqrt{\pi B^2}$, $\sqrt{3\pi B^2}$ and $\sqrt{7\pi B^2}$, respectively. Figure 2.7 shows the power spectrum of a signal generated using eq. 2.21. The plot follows the expected shape.

Often it is desirable to generate Gaussian spectral shapes. It is well known [11] that if n(t) in eq. 2.19 is not only Gaussian but slowly varying then $n_{FM}(t)$ will have a Gaussian-shaped spectral density. n(t) can be made to vary slowly by simply holding a sample value constant for many samples (i.e., n(i) = constant for i = ℓ, $\ell+\Delta$). Figure 2.8 illustrates the result of this process. The spectral density is Gaussian as expected. The approach outlined above provides an efficient technique for the generation of noise-loaded FM signals. The approach also provides a technique to directly generate jamming signals with various spectral shapes.

2.3.11 Filter Modeling (NFILT)

All RF, IF, and baseband filters are simulated using infinite impulse response (IIR) discrete-time filters [16]. These filters allow for complex inputs. A general filter package has been implemented which includes lowpass, highpass, bandpass and bandstop structures. Butterworth and Chebyshev, and

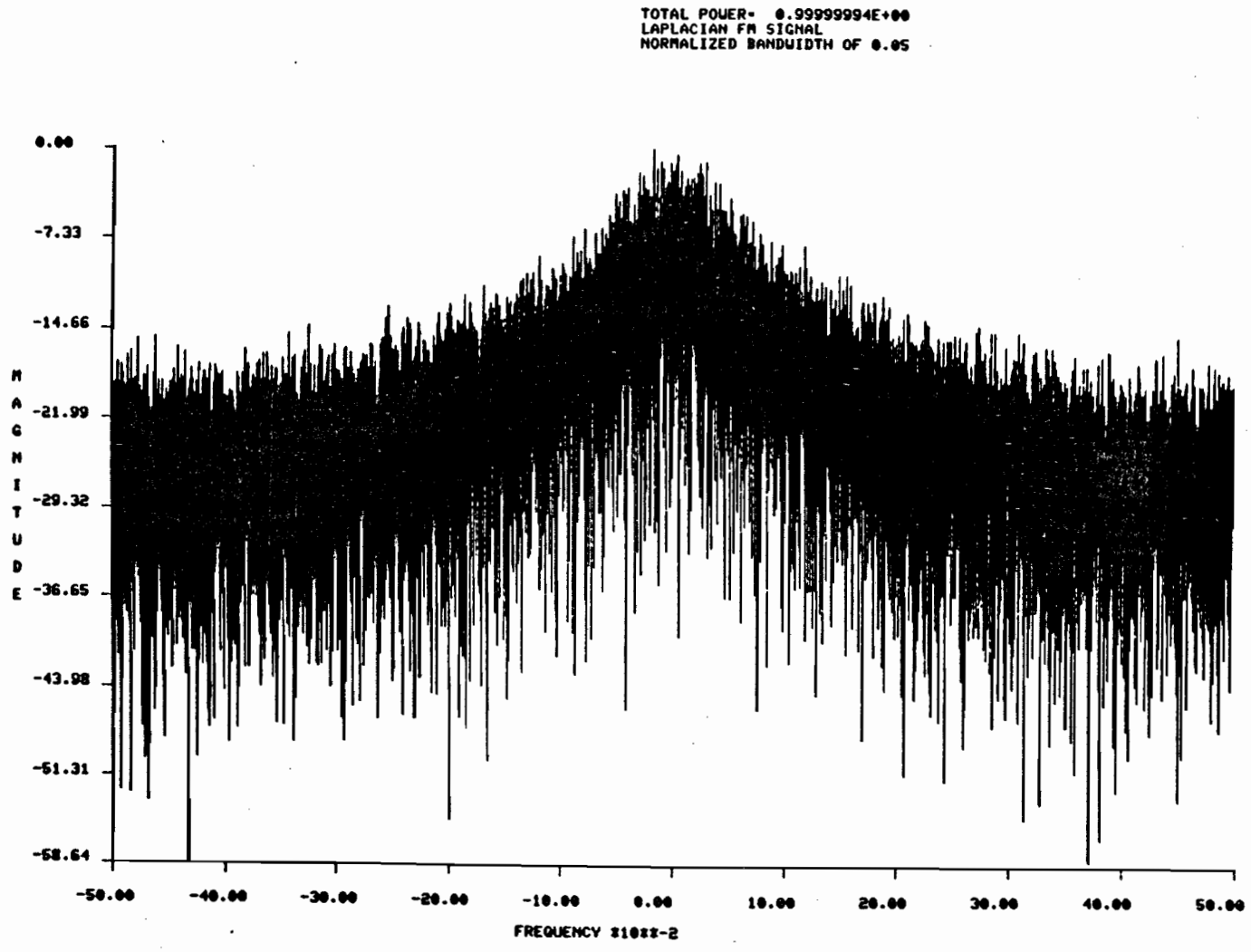


Figure 2.7 Power Spectral Density of a Noise-Loaded FM Jammer with a Wideband Noise Source

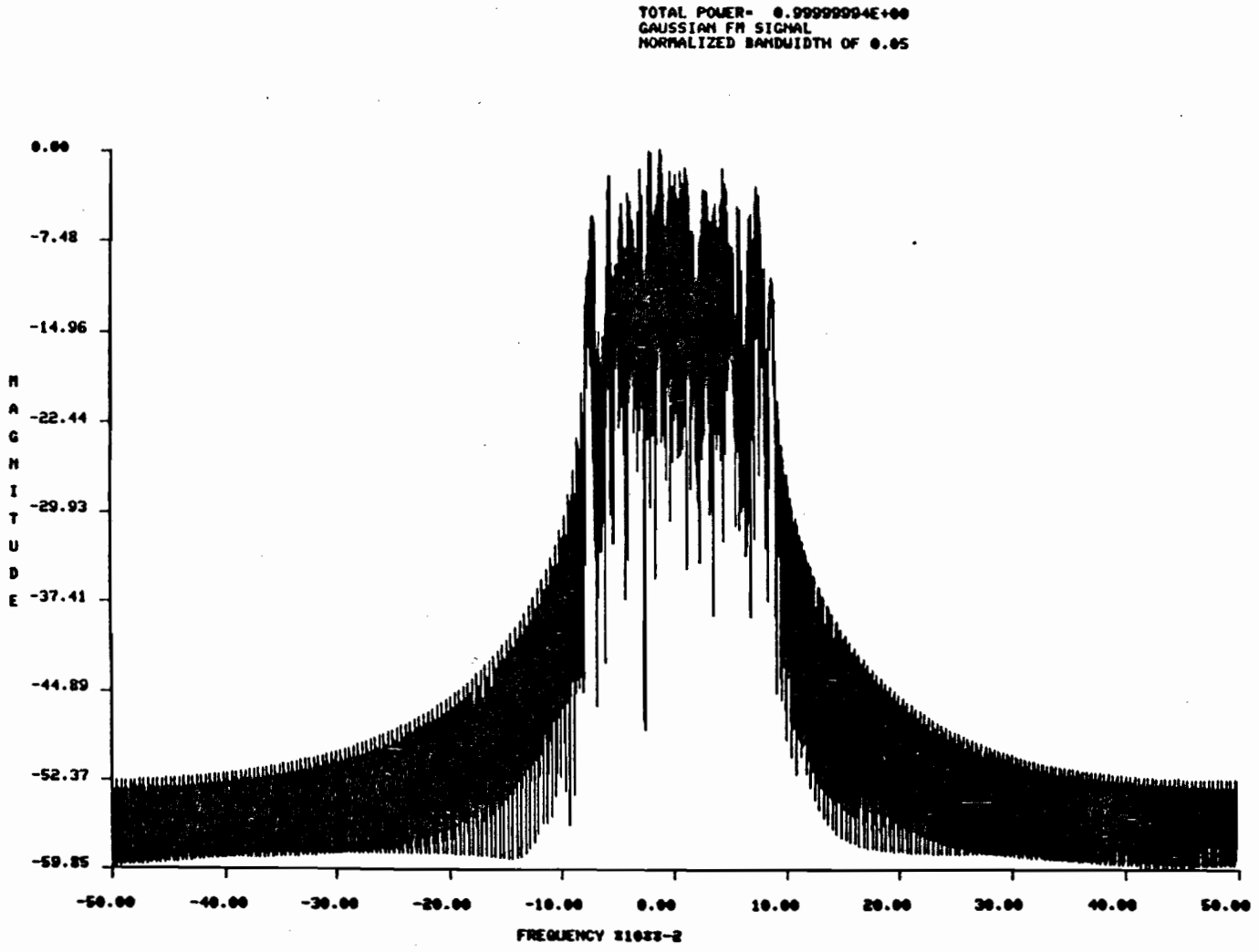


Figure 2.8 Power Spectral Density of a Noise-Loaded FM Jammer with a Narrowband Noise Source

elliptic filter characteristics are included. The bilinear z-transform technique is used to define the specifics of each filter structure. The bilinear z-transform provides the mechanism for transforming analog filters, e.g., a Butterworth filter specified in the s-plane, into a discrete-time IIR filter. (See Table 2.1 for the available filters and their parameters.)

2.3.12 Frequency Dehopping

In our current model we assume ideal frequency dehopping. This is equivalent to assuming that the receiver knows the FH PN code phase. FH PN code acquisition and tracking can be achieved with greater ease [6] because of the relatively slow FH PN code rate.

2.3.13 Direct Sequence Acquisition and Tracking

The phase of the DS PN code is unknown to the receiver and it must be estimated from the received signal. The phase of the DS PN code is estimated in this system by measuring the delay associated with the sync bursts. Because of the relatively short code length a correlator can be used to estimate this delay in a reasonable amount of time. A serial search correlation procedure is used to estimate the delay. In a serial search the local code phase (delay) set at some initial value, e.g., m time samples, and the following sum calculated

$$y = \sum_{n=1}^{k} s(n)\ P_L(n+m)$$

FILTERS

Types of Filters (Modules):

- Butterworth (lowpass, bandpass, highpass, bandstop)
- Chebyshev (lowpass, bandpass, highpass, bandstop)
- Elliptic (lowpass, bandpass, highpass, bandstop)

Parameters:

FILTYP	- Filter type
NORDER	- Order of the filter
BW	- Filter bandwidth
RIPL	- Inband ripple
CENTF	- Center frequency
INTYPE	- Type of input signal (real or complex)

Table 2.1

Filter characteristics

where

k = number time samples/burst,

$P_L(n)$ = local PN code,

S(n) = received signal time samples.

If y is greater than a set threshold then m is the estimated code phase and acquisition has been achieved. If y is less than the threshold then m is incremented by $N_s/2$ (i.e., 1/2 of a chip time) and the sum recalculated. For simulation purposes the delay is caused by the filter delays. These delays are estimated once and used to align the DS PN code. This delay is also used for bit synchronization. No code tracking is currently provided.

2.3.14 Despreader Model

The despreader is modeled by a multiplier. The incoming signal is multiplied by a locally generated Gold code sequence. Logic is provided to allow different bursts to be spread and despread with different Gold code sequences. Cyclic shift despreading has not been implemented.

2.3.15 Demodulation and Decision Logic

The function of the demodulator is to process the despread signal and produce the original information bit sequence. As stated previously no source or channel coding is modeled so the decision logic is relatively simple, i.e., a simple threshold. The major component of the demodulator is an integrate and dump filter (or a lowpass filter). The integration time is set equal to the bit duration. The structure will change slightly with OPSK and MSK modulation. These modulation schemes will require two integrate and dump filters and additional logic to reconstruct the transmitted bit sequence.

2.4 Summary

This section provides a description of a generic SS system and a model for each of its major functional blocks. Each functional block is described in terms of its input waveform (or bit sequence), its output waveform and the parameters which control its operation. In the following chapter the ICSSM implementation of these functional blocks is described, an example spread spectrum end-to-end system is configured, its parameters are specified and explained, and instructions are given as to obtaining graphic output of a single port in the configuration.

3.0 ICSSM IMPLEMENTATION OF A GENERIC SPREAD-SPECTRUM SYSTEM

3.1 ICSSM Programming Strategies

This section covers alternate methods for handling multi-port input, algorithm implementation, and feedback for ICSSM. While multi-port input is covered in the ICSSM manual, that method involves much more system overhead than is necessary. Most ICSSM modules from Hazeltine have algorithm code and ICSSM system subroutine calls intermixed. This tends to cloud the intent of the algorithm and therefore makes the algorithm implementation harder to understand and debug. An alternate approach addressing this problem is presented.

Regular ICSSM programming procedure for multi-port input is to use subroutine "fetchx" to see if every port is available. If a single port is not available the module usually exits. Upon the next processing entry (usually an "ION" event) all ports are checked again. There is an easier way to check that all ports are available.

According to event queue processing (see Appendix J of the ICSSM manual) each input port that is available sets the "kevnt" variable to "ION". If there are two input ports to a module and both become available the module will receive first one "ION" and then a second "ION", each "ION" coming as an input port becomes available. Two input ports will mean that two "ION"s will occur, three "ION"s mean that three ports are available, etc. One can take advantage of this fact in a program by using the following code:

```
      common /iparam/ ....,....,nion
c
c             source code
c
c             wait for two input ports
c
c             count the number of "ION"s (ports available)
      if(kevnt.eq.ion) nion=nion+1
```

```
c             count the number of "IOFF"s  (ports now unavailable)
      if(kevnt.eq.ioff) nion=nion-1
c             process only if a port is available
      if(kevnt.ne.ion) return
c             if both ports aren't available then return
c             (2 "ION"s means that 2 ports are available)
      if(nion.ne.2) return
c
c
c             more user code
c
      call fetchx(1,iev)
      if(iev.ne.0) stop (if we stop there is a drastic system error)
c             store the "u" array somewhere else because
c             next we'll fetch the second port
      call fetchx(2,iev)
      if(iev.ne.0) stop (again if we stop there is a drastic error)
c
```

Using this coding method all ports should be available by the time the module tries to fetch them.

In implementing our modules for ICSSM we have chosen to write, test, and debug all algorithms as subroutines independent of ICSSM. After verifying that the algorithms run correctly an ICSSM "shell" is built around a subroutine call to that algorithm. This minimizes ICSSM and algorithm code mixing. All ICSSM related input and output is only in the ICSSM "shell" routine and all code required for the algorithm itself is only in the subroutine that expresses the algorithm. However, since there is no ICSSM/algorithm code mixing there is no checkpointing in the "shell" or the subroutines themselves. Therefore, no user-defined restart or benchmarking points are available.

3.2 ICSSM Modeling of a Spread-Spectrum System

Block diagrams of each module, module documentation, and a block diagram of a complete system are included in the latter half of this chapter. The block diagrams of each module and documentation provide information on data

flow through the modules of both signals and variables, and provide definitions of user settable variables. The last section expounds upon parameter value determination and a method to obtain system delays of configurations before despreading is to be done.

3.2.1 System Overview

Our model of a spread-spectrum system consists of transmitter, modulator, channel, demodulator, receiver, and support module sections. There are some aspects of the simulation that need to be known to all modules in the simulation. Labeled commons /xblock/ and /yblock/ are used for these purposes in ICSSM. The /xblock/ common is used to describe the input signal; the /yblock/ common is used to describe the output signal. However, the naming conventions of those variables and the number of variables in those commons left unnamed are not adequate in our case. Table 3.1 presents the names of the variables in common /xblock/. The last 5 variables in /xblock/ are given "generic" names, i.e. "real1x", "real2x", "int1x", "int2x", and "int3x". These variables are intended for the programmer of an ICSSM module to use as he needs. For our purposes these 5 variables were not enough; we found it necessary to use other "named" variables to pass control information between modules to coordinate such things as "synchronization bursts". There is no "named" variable in /xblock/ to suit that purpose. Therefore, some variables in commons /xblock/ and /yblock/ are used for purposes other than those implied by the names of those variables. For instance, many modules need to know when a "burst" is a "sync burst". The frame formatter will set variables "fz" in /yblock/ to 1 to let downstream modules know that the current "burst" is a "sync burst". (In /xblock/ this variable is called "fzx".) This use of the variables "fz" and "fzx" has nothing to do with their ICSSM naming

VARIABLE USAGE IN COMMON /XBLOCK/

VARIABLE NAME	MEANING
tdurx	simulation time duration during module invocation
deltx	simulation time increment between waveform samples
tstopx	simulation stop time
ncofx	number of waveform samples available during module invocation
reallx	burst interval time
fzx	=0. generate data and data Gold code =1. suppress data generation and generate sync Gold code
int3x	number of chips in Gold code (Gold code length)
int2x	number of information bits per burst
amplx	amplitude of burst waveform (default value for jammers)
bwx	bandwidth of burst waveform (default value for jammers)
real2x	duty cycle of burst waveform
intlx	number of samples per Gold code chip
encdx	=0. real signal to process =1. complex signal to process (set by qpskmd or mskmod)

Table 3.1

conventions. Other modules may pass other information downstream in named or unnamed variables so that other modules may use the values of those variables.

3.2.2 Transmitter Section

The transmitter section consists of the frame formatter, Gold code generator, information generator, and the burst formatter modules (see Figure 3.1). Also included in this transmitter section is the expander module, which is used once to generate and delay the Gold code waveform and once to generate and delay the information bit sequence waveform.

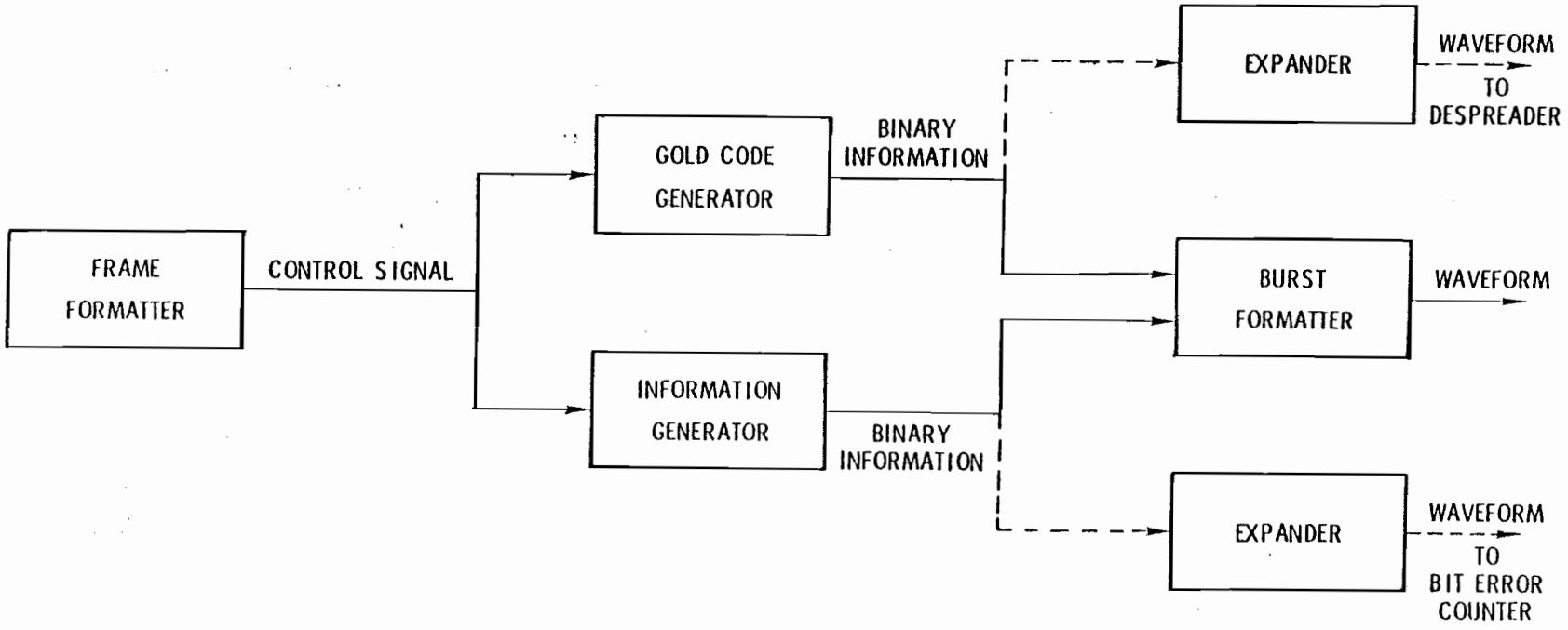
3.2.2.1 Frame Formatter (FRAMEF)

In our implementation the frame formatter is the self-updating module; hence it does not occur after the burst formatter, rather our frame formatter acts as system clock to tell the information generator when to generate a new information bit sequence, and also tells the Gold code generator when to produce a new Gold code and whether that Gold code should be a "sync" Gold code (fz in /yblock/ =1) or a "data" Gold code (fz in /yblock/ =0). Other modules downstream such as the despreader need access to this variable also. Other variables set by the frame formatter in /yblock/ that are needed downstream are "tdur" (ICSSM definition), "delt" (ICSSM definition), "tstop" (ICSSM definition), "ncof" (ICSSM definition), "reall" (burst interval time), and "fz" (as described above). (See Table 3.2 for ICSSM definitions of variables in common /xblock/.)

While the frame formatter has no data to output through its output port to the Gold code generator or the information bit sequence generator it uses the event code ("ION") of its call to subroutine "sigout" to tell the Gold code generator and the information sequence generator to "go", i.e. to generate a

Figure 3.1

SPREAD SPECTRUM WAVEFORM GENERATOR



```
Syntax:  COMMON/XBLOCK/TONX,TOFFX,FZX,BWX,TDURX,WVFMX,
         & SIGVX,TXZ,TERRX,IDX,RX,ELEVX,AZIMX,
         & PROPX, AMPLX, PHAZX, DPLRX, NCOFX, SCALX,
         & SPACX,DELTX,NODEX,NMBRX,TOKX,ENCDX,
         & TSTOPX,REAL1X,REAL2X,INT1X,INT2X,INT3X
```

COMMON block XBLOCK makes available Signal List elements describing the input signal at the Module input port. These input Signal List elements are the following:

FORTRAN VARIABLE NAME	MEANING
TONX TOFFX	SIMTIME values when Application Module was "energized" and was "de-energized" (such an interval is referred to as a "segment").
FZX	Center frequency
BWX	Signal bandwidth value
TDURX	Duration of Module activation (ie, segment duration)
WVFMX	Carrier waveform name code or type code
SIGVX	A signal voltage
TZX	Time of transmission
TERRX	Transmitter clock error
IDX	Transmitter (or source Module) ID Number
RX	Transmitter-receiver horizontal range
ELEVX	Elevation angle of transmitter antenna relative to receiver
AZIMX	Azimuth angle of transmitter relative to receiver
PROPX	Propagation Loss Factor
AMPLX	A signal amplitude or signal power value
PHAZX	A signal phase value
DPLRX	A Doppler rate or value
NCOFX	Number of samples in Coefficient Record
SCALX	A scale factor for Coefficient Record
SPACX	Code for Coefficient basis-space
DELTX	An incrementing value of time between Coefficient samples
NODEX	Node Code for associated output port
NMBRX	Index of Coefficient Record sample value relative to the first value
TOKX	T_{OK} for intra-Module SIMTIME
ENCDX	Encoding Type Code
TSTOPX	SIMTIME OF TSM termination
REAL1X REAL2X INT1X INT2X INT3X	(as required)

Table 3.2 ICSSM Variable Definitions

new Gold code or information bit sequence or to replay an old Gold code or information bit sequence.

3.2.2.2 Gold Code Generator (GOLDCD)

After the Gold code generator receives a "go" from the frame formatter (an "ION" at its input port) the Gold code generator determines whether to generate a new Gold code or to replay an old one.* The Gold code generated will either be a "sync" Gold code ("fzx" in /xblock/ =1) or a "data" Gold code ("fzx" in /xblock/ =0). The Gold code generator uses variables from /xblock/ "reallx" (burst interval) and "deltx" to determine whether this is a new burst or a burst already in progress (therefore do not change the Gold code). In this manner, with little modification, this Gold code module may be made to generate a different Gold code for every burst.

The burst formatter needs to know the length of the Gold code it will receive from the Gold code generator. This is done by finding the Gold code length in /xblock/ variable "int3x", and executing a call to ICSSM subroutine "dupxy" to copy /xblock/ into /yblock/. Now the variable "int3x" in /xblock/ is copied to "int3" in /yblock/. (Routine "dupxy" copies from the input labeled common /xblock/ to the output labeled common /yblock/.) Routine "yset" updates the ICSSM clock and other house-keeping variables in /yblock/.

3.2.2.3 Information Bit Sequence Generator (INFORM)

As with the Gold code generator the information bit sequence generator waits for an "ION" at its input port before it will run. It determines

* An old Gold code is replayed by being stored into hidden parameter block /IPARAM/ at the beginning of every burst interval. The proper portion for this invocation is then retrieved and output.

whether to generate a new bit sequence or to replay an old one. When the frame formatter indicates that the current burst should be a "sync burst" the information generator will call ICSSM subroutine "sigout" but the data at that port will not be used by the burst formatter. The burst formatter needs to know the number of information bits per burst it will get from the information bit sequence generator. The information generator uses "int2x" in /xblock/ for this value, passing it to the burst formatter in the same manner that the Gold code generator passes the Gold code length (through "dupxy" and "yset").

3.2.2.4 Burst Formatter (BURSTF)

The burst formatter waits for its 2 input ports to become available. It gets the Gold code from port 1 with the length of the Gold code specified by "int3x" in /xblock/. The Gold code is expressed as an "int3x" long sequence of zeros and ones. The burst formatter gets the information bit sequence from port 2. The information bit sequence is expressed as an "int2x" long sequence of zeros and ones. If, however, "fzx" in /xblock/ is 1 the information bit sequence port is waited for but ignored after it is present, since the current burst is a "sync burst". The burst formatter outputs a waveform at its output port. The burst formatter needs the following variables for /xblock/: "fzx", "real1x" (burst interval), "int2x" (number of information bits per burst), and "int3x" (the Gold code length). The burst formatter sets the following variables for use by downstream routines: "amplx" (average signal amplitude =1), "bwx" (the IF bandwidth), "real2x" (burst duty cycle), and "int1x" (number of samples per Gold code chip). The burst formatter outputs a NRZ waveform with a range of plus or minus 1, hence "amplx" is 1. The variable "bwx" is determined by the equation

bwx = Gold code length/((burst interval/deltat) * duty cycle)

Of these variables "amplx" and "bwx" are used as default values by the jammers.

3.2.2.5 Expanders (EXPAND)

The expander module is used twice in the transmitter section to provide data to be used for the receiver section. The expander module takes its binary (zero and one) inputs and converts them to waveforms where a binary zero is converted to a -1 and a binary 1 is converted to a +1. This module is usually used for two purposes: one is to produce a delayed information bit waveform for input to the bit error counter, and the other is to produce a delayed Gold code waveform for input to the despreader. Two variables are required from /xblock/; "reallx" (burst interval) and "deltx" (ICSSM definition).

3.2.3 Modulator Section

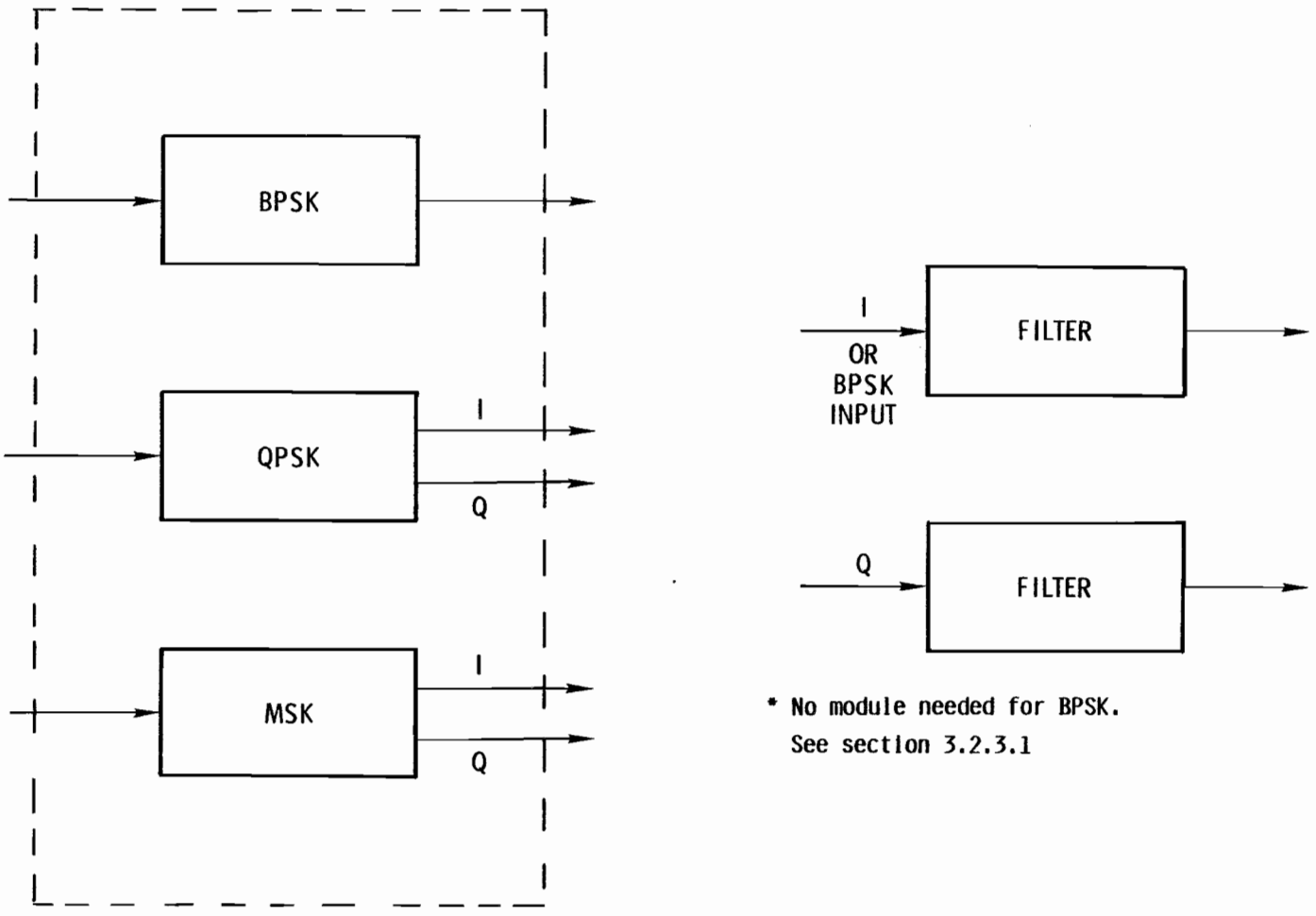
3.2.3.1 BPSK Modulator

Since the burst formatter outputs a waveform, that output is already in BPSK form. There is, therefore, no module that implements a BPSK modulator. (See Figure 3.2 for the modulator section block diagram.)

3.2.3.2 OPSK Modulator (OPSKMD)

The OPSK modulator looks at the system clock (sample time clock) and the number of samples per Gold code "chip" to determine if it has an odd numbered bit (output on the "I" channel) or an even numbered bit (output on the "Q" channel) by the following variables from /xblock/: "ncofx" (ICSSM definition), "reallx" (burst interval), "deltx" (ICSSM definition), "intlx"

Figure 3.2 MODULATOR SECTION



(number of samples per Gold code chip). At the input port each Gold code chip has "int1x" samples. Each output "chip" on the I and Q channels has twice the "int1x" number of samples. In this way the number of bits per baud is kept constant. For possible use by downstream modules (downstream filters, for example) the QPSK modulator sets "encdx" in /xblock/ to 1, to let modules downstream know that some change has taken place in the sampling rate of each chip.

3.2.3.3 MSK Modulator (MSKMOD)

The MSK modulator is a modified version of the QPSK modulator and expects the same variables to be set in /xblock/. Similarly "encdx" in /xblock/ is set to 1 to let downstream modules know that there has been a change in the sampling rate of each chip.

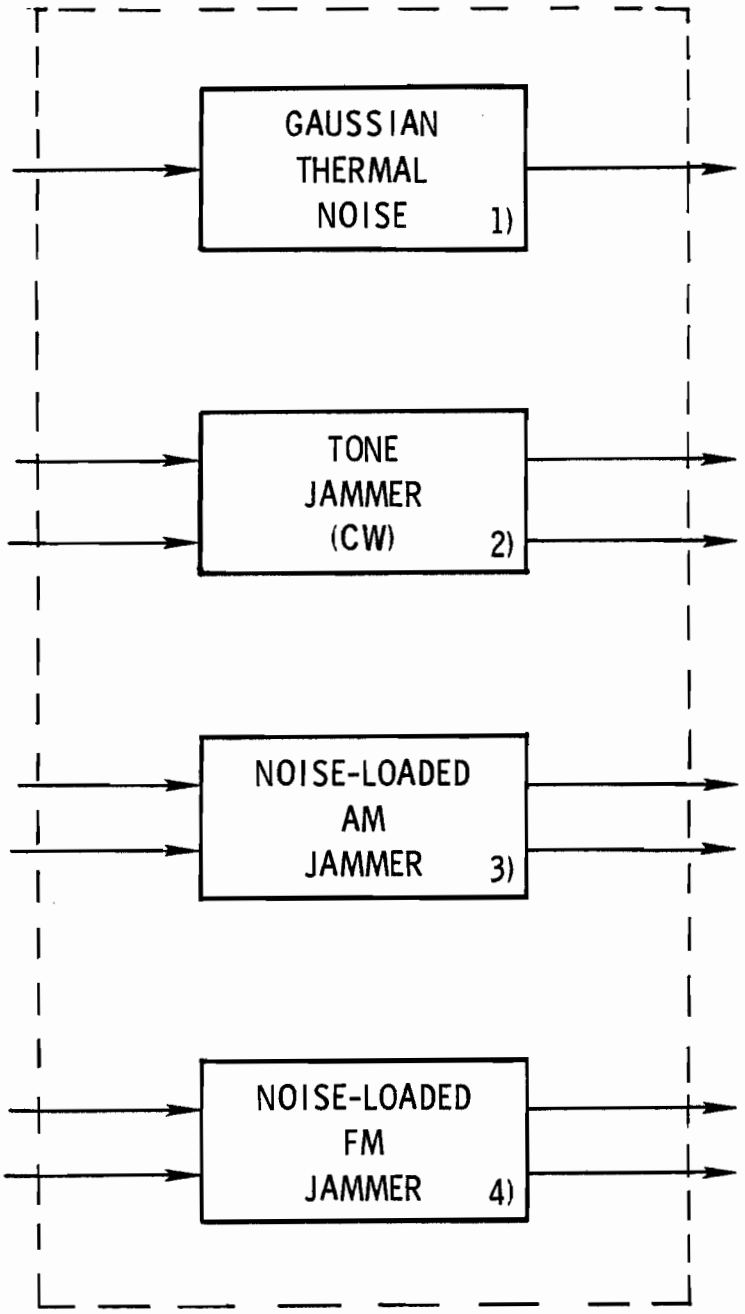
3.2.3.4 Filter (NFILT)

The filter module implements a Chebyshev, Butterworth or an elliptic filter (NFILT). The user can specify the type of filter (low-pass, high-pass, band-pass, or notch), the filter order, the ripple, and the lower and/or upper 3 dB frequencies. Only a real (not complex) filter is currently implemented.

3.2.4 Jammers

Figure 3.3 illustrates the jammer modules. The Gaussian white noise jammer always provides continuous noise output while the other jammers can output intermittently to simulate selective band jamming of frequencies in a frequency-hopped system.

Figure 3.3 JAMMERS



1) Modules NOISE and CNOISE
2) Modules CWJAMF and CWJAMR
3) Module AMJAMR
4) Module FMJAMR

3.2.4.1 Gaussian Thermal Noise (NOISE and CNOISE)

There are two implementations of noise generators, one is a real noise generator, the other is a complex noise generator. Both noise generators use the same formulae to compute the variance of the noise waveform samples. The formulae are as follows:

```
      etaa=s*(10.**-(snr/10.)/bwd)
      noise=sqrt(etaa/(2.*delt)) *(gaussian random number)
c
c           where s=average signal power
c                 snr=desired signal to noise ratio
c                 bwd=noise bandwidth
c                delt=simulation time between contiguous samples
c                     (this is the ICSSM definition)
c
```

For the complex noise implementation the "noise=" statement is repeated with different Gaussian random number for use in the imaginary part. The following variables are required from /xblock/: "bwx" (IF bandwidth used as a default value if the user enters none), "amplx" (average signal amplitude used as a default value if the user enters none), and "deltx" (standard ICSSM definition). These modules add noise to their inputs.

3.2.4.2 Other Jammers

The other jammers look at variable "encdx" in /xblock/ to see if a real (encdx=0) or complex (encdx=1) jammer is needed. These jammers are the noise-loaded AM jammer, the noise-loaded FM jammer, and the CW jammer (tone jammer). These jammers require two variables be set by the user to simulate jamming of some bands available for frequency hopping: the number of bands available for frequency hopping (NBANDS) and the number of bands to jam (NJAM). The jammer-to-signal power ratio will be evenly spread over all NJAM

bands, hence if a user wants all NBANDS to be jammed at full power, the user must set NBANDS=1 and NJAM=1.

A uniform random number generator is used to generate probabilities. If the probability generated is less than the ratio (1/NJAM), jamming is generated during the current burst interval. In this way, for baseband simulation purposes, frequency hopping is simulated, as some "bands" will be jammed while others will not.

Even though the jammers can be either real or complex, due to ICSSM topology requirements, both of the input ports and output ports must be connected. This is usually by-passed by connecting the real output port form the preceeding module to both input ports and the imaginary output to a terminator module. The following variables are required from /xblock/: "amplx" (ICSSM definition), "bwx" (IF bandwidth), "encdx" (as above), "ncofx" (ICSSM definition), "reallx" (burst interval), "real2x" (duty cycle of burst), and "intlx" (number of samples per Gold code chip). Variables "amplx" and "bwx" are default values in the event that the user does not set the user-settable amplitude and bandwidth variables.

3.2.4.2.1 AM Jammer (AMJAMR)

The noise-loaded AM jammer implements the following equations:

```
sj=dbsgpw +10.*alog10(njam)
cnum=gnois3(sj,i.f. bandwidth,average signal power,seed,deltx)
real signal=real signal + real(cnum)
imaginary signal=imaginary signal + aimag(cnum)
c
c       where gnois3 is a complex white noise generator as expressed
c       in the white noise section above
c
c       sj= modified jammer to signal ratio
c       njam= number of bands to jam
c       real(cx) gets the real part of a complex number cx
c       aimag(cx) gets the imaginary part a complex number cx
c
```

3.2.4.2.2 CW Jammer (CWJAMF and CWJAMR)

The CW jammer (tone jammer) has two versions, one in which the user fixes a single frequency and phase offset (cwjamf) and another that chooses uniformly distributed random frequency and phase offsets (cwjamr). Note that module cwjamr continuously chooses different offsets for every burst. The CW jammer implements the following equations:

```
    rsj=10.**(dbsgpw/10.) * avepwr
    aj=sqrt(rsj/float(njam))
c
c        if fixed frequency and phase offset is provided
c        by the user skip to 15
c
    if(fix) go to 15
    woff=uniform random(-bandwidth/2.,bandwidth/2.)
    theta=uniform random(-pi,pi)
c
15  temp=woff* sample clock +theta
    real signal=real signal + aj*cos(temp)
    imaginary signal=imaginary signal + aj*sin(temp)
c
c
c   where
c   dbsgpw = jammer to signal power in db
c   bandwidth = specified bandwidth within which to choose an offset
c   woff = frequency (omega) offset
c   theta = phase offset
```

3.2.4.2.3 FM Jammer (FMJAMR)

The FM jammer is essentially the CW jammer with two modifications, a three db bandwidth and a hold have been added. The hold is the number of samples that a random number is used to noise load the FM carrier. (If hold is set to 1 the noise spectrum generated will have a Laplacian shape, otherwise the noise spectrum will have a Gaussian shape.) The FM jammer implements the following additional equations:

```
    bw=bw3db*delt
    if(hold.ne.1) go to 30
    bw=bw/pi
```

```
      bw=sqrt(bw)
30    nend=ncofx/hold
      n=0
      temp2=bw*2.*pi
      sum=0.
      do 10 i=1,nend
      gauss=gaussian random number (mean 0, standard deviation 1)
      do 25 j=1,hold
      temp1=woff*clock1 +theta
      sum=sum+gauss
      temp=sum*temp2
      n=n+1
      term1=cos(temp)*cos(temp1) -sin(temp)*sin(temp1)
      real signal(n)=real signal(n) +aj*term1
      term2=cos(temp)*sin(temp1) +sin(temp)*cos(temp1)
      imaginary signal(n)=imaginary signal(n) +aj*term2
25    clock1=clock1+delt (sample clock)
10    continue
c     where
c     hold = number of samples to hold noise value
c     term1 = fm term for real part of signal
c     term2 = fm term for imaginary part of signal
c     bw3db = 3 db bandwidth of the fm jammer
c     woff = frequency offset
c     theta = phase offset
c
```

3.2.5 Demodulators

3.2.5.1 BPSK

Since BPSK in baseband modeling is the signal waveform itself there are no BPSK modulators or demodulators (see Figure 3.4).

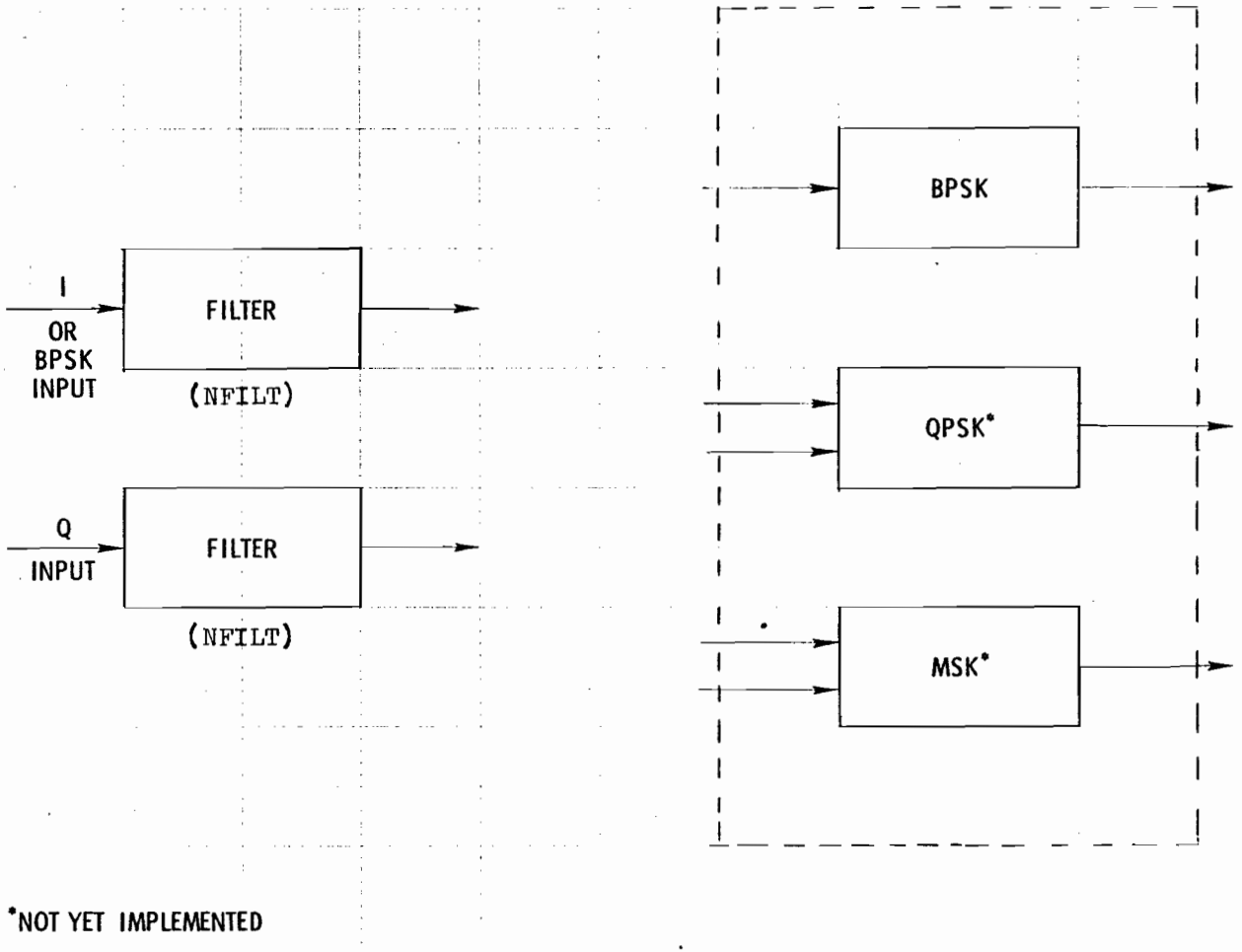
3.2.5.2 OPSK and MSK

Currently these are not implemented.

3.2.6 Receiver

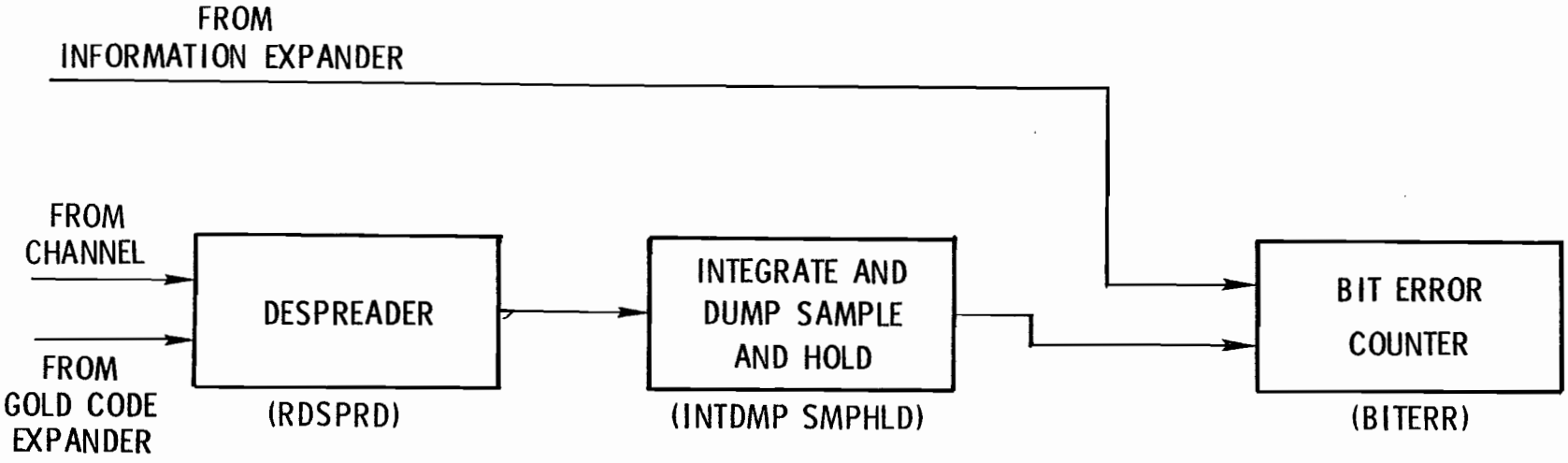
The receiver consists of a despreader, a sample and hold, and a bit error counter (see Figure 3.5).

Figure 3.4 DEMODULATORS



*NOT YET IMPLEMENTED

Figure 3.5 RECEIVER



3.2.6.1 Despreader (RDSPRD)

The despreader takes a filtered channel waveform and multiplies it by the Gold code waveform to compute the despreader's output. The despreader requires two variables from /xblock/: "fzx" (sync or data burst, set in the frame formatter) and "ncofx" (ICSSM definition). Note: there is no despreader for the cyclic shift spreading that is available in the burst formatter.

3.2.6.2 Integrate and Dump and Sample and Hold (INTDMP and SMPHLD)

The integrate and dump module will integrate an input signal over a given number of samples with a specified initial delay and dump the accumulated integration after the given number of samples have been integrated. At this point, if the sample and hold module is used, the sample and hold module will check the integration value. If the integration value is positive a "+1." value will be held over the succeeding integration time; if the integration value is negative a "-1." value will be held over the succeeding integration time. The variable "ncofx" (ICSSM definition) is required from /xblock/.

3.2.6.3 Bit Error Counter (BITERR)

The bit error counter inputs two bit streams and counts the differences between them at specified regular intervals in the bit stream. This is a terminator module. The counting of the differences is triggered by a real mode comparison for equality, i.e., by a FORTRAN "if" statement of the following form:

```
    if(x.eq.y) go to 30
c
c       where "x" is the output from the sample and hold
c       and "y" is the output from the expander module
c       applied to the information bit sequence generator
c
```

By this method, when an inequality occurs between "x" and "y" the count of bit errors is incremented. Were the bit error counter to be applied to an integrate and dump waveform as one of the inputs the real mode compare would most likely count each comparison point as an error, since the output waveform from the integrate and dump module seldom will reach the exact "+1." and "-1." values.

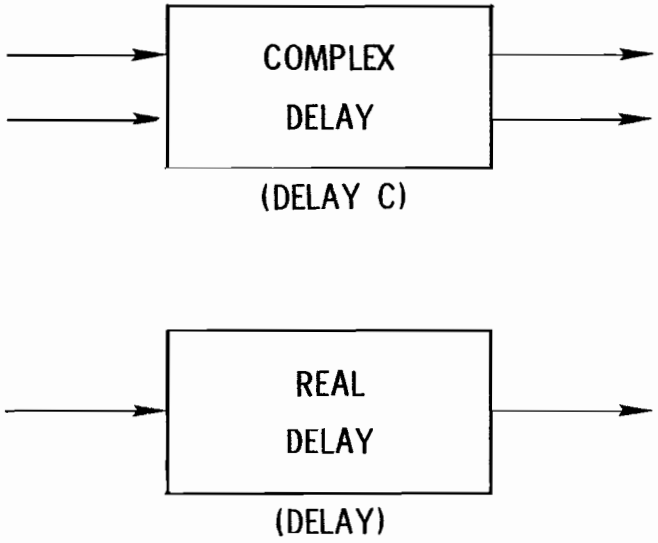
Variables needed from /xblock/ are "tdurx" (ICSSM definition) and "tstopx" (ICSSM definition). These two variables along with "tnow" from /exec/ are needed to tell this terminator module when its last run (the invocation corresponding to the final "tdur") will be so that the results of the bit error counter will be output to file "file36" during the last invocation. (There is also an option to specify the number of bit errors to terminate the simulation after, in which case the results will likewise be output to "file36".)

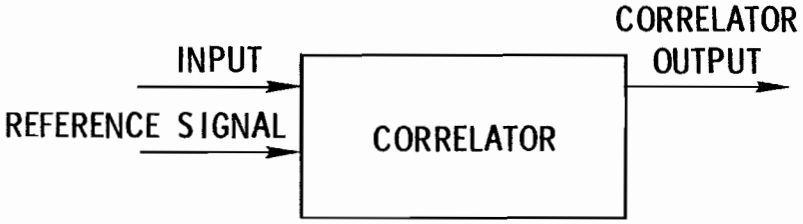
3.2.7 Support Routines

Since delays are already provided in the expander module, our spread-spectrum system does not use a stand-alone delay module (figure 3.6). However, future configurations may need delay modules, so they have been provided.

The correlator may be used to detect sync Gold code bursts in a noisy channel or to estimate the delay through the spread-spectrum system. As originally envisioned the correlator is in its own model partition, i.e., the simulation may be split into different segments with the correlator being the sole resident of one of the segments.

Figure 3.6 SUPPORT MODULES





3.2.7.1 Delays (DELAY and DELAYC)

The real delay module, DELAY, will allow delays of up to 2090 samples. This limit is determined by the amount of locally saved storage in hidden parameters of /param/. If delays of greater than 2090 samples are entered the module will output a message to the user's terminal and the simulation will stop.

The complex delay module will allow delays of up to 1045 samples and behave similarly to the real delay module in case this limit is exceeded. Both modules require "ncofx" from /xblock/.

3.2.7.2 Correlators (CORLR and HCHCLR)

The correlator module will do a sliding dot product of the reference waveform and another input waveform. The sliding dot product is output. This module is usually used either to detect a sync Gold code in a noisy channel or to estimate the system delay. Once this delay estimate is made, delays can be set in the delay section of the expander module for the Gold code so that the despreader will have the correct input from the Gold code expander.

The variable "ncofx" is required from /xblock/.

Because multiple invocations of the correlators are required to calculate the correlations (because the burst interval is usually at least twice as large as "tdur") previous input data from both input ports must be saved. These data are saved in files "file66" (for signal input) and "file67" (for the reference signal). Note that the reference signal is only ingested during the first burst interval time. Because the correlator calculates correlations for an entire burst interval time there will be too many samples to output

(more than ncof points) hence the remainder of these points must also be saved to a file, "file68".

All of the above 3 files are random access files.

3.3 ICSSM Module Documentation

This section contains block diagrams and documentation for each module. In each block diagram arrows going left to right indicate signal flow, while arrows going from top to bottom indicate variables needed by each module (incoming arrows at the top) from common /xblock/ and (outgoing arrows at the bottom) variables explicitly set by this module to be sent downstream to subsequent modules in common /yblock/.

Inside each module block are port numbers and user settable variables in commons /param/ and /iparam/. Following each block diagram is the internal documentation that further explains port usage, input and output variables, and user-settable variables.

Originally, names given to modules were descriptive, i.e. AMJAMR is the AM jammer. In order to conform to ICSSM naming conventions the modules were renamed (Appendix C, ICSSM User's Manual). The following list indicates the original module names and their current ICSSM names.

Below Table 3.3 is a cross correlation list:

KU	ICSSM	PURPOSE
AMJAMR	CJMR03	AM jammer
BITERR	MBCU01	Bit error estimator
BURSTF	TSGU01	Burst formatter
CNOISE	CWGC02	Complex gaussian noise
CORLR	UCRU02	Correlator
CWJAMF	CJMF01	Fixed frequency CW jammer
CWJAMR	CJMR01	Random frequency CW jammer
DELAY	UDLU01	Real delay
DELAYC	UDLC01	Complex delay
FMJAMR	CJMR02	FM jammer
GOLDCD	TGCU01	Gold code generator
HCHCLR	UHCU01	Half chip correlator
INFORM	TIGU01	Information generator
INTDMP	GITU01	Integrate and dump
NOISE	CWGU01	Gaussian noise
QPSKMD	GMDU01	QPSK modulator
SMPHLD	GSHU01	Sample and hold
MSKMOD	TMDU01	MSK modulator
NFILT	GFLU01	Filter
EXPAND	UEXU01	Binary to waveform expander
FRAMEF	TINU01	Frame formatter
RDSPRD	GDSU01	Real despreader

Table 3.3

Module Name Crosscorrelation Table

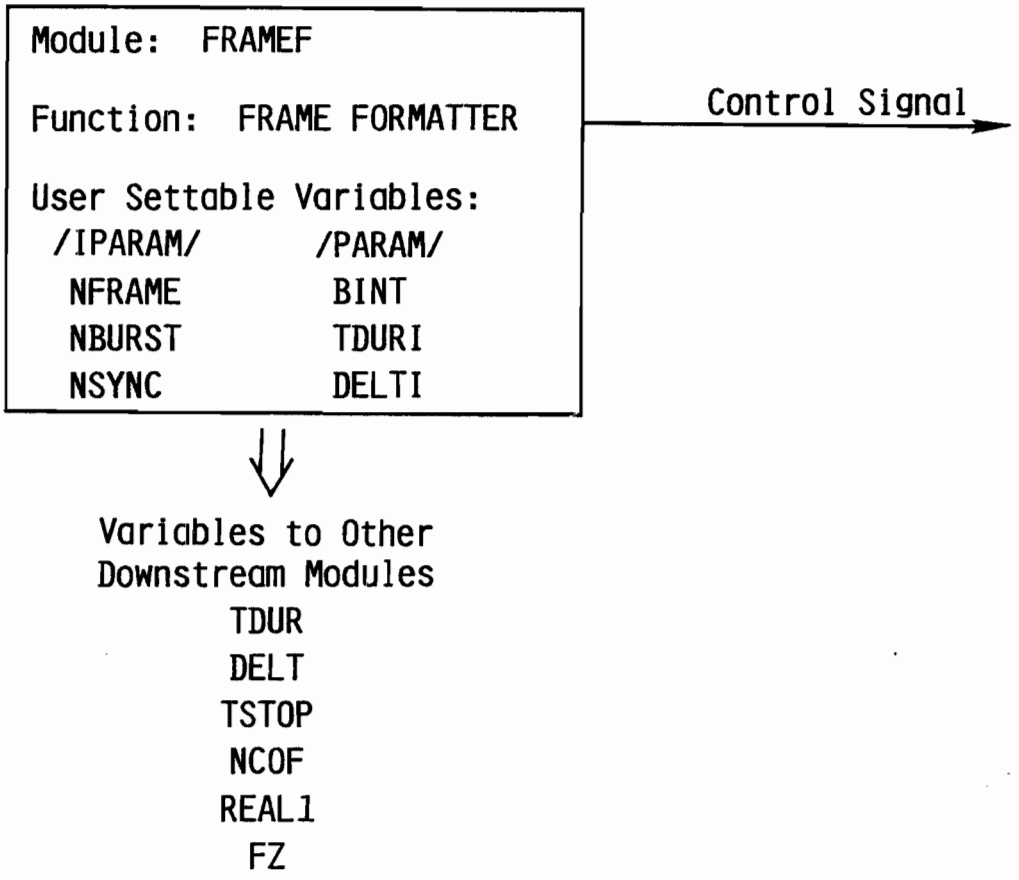


Figure 3.7 Frame Formatter

```
c  framef       frame formatter
c
c  identification
c
c       title         framef
c       author        c.paul
c       date          2/22/83
c       language      fortran
c       computer      honeywell
c       operating environment - icssm
c       opsys         multics
c  purpose
c
c       this routine drives the burst formatter and information generator
c       for the signal transmitter model
c
c   port description
c
c     port    direction   description
c
c       1         output     control signal to information generator
c                            and gold code generator
c
c  user settable parameters
c
c  in common /iparam/
c
c       nframe  int      number of frames to produce
c       nburst  int      number of bursts per frame (including sync)
c       nsync   int      number of sync bursts per frame
c
c  in common /param/
c
c       bint    real     busrst interval time
c       tduri   real     time duration during routine invocation
c       delti   real     time increment between samples
c
c  ** note **
c
c       due to icssm limitations tduri/delti must be less than
c       or equal to 4096
c
c  ** note **
c
c       due to transmitter model limitations bint must be evenly divisible
c       by tduri, else the frame formatter will execute a stop statement
c
c  variables in /yblock/ set by this routine
c
c       tdur    real     tduri
c       delt    real     delti
c       tstop   real     simulation stop time
c       ncof    int      number of samples per invocation to produce
c       reall   real     burst interval (bint)
c       fz      real     =0. tells burst formatter to expect information
c                        at burst formatter input port 2
c                             and tells information producer to produce info
```

Figure 3.7a Frame Formatter

```
c                       =1. tells burst formatter to ignore burst formatter
c                           input port 2 because the current burst is a
c                           sync burst
c                            and tells the information producer to return
c                            without producing information (it won°t sigout
c                            to its output port)
c
c  subroutines required outside of icssm
c
c      modf    integer function that is modulus of two real numbers
c
c*************************************************************************
c
```

Figure 3.7b Frame Formatter

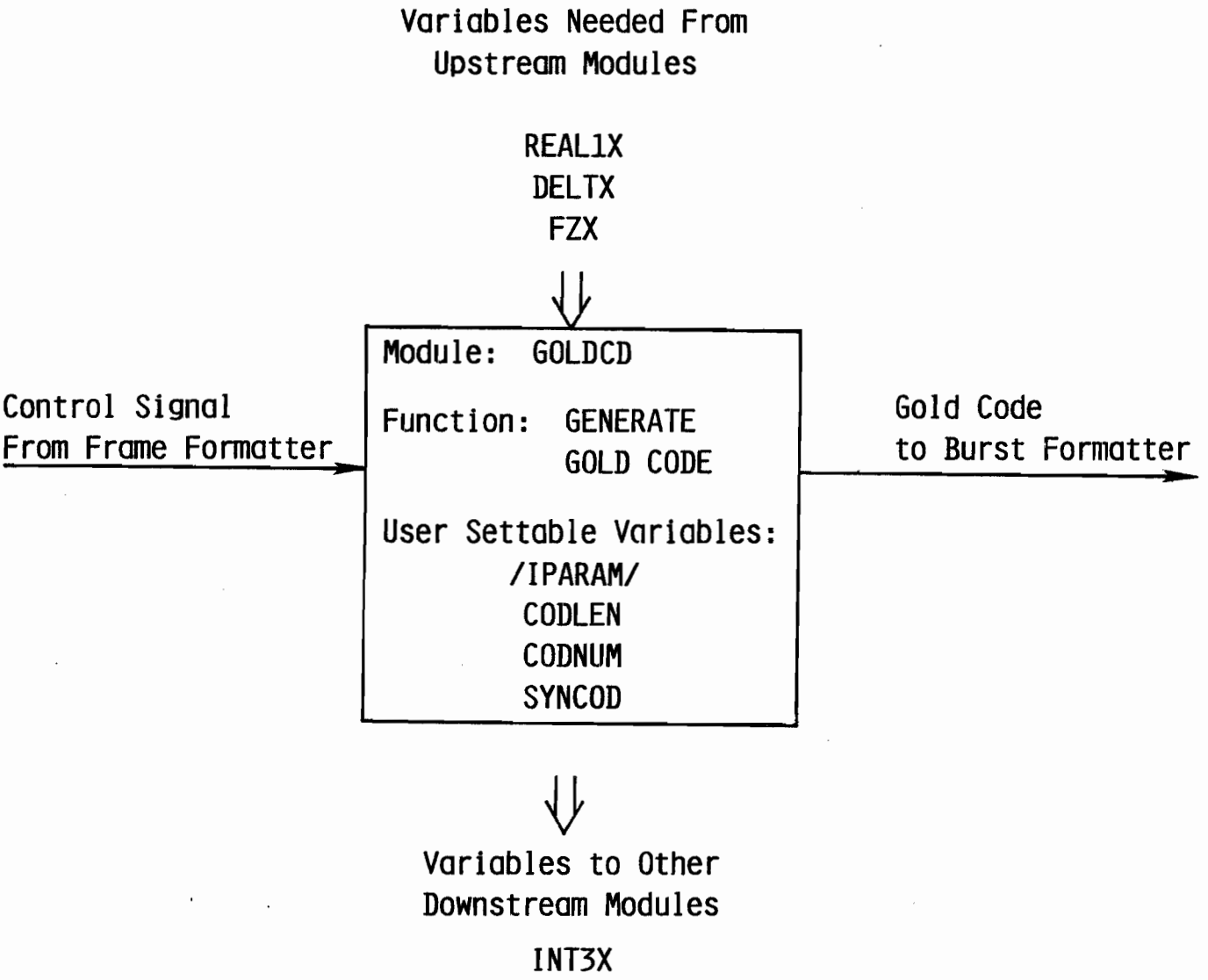


Figure 3.8 Gold Code Generator

```
c  goldcd      generate gold code to send to burst formatter
c
c  identification
c
c      title       goldcd
c      author      c.paul
c      date        2/22/83
c      language    fortran
c      computer    honeywell
c      operating environment - icssm
c      opsys       multics
c  purpose
c
c      this routine will generate a gold code to send
c      to the burst formatter
c
c    port description
c
c    port   direction   description
c
c     1       input      control signal from frame formatter
c     2       output     gold code bit stream output
c
c  user settable parameters in common /iparam/
c
c      codlen  int     gold code length (power of 2 but not 2**4,2**8, etc)
c      codnum  int     data gold code
c      isyncd  int     sync gold code
c
c  variables in /xblock/ set by this routine
c
c      int3x   int     gold code length
c
c  variables needed from /xblock/
c
c      reallx  real    burst interval time (bint) set by frame formatter
c      deltx   real    time increment between samples (set by frame formatter)
c      fzx     rea;    =0. gen. data gold code, =1. gen. sync gold code
c
c  subroutines required outside of icssm
c
c      gengld  generate gold code
c      modf    integer function modulus of 2 real numbers
c
c*************************************************************************
c
```

Figure 3.8a Gold Code Generator

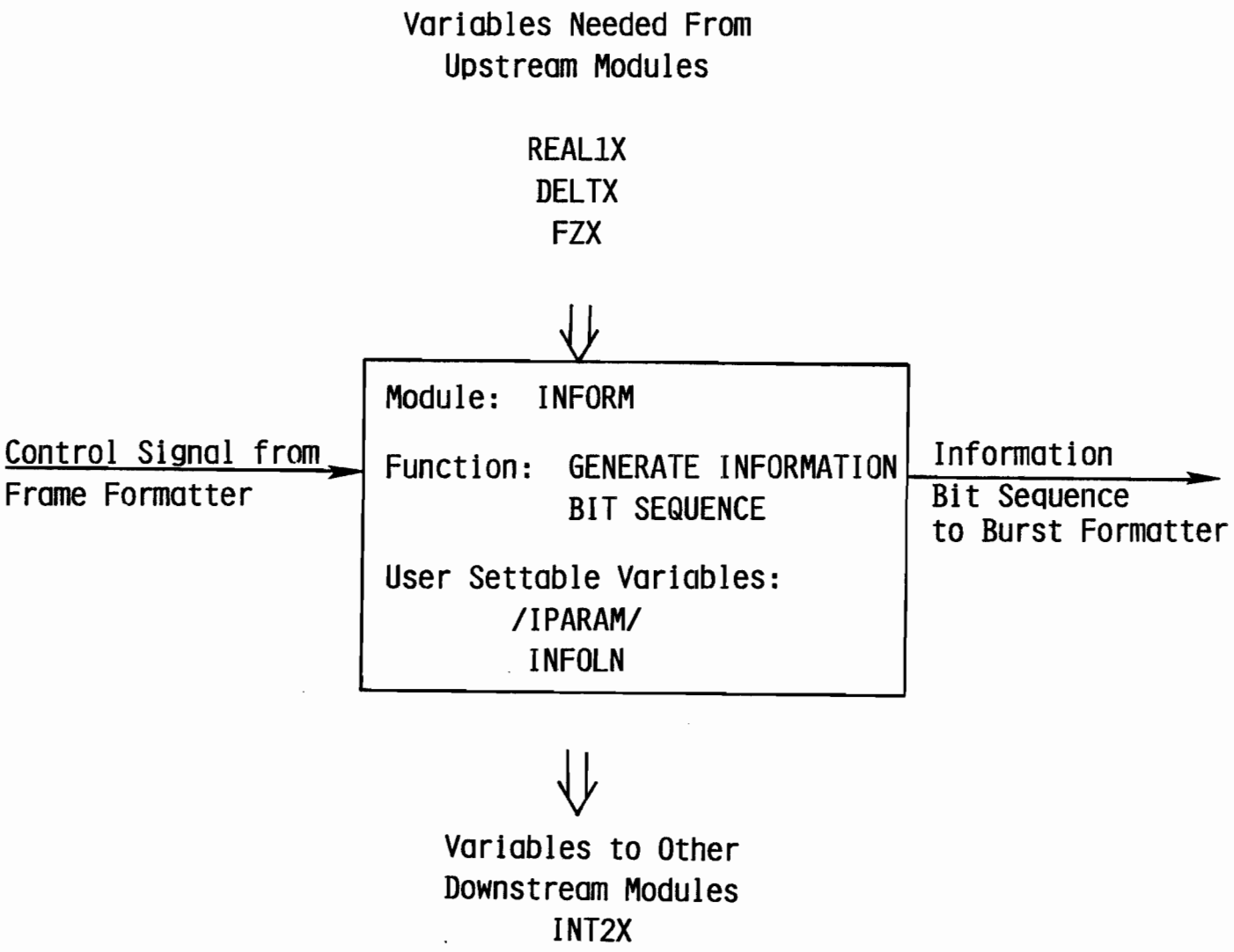


Figure 3.9 Information Bit Sequence Generator

```
c  inform      generate information to send to burst formatter
c
c  identification
c
c      title        inform
c      author       c.paul
c      date         2/22/83
c      language     fortran
c      computer     honeywell
c      operating environment - icssm
c      opsys        multics
c  purpose
c
c      this routine will generate information to send
c      to the burst formatter
c
c   port description
c
c     port   direction   description
c
c      1       input      control signal from frame formatter
c      2       output     information bit sequence output
c
c  user settable parameters in common /iparam/
c
c      infoln  int      number of information bits per burst
c
c  variables in /xblock/ set by this routine
c
c      int2x   int      # of information bits per burst
c
c  variables needed from /xblock/
c
c      reallx  real     burst interval time (bint) set by frame formatter
c      deltx   real     time increment between samples (set by frame formatter)
c      fzx     real     =1. don't produce info, sync is needed
c                       =0. produce info
c                       fzx is set by the frame formatter at the
c                       appropriate time
c
c  subroutines required outside of icssm
c
c      modf     integer function modulus of 2 real numbers
c
c*************************************************************************
c
```

Figure 3.9a Information Bit Sequence Generator

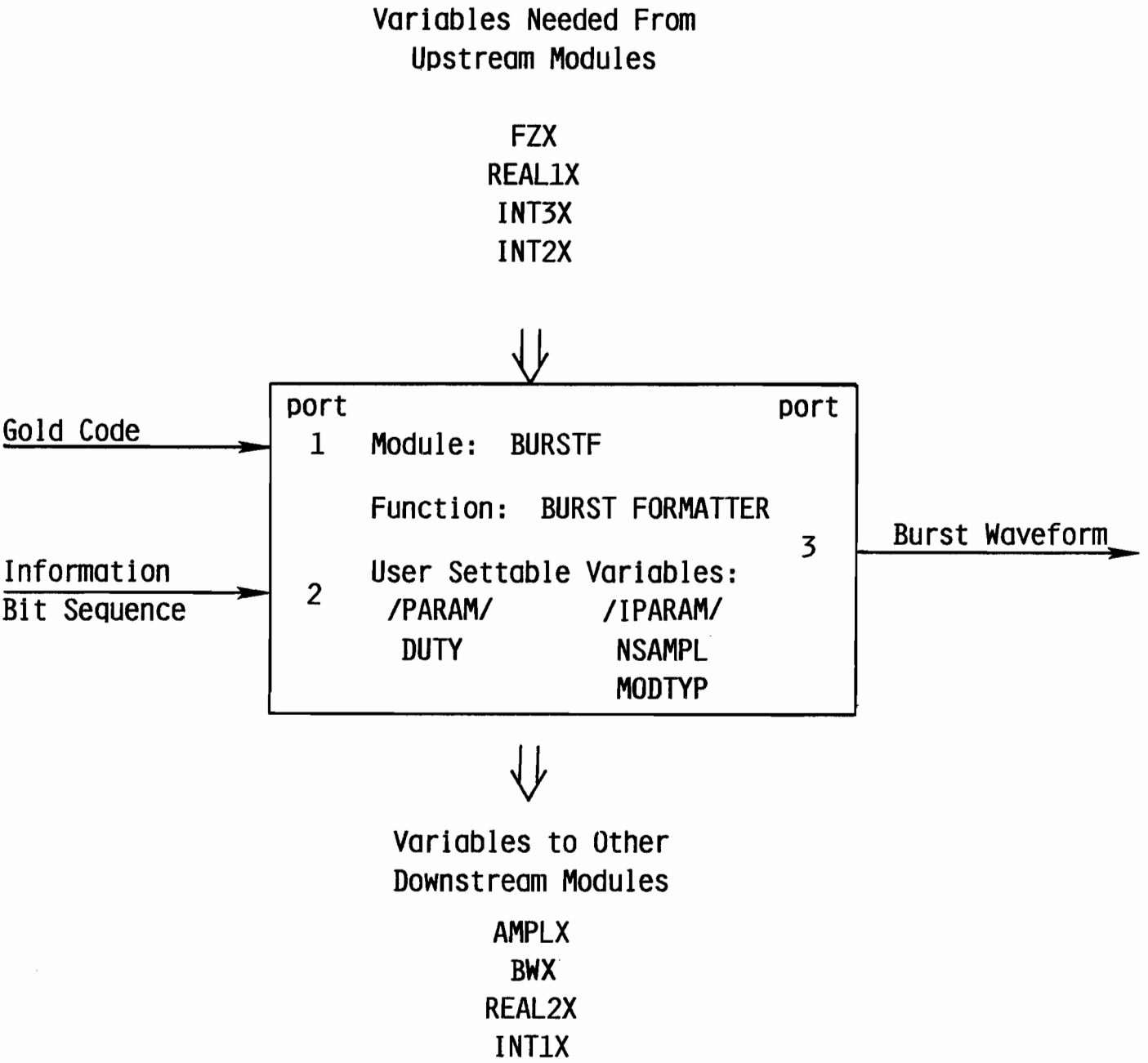


Figure 3.10 Burst Formatter

```
c  burstf      burst formatter shell for icssm
c
c  identification
c
c       title    burstf
c       author   c.paul
c       language fortran
c       date     2/22/83
c       computer honeywell
c       operating environment - icssm
c       opsys    multics
c  purpose
c
c       this is the icssm shell for the burst formatter
c       burst amplitude is 2 mean 0 (1 bit is +1. 0 bit is -1.)
c
c  port callout
c
c     port#      direction   description
c
c       1          input       retrieve gold code, length specified in int3x
c                              of /xblock/
c       2          input       retrieve information, length specified in int2x
c                              of /xblock/
c                              this port is inactive if fzx=1.
c                              fzx is set to 1. by the frame formatter when
c                              this burst is to be sync
c       3          output      burst output (ncofx long)
c
c  user settable parameters
c
c  in common /iparam/
c
c       nsampl   int       number of samples per gold code chip
c       modtyp   int       modulation type. =0 cyclic shift coding
c                          =1 direct multiplication
c
c  in common /param/
c
c       duty     real      duty cycle of burst (fraction of burst
c                          that the gold code occupies)
c
c  variables required from common /xblock/
c
c       fzx      real      set to 1. by frame formatter to indicate
c                          that this burst will be sync, hence no information
c                          is retrieved from the information port
c       reallx   real      burst interval time (bint=reallx)
c                          set by frame formatter
c       int3x    int     gold code length (set by gold code generator)
c       int2x    int     # of info bits per burst (set by info generator)
c
c  variables set here for common /xblock/
c
c       amplx    real      average signal amplitude  =1.
c       bwx      real      i.f. bandwidth = (codlen/((bint/delt)*duty))
c       real2x   real      duty cycle of burst
```

Figure 3.10a Burst Formatter

```
c                          (user specified in /param/ duty)
c      intlx    int        # of samples per gold code chip
c                          (user specified in /iparam/ nsampl)
c
c  subroutines required outside of icssm
c
c      brstfm   burst formatter
c
c*************************************************************************
c
```

Figure 3.10b Burst Formatter

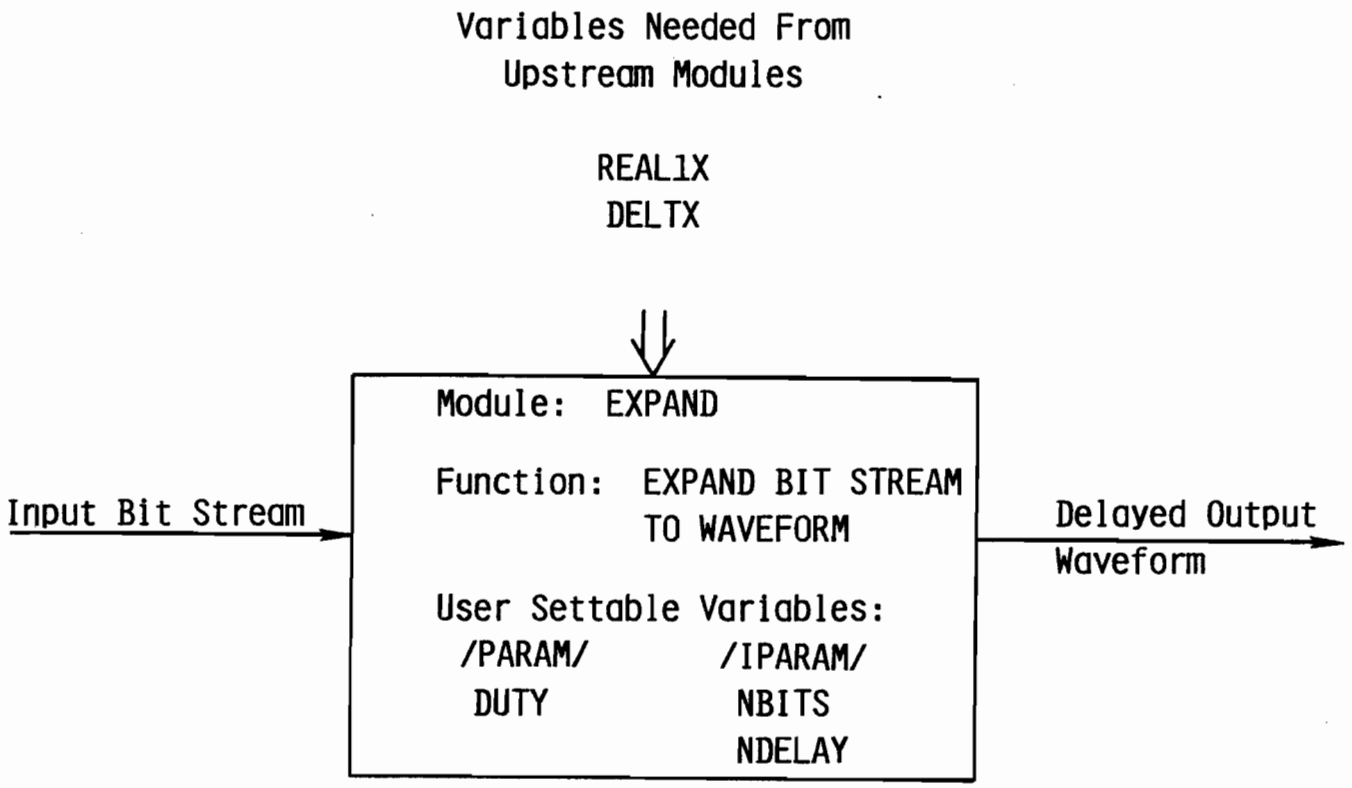


Figure 3.11 Bit Stream Expander

```
c  expand    expand a real binary bit stream to a signal format
c
c  identification
c
c     title    expand
c     version  a
c     author   c.paul
c     date     4/20/83
c     language fortran66
c     computer honeywell
c     opsys    multics
c  purpose
c
c      this routine will take in a binary bit stream in real
c      form (0. for binary 0 and 1. for binary 1) and output
c      a waveform where binary 0s are changed to -1. and binary
c      1s are changed to +1.  The bit stream is also expanded
c      to fill the number of samples occupied by the burst interval
c      active section (bint/delt *duty)
c
c   user settable parameters in /param/
c
c      duty   real   duty cycle of burst
c
c  user settable parameters in /iparam/
c
c      nbits    int     number of bits in input binary stream
c      ndelay   int     number of samples to delay output
c
c  arguments required from common /xblock/
c
c      reallx   real    burst interval (bint)
c
c  port description
c
c    port   direction   description
c
c      1       input       bit stream input
c      2       output      signal stream output
c
c ** note **
c
c      this routine will usually be used for two purposes:
c
c      1)  to produce an information bit stream for input
c          to the bit error counter
c
c      2)  to produce a gold code bit stream for input to the
c          despreader
c
c  subroutines required outside of icssm
c
c    modf   real modulus function
c    rdelay real delay
c
c*********************************************************
```

Figure 3.11a Bit Stream Expander

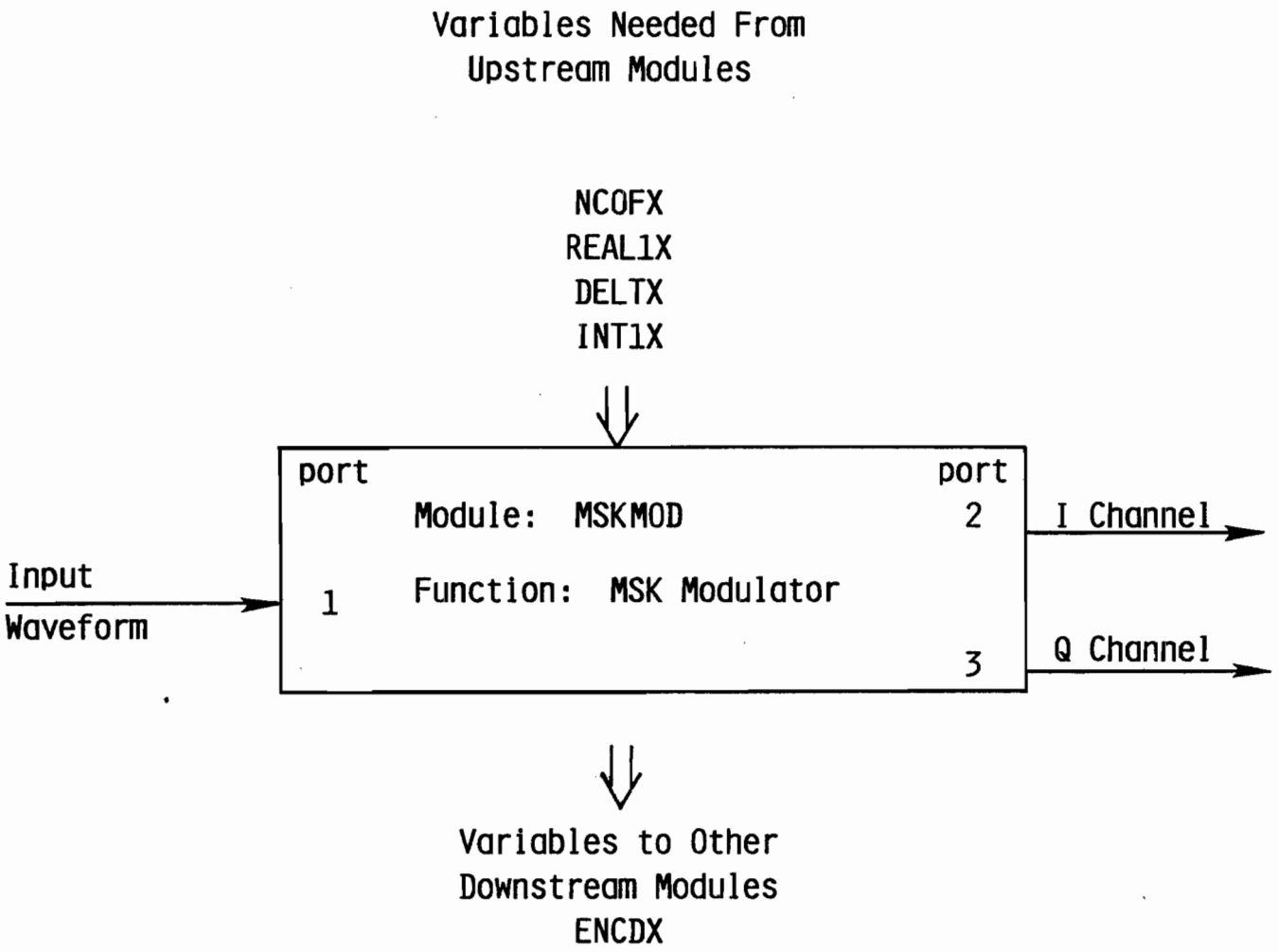


Figure 3.12 MSK Modulator

```
c  mskmod    msk modulator
c
c  identification
c
c       title       mskmod
c       date        2/11/83
c       author      c.paul
c       language    fortran
c       computer    harris (honeywell)
c       operating environment  icssm
c purpose
c
c       given an input stream two output streams will be created
c       with each stream receiving alternate bits.  note that
c       if there are x samples per input bit (nsampl) each output bit
c       will have 2x samples (nsampl*2).
c       In this implementation of the msk modulator
c       the qpsk routine is called, then
c       the q stream is delayed by nsampl samples.
c       Then both waveforms are multiplied by half
c       sine waves (q channel delayed) to arrive
c       at the msk modulator.
c
c  port description
c
c    port   direction  description
c
c      1        input      signal input
c      2        output      i channel
c      3        output       q channel
c
c  arguments from xblock
c
c       ncofx    int      number of samples to process in ublock
c       reallx   real     reallx=bint  burst interval time
c       deltx    real     time increment per sample
c       intl     int      intl=nsampl number of samples per bit
c
c   arguments from exec
c
c       tnow     real     current simulation time
c
c  arguments set by this routine
c
c       in common /xblock/
c
c       encdx    real     let everyone downstream know there is msk
c
c  subroutine required
c
c       msk     msk modulator subroutine
c       modf     real modulus function (called by msk)
c
c*************************************************************************
c
```

Figure 3.12a MSK Modulator

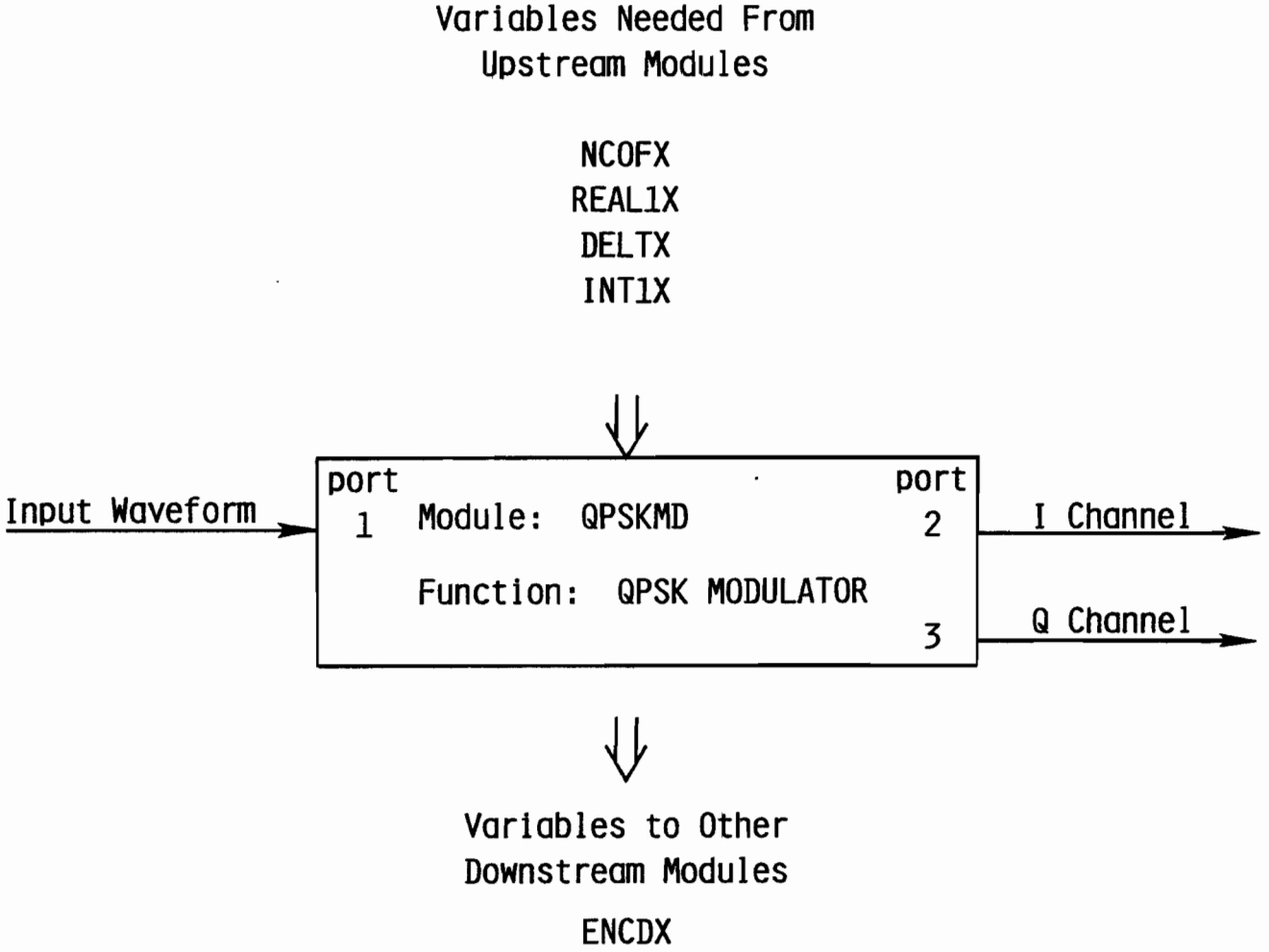


Figure 3.13 QPSK Modulator

```
c  qpskmd    qpsk modulator
c
c  identification
c
c       title       qpskmd
c       date        2/11/83
c       author      c.paul
c       language    fortran
c       computer    harris (honeywell)
c       operating environment  icssm
c purpose
c
c       given an input stream two output streams will be created
c       with each stream receiving alternate bits.  note that
c       if there are x samples per input bit (nsampl) each output bit
c       will have 2x samples (nsampl*2)
c
c  port description
c
c    port   direction  description
c
c      1        input      signal input
c      2        output      i channel
c      3        output       q channel
c
c  arguments from xblock
c
c       ncofx   int      number of samples to process in ublock
c       reallx  real     reallx=bint  burst interval time
c       deltx   real     time increment per sample
c       intl    int      intl=nsampl number of samples per bit
c
c   arguments from exec
c
c       tnow    real     current simulation time
c
c  arguments set by this routine
c
c       in common /xblock/
c
c       encdx   real     let everyone downstream know there is qpsk
c
c  subroutine required
c
c       qpsk    qpsk modulator subroutine
c       modf    real modulus function (called by qpsk)
c
c*************************************************************************
c
```

Figure 3.13a QPSK Modulator

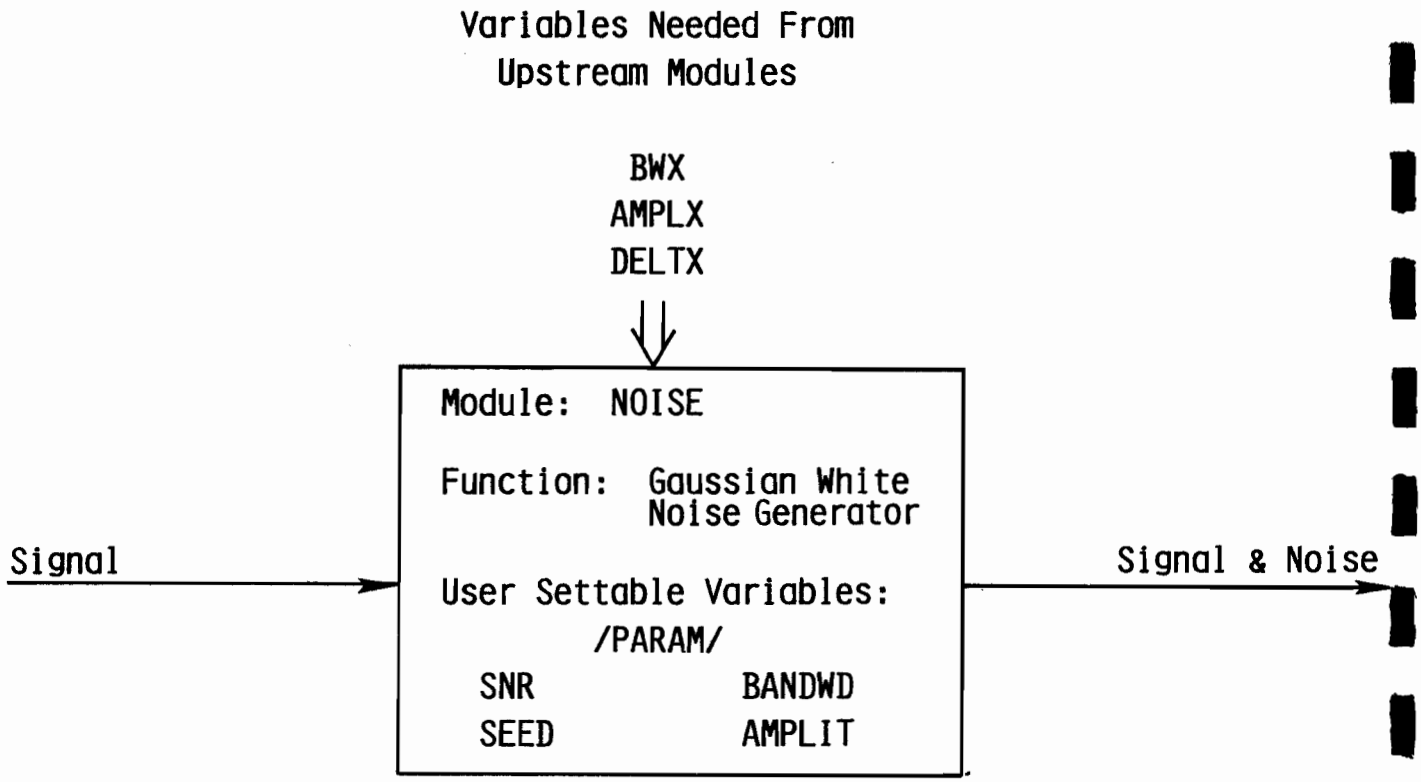


Figure 3.14 Gaussian White Noise Generator

```
c  noise        real gaussian noise generator
c
c  identification
c
c      title    noise
c      version  a
c      date     2/9/83
c      author   c.paul
c      language fortran
c      operating environment - icssm
c      opsys    multics
c      machine  hw
c  purpose
c
c      this routine uses a gaussian random # generator
c      and produces gaussian white noise with the proper variance
c
c   port description
c
c   port   direction    description
c
c    1       input       signal input
c    2       output      signal output with noise
c
c
c  user settable parameters
c
c  in /param/
c
c      snr      real    signal to noise ratio in db
c      seed     real    seed for random # generator
c      amplit   real    average signal amplitude
c                       ** if set to 0. this will default to
c                          amplx in /xblock/
c      bandwd   real    rf or if bandwidth
c                       ** if set to 0. this will default to
c                          bwx in /xblock/
c
c  variables required from common /xblock/
c
c       enb=bwx          if bandwidth (usually nchips/((bint/delt)*duty))
c       s=amplx          signal power
c       delt=deltx       time increment between samples
c
c
c  subroutines required outside of icssm
c
c      gnois3 complex gaussian white noise generator
c      gnois2 white noise generator
c      rgauss uniform to gaussian # generator
c      ranup  update seed for random num. generator
c      ranu   uniform random number generator
c
c*****************************************************************
c
```

Figure 3.14a Gaussian White Noise Generator

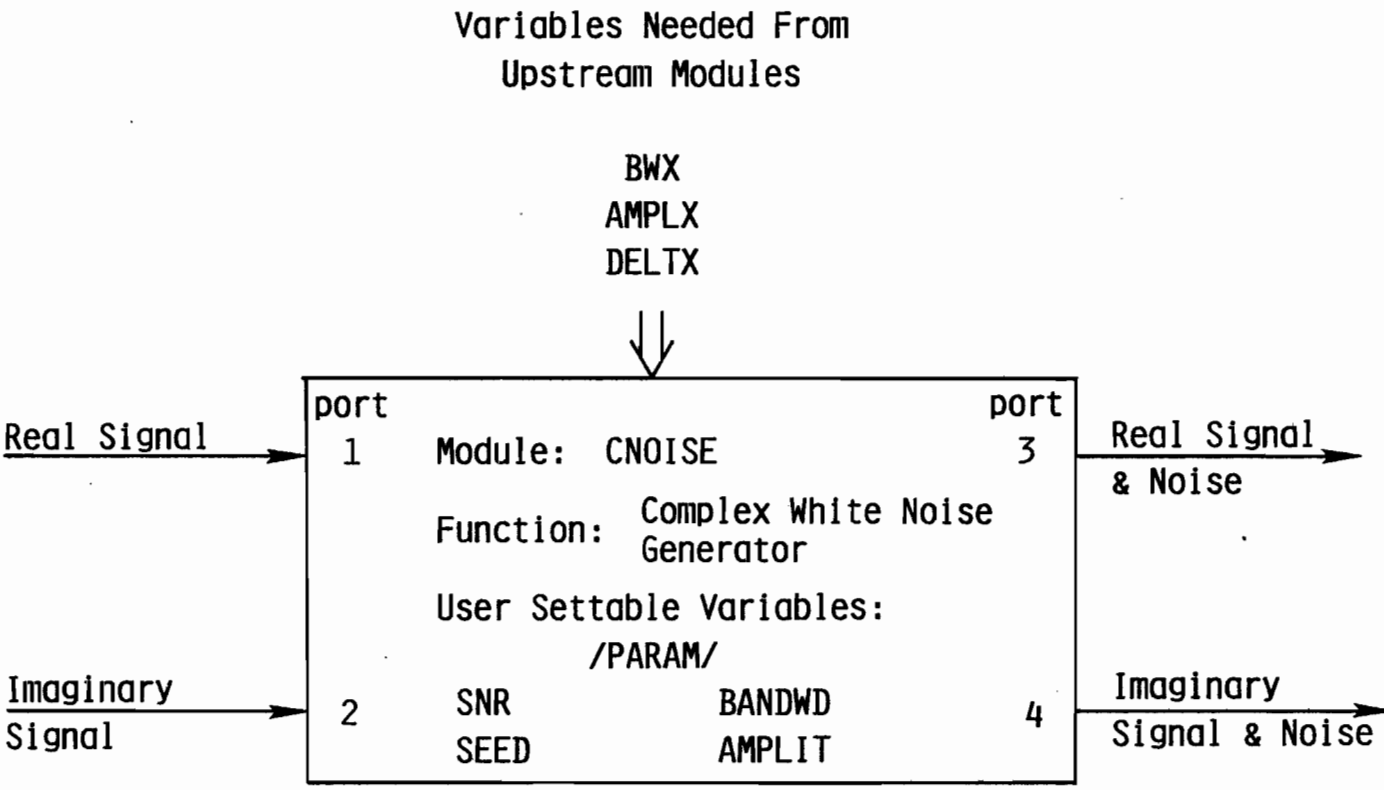


Figure 3.15 Complex White Noise Generator

```
c  cnoise        complex gaussian noise generator
c
c  identification
c
c       title    cnoise
c       version  a
c       date     2/9/83
c       author   c.paul
c       language fortran
c       operating environment - icssm
c       opsys    multics
c       machine  hw
c  purpose
c
c       this routine uses a gaussian random # generator
c       and produces complex gaussian white noise with the proper variance
c
c  port description
c
c    port   direction   description
c
c      1       input       real signal input
c      2       input       imaginary signal input
c      3       output      real signal output
c      4       output      imaginary signal output
c
c
c  user settable parameters
c
c  in /param/
c
c       snr      real    signal to noise ratio in db
c       seed     real    seed for random # generator
c       amplit   real    average incoming signal amplitude
c                        ** if set to 0. this will default to
c                           amplx in /xblock/
c       bandwd   real    rf or if bandwidth
c                        ** if set to 0. this will default to
c                           bwx in /xblock/
c
c  variables required from common /xblock/
c
c        enb=bwx          if bandwidth (usually nchips/((bint/delt)*duty))
c        s=amplx          signal power
c        delt=deltx       time increment between samples
c
c
c  subroutines required outside of icssm
c
c       gnois3 complex gaussian white noise generator
c       gnois2 white noise generator
c       rgauss uniform to gaussian # generator
c       ranup  update random number generator seed
c       ranu   uniform random number generator
c
c*****************************************************************
c
```

Figure 3.15a Complex White Noise Generator

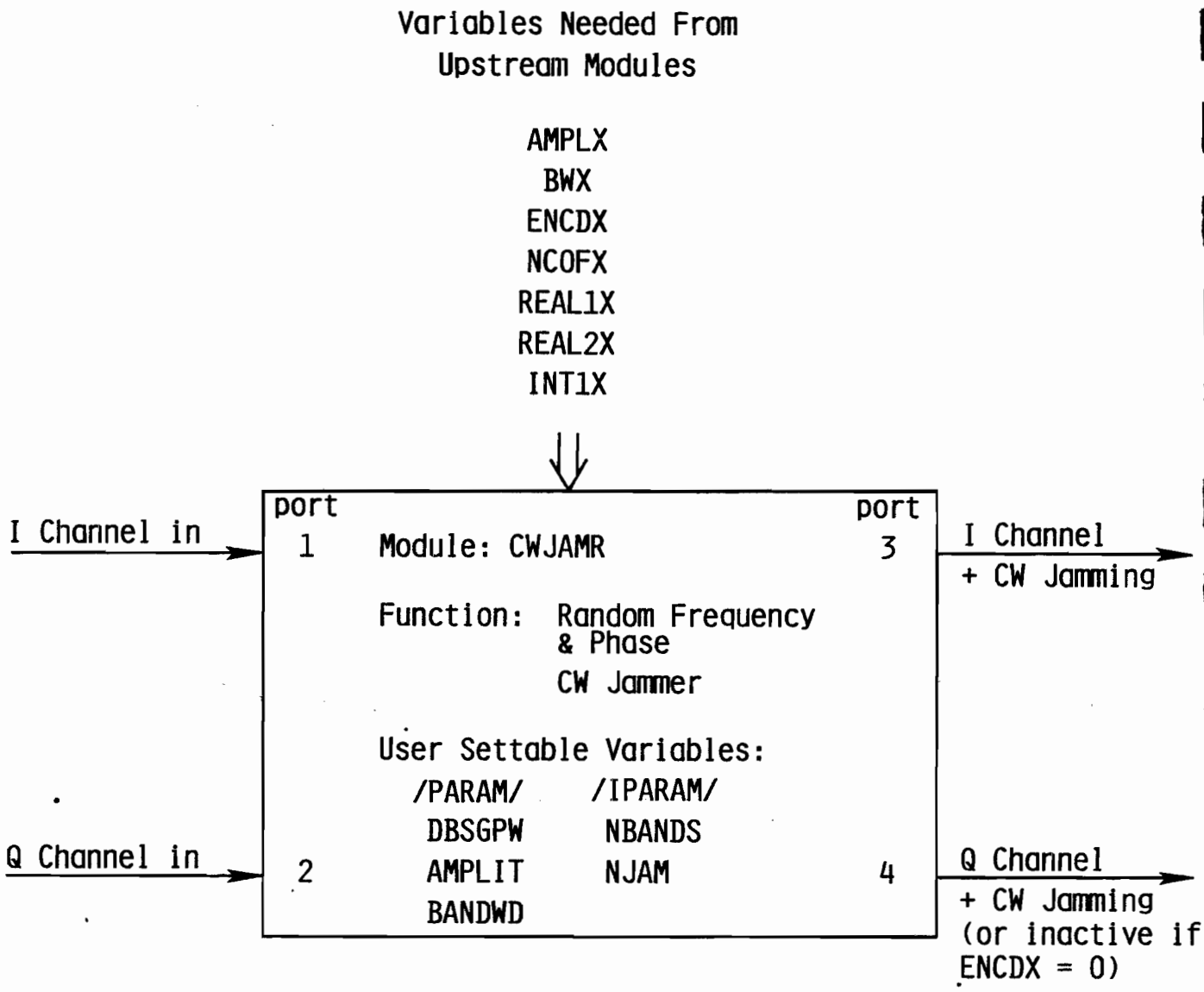


Figure 3.16 Random Frequency & Phase CW Jammer

```
c cwjamr     random frequency and phase cw jammer
c
c identification
c
c      title       cwjamr
c      date        1/17/83
c      author      c.paul
c      computer    hw
c      operating environment- icssm
c      opsys       multics
c  purpose
c
c      this driver will input real or complex signal
c      jam them randomly and output a real or complex signal
c
c  port specification
c
c            two forms possible
c
c     for real signal (encdx from /xblock/ =0.)
c
c  port#     direction      description
c
c    1       input          real signal (usually bpsk)
c    3       output         real signal with jamming
c
c     for complex signal (encdx from /xblock/ =1.)
c
c  port#     direction      description
c
c    1       input          real part of complex signal (usually qpsk signal i)
c    2       input          imaginary part of complex signal (usually qpsk sig
 cl q)
c    3       output         jammed real signal (qpsk signal  i)
c    4       output         jammed imaginary signal (qpsk signal q)
c
c  user settable variables
c
c  in common /param/
c
c      dbsgpw  real    jammer to signal power ratio in db
c      amplit  real    average signal amplitude
c                      ** if left set to 0. this will default
c                         to amplx in /xblock/
c      bandwd  real    r.f. or i.f. bandwidth
c                      ** if left set to 0. this will default
c                         to bwx in /xblock/
c
c  in common /iparam/
c
c      nbands  int     number of bands available for frequency hopping
c      njam    int     number of bands to jam
c
c  variables used from common /xblock/
c
c      amplx   real    average signal amplitude (avepwr)
c      bwx     real    if bandwidth (nchips/((bint/delt)*duty)
```

Figure 3.16a Random Frequency & Phase CW Jammer

```
c                       this variable is called bwdtif in cwjm
c      encdx    real    =0. for real input (bpsk)
c                       =1. for complex input (qpsk)
c                       this variable is used to set qporbp in cwjm
c      ncofx    int     number of samples to process during tdur
c                       (length of arrays in ublock and vblock)
c      reall    real    burst interval time (called bint)
c      real2    real    duty cycle of burst (called duty)
c      intl     int     number of samples per chip (called nsampl)
c
c
c subroutines required out of icssm
c
c      cwjm     cw jamming
c      ranu     uniform random number generator
c      ranup    update random number generator seed
c*****************************************************************
c
```

Figure 3.16b Random Frequency & Phase CW Jammer

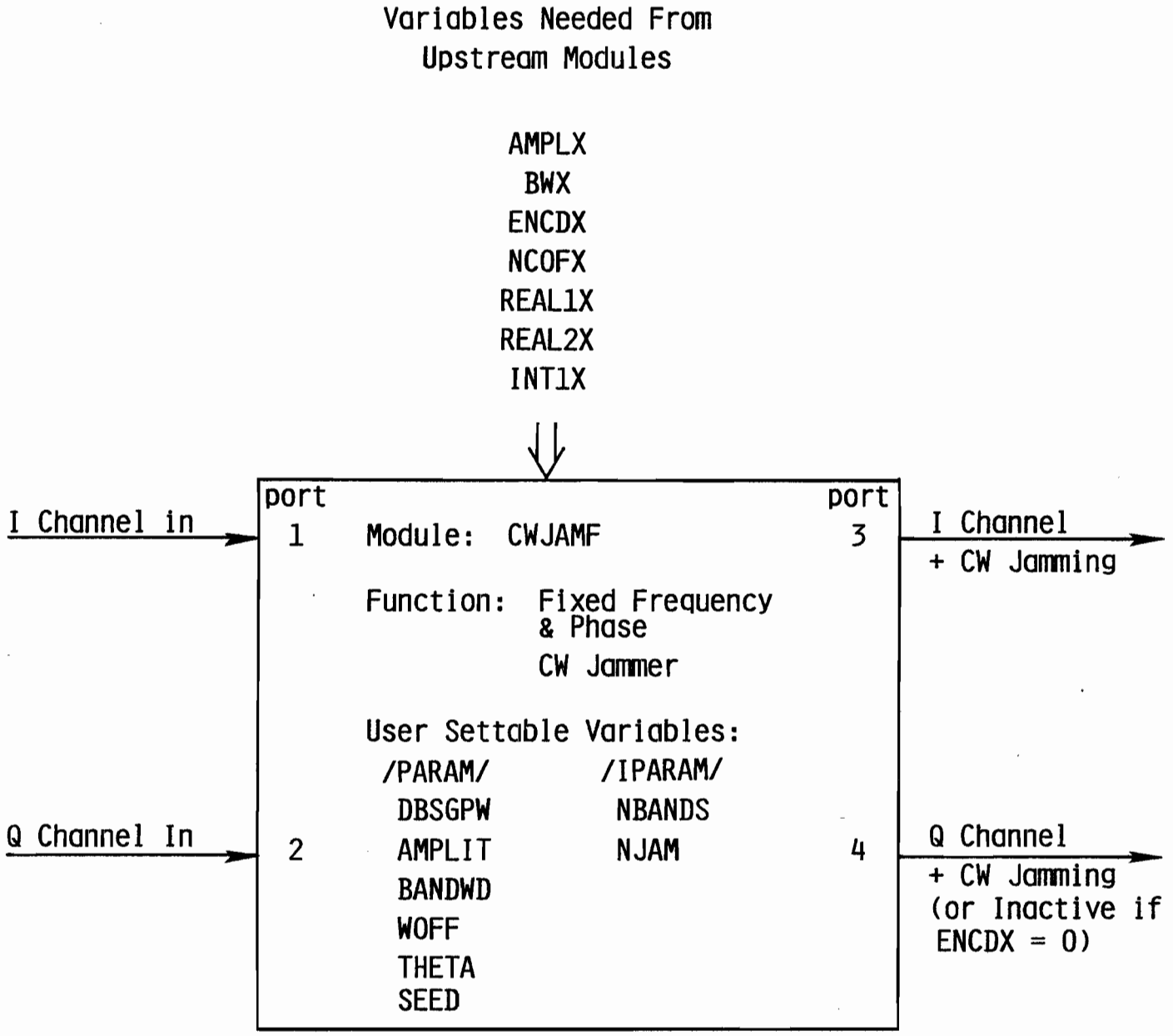


Figure 3.17 Fixed Frequency and Phase CW Jammer

```
c cwjamf     fixed frequency and phase cw jammer
c
c identification
c
c       title       cwjamf
c       date        1/17/83
c       author      c.paul
c       computer    hw
c       operating environment- icssm
c       opsys       multics
c  purpose
c
c       this driver will input real or complex signal
c       jam them and output a real or complex signal
c
c  port specification
c
c             two forms possible
c
c     for real signal (encdx from /xblock/ =0.)
c
c  port#      direction       description
c
c    1        input           real signal (usually bpsk)
c    3        output          real signal with jamming
c
c     for complex signal (encdx from /xblock/ =1.)
c
c  port#      direction       description
c
c    1        input           real part of complex signal (usually qpsk signal i)
c    2        input           imag. part of complex signal (usually qpsk signal q
c    3        output          jammed real signal (qpsk signal i)
c    4        output          jammed imaginary signal (qpsk signal q)
c
c  user settable variables
c
c  in common /param/
c
c       dbsgpw  real    jammer to signal power ratio in db
c       amplit  real    average signal amplitude
c                       ** if set to 0. this will default to
c                       amplx in /xblock/
c       bandwd  real    rf or if bandwidth
c                       ** if set to 0. this will default
c                       to bwx in /xblock/
c       woff    real    frequency offset in range
c                       -bif/2 to bif/2 where bif=nchips/(bint*duty)
c       theta   real    phase offset between -pi and pi
c
c  in common /iparam/
c
c       nbands  int     number of bands available for frequency hopping
c       njam    int     number of bands to jam
c
c  variables used from common /xblock/
c
```

Figure 3.17a Fixed Frequency and Phase CW Jammer

```
c      amplx   real    average signal power (avepwr)
c      bwx     real    if bandwidth (nchips/((bint/delt)*duty)
c                      this variable is called bwdtif in cwjm
c      encdx   real    =0. for bpsk
c                      =1. for qpsk
c                      this variable is used to set qporbp in cwjm
c      ncofx   int     number of samples to process during tdur
c                      (length of arrays in ublock and vblock)
c      real1   real    burst interval time (called bint)
c      real2   real    duty cycle of burst (called duty)
c      int1    int     number of samples per chip (called nsampl)
c
c
c subroutines required outside of icssm
c
c      cwjm    cw jamming
c      ranu    uniform random number generator
c      ranup   update uniform random number seed
c
c*****************************************************************
c
```

Figure 3.17b Fixed Frequency and Phase CW Jammer

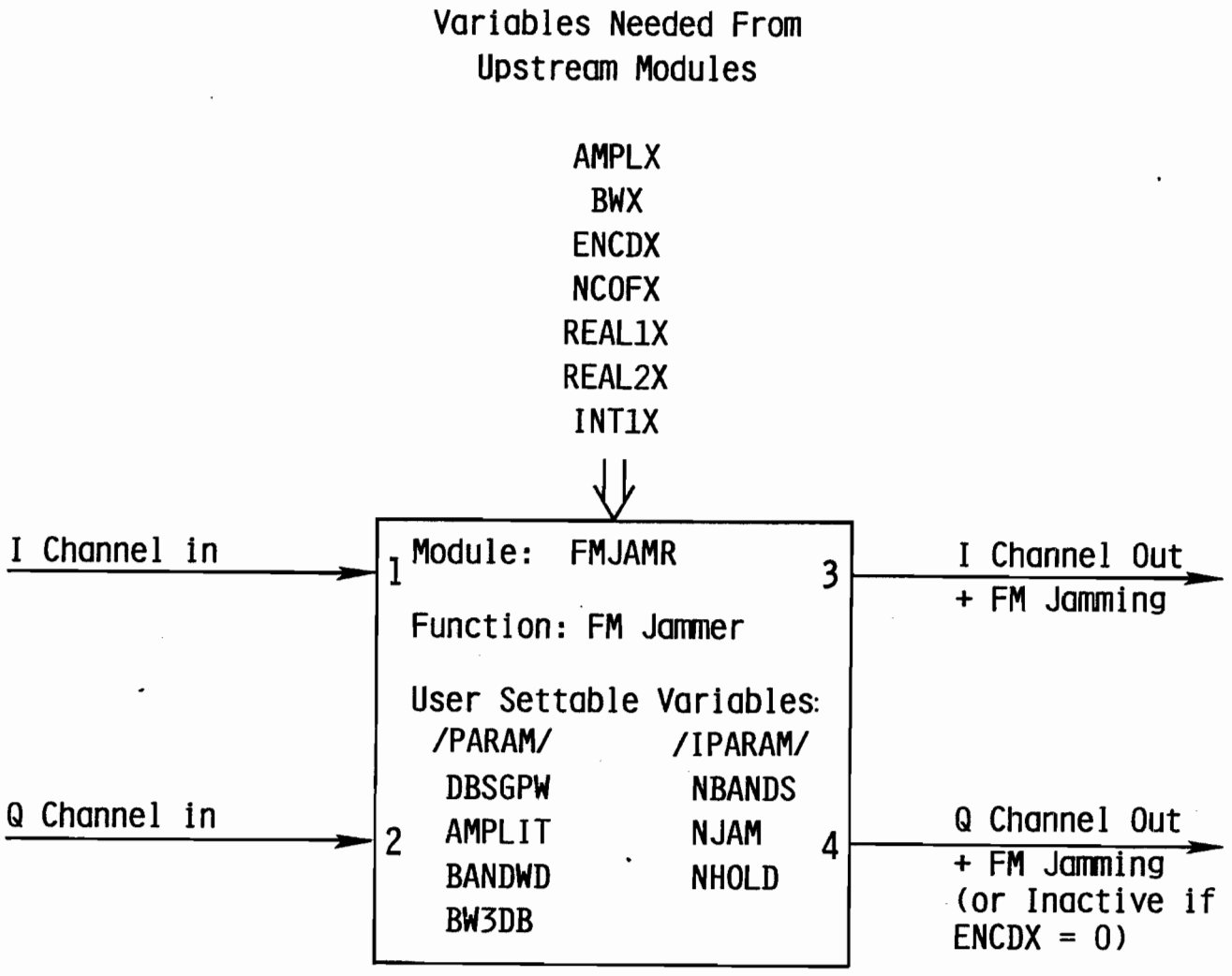


Figure 3.18 FM Jammer

```
c fmjamr     random frequency fm jammer
c
c identification
c
c      title       fmjamr
c      date        1/17/83
c      author      c.paul
c      computer    hw
c      operating environment- icssm
c      opsys       multics
c  purpose
c
c      this driver will input real or complex signal
c      jam them randomly and output a real or complex signal
c
c  port specification
c
c            two forms possible
c
c     for real signal (encdx from /xblock/ =0.)
c
c  port#     direction      description
c
c    1       input          real signal (usually bpsk)
c    3       output         real signal with jamming
c
c     for complex signal (encdx from /xblock/ =1.)
c
c  port#     direction      description
c
c    1       input          real part of complex signal (usually qpsk signal i)
c    2       input          imaginary part of complex signal (usually qpsk sigr
 cl q)
c    3       output         jammed real signal (qpsk signal  i)
c    4       output         jammed imaginary signal (qpsk signal q)
c
c  user settable variables
c
c  in common /param/
c
c      dbsgpw  real    jammer to signal power ratio in db
c      amplit  real    average signal amplitude
c                      ** if left set to 0. this will default
c                         to amplx in /xblock/
c      bandwd  real    r.f. or i.f. bandwidth
c                      ** if left set to 0. this will default
c      bw3db   real    3db bandwidth of fm jammer
c
c  in common /iparam/
c
c      nbands  int     number of bands available for frequency hopping
c      njam    int     number of bands to jam
c      nhold   int     number of samples to hold a random number
c                      to noise load the fm
c                      =1 laplacian noise, †1 gaussian noise
c                      ** note if nhold does not evenly
c                         divide ncofx routine fmjm will notify
```

Figure 3.18a FM Jammer

```
c                          the user of this error and stop
c
c  variables used from common /xblock/
c
c       amplx    real     average signal amplitude (avepwr)
c       bwx      real     if bandwidth (nchips/((bint/delt)*duty)
c                         this variable is called bwdtif in fmjm
c       encdx    real     =0. for real input (bpsk)
c                         =1. for complex input (qpsk)
c                         this variable is used to set qporbp in fmjm
c       ncofx    int      number of samples to process during tdur
c                         (length of arrays in ublock and vblock)
c       reall    real     burst interval time (called bint)
c       real2    real     duty cycle of burst (called duty)
c       intl     int      number of samples per chip (called nsampl)
c
c
c subroutines required out of icssm
c
c       fmjm     fm jamming
c       ranu     uniform random number generator
c       ranup    update random number generator seed
c       rgauss   gaussian random number generator
c
c*****************************************************************
c
```

Figure 3.18b FM Jammer

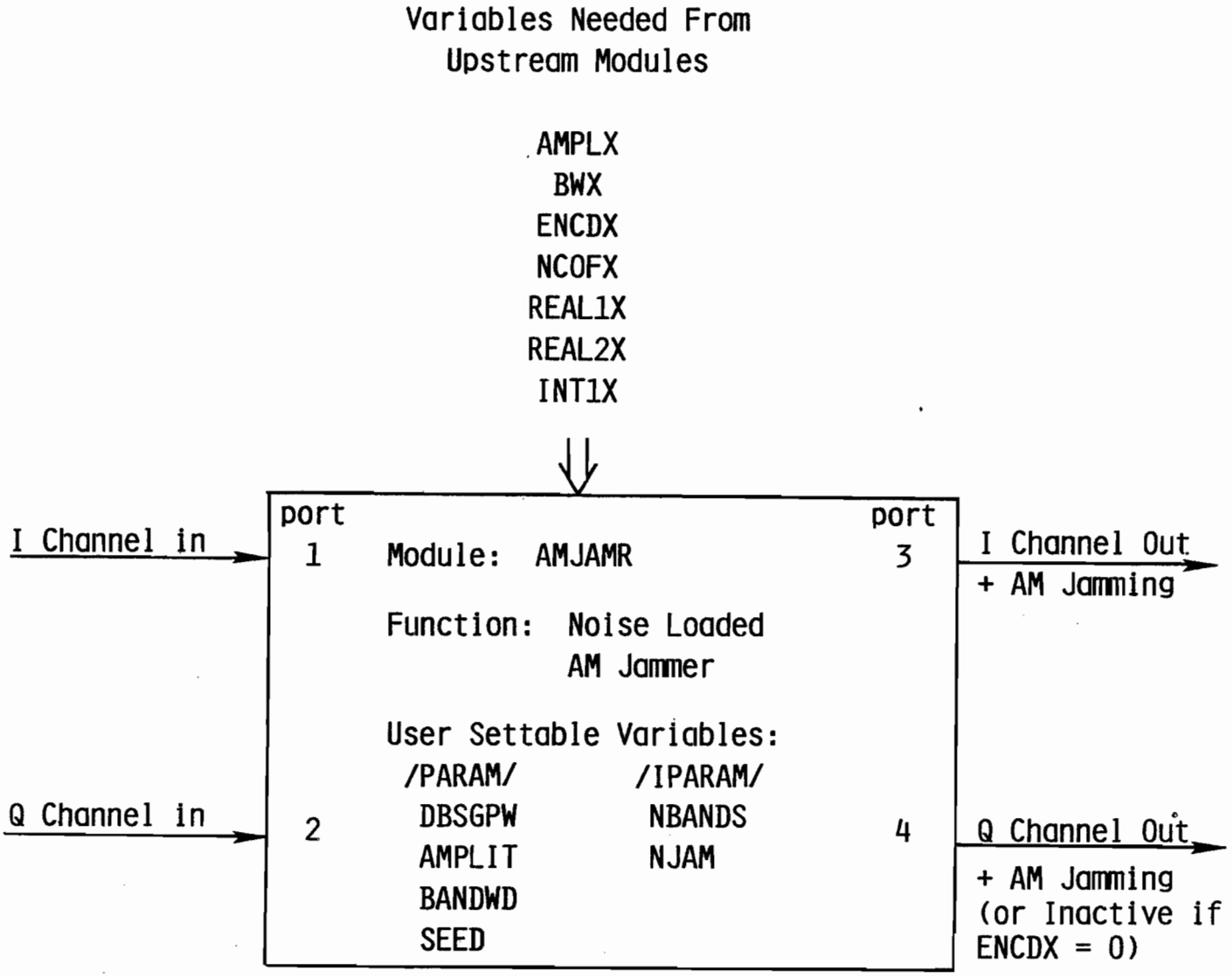


Figure 3.19 Noise-Loaded AM Jammer

```
c amjamr    icssm driver for am jammer
c
c identification
c
c       title       amjamr
c       date        1/14/83
c       author      c.paul
c       computer    hw
c       operating environment- icssm
c       opsys       multics
c  purpose
c
c       this driver will input real or complex signal
c       jam them randomly and output a real or complex signal
c
c  port specification
c
c             two forms possible
c
c     for real input (encdx from /xblock/ =0.)
c
c  port#      direction       description
c
c    1         input           real signal (usually bpsk)
c    3         output          real signal with jamming
c
c     for complex input (encdx from /xblock/ =1.)
c
c  port#      direction       description
c
c    1         input           real part of complex signal (usually qpsk i)
c    2         input           imaginary part of complex signal (usually qpsk q)
c    3         output          jammed qpsk signal i
c    4         output          jammed qpsk signal q
c
c  user settable variables
c
c  in common /param/
c
c       dbsgpw  real    jammer to signal power ratio in db
c       aveamp  real    average signal amplitude
c                       ** if left set to 0. this will default
c                          to amplx in /xblock/
c       bandwd  real    rf or if bandwidth
c                       ** if set to 0. this will default
c                          to bwx in /xblock/
c       seed    real    seed for random number generator between 0. and 9.
c
c  in common /iparam/
c
c       nbands  int     number of bands available for frequency hopping
c       njam    int     number of bands to jam
c
c  variables used from common /xblock/
c
c       amplx   real    default value for average signal power (avepwr)
c       bwx     real    default value for if bandwidth (nchips/((bint/delt)*du
```

Figure 3.19a Noise-Loaded AM Jammer

```
c                       this variable is called bwdtif in amjm
c      encdx    real    =0. for real input (bpsk)
c                       =1. for complex input (qpsk)
c                       this variable is used to set qporbp in amjm
c      ncofx    int     number of samples to process during tdur
c                       (length of arrays in ublock and vblock)
c      real1    real    burst interval time (called bint)
c      real2    real    duty cycle of burst (called duty)
c      intl     int     number of samples per chip (called nsampl)
c
c
c subroutines required out of icssm
c
c      amjm     am jamming
c      gnois3   complex gaussian white noise
c      gnois2   white noise generator
c      rgauss   gaussian number generator
c      ranu     uniform random number generator
c      ranup    update random number seed
c
c*****************************************************************
c
```

Figure 3.19b Noise-Loaded AM Jammer

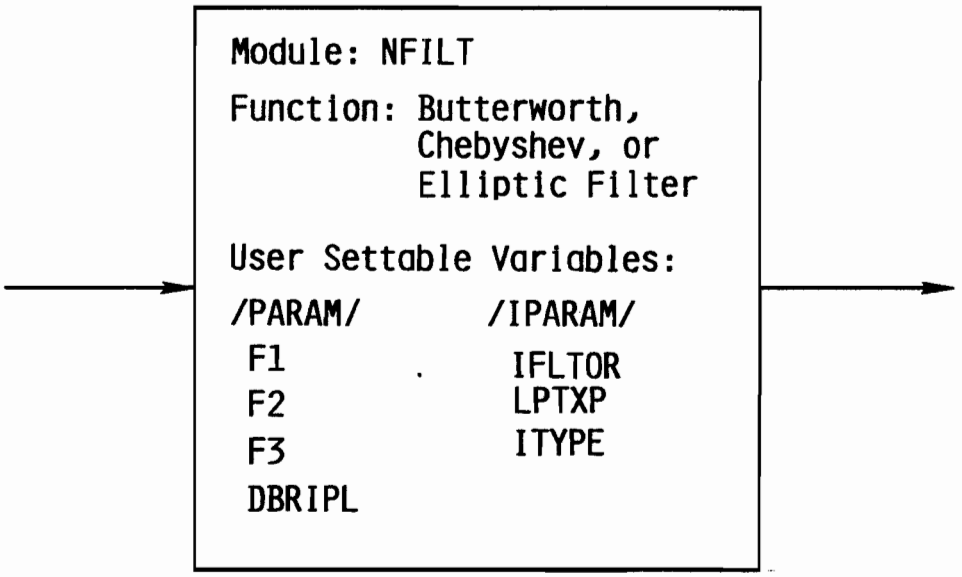


Figure 3.20 Butterworth, Chebyshev or Elliptic Filter

```
c  nfilt        new butterworth, chebychev, and elliptic filter
c
c  identification
c
c      title    nfilt
c      version  a
c      date     1/12/83
c      author   c.paul
c      language fortran
c      operating environment - icssm
c      opsys    multics
c      machine  hw
c  purpose
c
c      this routine drives the general filter based
c      on the bilinear z transform
c
c   port description
c
c     port   direction  description
c
c      1       input     signal input
c      2       output    filtered signal output
c
c  user settable parameters
c
c  in /param/
c
c      f1       real    -3db point for low-pass and high-pass
c                        lower -3db point for band-pass and band-stop
c      f2       real    upper -3db point for band-pass and band-stop
c      f3       real    stop band edge for elliptic filter
c      dbripl   real    in band ripple in db
c
c  in /iparam/
c
c      ifltor   int     filter order
c  ** note **           this integer should be even and less than 11
c      lptxp    int    low pass t intended pass
c                       =0  low pass with -3db at f1
c                       =1  high pass with -3db at f1
c                       =2  band pass
c                           between f1 and f2
c                       =3  band stop
c                           notch out frequencies between f1 and f2
c      itype            =0 butterworth, =1 chebyshev, =2 elliptic
c
c  subroutines required outside of icssm
c
c      filtl  actual filter routine
c      filbll build desired filter (used only first time through)
c      lptoxp transforms low pass filter to x pass filter
c      jacelp
c      fkofk
c      fk
c      asn
c      agm
```

Figure 3.20a Butterworth, Chebyshev or Elliptic Filter

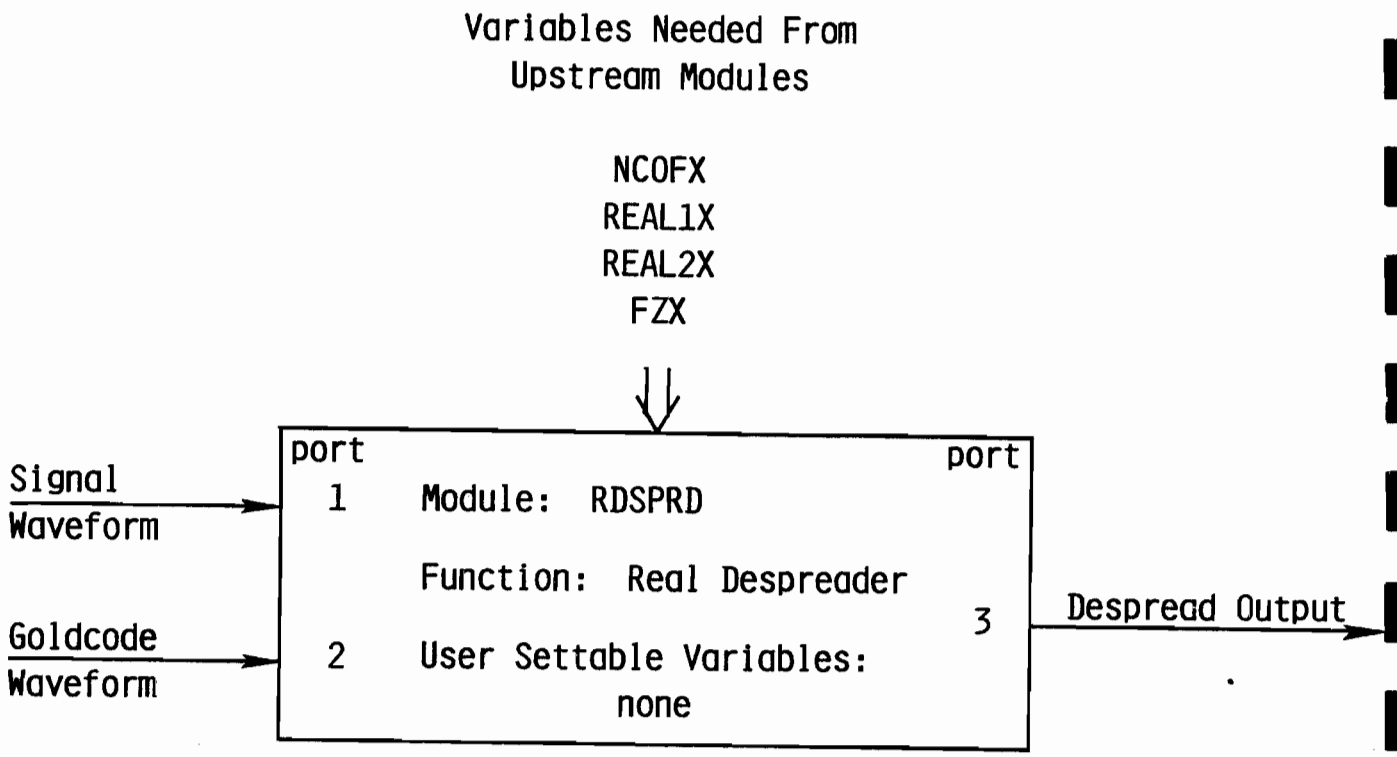


Figure 3.21 Real Despreader

```
c  rdsprd   real despreader
c
c  identification
c
c      title         rdsprd
c      date          2/21/83
c      author        c.paul
c      language      fortran
c      computer      honeywell
c  purpose
c
c       this routine will despread an input stream
c       with an externally generated reference signal
c       (in this case a gold code)
c
c  user settable parameters
c
c       none
c
c  port specification
c
c  port    direction  use
c
c   1      input      spread signal input
c   2      input      reference signal to despread with
c   3      output     despread signal output
c
c
c  parameters required from /xblock/
c
c     ncofx  int   number of points to process
c
c  subroutines required outside of icssm
c
c      dsprdr   real despreading routine
c
c*****************************************************************
c
```

Figure 3.21a Real Despreader

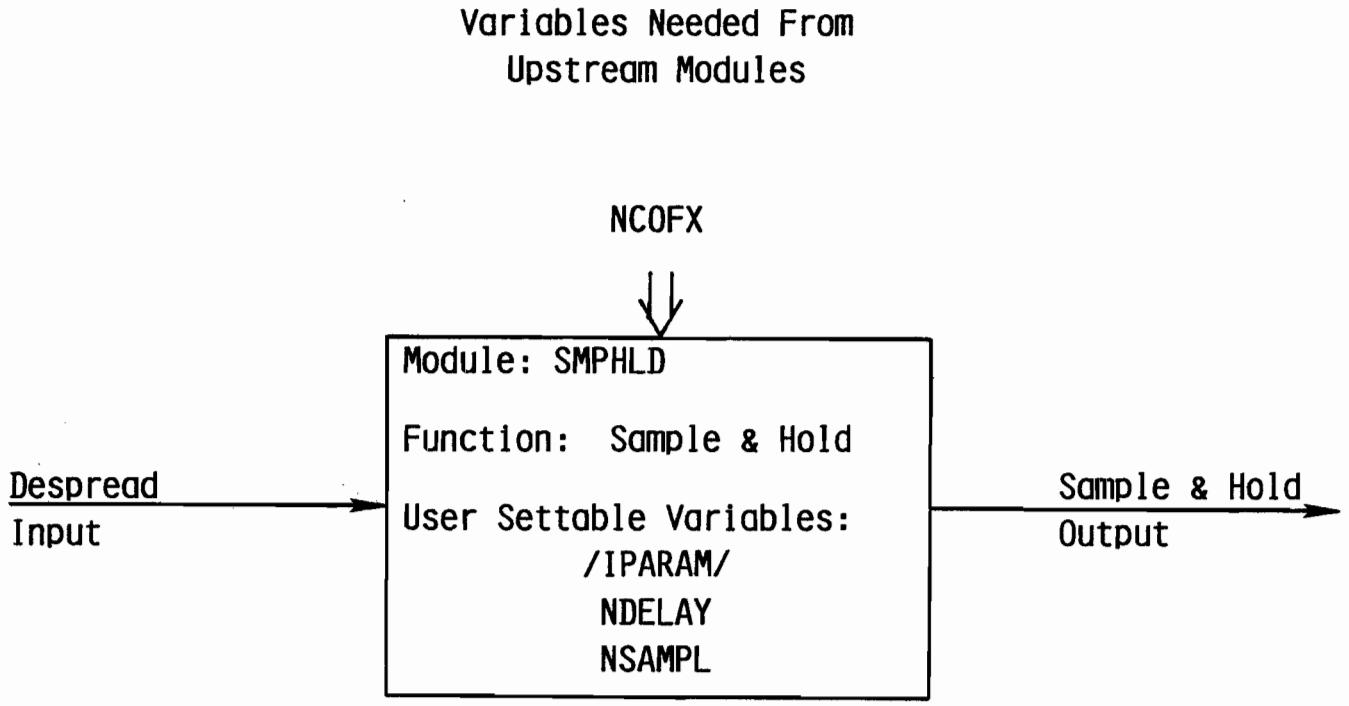


Figure 3.22 Sample and Hold

```
c  smphld     sample and hold
c
c  identification
c
c       title          smphld
c       date           2/14/83
c       author         c.paul
c       language       fortran
c       computer       harris (honeywell)
c       operating environment  icssm
c       opsys          multics
c  purpose
c
c       this routine will integrate an input signal
c       over a given number of samples with a specified
c       initial delay and dump the accumulated integration
c       after the given number of samples have been integrated
c       just before the integration is dumped, the integration
c       value is check for a positive value.  if the value is positive
c       the output of the sample and hold is 1. if the value is negative
c       the output of the sample and hold is -1.  the held value is held
c       for nsamples.  since the first sample and hold value is not
c       complete until ndelay+nsampl -1st sample is accumulated the
c       sample and hold value will be zero until that time, hence
c       there is a delay of ndelay+nsampl through this module.
c
c    port description
c
c    port    direction    description
c
c     1         inputdespread signal input
c     2         output       sample and hold output
c
c
c  user settable parameters
c
c  in common /iparam/
c
c       ndelay  int      number of samples for initial delay
c       nsampl  int.     number of samples to include per integration
c                        (this is usually the number of samples per
c                         information bit)
c
c  parameters required from /xblock/
c
c       ncofx   int      number of points to process
c
c
c  subroutines required
c
c       intdum  integrate and dump or sample and hold
c
c**********************************************************
c
```

Figure 3.22a Sample and Hold

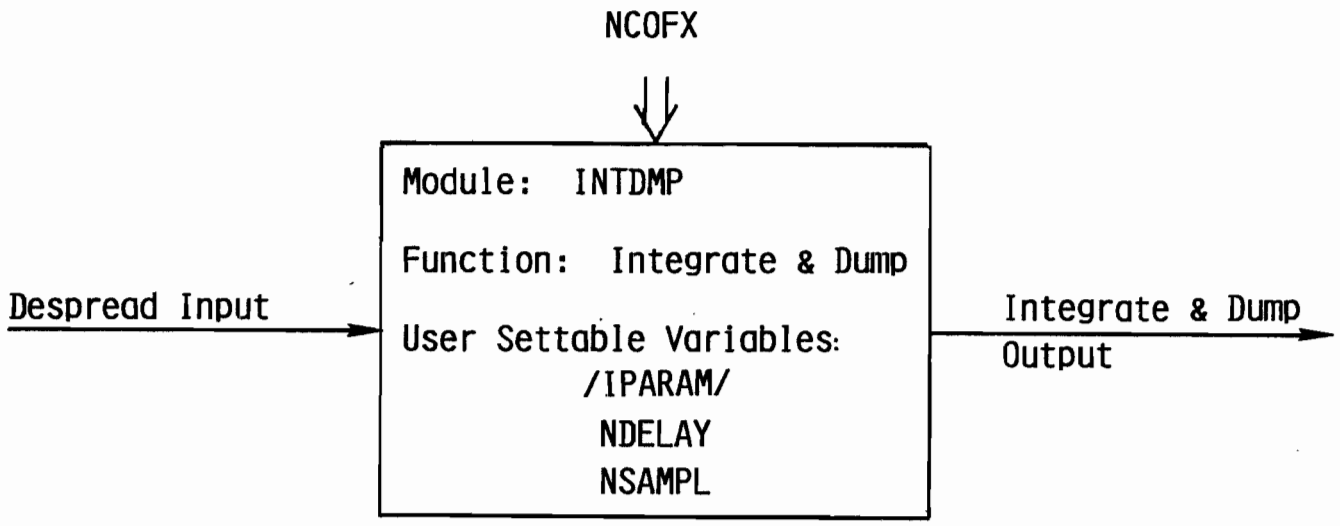


Figure 3.23 Integrate & Dump

```
c  intdmp     integrate (ingratiate) and dump
c
c  identification
c
c       title         intdmp
c       date          2/14/83
c       author        c.paul
c       language      fortran
c       computer      harris (honeywell)
c       operating environment  icssm
c       opsys         multics
c  purpose
c
c       this routine will integrate and input signal
c       over a given number of samples with a specified
c       initial delay and dump the accumulated integration
c       after the given number of samples have been integrated
c
c  user settable parameters
c
c  in common /iparam/     .
c
c       ndelay  int      number of samples for initial delay
c       nsampl  int      number of samples to include per integration
c                        (this is usually the number of samples per
c                         information bit)
c
c  parameters required from /xblock/
c
c       ncofx   int      number of points to process
c
c
c  subroutines required
c
c       intdum  integrate and dump or sample and hold
c
c*************************************************************
c
```

Figure 3.23a Integrate & Dump

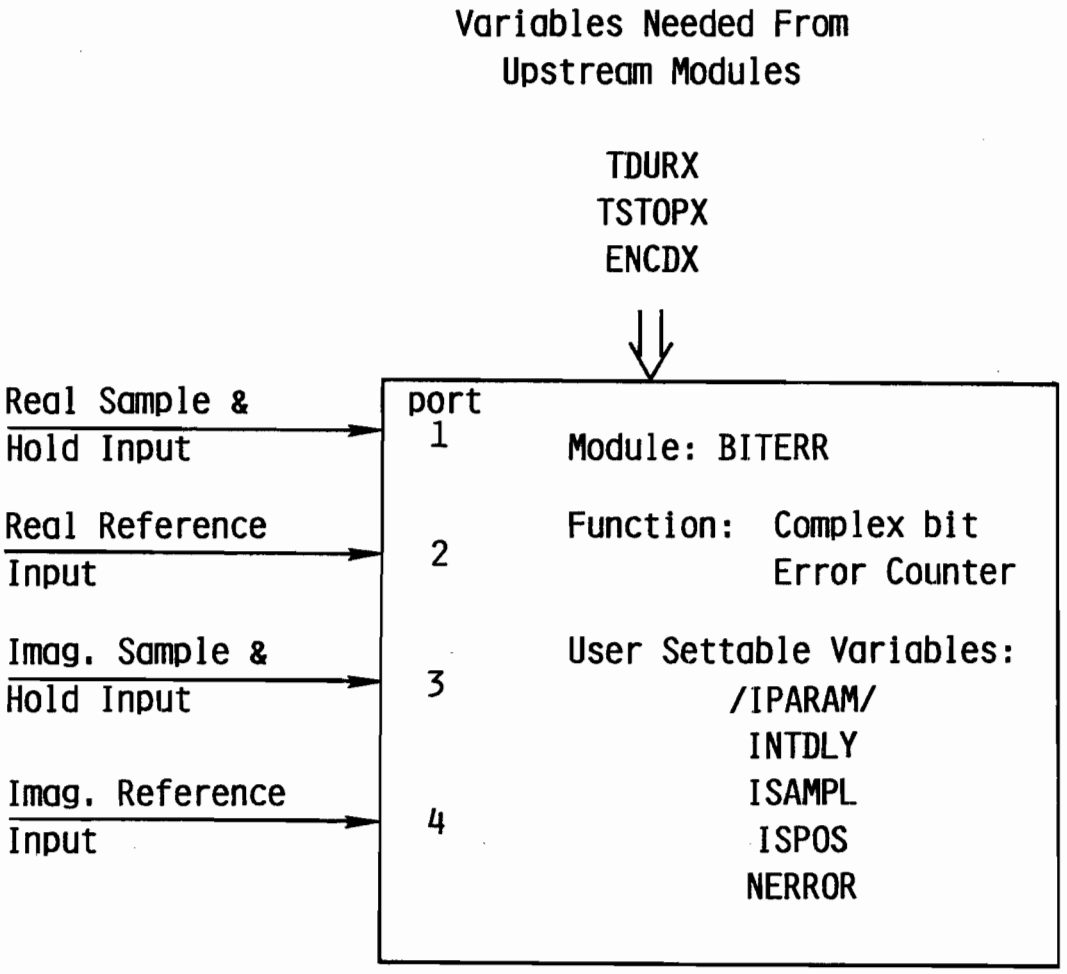


Figure 3.24 Complex Bit Error Counter

```
c  biterr   icssm driver for bit error estimator
c
c  identification
c
c       title       biterr
c       version     a
c       date        2/10/83
c       author      c.paul
c       computer    hw
c       operating environment  icssm
c       op.sys      multics
c
c  purpose
c
c           this routine will take two complex data streams, sample them
c           at specific spots and count the number of differences
c           between the two along with the number of places sampled.
c           this will yield the bit error rate of the two data streams
c
c    port specification
c
c    port        direction   description
c
c     1          input       real part of complex bit stream
c     2          input       real part of another complex bit stream
c     3          input       imaginary part of complex bit steam
c     4          input       imaginary part of another complex bit stream
c
c  user settable variables
c
c   in common /iparam/
c
c      intdly  int     initial delay. number of samples to wait before
c                      counting is to begin
c      isampl  int     number of samples per information bit
c      ispos   int     sample position withing information bit to count
c                      1 §= ispos §= isampl
c      nstper  int     num. of errors to stop counting after
c
c   variables required from common /xblock/
c
c      tdurx   real    time duration per sample block
c      tstopx  real    simulation stop time
c
c   variable required from common /exec/
c
c      tnow    real    current simulation time
c
c   ** note **
c
c      tdurx, tstopx, and tnow are needed to tell this terminator
c      module when its last run will be so that the results of the
c      bit error estimator will be output to a file during the last
c      invocation
c
c*************************************************************************
c
```

Figure 3.24a Complex Bit Error Counter

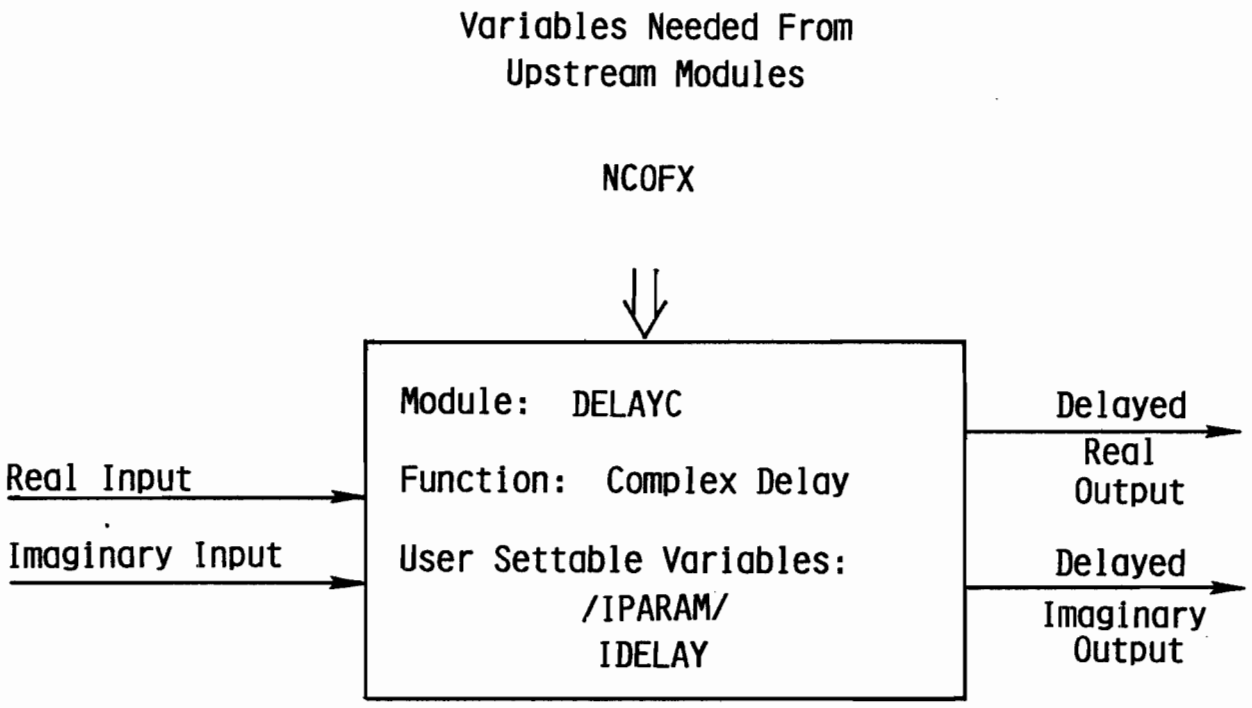


Figure 3.25 Complex Delay

```
c  delayc    icssm real signal delay routine
c
c  identification
c
c       title       delayc
c       date        2/18/83
c       author      c.paul
c       language    fortran
c       computer    honeywell
c       operating environment - icssm
c  purpose
c
c       this routine will delay the signal at the input ports
c       by a specified number of samples
c
c  user settable parameters
c
c  in common /iparam/
c
c        idelay  int     number of samples to delay
c ** note **    due to size limitations of common param
c               and since this routine needs to be re-entrant
c               a delay of 1045 maximum is the limit.
c               otherwise the routine will execute a stop statement
c               after notifying the user why it has stopped
c
c  parameters required from common /xblock/
c
c        ncofx   int     number of samples to process
c
c  subroutines required outside of icssm
c
c       cdelay  delay      signal
c
c*************************************************************************
c
```

Figure 3.25a Complex Delay

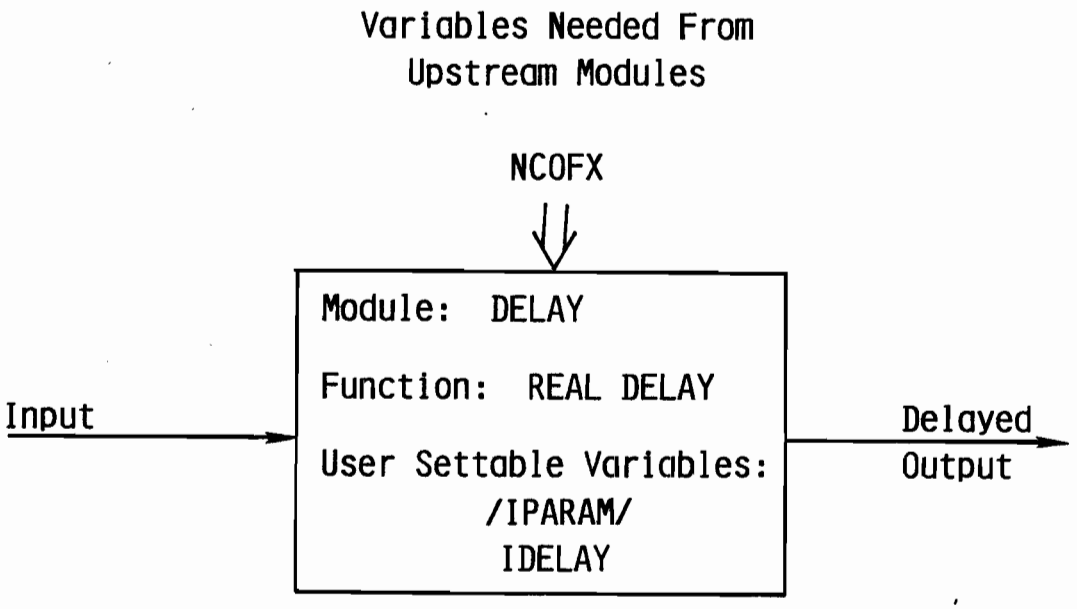
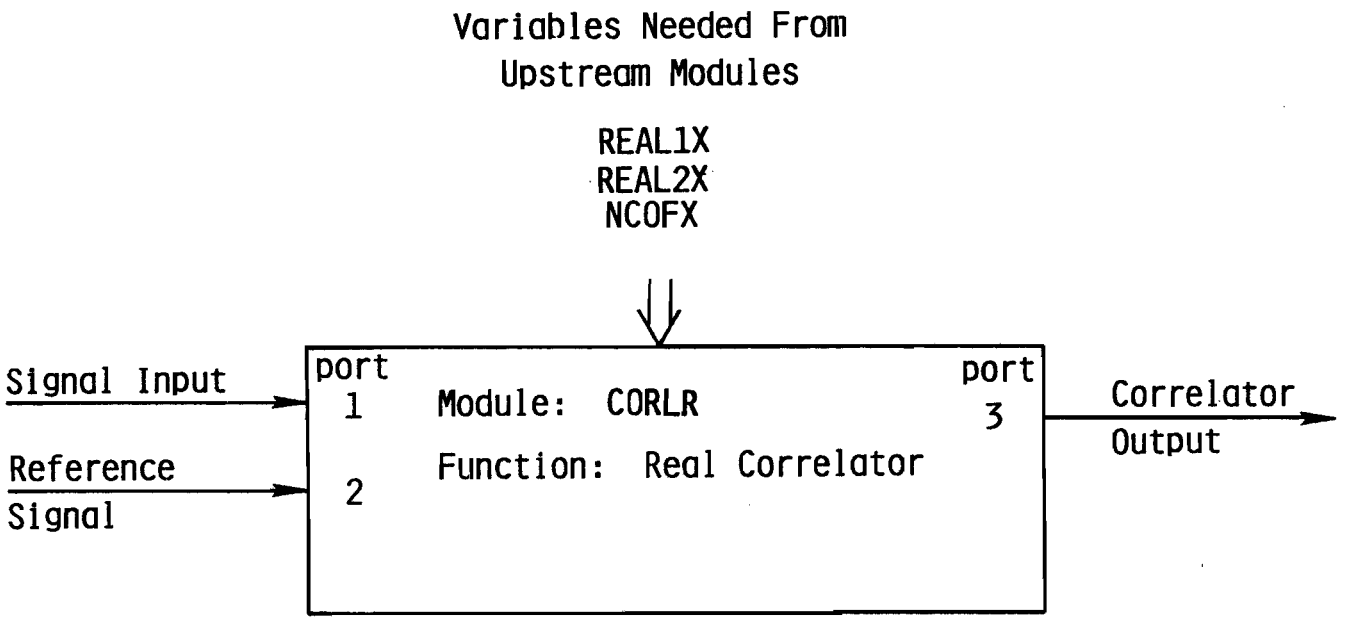


Figure 3.26 Real Delay

```
c  delay    icssm real signal delay routine
c
c  identification
c
c       title        delay
c       date         2/18/83
c       author       c.paul
c       language     fortran
c       computer     honeywell
c       operating environment - icssm
c  purpose
c
c       this routine will delay the signal at the input port
c       by a specified number of samples
c
c     port description
c
c     port   direction  description
c
c      1       input     signal input
c      2       output    delayed signal output
c
c  user settable parameters
c
c  in common /iparam/
c
c        idelay  int     number of samples to delay
c ** note **     due to size limitations of common param
c                and since this routine needs to be re-entrant
c                a delay of 2090 maximum is the limit.
c                otherwise the routine will execute a stop statement
c                after notifying the user why it has stopped
c
c  parameters required from common /xblock/
c
c        ncofx   int     number of samples to process
c
c  subroutines required outside of icssm
c
c       rdelay  delay real signal
c
c*************************************************************************
c
```

Figure 3.26a Real Delay

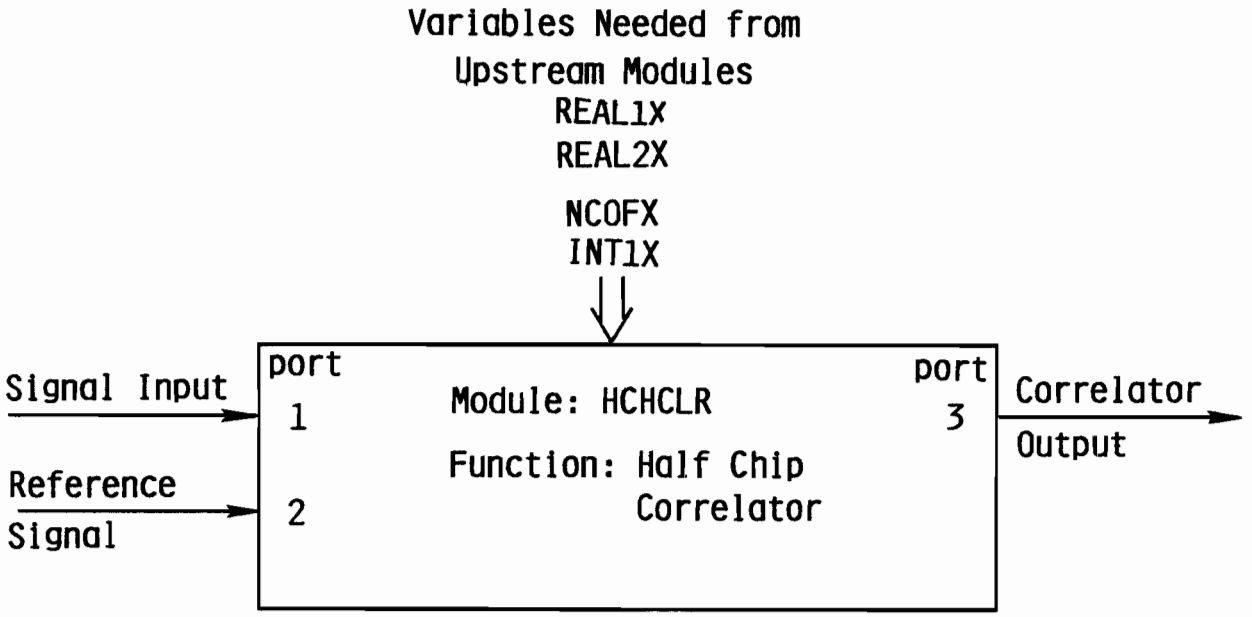


Files "file66", "file67",
and 'file68" are used
to feedback and hold
input and output data.

Figure 3.27 Real Correlator

```
c  corlr  correlator
c
c  identification
c
c      title         corlr
c      date          2/21/83
c      author        c.paul
c      language      fortran
c      computer      honeywell
c  purpose
c
c       this routine will correlate an input stream
c       with another (reference) signal
c
c  parameters required from /xblock/
c
c     ncofx  int   number of points to process
c     reall  real  burst interval time (called bint)
c     real2  real  duty cycle of burst (called duty)
c
c  port specification
c
c     port#    direction    specification
c
c       1         input       input signal to correlate
c       2         input       reference signal stream
c                             after one burst is injested this port
c                             will be disregarded
c       3         output      correlation array
c                             active only after each complete burst
c                             has been injested, otherwise outputs zeros
c
c  ** note **
c
c       this routine uses logical units 66, 67, and 68 to manage
c       multiple invocation feedback via read/write random
c       (direct access files)
c
c*****************************************************************
c
```

Figure 3.27a Real Correlator



Files "file66", "file67",
and "file68" are used
to feedback and hold
input and output data.

Figure 3.28 Half Chip Correlator

```
c  hchclr  half chip correlator
c
c  identification
c
c       title         hchclr
c       date          2/21/83
c       author        c.paul
c       language      fortran
c       computer      honeywell
c  purpose
c
c        this routine will correlate an input stream
c        with another (reference) signal
c        in half chip increments, hence only ((reall/deltx)/nsampl) *2
c        active points will be output by the correlator.
c        The other ((reall/deltx)-(((reall/deltx)/nsampl)*2) points will be zero
 c.
c
c  parameters required from /xblock/
c
c     ncofx  int   number of points to process
c     real1  real  burst interval time (called bint)
c     real2  real  duty cycle of burst (called duty)
c     intl   int   number of samples per chip (called nsampl)
c
c  port specification
c
c     port#    direction    specification
c
c      1         input       input signal to correlate
c      2         input       reference signal stream
c                            after one burst is injested this port
c                            will be disregarded
c      3         output      correlation array
c                            active only after each complete burst
c                            has been injested, otherwise outputs zeros
c
c  ** note **
c
c      this routine uses logical units 66, 67, and 68 to manage
c      multiple invocation feedback via read/write random
c      (direct access files)
c
c*****************************************************************
c
```

Figure 3.28a Half Chip Correlator

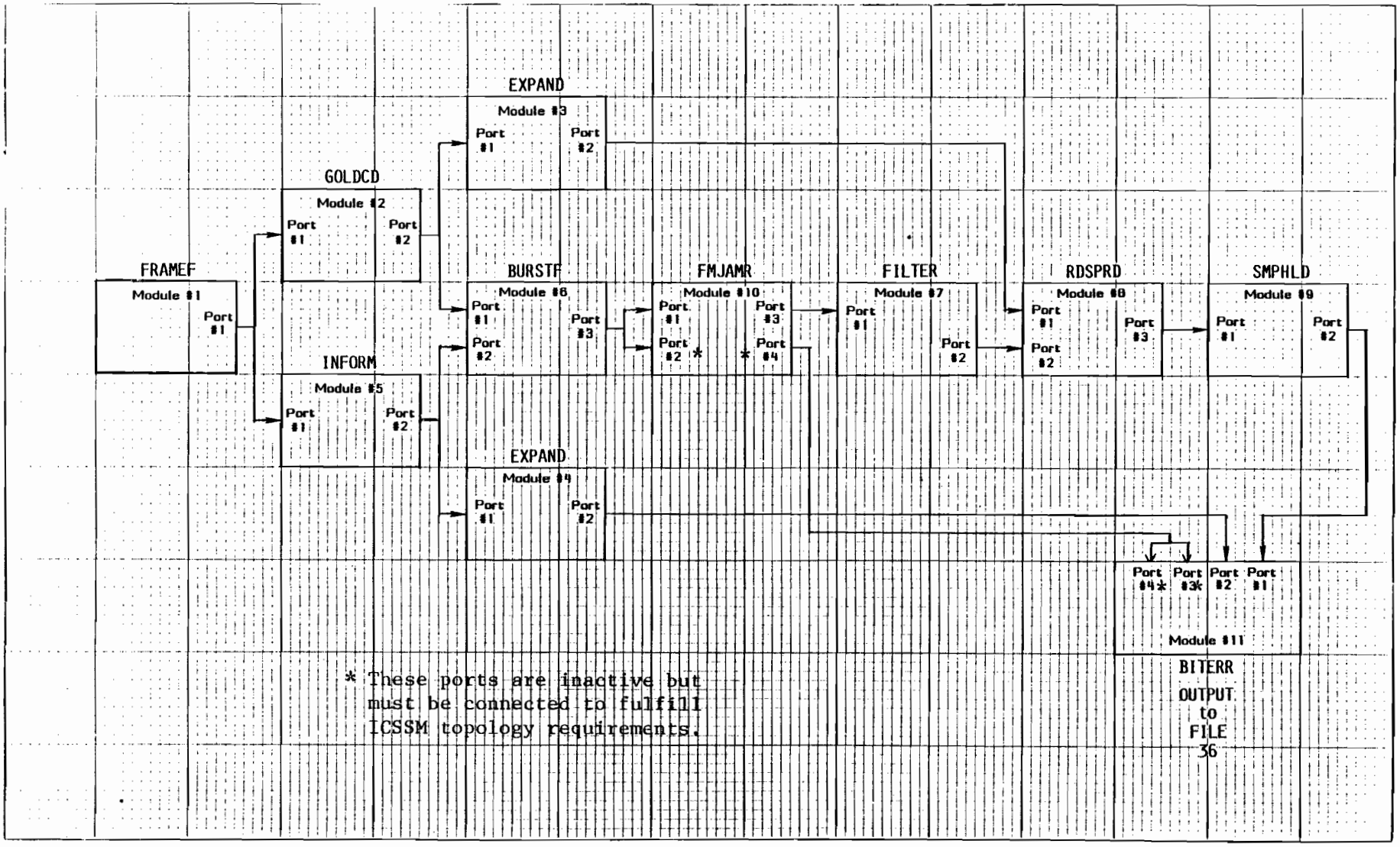


Figure 3.29 Actual Simulation Configuration to Test Error Rate vs. FM Jamming Powers

CONFIGURATION SUMMARY

```
DATE:  8/25/83
TIME: 14:32:12
MODULE   LIBRARY   USER
NUMBER   NAME      NAME
   1     framef    frame1

FLOATING-POINT PARAMETERS:
                          burst interval time                2048.000
                          time duration per invocation       1024.000
                          time increment between samples        1.000
INTEGER PARAMETERS:
                          # of frames to generate                 1
                          # bursts/frame (with sync)            250
                          # sync bursts/frame                     0
CHECKPOINT TRIGGERS:
                                                          NONE
OUTPUT CONNECTORS:
                          PORT          MODULE   PORT
                          NUMBER        NUMBER   NUMBER
                             1    to       2        1
                             1    to       5        1
PORTS OPEN FOR OUTPUT DATA TRANSFER:
                                                          NONE


MODULE   LIBRARY   USER
NUMBER   NAME      NAME
   2     goldcd    gold1
```

Figure 3.29a Actual Simulation Configuration to Test Error Rate vs. FM Jamming Powers

```
FLOATING-POINT PARAMETERS:
                                                          NONE
INTEGER PARAMETERS:
                        gold code length                          128
                        data gold code number                       0
                        sync gold code number                       1
CHECKPOINT TRIGGERS:
                                                          NONE
OUTPUT CONNECTORS:
                        PORT          MODULE   PORT
                        NUMBER        NUMBER   NUMBER
                           2    to       3        1
                           2    to       6        1
PORTS OPEN FOR OUTPUT DATA TRANSFER:
                                                          NONE


MODULE   LIBRARY   USER
NUMBER   NAME      NAME
   3     expand    expdg

FLOATING-POINT PARAMETERS:
                        duty cycle of burst                     1.000
INTEGER PARAMETERS:
                        num. of bits in input stream              128
                        num. of samples to delay outpt             11
CHECKPOINT TRIGGERS:
                                                          NONE
OUTPUT CONNECTORS:
```

Figure 3.29b Actual Simulation Configuration to Test Error Rate vs. FM Jamming Powers

```
                              PORT        MODULE   PORT
                              NUMBER      NUMBER   NUMBER
                                 2    to      8        2
PORTS OPEN FOR OUTPUT DATA TRANSFER:
                                                      NONE


MODULE   LIBRARY   USER
NUMBER   NAME      NAME
   4     expand    expdi

FLOATING-POINT PARAMETERS:
                              duty cycle of burst                  1.000
INTEGER PARAMETERS:
                              num. of bits in input stream            4
                              num. of samples to delay outpt        523
CHECKPOINT TRIGGERS:
                                                      NONE
OUTPUT CONNECTORS:
                              PORT        MODULE   PORT
                              NUMBER
   NUMBER   NUMBER
                                 2    to     11       1
                                 2    to     11       3
PORTS OPEN FOR OUTPUT DATA TRANSFER:
                                    2


MODULE   LIBRARY   USER
NUMBER   NAME      NAME
   5     inform    info1
```

Figure 3.29c Actual Simulation Configuration to Test Error Rate vs. FM Jamming Powers

```
FLOATING-POINT PARAMETERS:
                                                        NONE
INTEGER PARAMETERS:
                    # of info bits                          4
CHECKPOINT TRIGGERS:
                                                        NONE
OUTPUT CONNECTORS:
                    PORT          MODULE   PORT
                    NUMBER        NUMBER   NUMBER
                       2    to       4        1
                       2    to       6        2
PORTS OPEN FOR OUTPUT DATA TRANSFER:
                                                        NONE


MODULE  LIBRARY  USER
NUMBER  NAME     NAME
   6    burstf   burst1

FLOATING-POINT PARAMETERS:
                    duty cycle                          1.000
INTEGER PARAMETERS:
                    number of samples per chip            16
                    =1 direct coding =0 cyc. shift         1
CHECKPOINT TRIGGERS:
                                                        NONE
OUTPUT CONNECTORS:
                    PORT          MODULE   PORT
                    NUMBER        NUMBER   NUMBER
```

Figure 3.29d Actual Simulation Configuration to Test Error Rate vs. FM Jamming Powers

```
                              3   to    10        1
                              3   to    10        2
PORTS OPEN FOR OUTPUT DATA TRANSFER:
                                                      NONE


MODULE  LIBRARY  USER
NUMBER  NAME     NAME
   7    filter   filt1

FLOATING-POINT PARAMETERS:
                         low pass corner frequency                0.063
                         ripple in db                             0.000
                         =1. butterworth, =2. chebyshev           1.000
                         hp 3db, bp or notch lower 3db            0.000
                         lp 3db, bp or notch upper 3db            0.000
INTEGER PARAMETERS:
                         filter order                                6
                         1 lp-lp,2 lp-hp,3 lp-bp,4 lp-n              0
CHECKPOINT TRIGGERS:
                                                      NONE
OUTPUT CONNECTORS:
                         PORT        MODULE  PORT
                         NUMBER      NUMBER  NUMBER
                            2   to      8       1
PORTS OPEN FOR OUTPUT DATA TRANSFER:
                                                      NONE


MODULE  LIBRARY  USER
```

Figure 3.29e Actual Simulation Configuration to Test Error Rate vs. FM Jamming Powers

```
NUMBER  NAME      NAME
   8    rdsprd    dsprd1

FLOATING-POINT PARAMETERS:                                    NONE

INTEGER PARAMETERS:                                           NONE

CHECKPOINT TRIGGERS:                                          NONE

OUTPUT CONNECTORS:
                          PORT        MODULE  PORT
                          NUMBER      NUMBER  NUMBER
                             3   to       9       1
PORTS OPEN FOR OUTPUT DATA TRANSFER:
                                                              NONE


MODULE  LIBRARY  USER
NUMBER  NAME     NAME
   9    smphld   samp

FLOATING-POINT PARAMETERS:                                    NONE

INTEGER PARAMETERS:
                          # of samples for initial delay        11
                          # of samples per integration         512
CHECKPOINT TRIGGERS:
                                                              NONE

OUTPUT CONNECTORS:
                          PORT        MODULE  PORT
```

Figure 3.29f Actual Simulation Configuration to Test Error Rate vs. FM Jamming Powers

```
                              NUMBER      NUMBER   NUMBER
                                 2    to    11       2
PORTS OPEN FOR OUTPUT DATA TRANSFER:
                                 2


MODULE   LIBRARY   USER
NUMBER   NAME      NAME
  10     fmjamr    fm1

FLOATING-POINT PARAMETERS:
                              jammer to signal power in db          9.000
                              average signal amplitude              1.000
                              i.f. or r.f. bandwidth                0.000
                              3db bandwidth of fm jammer            0.016
INTEGER PARAMETERS:
                              num. bands for frequency hop.            1
                              num. bands to jam                        1
                              num. samples to hold for fm            128
CHECKPOINT TRIGGERS:
                                                                  NONE
OUTPUT CONNECTORS:
                              PORT        MODULE   PORT
                              NUMBER      NUMBER   NUMBER
                                 3    to     7        1
                                 4    to    11        4
PORTS OPEN FOR OUTPUT DATA TRANSFER:
                                 4
```

Figure 3.29g Actual Simulation Configuration to Test Error Rate vs. FM Jamming Powers

MODULE NUMBER	LIBRARY NAME	USER NAME
11	biterr	error

FLOATING-POINT PARAMETERS:

NONE

INTEGER PARAMETERS:

initial delay in samples	523
# of samples per info bit	512
position within info bit	255
num of errors to stop after	5

CHECKPOINT TRIGGERS:

NONE

OUTPUT CONNECTORS:

NONE

EXERCISOR CHECKPOINT TRIGGERS:

NONE

Figure 3.29h Actual Simulation Configuration to Test Error Rate vs. Fil Jamming Powers

3.4 Example ICSSM Configuration

The purpose of this section is to illustrate how the ICSSM library modules previously described can be used to simulate a spread spectrum communications system in a jamming environment. The simulation examples presented in section 4 use the same basic ICSSM configuration and parameters discussed here with only some minor differences. The extension of the ICSSM configuration described here to those used in section 4 is straightforward thus specific ICSSM configurations for each case discussed in section 4 will not be included.

The spread spectrum transmitter and its simulation model will be discussed in section 3.4.1. The jammer will be presented in section 3.4.2 while the receiver will be discussed in 3.4.3. For each system component and the corresponding ICSSM module the relationship between the simulation and system parameters will be discussed.

The example considered here is a direct sequence spread spectrum system operating in noise loaded FM jamming. Figure 3.29 shows the ICSSM configuration for this case.

Since this configuration was run a new filter has been implemented (NFILT). It should be used in future configurations (see figure 3.20).

3.4.1 The Spread Spectrum Transmitter

A direct sequence system with a processing gain of ~32 (15dB), information bit rate of 64kbps (digital voice), and a chip rate of 2.048 Mcps was to be simulated. The system operates with continuous transmission (i.e. no time hopping) and on a single carrier frequency (i.e. no frequency hopping). To simulate this SS transmitter the frame formatter (FRAMEF), burst formatter (BURSTF), gold code generator (GOLDCD) and the information bit generator

(INFORM) must be configured as shown in figure 3.29. The primary parameters and their settings that govern the waveform characteristic are:

Time/Burst = BINT = 2048 sec

Time/sample = DELTI = 1 sec

Gold code length/Burst = CODLEN = 128

Number of Information bits/Burst = INFOLN = 4

Duty cycle = Duty = 1

Number of time samples/chip = NSAMP = 16

To achieve a processing gain of 15dB requires that there be 32 spreading chips per information bit. In this example a 128 chip code is directly multiplied by a 4 bit information sequence. To minimize the effects of aliasing a sample rate of 16 samples/chip were selected. Further to simplify the parameters specifications the time increment/sample, DELTI, was set to 1 sec. For the desired system DELTI should be .03 ns. This form of time scaling results in a chip rate of (1 chip/16 sec) = r_c. All bandwidth calculations are then made relative to the simulated r_c = 1/16. The simulated bit rate, r_b is then r_b = 1/512. Even though the system of interest here does not use burst transmissions for flexibility the library modules were designed around a burst system. Therefore to simulate continuous transmission the duty cycle is set to 1 and the waveform is described on a per burst basis. One burst time is set to the length of the PN code so time per burst = 2048 sec.

There are also a set of secondary parameters which must be set. These variables deal with the specifics of ICSSM. For the example considered here they are:

Time/invocation = TDURI = 1024 sec

Number of frames to generate = NFRAME = 1

Number of bursts/frame = NBURST = 250

Number of sync burst/frame = NSYNC = 0

Data Gold code number = CODNUM = 0

Sync Gold code number = SYNCOD = 1

ICSSM is a block based simulation, the length in time of each block is called the time/invocation. The burst formatter requires that

$$\frac{BINT}{TDURI} = \text{integer}$$

The length of the simulation, i.e. the number of information bits to process, is set by the product of NFRAME and NBURST along with the number of bits/burst. Here the simulation will generate 1 frame with 250 bursts with 4 bits/burst so 1000 information bits will be processed. The simulation structure allows for synchronization bursts. In this example none are needed so NSYNC=0 and the Gold code for the sync burst is meaningless. The PN code generation is designed to produce any member of the family codes for a specific code length, hence the first member of the Gold code family for length 128 is used so CODNUM is set to 0.

3.4.2 Jammer

A noise loaded FM jammer was to be considered. For this example the jammer to signal was to be set to 9dB with a jammer bandwidth of 512 khz (or $r_c/4$). The parameters to be set for the FMJAMR module are

Jammer to signal power = DBSGPW = 9

Jammer bandwidth = BW3DB = .016 (1/64)

Number of hop frequencies = NBANDS = 1

Number of bands to jam = NJAM = 1

Average signal amplitude = AMPLIT = 1 (same as AMPLX see section 3.2.4.2)

IF signal bandwidth = BANDWD = 0.0 (defaults to BWX see section 3.2.4.2)

Number of samples to hold = NHOLD = 128

For a jammer to signal ratio of 9dB the parameter DBSGPW is set to 9. With the given time scaling (i.e. r_c = 1/16) a jammer bandwidth of 512 kHz is simulated by setting the parameter BW3DB to 1/64. For the example considered here i.e. a DS system, there is no frequency hopping thus the number of hopping frequencies NBANDS is set to 1. To model a continuous jammer the number of bands to jam, NJAM, is set to NBANDS; in this case NJAM=1. To properly scale the jammer power this library module must know the signal amplitude, AMPLX, and the signal bandwidth, BWX. The parameters are set in the burst formatter module, BURSTF, and sent downstream. The FM jammer module allows the user to respecify these parameters, if the user wishes to use the original numbers (as will be the most common case) then the parameters signal amplitude, AMPLIT, and IF bandwidth, BANDWD are set to zero (see section 3.2.4.2 for details). To insure that the spectral shape of the jammer is Gaussian the parameter number of samples to hold, NHOLD, must be set to some large number; for this example 128 was adequate (see section 2.3.10 for details).

3.4.3 The Spread Spectrum Receiver

The spread spectrum receiver consists of six library modules, EXPAND (which is used twice), FILTER, RDSPRD, SMPHLD, and BITERR. These modules must be configured as shown in figure 3.29. The receiver must filter the received signal, obtain code and bit synchronization, despread the DS signal, perform bit detection and count errors. All delays required for synchronization were

estimated off line using the real correlation modulator CORLR (see section 3.2.7.2 for details)

Code synchronization is obtained by first estimating the code delay through the system. In this case the delay was 11 samples. Next the transmitted code must be delayed by this amount. This function is performed by the EXPAND module (module #3 in Fig. 3.29). In addition to delaying the chip waveform this module converts (expands) the generated code sequence of 0's and 1's into the correct delayed waveform. The output of this module is one of the inputs to the despreader module, RDSPRD. The parameters of this module are:

Duty cycle = DUTY = 1

Number of elements in the input sequence of 0's and 1's = NBITS = 128

Number of samples to delay the waveform = NDELAY = 11

Bit synchronization is also obtained by using a EXPAND module (module #4 in figure 3.29). In this case the output of the EXPAND module is the input of the bit error counter BITERR. That is, this module provides the correct timing for deciding whether an error occured. The parameters of this module are:

Duty cycle = DUTY = 1

Number of elements in the input sequence of 0's and 1's = NBITS = 4

Number of samples to delay the waveform = NDELAY = 523

The delay is calculated from the waveform delay of 11 samples plus one bit time of 512 samples.

For this example the receiver's IF filter is modeled as a 6th order butterworth with a 3dB bandwidth equal to the chip rate. The parameters of the filter module FILTER are then

Lowpass corner frequency = FCT = .063 (1/16)

ripple = DBRIPL = 0

Type = FL = 1.0

Filter order = IFLTOR = 6

The output of the FILTER module along with a delayed version of the PN code waveform are input to the despreader (RDSPRD) module. This module performs the despreading function and has no parameters.

The despread signal is the input to the sample and hold module (SMPHLD). This module includes the demodulation/integration function. This module requires the integration time be specified which is usually one bit time, here 512 samples. Also a settle time (delay) is specified. This is the waveform delay. The parameters for the SMPHLD module are

Number of samples for initial delay = NDELAY = 11

Number of samples per integration = NSAMP = 512

The output of the sample and hold is the received bit waveform. The bit error estimator compares the transmitted bit waveform (the output of the EXPAND module, module #4) to the received bit waveform and counts the number of errors. The user is given the option to stop the simulation after a set number of errors have been detected. For example this simulation will stop after 1000 bits have been processed or 5 errors have been detected. The user is also given the option of placing the point in time where the two waveforms are compared. This module also requires a settle time which is the same as the bit time and the waveform delay. The parameters for the BITERR module are:

Initial delay in samples = INTDLY = 523

Number of samples per bit = ISAMPL = 512

Sampling Position = ISPOS = 255

Number of errors to count = NERROR = 5

The BITERR module is the terminator module for the ICSSM configuration. This module writes the results to a file as previously explained.

3.5 Graphics Package and its Support Package

ICSSM post processing routines lacked graphics output. Toward this end two stand alone packages are provided for inspection of ICSSM generated signals (ICSSM terms them coefficients). The graphics package implements signal plots, eye diagrams, and spectral plots. It should be emphatically noted that currently one may only observe one output port at a time.

During configuration one should specify only one output port be available for post-processing; this is the port that one is interested in plotting.

After the configuration has been run (exercised) no post-processing by ICSSM is required. To transform the signal file (exer4_dat) containing the single output port to a format suitable to plot one needs simply to specify program "split" to run. Program "split" asks no questions of the user. It reads from file "exer4_dat" and writes to file "file01".

To see a signal plot, eye diagram, or magnitude spectral plot of the signal in "file01" run program "pltsig" while using a Tektronics 4014 terminal. Program "pltsig" prompts the user for all inputs, such as starting sample, number of samples to plot, plot width in the case that the eye diagram is specified and the number of samples per "bit" (defined as "chip" in the burst formatter).

The magnitude spectral plot provides signal power in the upper right hand corner of the screen after the spectrum has been plotted.

3.6 System Delay Calculation

To determine the system delay of the system in Fig. 3.29 a similar system should be configured with modules #4, #8, #9, and #10 deleted and the correlator module (CORLR) inserted after the filter. To satisfy ICSSM topology requirements a single port terminator must also be added to "absorb" the output of the correlator.

Most often, for the correlator to acquire a complete burst, it must be invoked more than once (since the burst interval time is frequently greater than "tdur"). Because of this, in the case of no system delay between the reference signal and the input signal, the correlator output will not reach a maximum until both a complete burst is ingested and the last sample in the input stream of the completed burst is used to calculate the correlation. To quantify this correlation maximum as to its sample ordinate, its ordinate will be (burst interval/deltx)-1. This sample ordinate should be used as the zero delay reference point. When there is a delay in the system the correlation maximum will occur at a later sample.

A terminator module, given the above information, would be easy to construct to determine the system delay from the correlator output.

System delay may also be determined by visual inspection of the correlation output by using programs "split" and "pltsig". These two programs are described in section 3.5.

4.0 Simulation Examples

Several experiments have been conducted to evaluate the performance of the spread-spectrum system simulator. The validity of the simulation is demonstrated through a comparison of the bit error probability predicted by theory or measured from hardware to that calculated using the simulation. The comparison is made as a function of the jammer-to-signal power ratio (J/S) or the signal-to-noise (S/N) power ratio. A single SS system configuration operating in different interference environments was simulated. Tone and noise-loaded FM jamming were considered along with broadband white Gaussian noise. The results presented here establish the validity of the spread-spectrum simulation software.

The SS system transmitter and receiver simulated in these experiments is configured as shown in Figures 4.1 and 4.2, respectively. The parameters of the system are listed in Table 4.1. A typical burst is shown in Figure 4.3.

The performance of the SS system described above was first evaluated in the presence of broadband white Gaussian noise. The probability of error, P_e, of this SS system operating in broadband noise is identical to a conventional BPSK system. The P_e is given in this case by [17]

$$P_e = Q\sqrt{(S/N)}$$

where

$$Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty e^{-z^2/2} dz$$

Figure 4.4 shows the simulated and the calculated P_e vs. S/N. For a given P_e the difference between the simulated and theoretically derived S/N was less than .4 dB. This difference can be attributed to system losses. These losses are due to filtering and synchronization errors.

Figure 4.1

SYSTEM BLOCK DIAGRAM

TRANSMITTER:

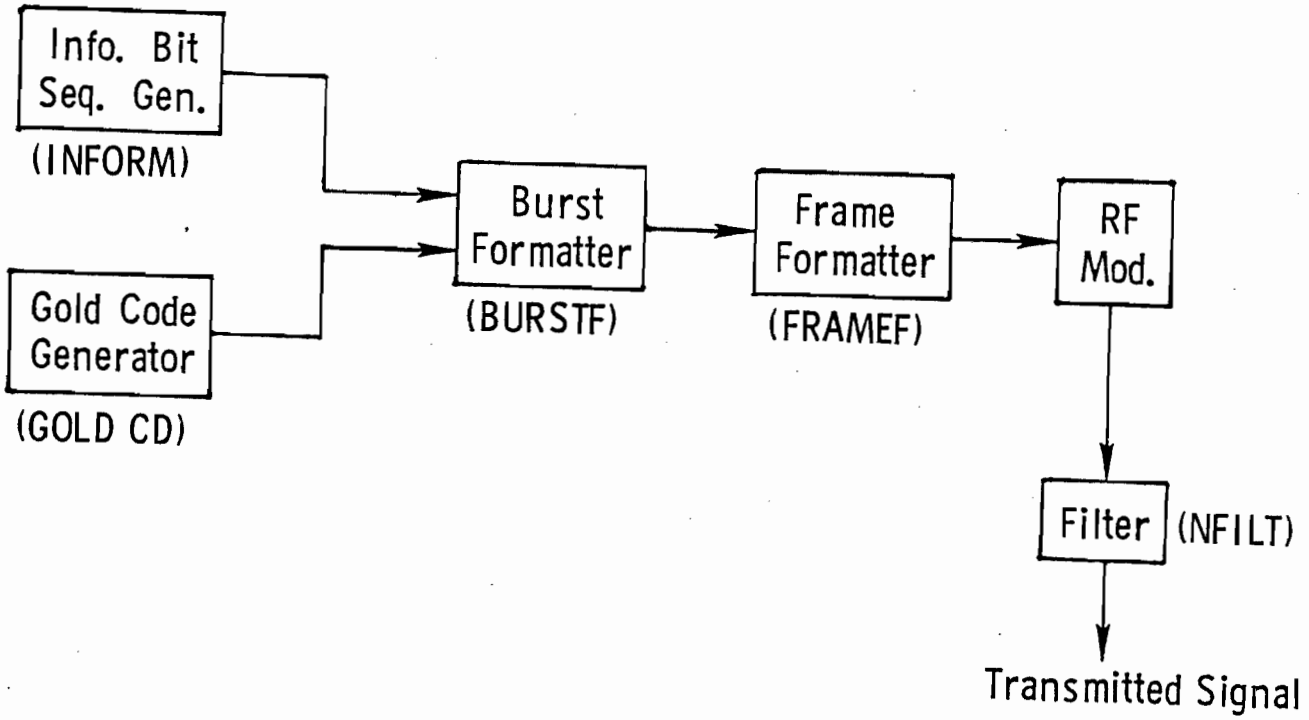
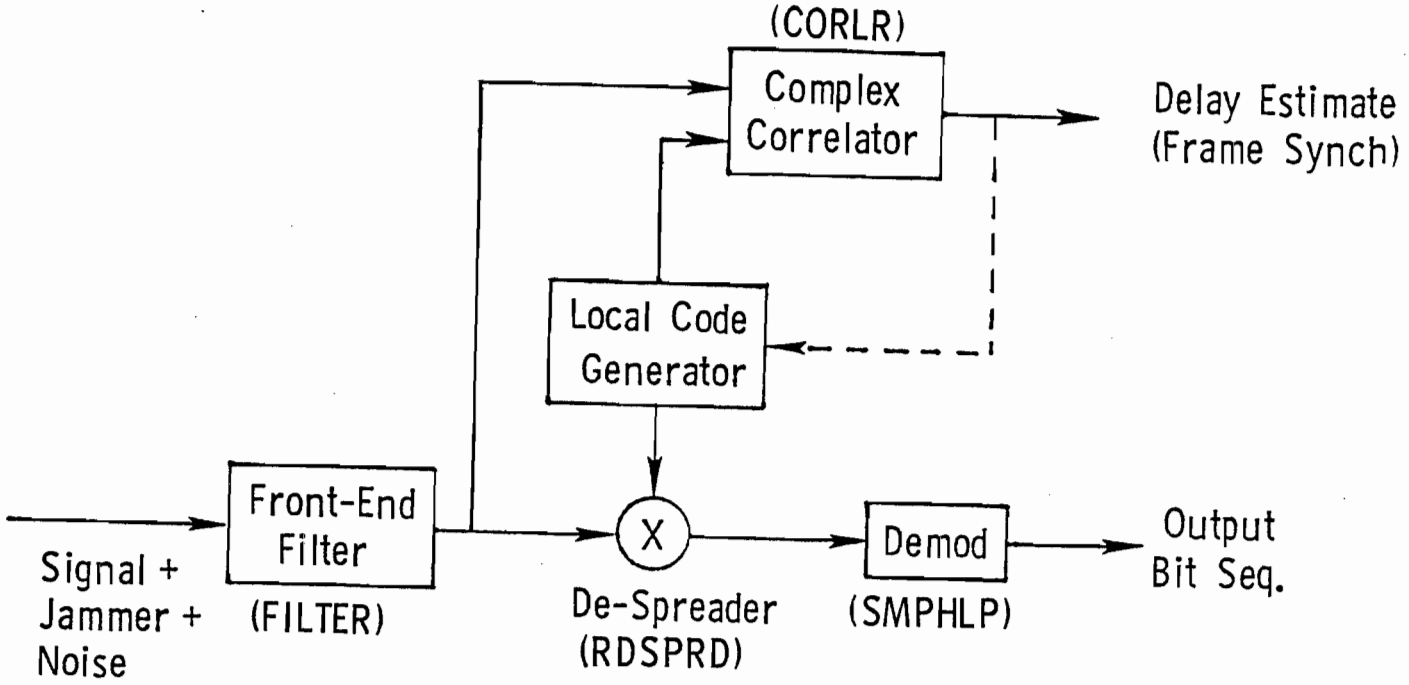


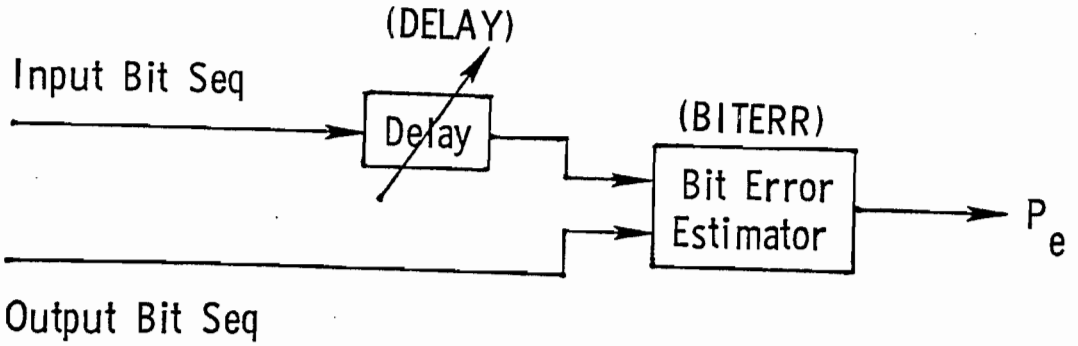
Figure 4.2

SYSTEM BLOCK DIAGRAM

3. Receiver

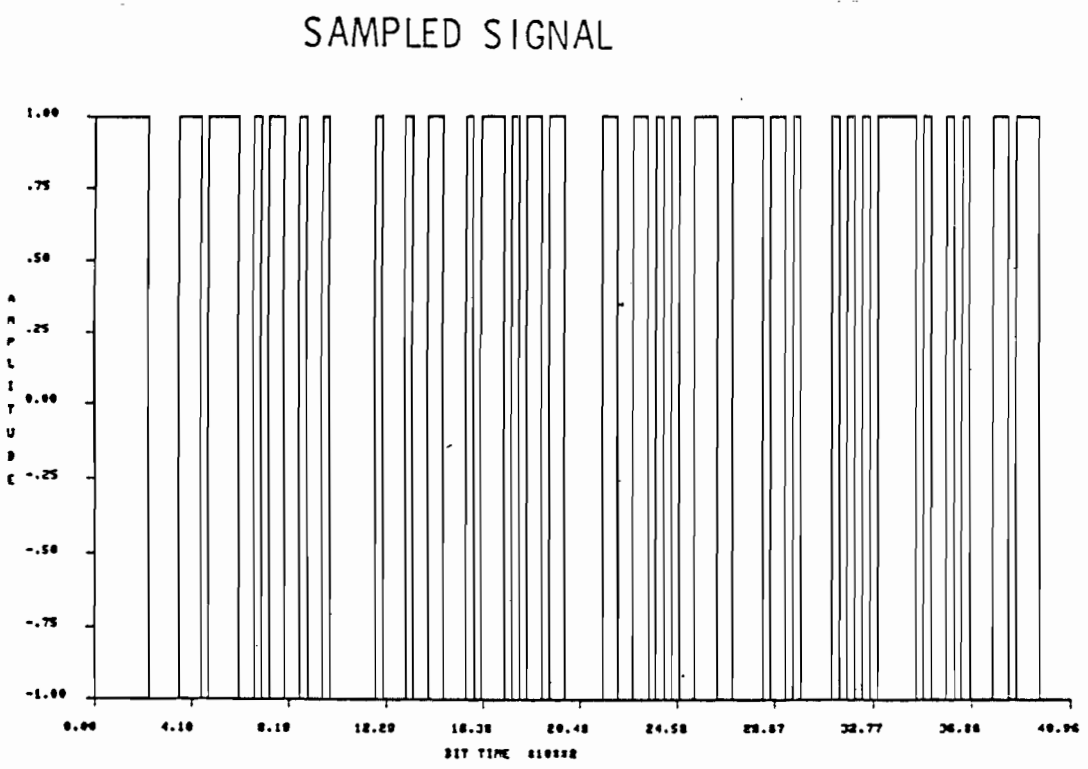


(a)



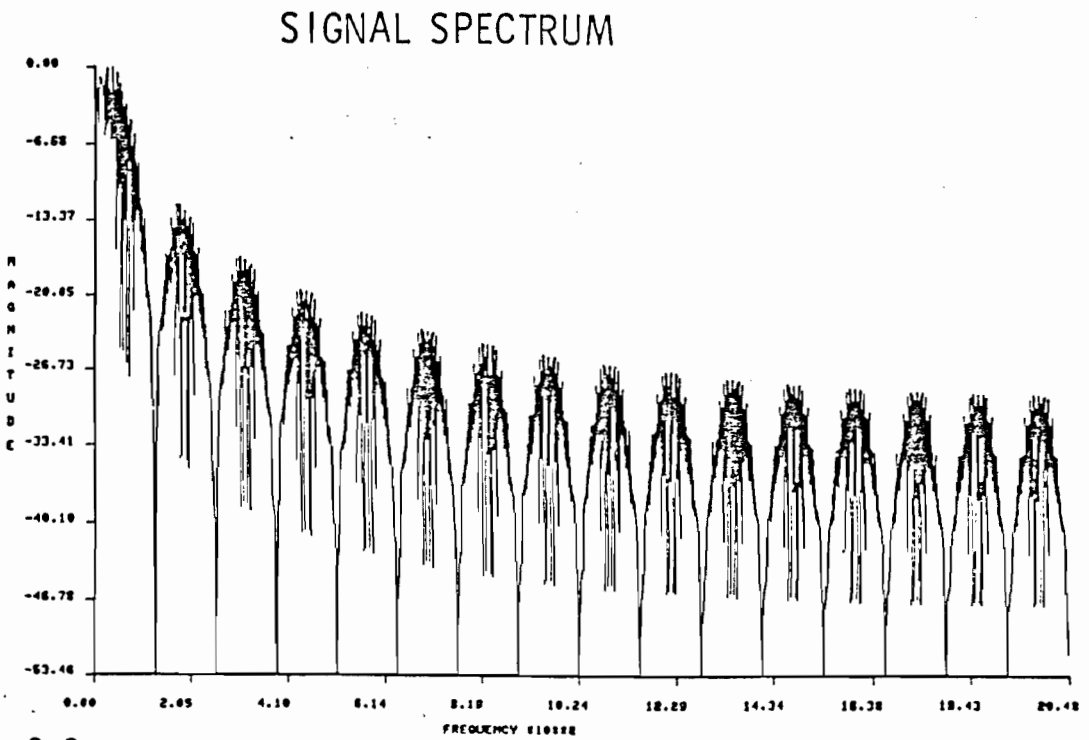
(b)

Figure 4.3 Time Sampled Signal and its Spectrum
for a Single Burst



(a)

TOTAL POWER= 0.000000000E+00



0.0

(b)

$\frac{f_s}{0.5}$

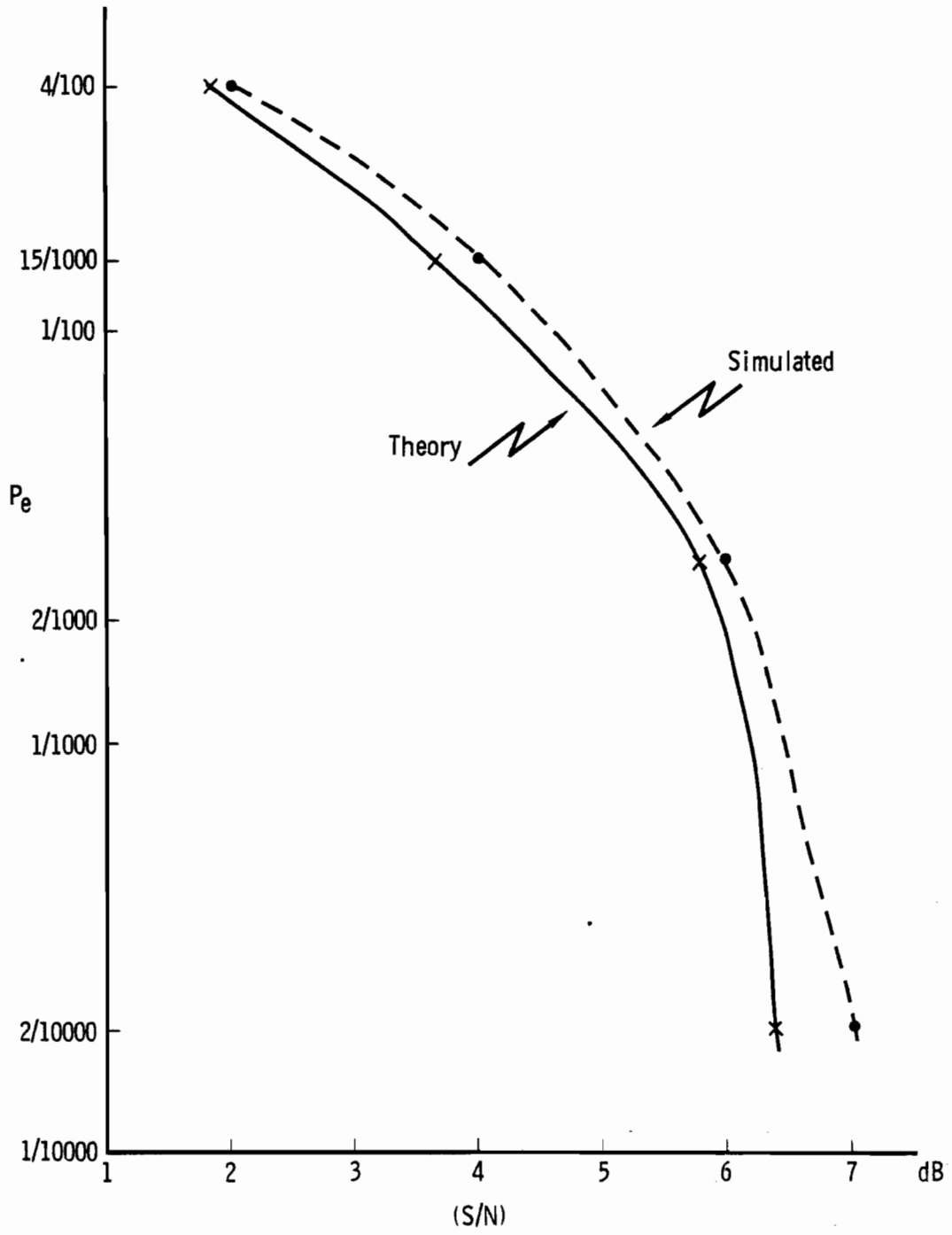


Figure 4.4 P_e vs. (S/N) for SS System in Broadband White Gaussian Noise

Bit Rate = R_b = 64 kbps

Chip Rate = R_c = 2.04 MMcps

Duty Cycle = 1

Gold Code Length = 127

Number of Code Chips/Burst = 128

Number of Information Bits/Burst = 4

Spreading Modulation Format = Multiply

Number of Time Samples/Chip = 16

Number of Sync Bursts/Frame = 3

Number of Information Bits/Frame = 92

Type of RF Modulation = BPSK

Receiver Filter parameters

- Type = Butterworth
- Order = 6
- Bandwidth = R_c

Type of Demodulation = Integrate and Dump

Number of Bands Jammed = 23 (the jammer is present on all bursts)

Ideal Dehopping

Table 4.1

Spread-Spectrum System Parameters

The SS system was next simulated in a tone jamming environment. For a tone jammer at the carrier frequency the P_e is given by

$$P_e = Q\sqrt{G_p(S/J)\cdot L}$$

where

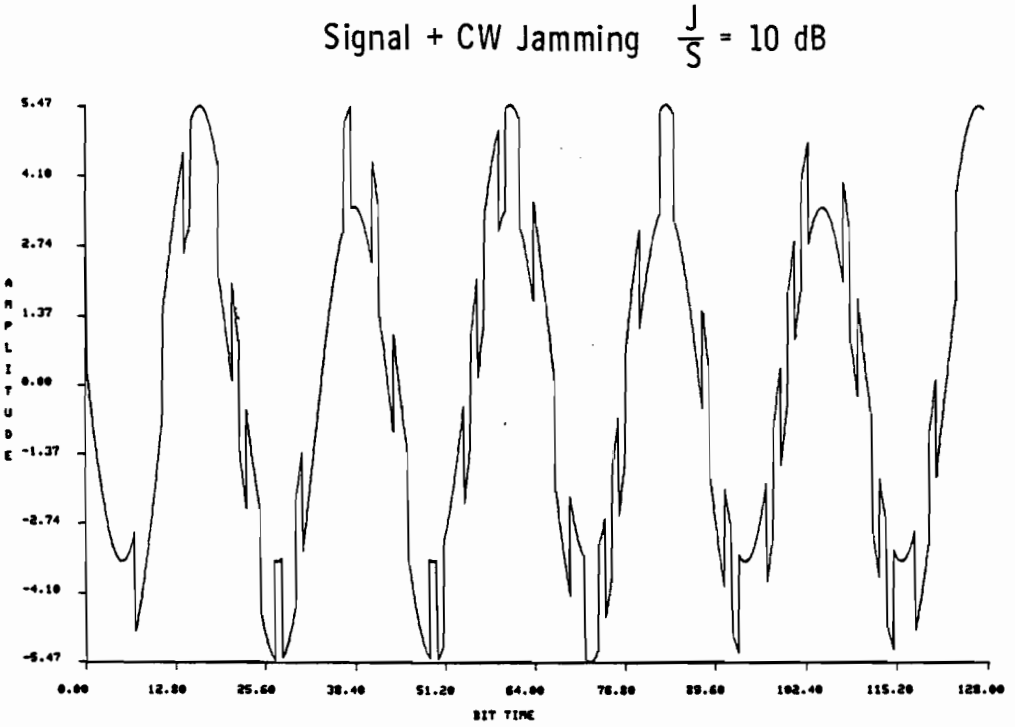
S/J = signal-to-jammer power ratio,

G_p = processing gain,

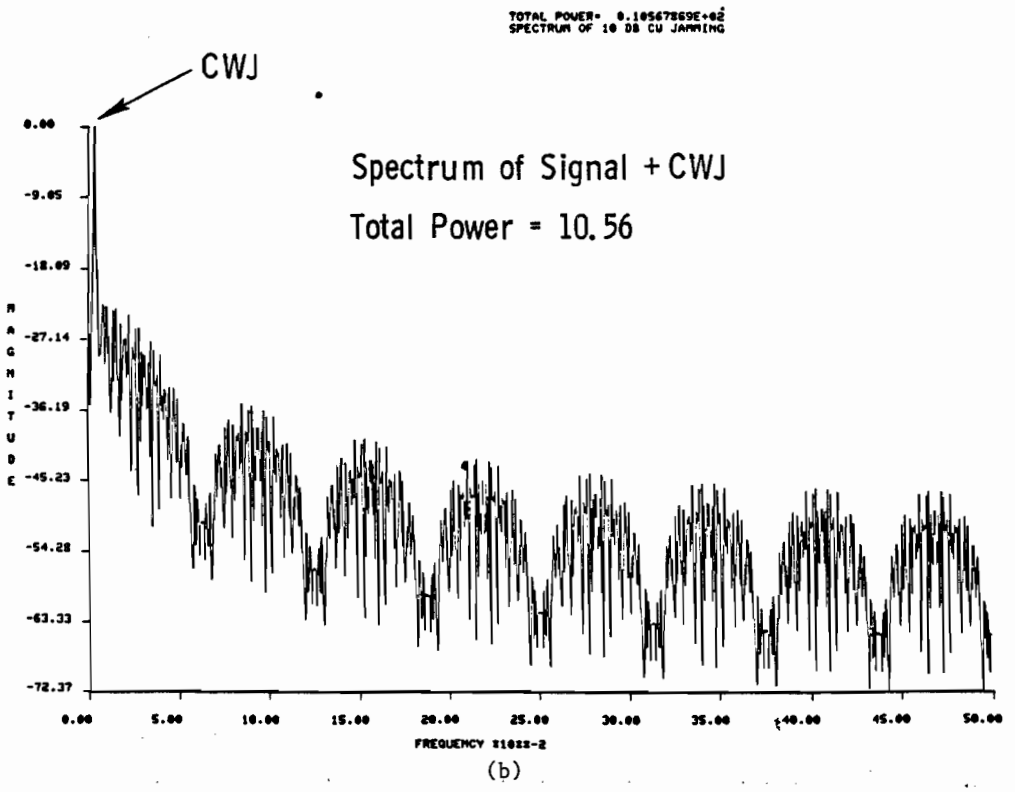
L = system losses.

For the results presented here $G_p = 32$. Figure 4.5 shows the waveform and spectrum at the receiver input; this signal contains both the desired SS signal and a tone jammer. The location of the tone within the RF/IF bandwidth was randomly selected. Figure 4.6 shows the effect of the receiver front-end filter. In Figures 4.5 and 4.6 the tone jammer is clearly evident. In Figure 4.7 the signal and its spectrum after despreading are shown. The effect of the tone jammer is greatly reduced. Figure 4.8 displays the time waveform at the output of the integrate and dump filter. By sampling and thresholding the integrate and dump waveform, a received bit sequence is generated and a P_e calculated. Figure 4.9 shows a comparison of the theoretical to the simulated P_e. The differences between the calculated and simulated P_e are within our expectations.

A noise-loaded FM jamming environment was also simulated. The jammer's bandwidth was set at $R_c/4$ (where R_c = chip rate). The center frequency of the jammer was set at the burst center frequency for one experiment and at an offset of $R_c/4$ for a second experiment. The receiver input waveform and spectrum is shown in Figure 4.10. The receiver filter output is displayed in Figure 4.11. The despread signal and integrate and dump output are shown in Figures 4.12 and 4.13, respectively. Measurements, i.e., P_e vs. J/S data, were obtained from a system with similar parameters. A comparison of the

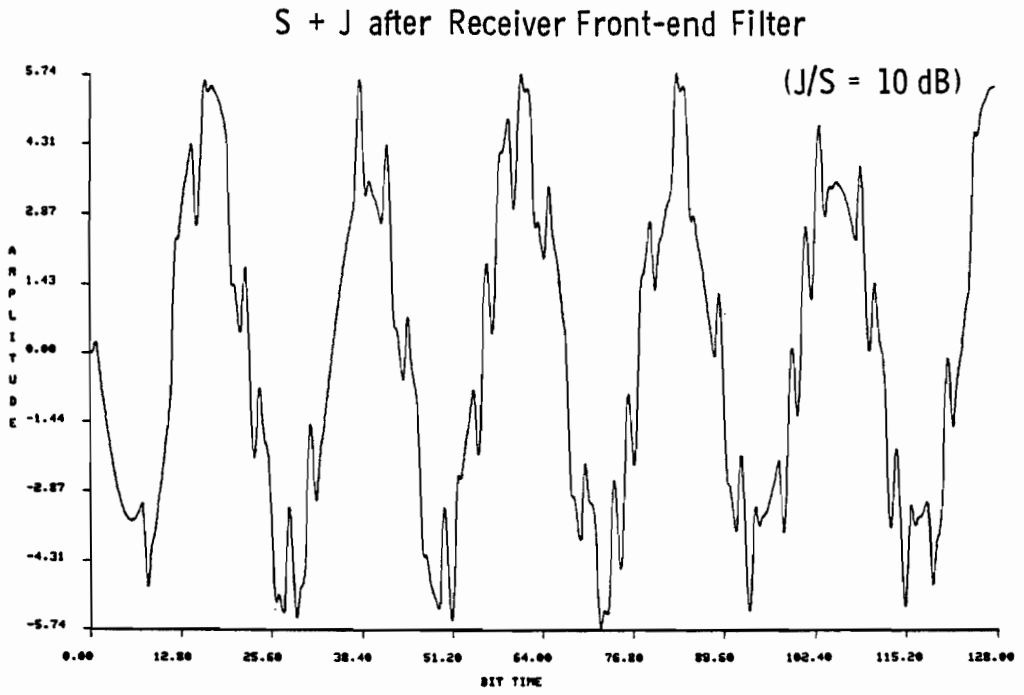


(a)

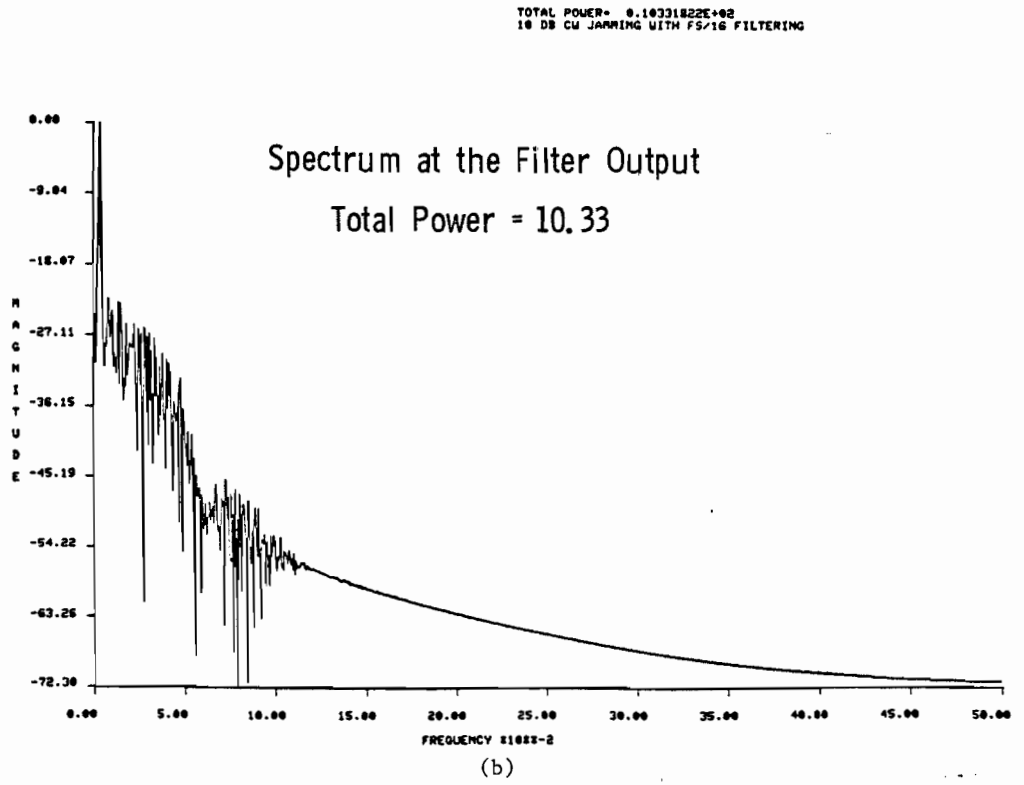


(b)

Figure 4.5 Time and Frequency Display of SS Signal with a Tone Jammer (Δf Random)

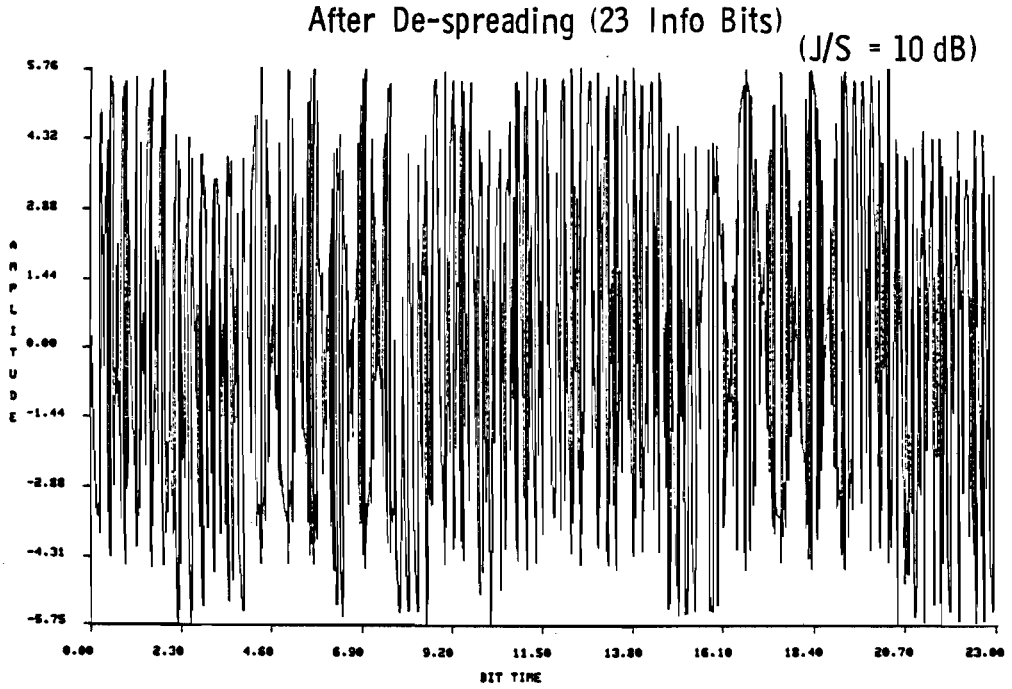


(a)

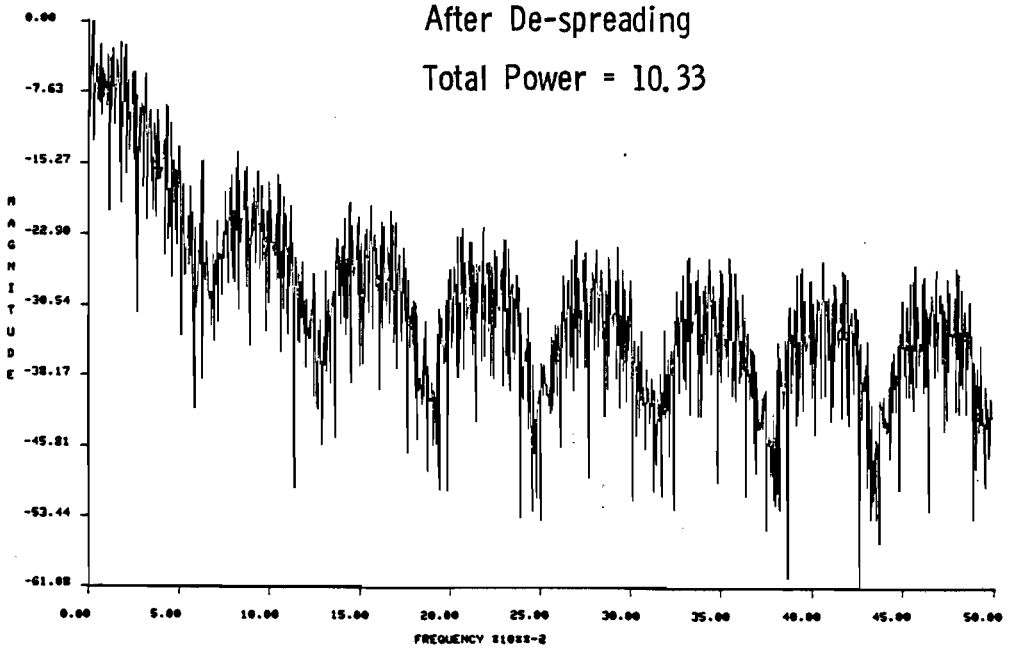


(b)

Figure 4.6 Time and Frequency Display of SS Signal with a Tone Jammer after IF Filter



(a)



(b)

Figure 4.7 Time and Frequency Display of SS Signal with a Tone Jammer after Signal Despreading

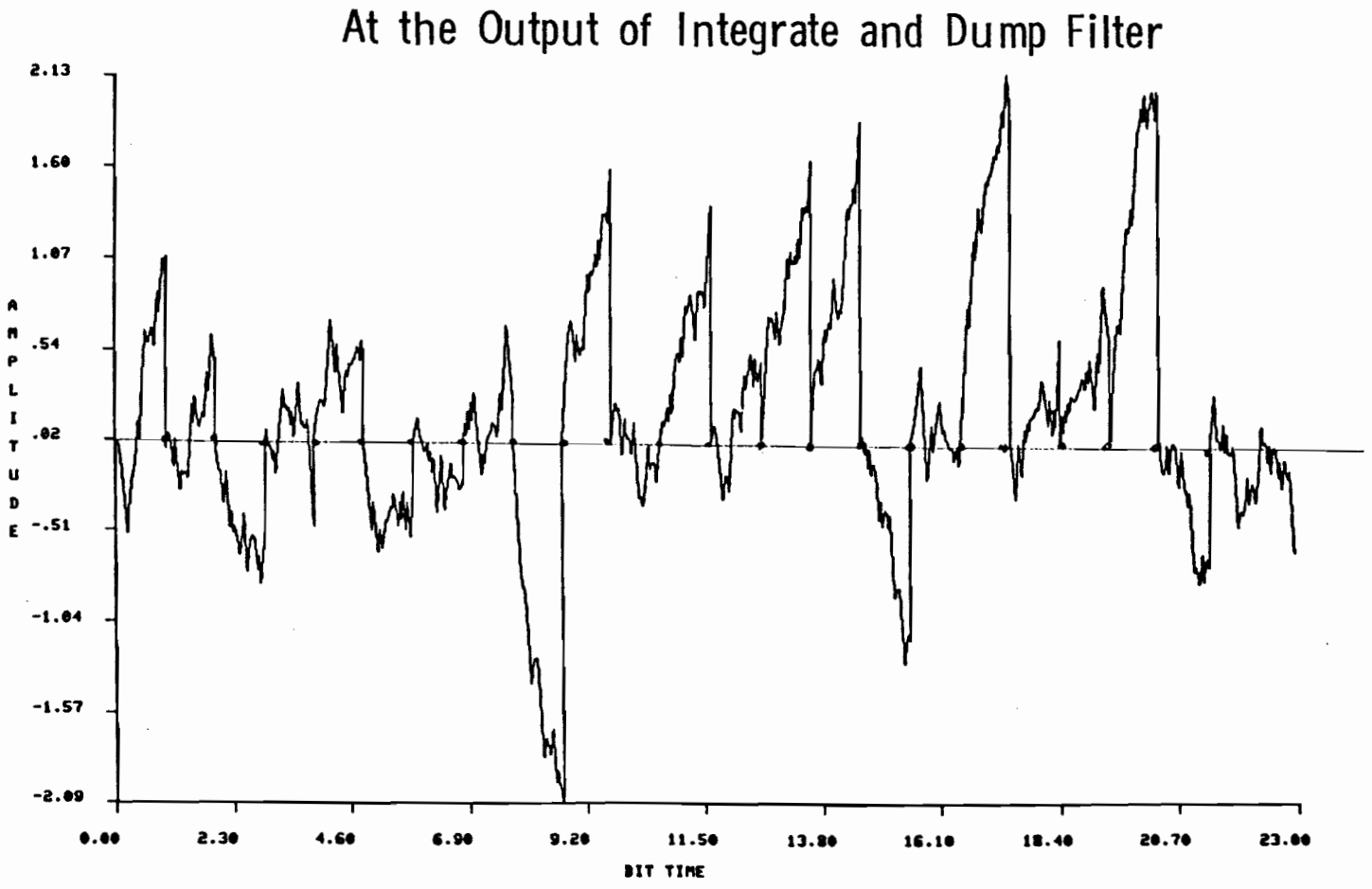
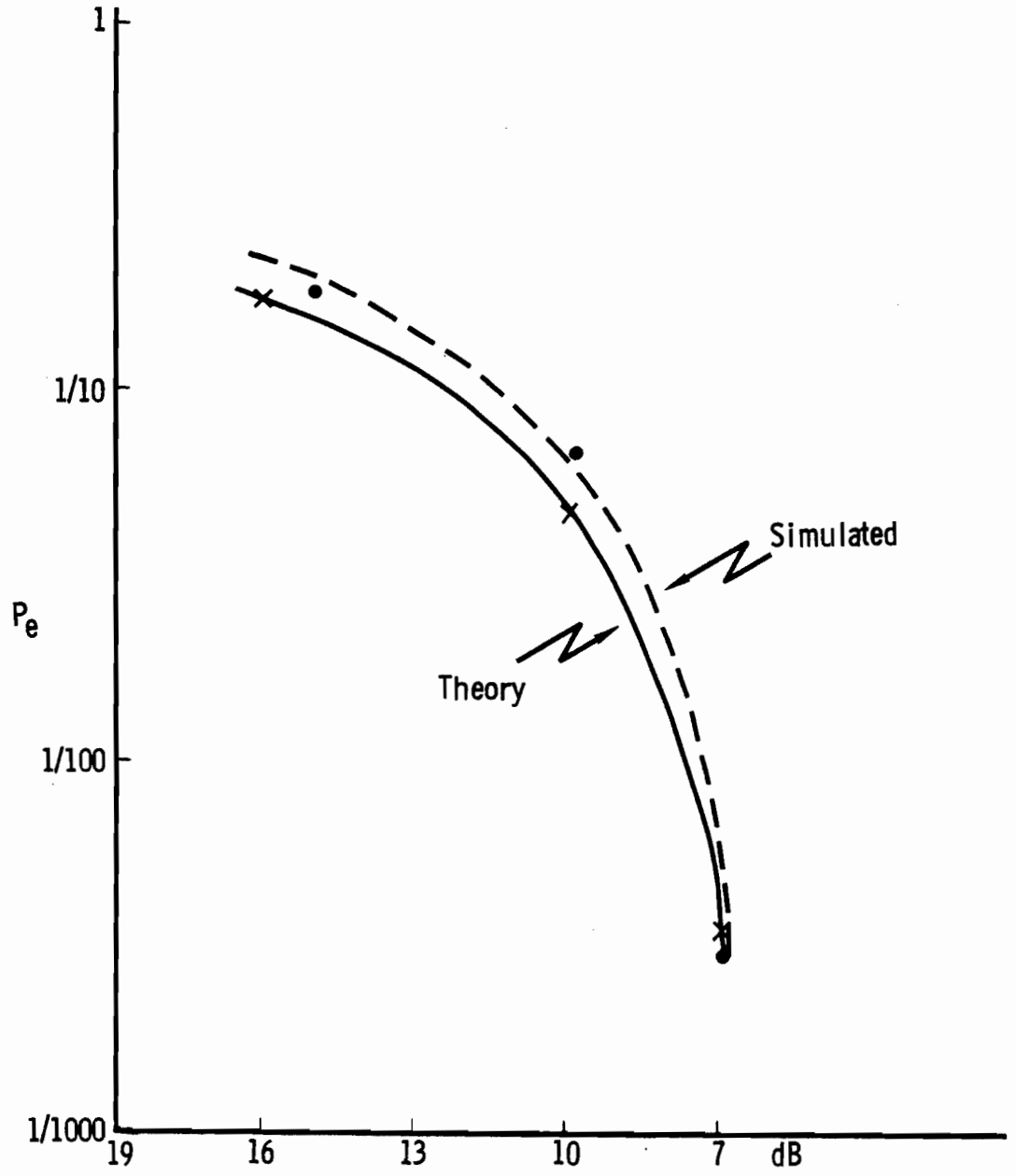
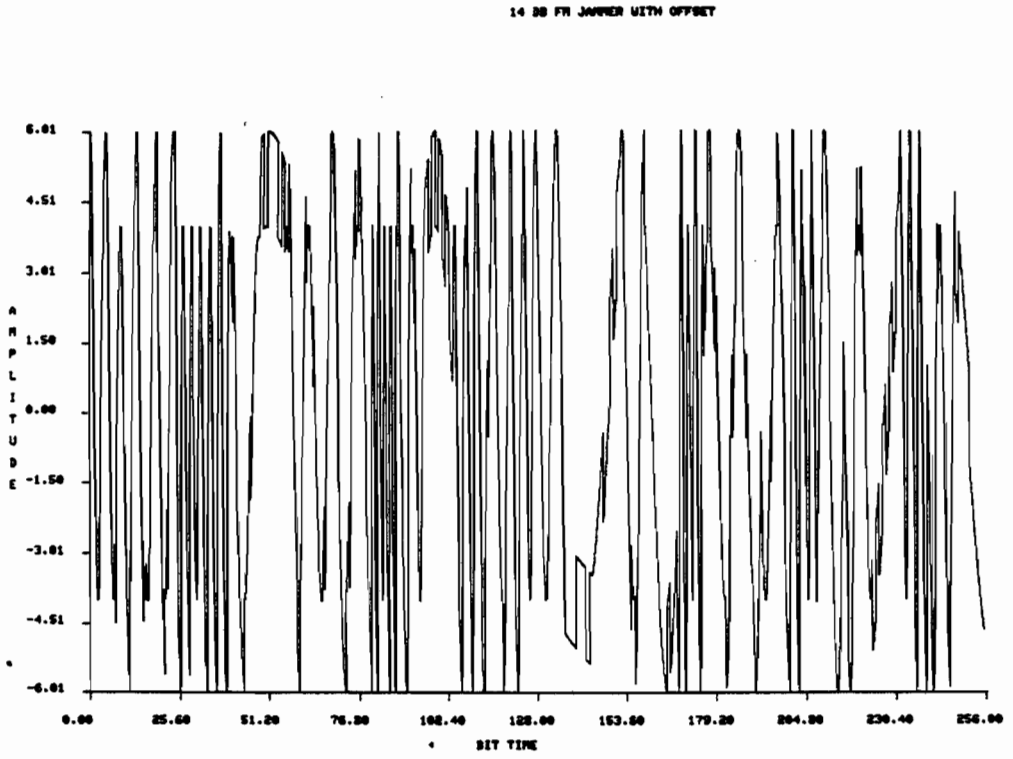


Figure 4.8 Time Waveform at the Output of Integrate and Dump Filter

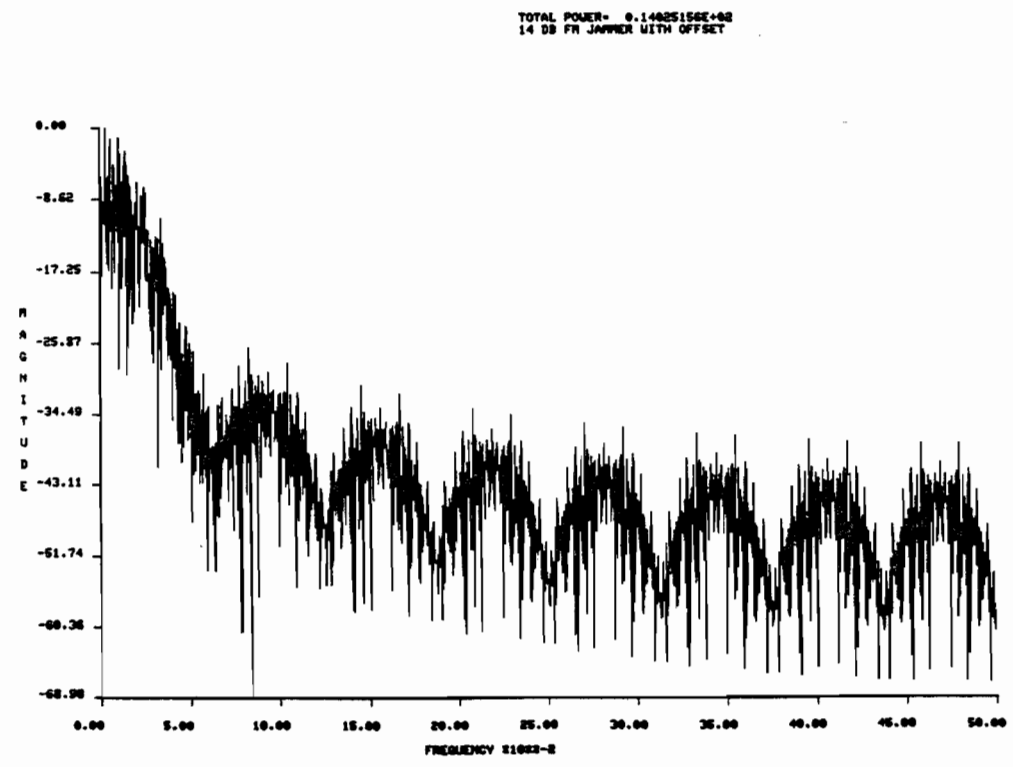


Tone Jamming; Duty Cycle = 1
Theory - Jammer at middle of signal band
Simulated - Jammer frequency is random

Figure 4.9 P_e vs. (J/S) for SS System with a Tone Jammer

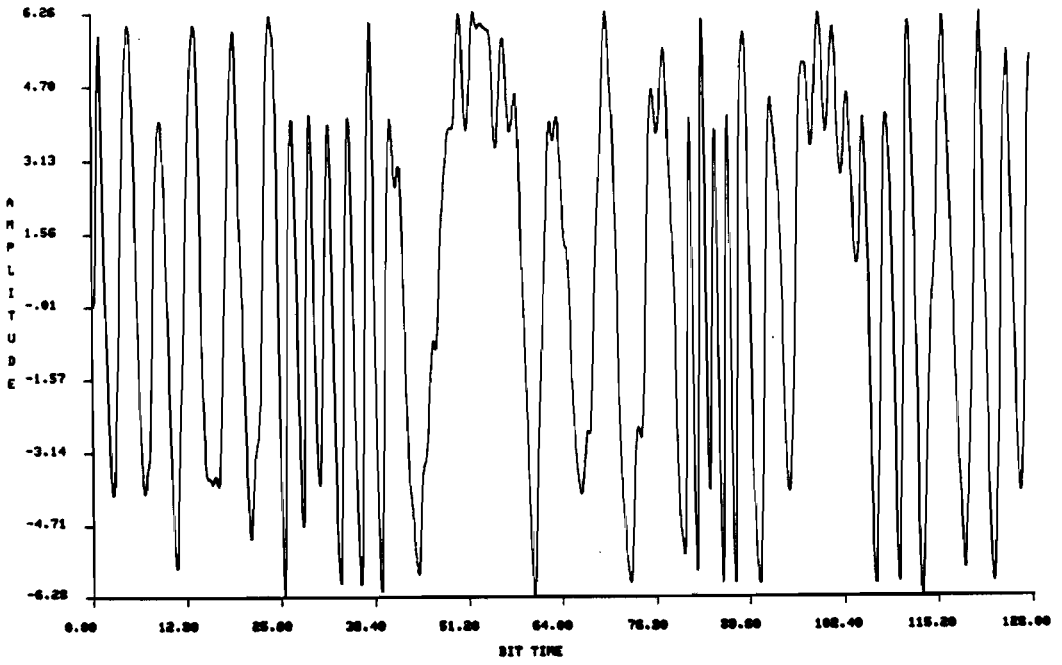


(a)

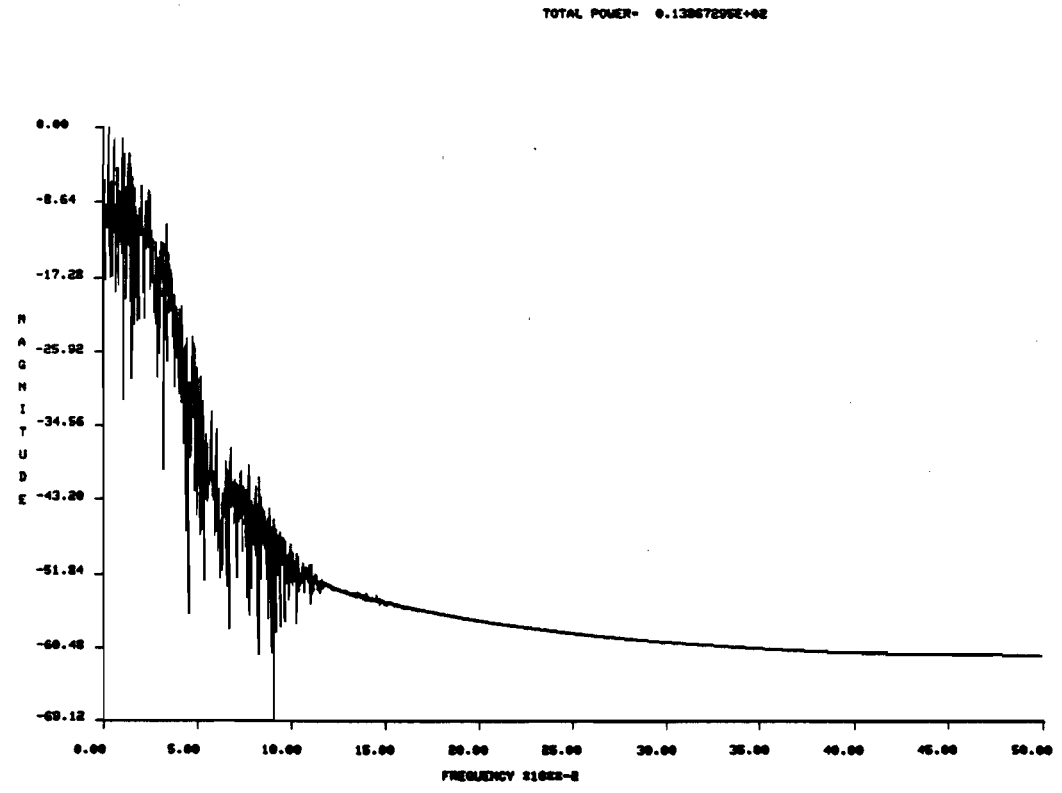


(b)

Figure 4.10 Time and Frequency Display of a SS Signal with Noise-Loaded FM Jamming

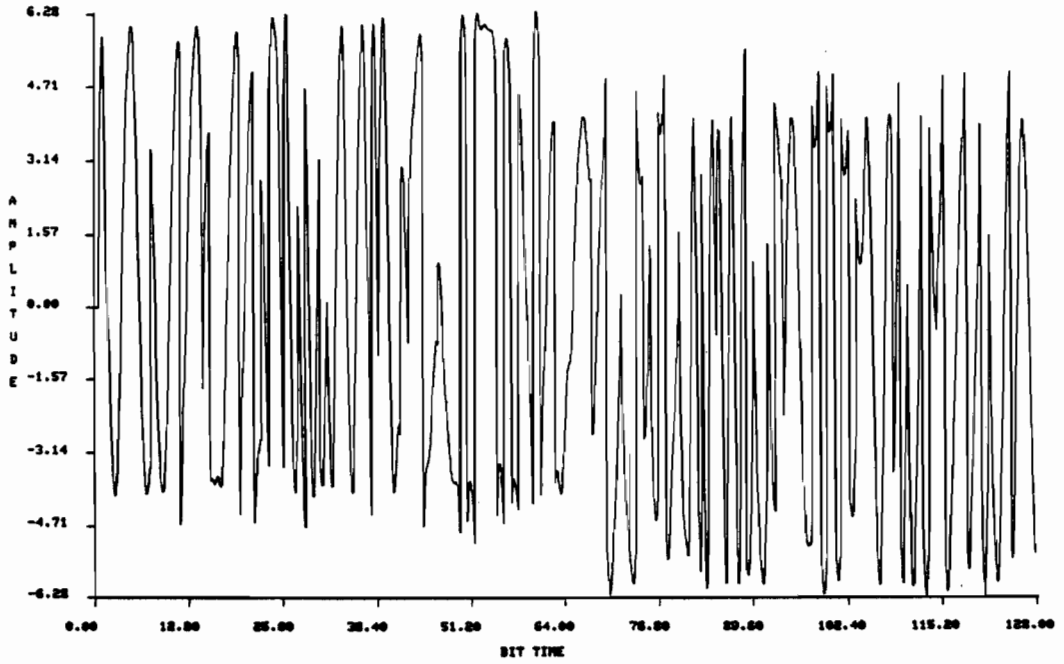


(a)

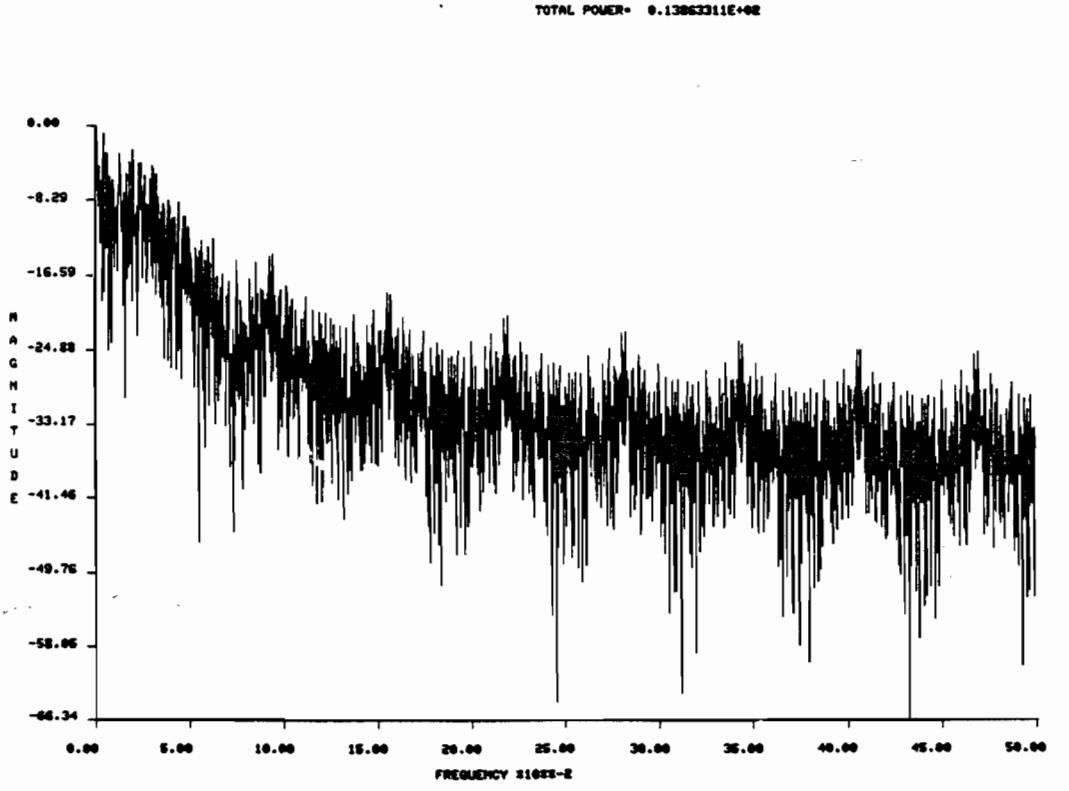


(b)

Figure 4.11 Time and Frequency Display of a SS Signal with Noise-Loaded FM Jamming after IF Filter



(a)



(b)

Figure 4.12 Time and Frequency Display of a SS Signal with Noise-Loaded FM Jamming after Despreading

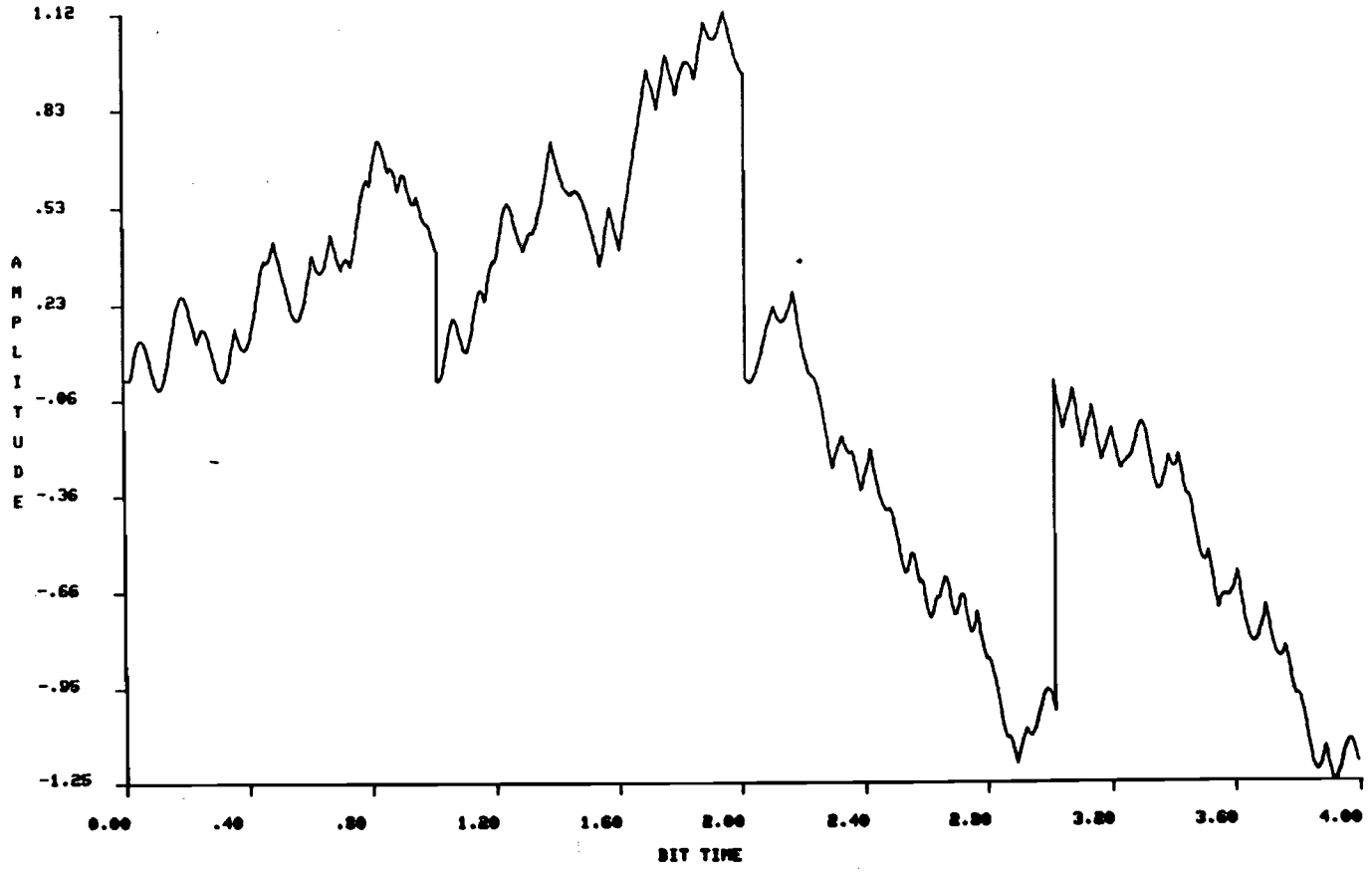


Figure 4.13 Time Waveform at the Output of the Integrate and Dump Filter (Contains 4 Information Bits)

simulated and measured results are shown in Figure 4.14 for the case of no frequency offset. Note the simulated interference environment contained only the jammer, no thermal noise was included. For a given P_e the maximum difference in J/S between the measured and simulated data was -1.75 dB. Figure 4.15 shows the comparison for the $R_c/4$ frequency offset. In this case the maximum J/S difference is -2.25 dB. Again these results are within our expectations.

To illustrate the synchronization capabilities of the simulation an experiment was conducted using the complex correlator. The PN code phase is unknown to the receiver and must be estimated from the received signal. The code phase is estimated in the SS system considered here by correlating on the sync burst codes. Figures 4.16a shows the output of the correlator for a frame with three sync bursts. The only difference between the information and sync burst is the Gold code, i.e., no data is on the information burst. Figure 4.16b presents the case with data. Note the increase in the cross-correlation level. This can be improved by increasing the processing gain, i.e., by spreading each bit over more chips. Figures 4.17a and b show the performance of the correlation in broadband white noise. As expected the performance is quite good.

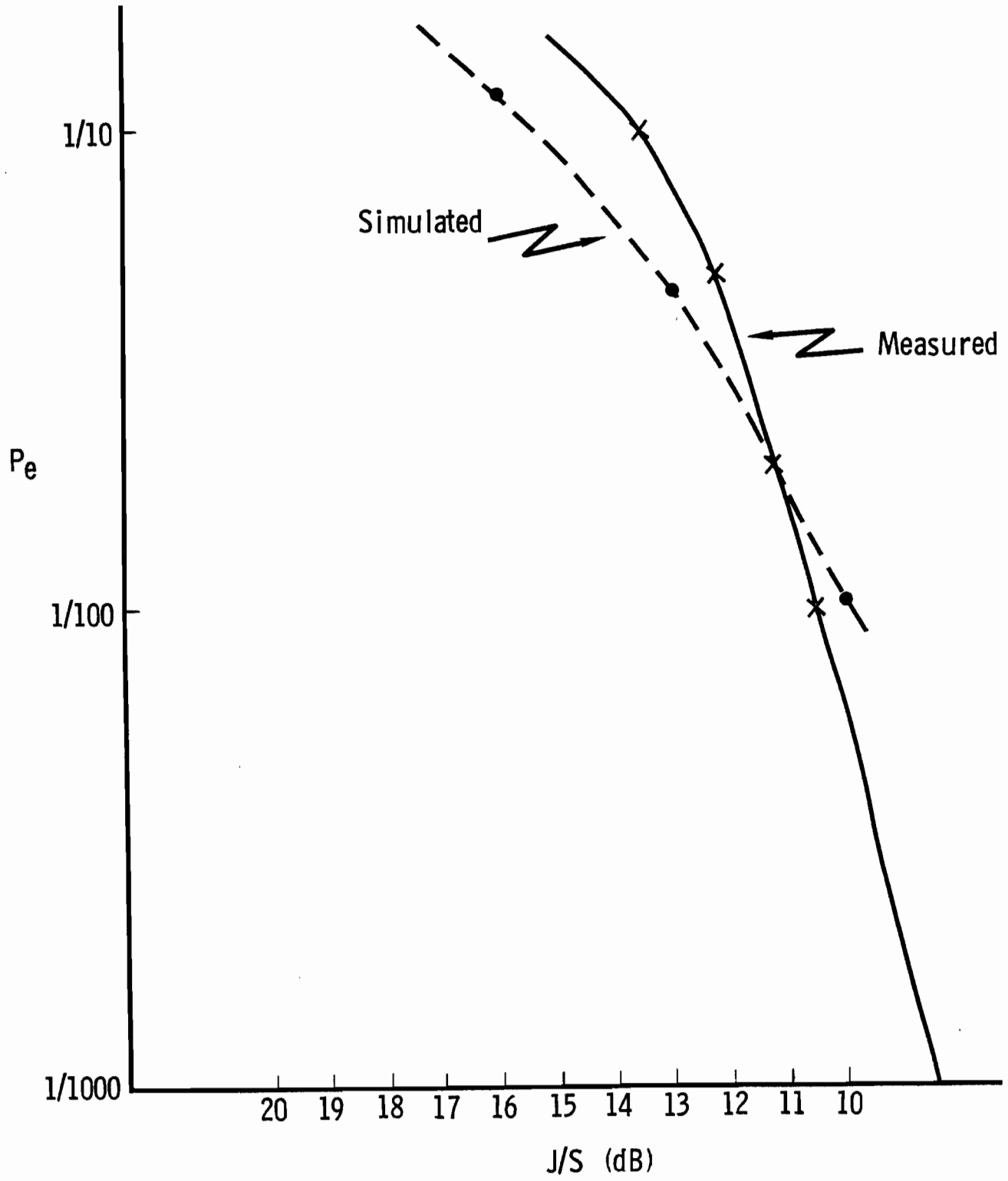


Figure 4.14 P_e vs. (J/S) for a Spread Spectrum System with Noise-Loaded FM Jamming ($\Delta f = 0$)

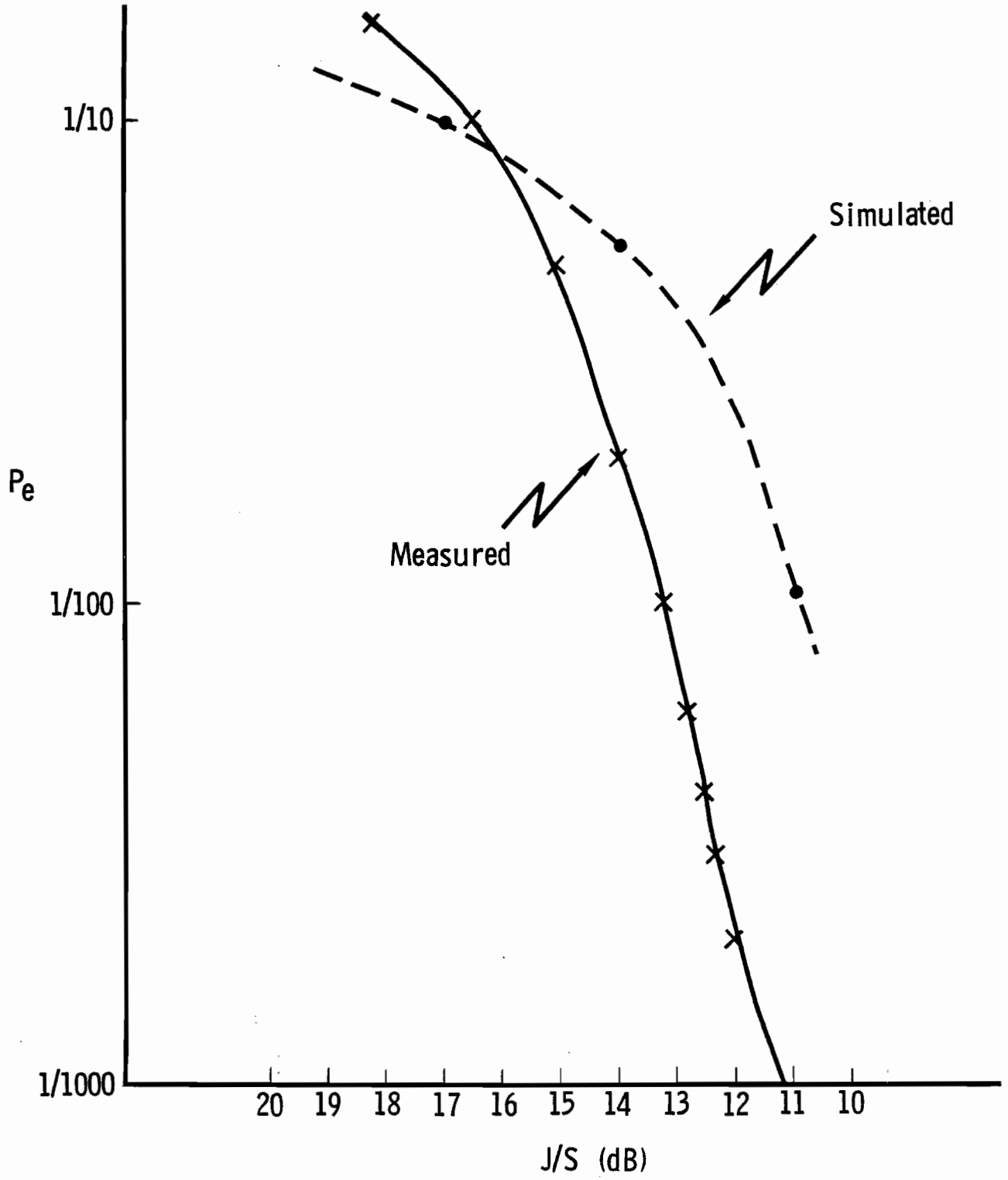
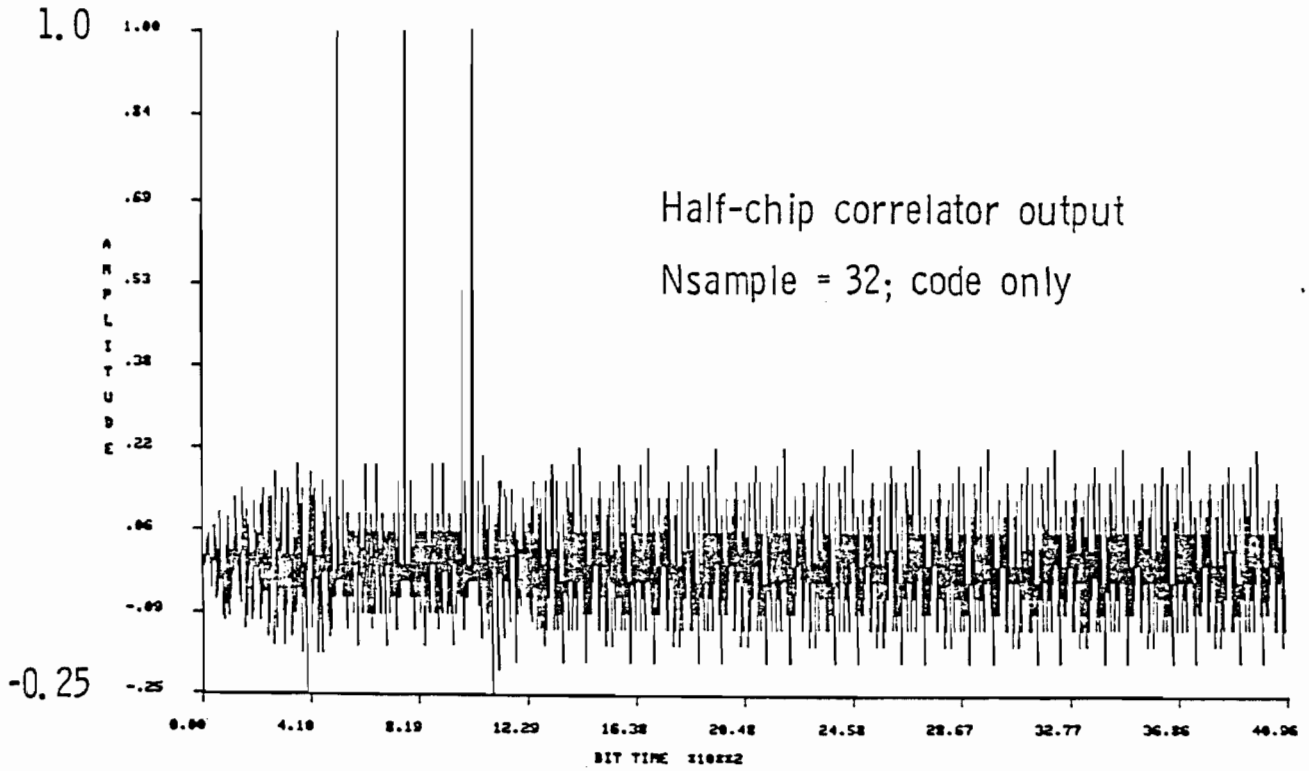


Figure 4.15 P_e vs. (J/S) for a Spread Spectrum System with Noise-Loaded FM Jamming ($\Delta f = R_c/4$)

Figure 4.16 Output of Complex Correlator
(No Noise)



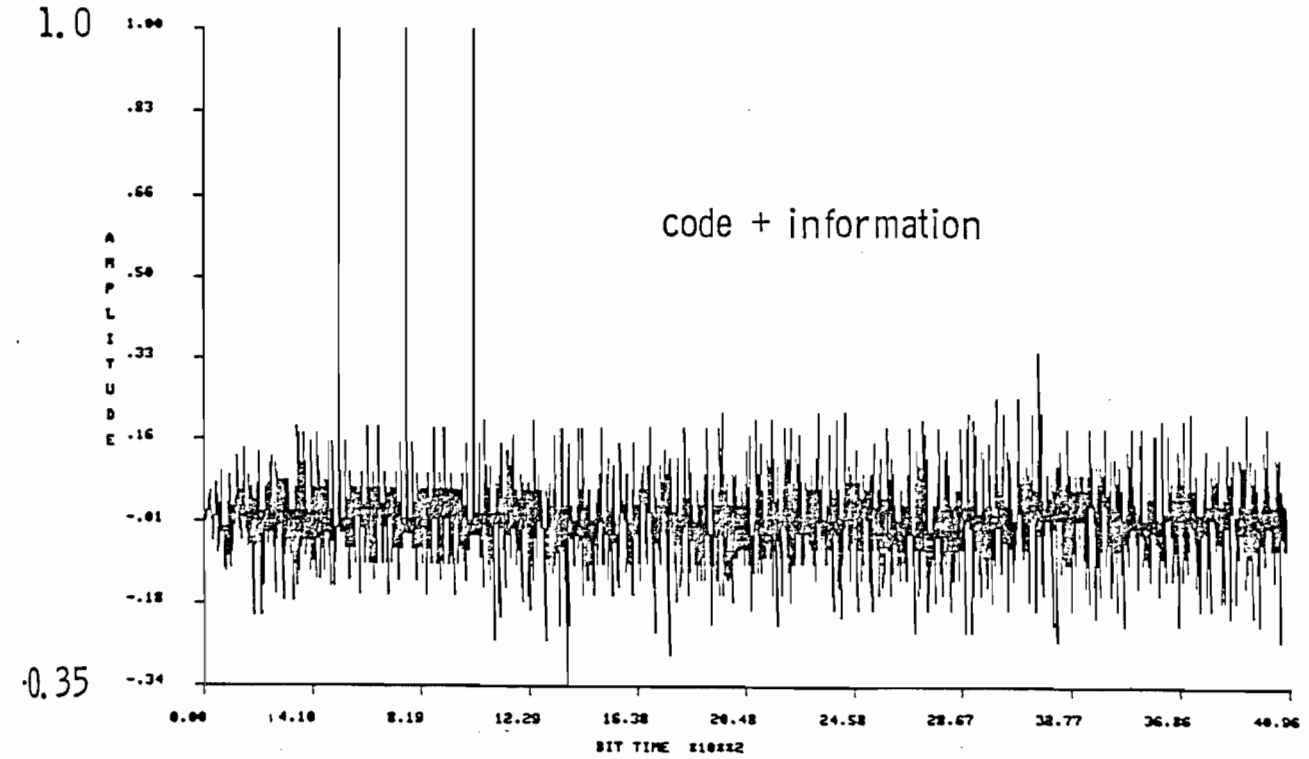
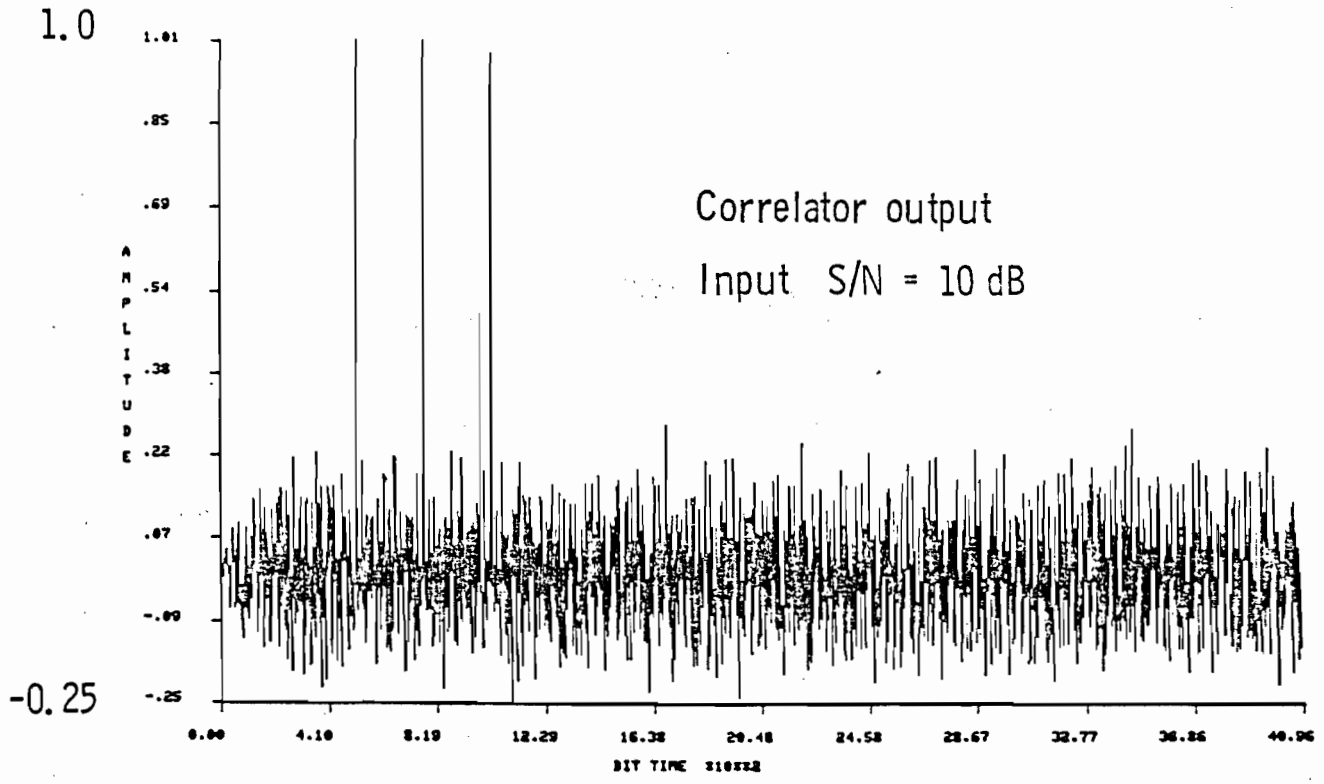
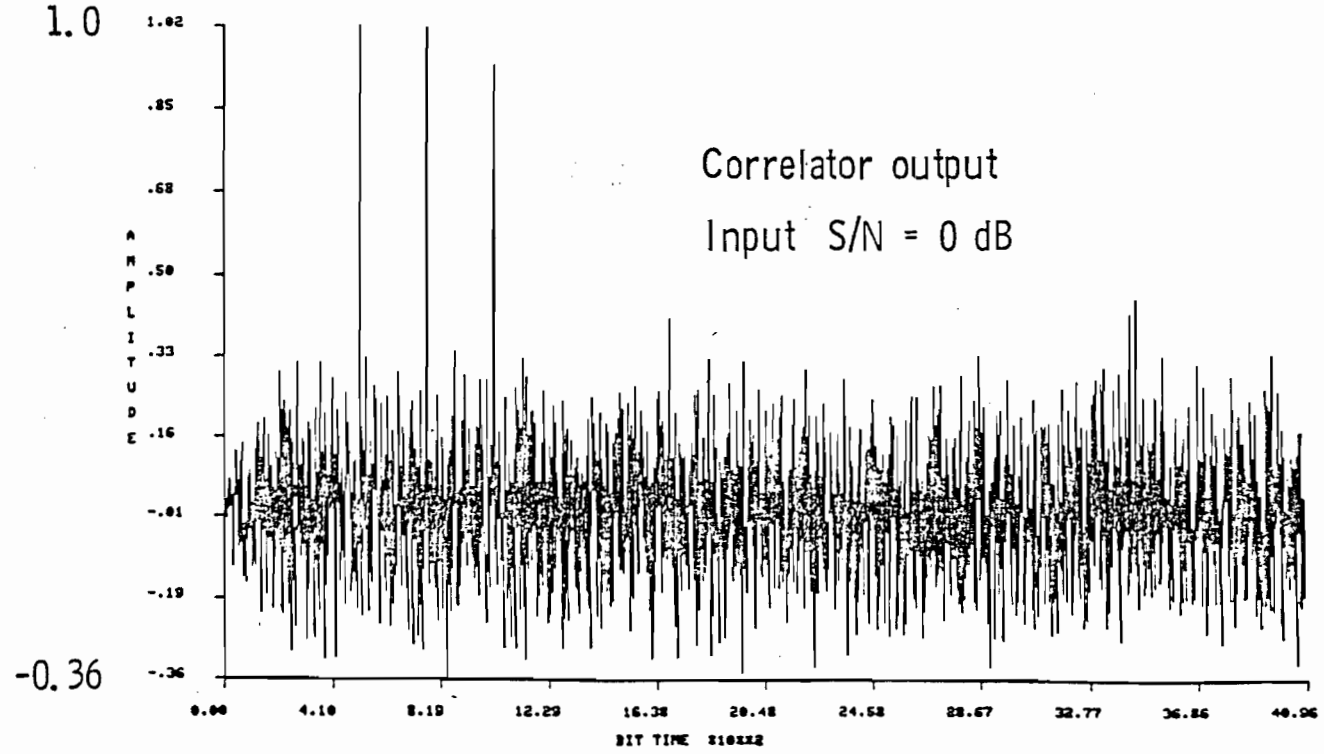


Figure 4.17 Output of Complex Correlator with Noise





5.0 Conclusions and Recommendations

Models have been developed and implemented within the ICSSM structure to simulate the major functional blocks of a generic SS system. The ICSSM modules have been documented and simulation results generated which favorably compare to theory and hardware measurements. These modules provide the capability to assess the vulnerability of a range of SS system configurations to jamming.

The modules developed and implemented here do not represent a complete set. There are several other modules needed to further enhance the capability of ICSSM for CVA. The needed modules include source and channel coders. Enhanced channel models are needed to represent the effects of multipath and dispersion. Receiver nonlinearity models are needed. One major problem in SS systems involves the performance of AGC loops. Models for different AGC loops need to be developed and implemented. The dynamics of code acquisition models must be enhanced and code tracking algorithms implemented. Additional jamming margin can be gained through the application of spatial signal processing, i.e., using an adaptive antenna. It is not efficient to just attach an adaptive antenna to a SS system [18]. The special characteristics of SS systems can be exploited to further improve the overall system performance. The capability to simulate such systems is needed.

REFERENCES

[1] J.W. Modestino et al., "Digital Simulation of Communications Systems," presented at The National Communications Conference, paper 06.2, 1979.

[2] K.S. Shanmugan and P. Balaban, "A Modified Monte-Carlo Simulation Technique for the Evaluation of Error Rates in Communications Systems," IEEE Trans. on Comm., vol. COM-28, no. 11, November 1980, pp. 1916-1924.

[3] M. Fashano, "Communications Systems Simulation," presented at The National Communications Conference, paper 06.3, 1979.

[4] B. Coetzer et al., "Digital Computer Simulation of Spread Spectrum Communications Systems," presented at Int. Conference on Communication, paper 45.4, 1981.

[5] G.C. Lin and J.F. Rowe, "A Computer Simulation of a Jammed Direct Sequence Spread Spectrum Communications Link," presented at The National Communications Conference, paper 15.6, 1979.

[6] R. Dixon, Spread Spectrum System, Wiley, New York, 1976.

[7] R. Dixon, Spread Spectrum Techniques, IEEE Press, 1976.

[8] IEEE Trans. on Communications - Special Issue on Spread Spectrum Communication, COM-25, August 1977.

[9] IEEE Trans. on Communications - Special Issue on Spread Spectrum Communication, COM-30, May 1982.

[10] J.K. Holmes, Coherent Spread Spectrum Systems, John Wiley, New York, 1982.

[11] A.B. Carlson, Communications Systems, McGraw Hill, New York, 1975.

[12] H.L. Van Trees, Detection, Estimation, and Modulation Theory, Vol. 3, John Wiley and Sons, 1971.

[13] L.B. Milstein et al., "The Effect of Multiple Tone Interfering Signals on a Direct Sequence Spread Spectrum Communication System," IEEE Trans. on Comm., vol. COM-30, no. 3, March 1982, pp. 436-445.

[14] I.S. Gradshteyn and I.M. Ryzhik, Table of Integrals, Series and Products, Academic Press, New York, 1980.

[15] H. Taub and D.C. Schilling, Principles of Communications, McGraw Hill, New York, 1971.

[16] A.V. Oppenheim and R.W. Schafer Digital Signal Processing, Prentice-Hall, Englewood Cliffs, New Jersey, 1975.

[17] K.S. Shanmugan, Digital and Analog Communications Systems, John Wiley, New York, 1979.

[18] J.M. Winters, "Spread Spectrum in a Four Phase Communication System Employing Adaptive Antennas," IEEE Trans. on Comm., vol. COM-30, no. 5, May 1982, pp. 929-936.

[19] J.C. Holtzman, et al. "Prediction of the Interference of Spread Spectrum Systems or Non-Spread Spectrum Systems", TISL TR 588 - Final Report, August 1983, Telecommunications and Information Sciences Laboratory, University of Kansas.