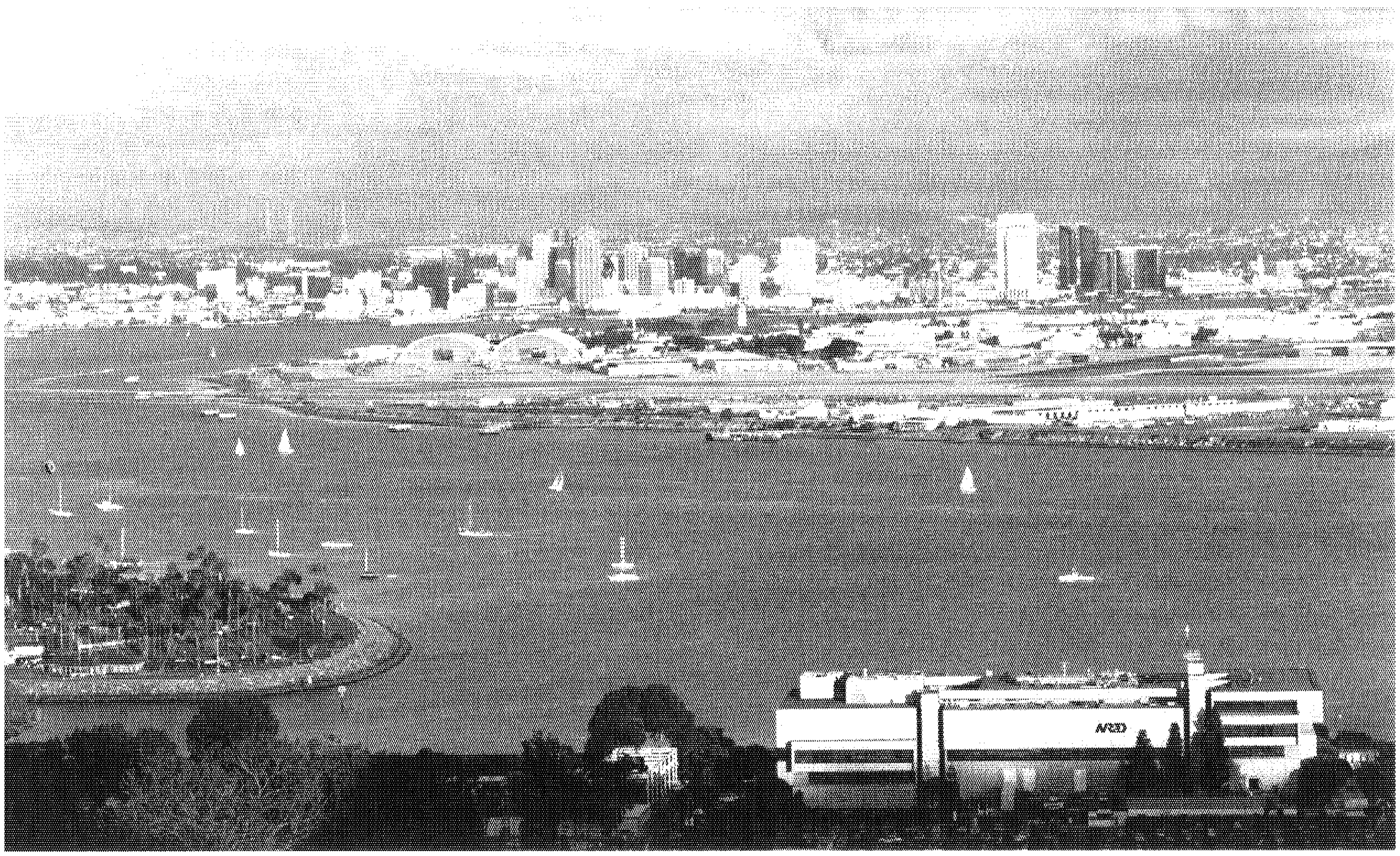


Technical Document 2739
June 1995

Simulation of Time-Varying Filters for Spread Spectrum Communication

John Custy

Naval Command, Control and
Ocean Surveillance Center
RDT&E Division

San Diego, CA
92152-5001

Technical Document 2739
June 1995

Simulation of Time-Varying Filters for Spread Spectrum Communication

John Custy

NAVAL COMMAND, CONTROL AND OCEAN SURVEILLANCE CENTER
RDT&E DIVISION
San Diego, California 92152-5001

ADMINISTRATIVE INFORMATION

The work detailed in this document was performed by the Naval Command, Control and Ocean Surveillance Center, RDT&E Division (NRaD), at NRaD's Warminster Detachment, Communication Systems Analysis Branch, Code 342. The work was funded by the ONR Technology Program at NRaD under program element 0602232N, project number RC32C13, work unit number 30–NNB101, and accession number DN081123.

Released by
L. E. Smith, Head
Communication Systems
Analysis Branch

Under authority of
P. J. Finnegan, Head
Aircraft Communication
Technology Division

RT

CONTENTS

Figures

1.0 INTRODUCTION

As described in detail in Dyckman (1995), time-varying filters (TVFs) can be used to generate a broad class of signals for spread spectrum communication. This document describes simulations which have been developed as adjuncts to the mathematical analysis of these filters. Our specific goal has been to obtain tools for the quantitative comparisons of TVF modulation against other spead spectrum techniques in terms of bit error-rate performance and low probability of intercept/antijam (LPI/AJ) performance. These simulations were developed to also aid in studying synchronization.

Several architectures for TVF modulators were identified in Dyckman (1995). The unitary transformation architecture was chosen for these simulations primarily because this architecture appears to hold promise for implementation in distributed hardware. Also, the optimal-matched filter detector in additive white noise can be implemented as a simple inverse transformation.

Two simulations have been developed: one on the Macintosh and one under Comdisco's Signal Processing Worksystem (SPW) on a Sun. The Macintosh simulation can be used to spread a variety of baseband signals with a TVF modulator and to carry out the inverse transformation using a TVF demodulator. Modulated signals can be corrupted by noise or multipath effects before demodulation. Bit error-rate statistics can be accumulated for all channel configurations. Modulated waveforms which have passed through the channel can be saved to disk in Matlab readable format to perform analysis. The simulation is object oriented, making it easy to understand and modify.

The simulations on the Sun were developed under SPW, which is a simulation environment for communication systems. These simulations can be used to modulate and demodulate signals using TVFs. Several channel models are provided with SPW. In addition, a variety of intercept receiver and jammer models have been obtained from Wright Patterson Air Force Base for use under SPW; thus, these simulations may be used for studying AJ and LPI charactersitics of TVF-generated waveforms.

2.0 TIME-VARYING FILTER SIMULATIONS ON THE MACINTOSH

2.1 OVERVIEW OF THE TIME-VARYING FILTER SIMULATION ON THE MACINTOSH

The Macintosh simulations were written with Symantec C++, version 6.0. 1, and were recently upgraded with Symantec C++, version 7.0. The simulation consists of a collection of objects, where an object (for example, a signal generator) consists of a collection of data and a group of functions that operate on that data. Communication between objects occurs in terms of messages; that is, an object can send a message to change the internal data of another object. For example, a message can be sent to a signal generator object to change the frequency of the signal it is generating. Thus, the characteristics of an object are changed only through a well-defined message structure, and a user does not have to be concerned with the structure of the data within an object, or in the way in which an object's response to a message is implemented.

The object-oriented nature of the simulation provides important benefits. Since data are tightly coupled with the functions that operate on them, modifications to the simulation can be carried out with limited risk of unexpected side effects. Objects allow implementation details to be hidden from the user. Also, the simulation is easy to understand from both a user's standpoint and a programmer's standpoint, because of its structural similarity to a real-world communication system.

As shown in figure 1, the system being simulated consists of a baseband signal generator, a time-varying filter modulator, a channel, a time-varying filter demodulator, and a detector. There is also a provision for measuring bit error rates by comparing the baseband signal against the received demodulated signal. The baseband signal generator, detector, and comparator are all part of a single object because of the need for these items to share data as bit error-rate (BER) measurements are made. In addition to the objects shown in the figure, window objects are also available for displaying text and graphics. Since all waveforms in the simulation take on complex values, a complex data

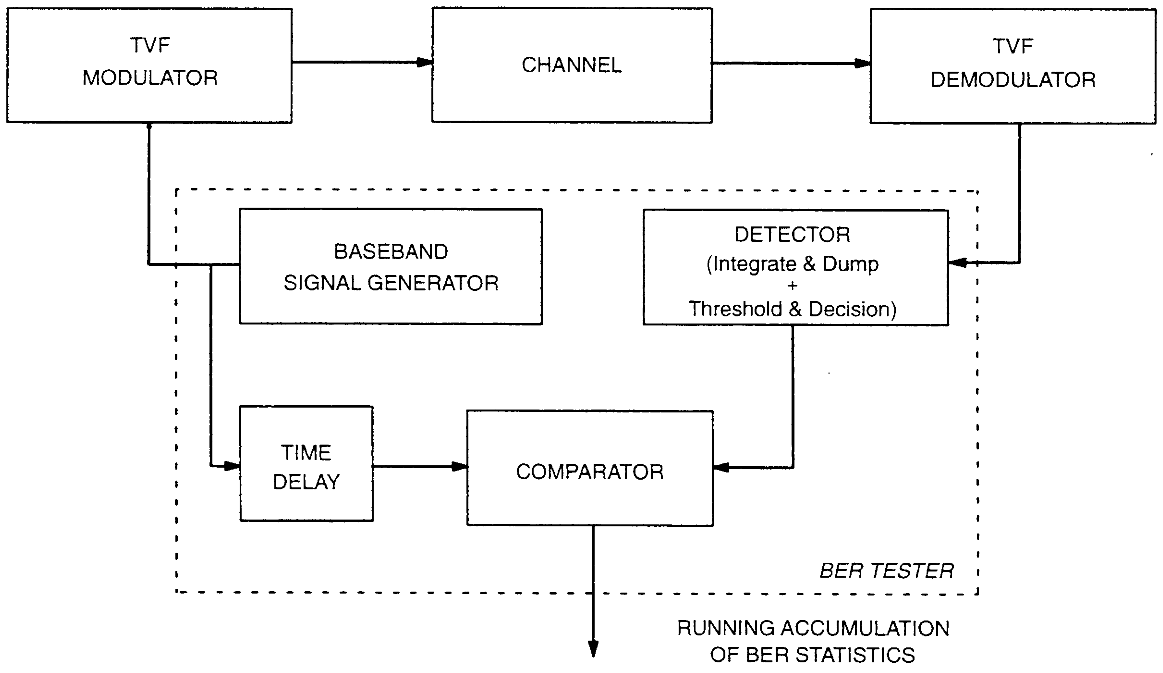

TVF MODULATOR
CHANNEL
TVF DEMODULATOR
BASEBAND SIGNAL GENERATOR
DETECTOR (Integrate & Dump + Threshold & Decision)
TIME DELAY
COMPARATOR
BER TESTER
RUNNING ACCUMULATION OF BER STATISTICS

Figure 1. Structure of the TVF simulation. The TVF modulator, channel, TVF demodulator, and BER tests are objects in the simulation. The TVF modulator and demodulator are composed of layer and transform objects.

type and functions for operating on it were developed. (The complex data structure and library provided with Symantec C++ 6.0. 1 had some bugs associated with it.) The signals coming out of the modulator, passing through the channel, and going into the demodulator are baseband complex envelope representations of RF-modulated signals (Jeruchim et al., 1992, p. 36).

The baseband generator generates complex valued waveforms that can take the form of a sinusoid square wave, or pseudonoise sequence. The user can change the type of signal being generated and its amplitude as the program is running. Bit error rates are measured by comparing the baseband pseudonoise sequence against the output of an integrator that acts as a detector.

The output of the baseband generator is fed into a time-varying filter modulator that spreads the signal. The structure of the modulator used in the simulation is shown in figure 2. It consists of two layers, each of length 100, and of 50 unitary transform objects. As the simulation stands now, the coefficients of the transformations yield pseudorandom unitary transformations. In general, these coefficients can be set to any value by modifying the appropriate code.

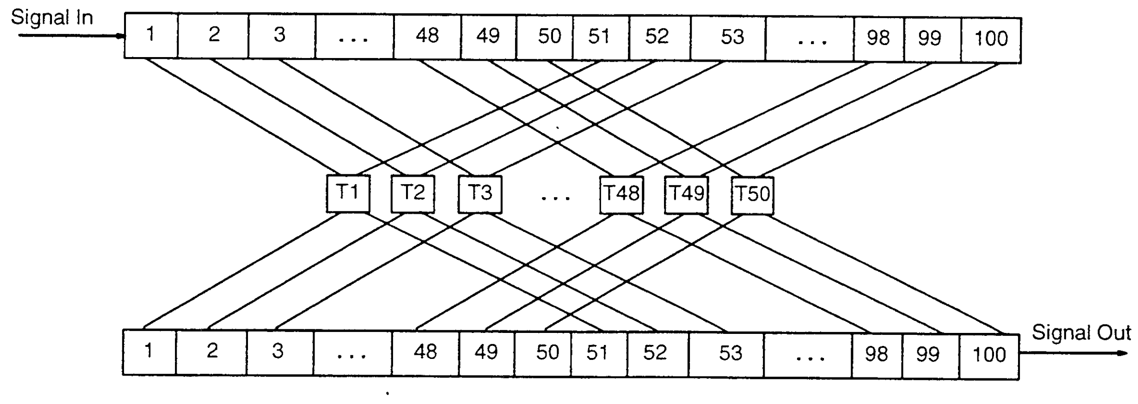


Figure 2. Unitary transformation architecture used in the simulation for the modulator and demodulator. The 50 unitary transformations, marked T1 through T50, act on a block of 100 samples.

The filter operates by shifting the input signal into the top layer, one sample at a time. After 100 new samples have been shifted in, they are fed through the transformations to form samples in the output layer. The resulting samples in the output layer are shifted out, one sample at a time; simultaneously, new samples arrive at the input layer. Thus, the transforms can be thought of as “firing” every time a complete set of new samples has been shifted into the input layer. The processing gain is directly proportional to the amount of oversampling of the input signal.

Both the modulator and demodulator have the same architecture (as shown in figure 2); the only difference lies in the coefficients of the transformations. The transformations of the demodulator are simply the matrix inverses of the modulator transformations. Dyckman (1995) explains rigorously the sense in which this architecture is time varying, spectrum spreading, and time spreading.

The signal generated by the TVF modulator is fed into a channel object, which is based on a shift register. It introduces a time delay corresponding to transit time through the channel and can also introduce Gaussian noise and multipath effects.

The output of the channel is fed into a time-varying filter demodulator. This demodulator carries out the inverse transformation performed by the modulator. Thus, the demodulator has the same architecture as the modulator (shown in figure 2), but the unitary transformations of the demodulator are the inverse transformations of the modulator. The modulator and demodulator are synchronized by using knowledge of the exact time delay introduced by the channel. (The synchronization scheme was kept as simple as possible at this point in anticipation of more practical schemes to be developed.)

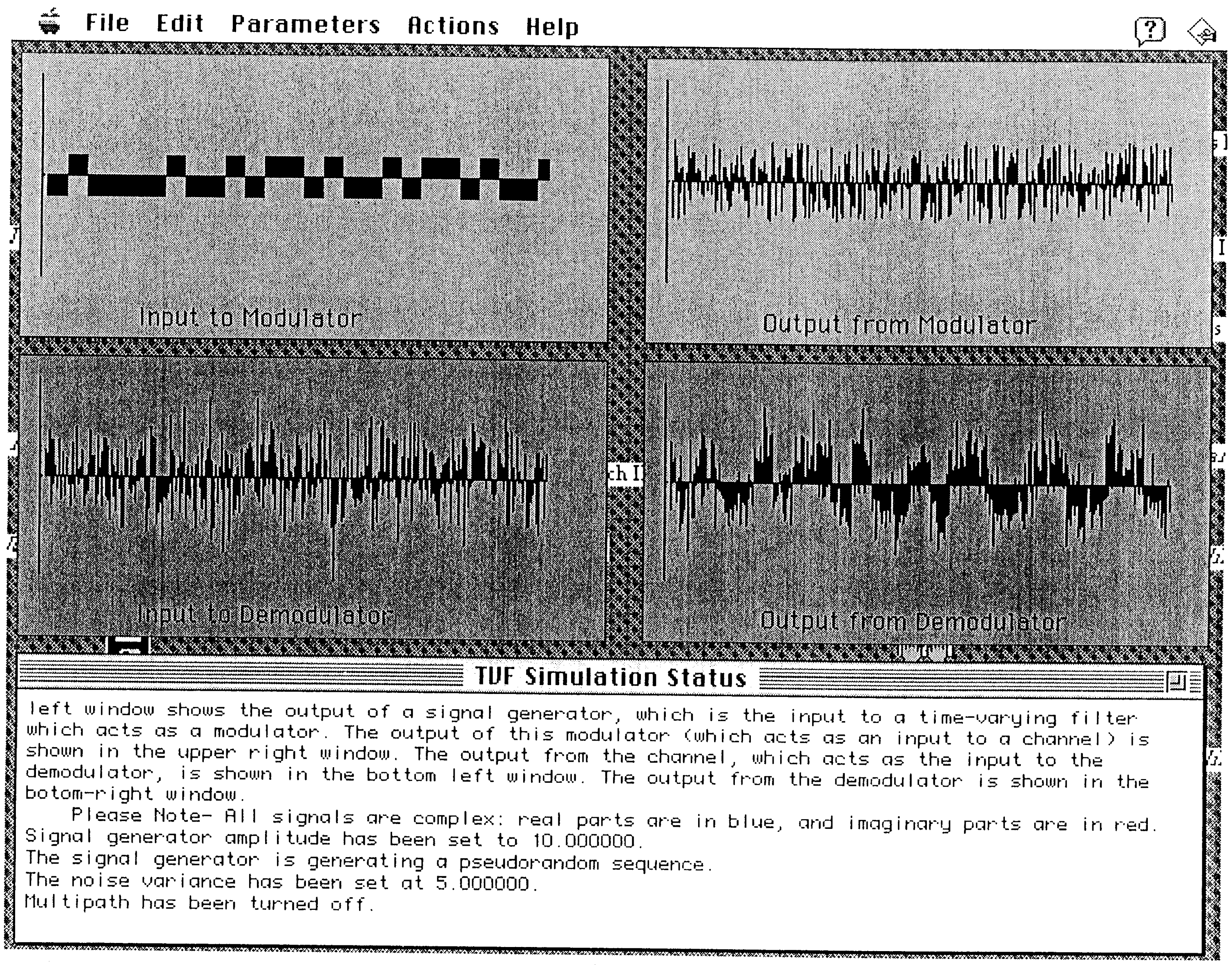
The output of the demodulator is a re-creation of the baseband signal, except for the effects of noise and multipath introduced by the channel. If the baseband generator is generating a pseudonoise sequence, then data are collected for bit error-rate measurements; this is done by performing an integrate and dump operation on the demodulated waveform. A decision about which bit was transmitted (+1 or –1) is made by comparing the output of the integrate and dump against a fixed threshold of value zero. This result is compared against the true bit sent; a running count is kept of the total number of bits transmitted, number of bits in error, and number of bits received correctly.

2.2 HOW TO USE THE SIMULATION

The simulation runs on the Macintosh-II family of computers under System 7. Upon double-clicking on the TVF icon, the screen shown in figure 3 appears. The default configuration of the simulation is as follows:

Baseband signal type:	Pseudorandom sequence
Baseband signal amplitude:	10 units
Channel noise variance:	0 units
Multipath in channel:	None

Four graphics windows and one text window are associated with the simulation. The graphics window in the upper-left corner shows the output from the baseband generator. This is the waveform being fed into the TVF modulator. The output from the TVF modulator, shown in the upper right window, is fed into the channel; the output from the channel is shown in the lower left window. This waveform is different from the output of the modulator due to the time delay through the channel, the effects of noise, and the effects of multipath. This waveform is fed into the TVF demodulator, whose output is shown in the lower right window. This waveform would be the same as the output of the baseband generator, except for a time delay and for the effects of channel noise and multipath. The bottom-most window is a text window that provides feedback to the user whenever a simulation parameter (that is, signal type, signal amplitude, noise, and/or multipath status) is changed.



```
left window shows the output of a signal generator, which is the input to a time-varying filter
which acts as a modulator. The output of this modulator (which acts as an input to a channel) is
shown in the upper right window. The output from the channel, which acts as the input to the
demodulator, is shown in the bottom left window. The output from the demodulator is shown in the
botom-right window.
    Please Note- All signals are complex: real parts are in blue, and imaginary parts are in red.
Signal generator amplitude has been set to 10.000000.
The signal generator is generating a pseudorandom sequence.
The noise variance has been set at 5.000000.
Multipath has been turned off.
```

Figure 3. Screen dump showing the time-varying filter simulation windows. All waveforms scroll from left to right. The upper-left window shows the baseband input to the time-varying filter modulator. The baseband waveform is a pseudonoise sequence generated by a 29-stage shift-register generator. The output from the TVF modulator is shown in the upper-right window. This waveform is fed into the channel, which can introduce noise or multipath; here, only noise is added. The output from the channel is shown at the lower left. This signal is fed into the demodulator, which carries out the inverse TVF transformation. The output from the demodulator is shown in the lower-right window.

Several menus are available to the user that allow the simulation to be modified as it is running. The available menus are the Apple menu, the File menu, the Edit menu (not shown), the Parameters menu, the Actions menu, and the Help menu. Author and version information is available under the Apple () menu (figure 4). Desk Accessories are not available while the TVF simulation is running.

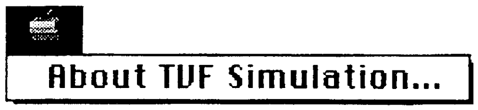


Figure 4. The Apple menu for the TVF simulation.

The File menu (figure 5) provides the Save Data As... command and the Quit command. The Save Data As... command allows the last 1024 data points coming out of the channel to be saved to disk as a Matlab M-file. This is a data file that can easily be read into Matlab for analysis, as is later described.

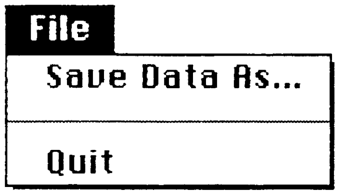


Figure 5. The File menu for the TVF simulation.

The Edit menu is not used in this simulation; it is included in an inactive state for future compatibility with desk accessories, and also to conform to Macintosh user-interface guidelines.

The Parameters menu (figure 6) has a single choice associated with it: Change Simulation Parameters.... This menu choice brings up a dialog box (figure 7) that allows the user to change the type and amplitude of the baseband waveform, the noise level in the channel, and to indicate whether or not multipath is present.

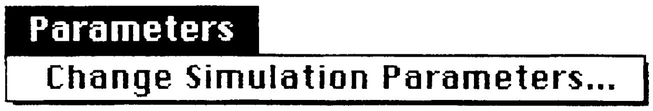
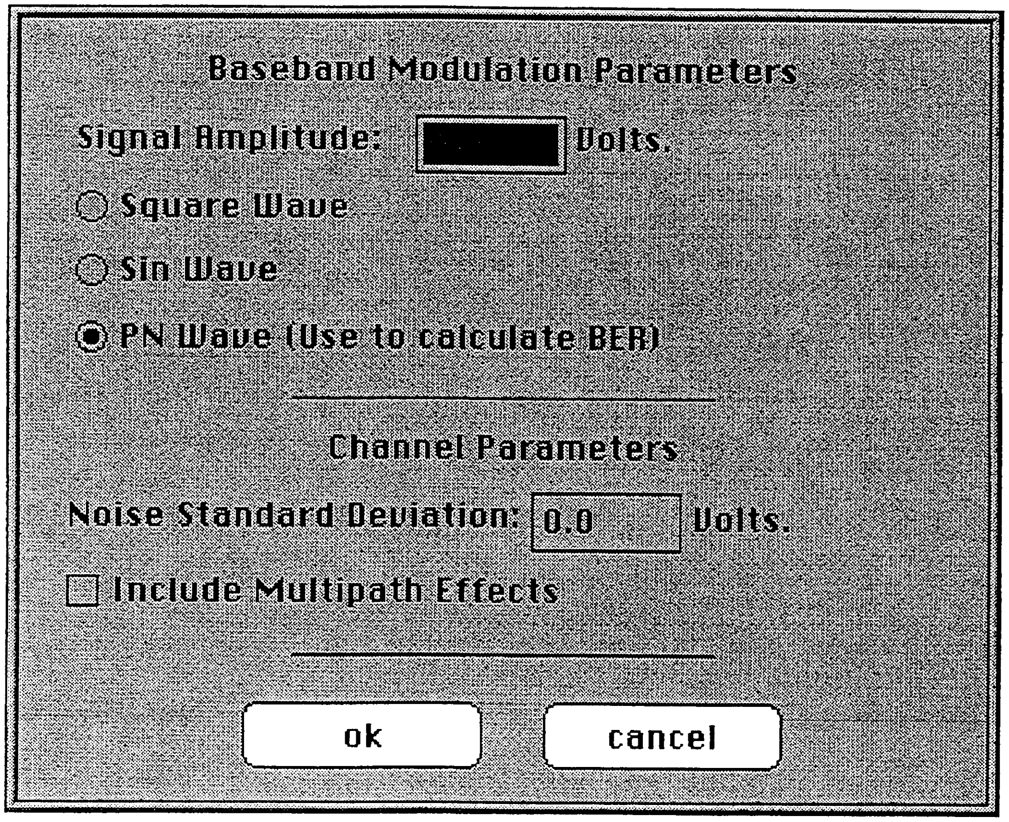


Figure 6. The Parameters menu for the TVF simulation.



Baseband Modulation Parameters

Signal Amplitude: Volts.

○ Square Wave

○ Sin Wave

● PN Wave (Use to calculate BER)

Channel Parameters

Noise Standard Deviation: 0.0 Volts.

☐ Include Multipath Effects

Figure 7. The Parameters dialog box.

The Actions... menu, shown in figure 8, allows the user to perform a number of actions on the simulation.

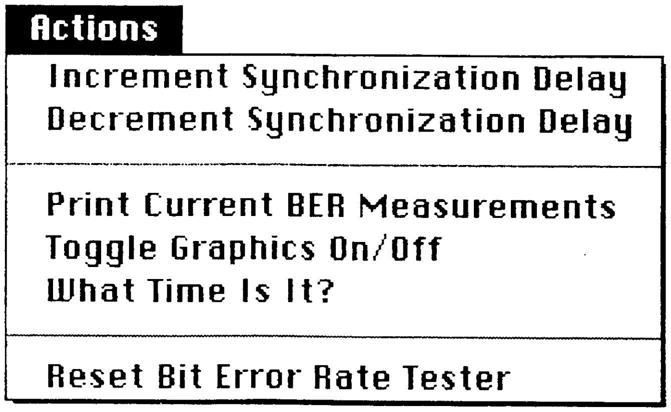

Actions
Increment Synchronization Delay
Decrement Synchronization Delay
Print Current BER Measurements
Toggle Graphics On/Off
What Time Is It?
Reset Bit Error Rate Tester

Figure 8. The Actions menu for the TVF simulation.

The Increment Synchronization Delay and the Decrement Synchronization Delay items change by one unit the time delay between the modulator firing and the demodulator firing. By default, this value is set to the sum of the time delay through the channel, the time delay through the modulator, and the time delay through the demodulator, which results in proper synchronization. The total synchronization delay is printed out whenever these items are changed.

The Print Current BER Measurement prints out all of the bit error-rate statistics kept by the system; i.e., the number of bits received correctly, the number of bits received incorrectly, and the total number of bits sent.

The Toggle Graphics On/Off item stops the four graphics windows from being updated. This is useful for speeding up BER measurements, because the simulation spends most of its time drawing to the screen.

The What Time Is It? item causes the total number of elapsed-time increments to be printed. Note that this is greater than the number of data bits transmitted by a factor equal to the processing gain (i.e., the rate at which the input baseband signal is oversampled).

The Reset Bit Error-Rate Tester should be used after changing signal amplitude, noise amplitude, or multipath status. This menu choice resets to zero all accumulated BER data associated with the bit error-rate tester.

The Help menu (figure 9) provides information about the TVF concept in general, how to use the simulation, and how to modify the simulation. The information presented by these items is stored in TEXT resources, thus facilitating updates when the program is modified.

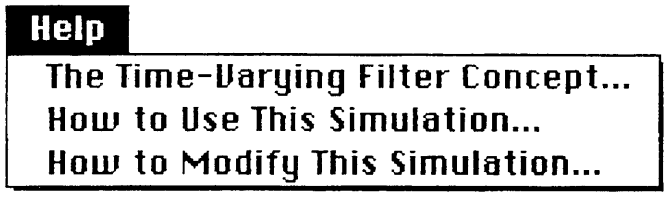

Help
The Time-Varying Filter Concept...
How to Use This Simulation...
How to Modify This Simulation...

Figure 9. The Help menu for the TVF simulation.

2.3 ANALYSIS OF TVF DATA IN MATLAB

One of the most useful aspects of the Macintosh simulation is that data coming out of the channel can be saved to disk as a text file for analysis in Matlab. To perform an analysis, configure the TVF simulation program as desired in terms of baseband waveform, signal and noise level, and multipath status. Next, allow the program to run until at least 1024 samples have been generated (this can be checked by using the What time is it? command under the Actions menu). Then, simply use the Save as... command under the File menu to save the data points to a file. Saving the file somewhere in the Matlab search path is most convenient. By typing the file name at the Matlab prompt, the one-dimensional complex vector of length 1024 is read in and named TVF. For example, for a file saved in the Matlab search path with the default name TVF_Data.m, the following commands read in the data and perform an FFT on it:

```
»TVF_Data                          %Read in data
»fft_tvf=fft(TVF);                 %Take FFT of data vector
»plot(abs(fft_tvf(1:1000)),'black'); %Plot the magnitude of the FFT
»whitebg;                          %Display with white background
»title('Frequency Spectrum of TVF Waveform')
»ylabel('Amplitude')
»xlabel('Frequency')
```

The resulting frequency spectrum is shown in figure 10.

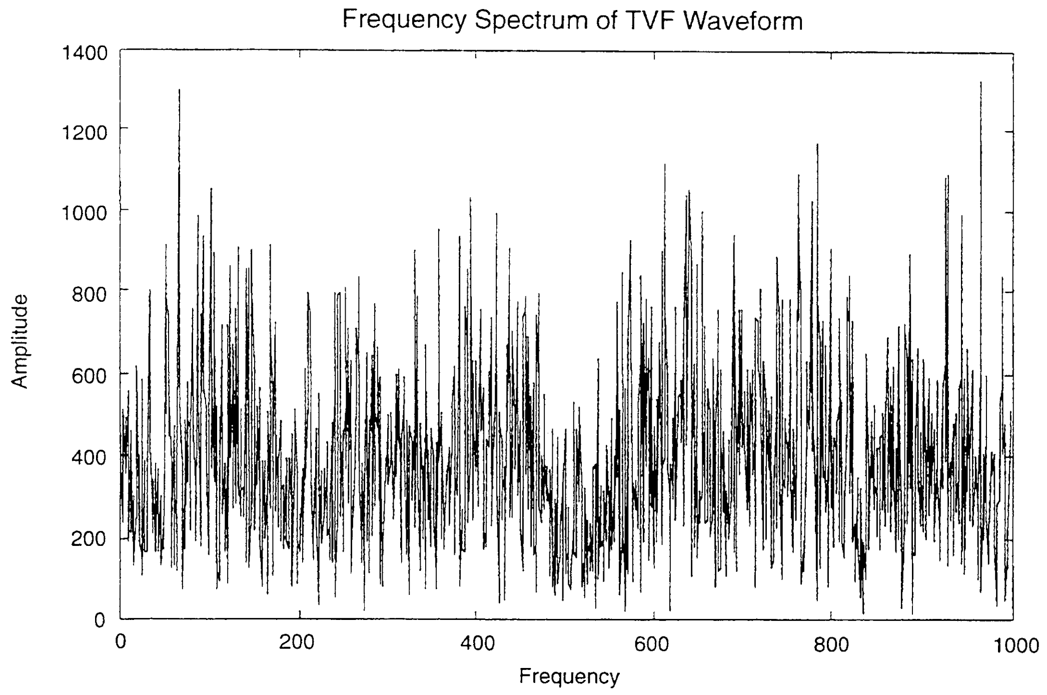


Figure 10. Frequency spectrum of data going into the TVF demodulator. The input waveform was a pseudonoise sequence, and processing gain was set to 10. There was no channel noise, and multipath was off. The modulator was that shown in figure 2.

2.4 DESCRIPTION OF THE MAIN-EVENT LOOP AND OBJECTS IN THE SIMULATION

This section describes each object used in the simulation at a level that enables the main-event loop to be understood and modified to suit a particular purpose. The descriptions here are not meant to be comprehensive, but rather to simply indicate how the simulation works, and also to provide some guidance for changing the main-event loop.

As shown in figure 11, the main-event loop indicates the structure of the simulation. Each cycle through the loop corresponds to one increment of simulated time. The user's interaction with the simulation is carried out through the operating system; i.e., the operating system is the intermediary between user events, such as mouse-down events and menu choices, and the behavior of the simulation. For example, if the user presses the mouse button as the simulation is running, the operating system stores information about this event, where it occurred (e.g., in the menu bar, in a window, etc.), for use by the program. (Examples of other events include mouse-up events and disk-insertion events.)

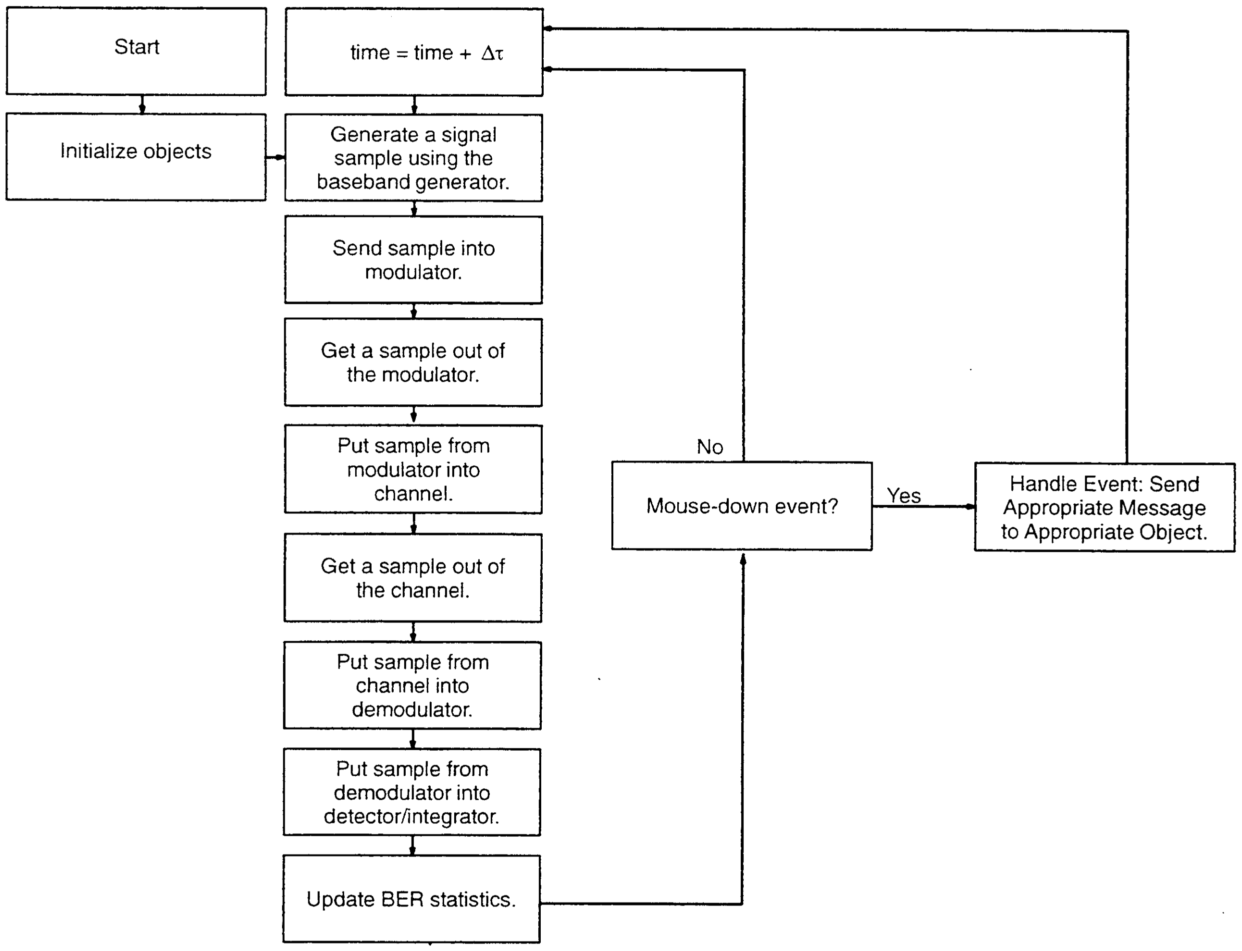

Start
Initialize objects
time = time + Δτ
Generate a signal sample using the baseband generator.
Send sample into modulator.
Get a sample out of the modulator.
Put sample from modulator into channel.
Get a sample out of the channel.
Put sample from channel into demodulator.
Put sample from demodulator into detector/integrator.
Update BER statistics.
Mouse-down event?
No
Yes
Handle Event: Send Appropriate Message to Appropriate Object.

Figure 11. Main-event loop in TVF simulation. Each cycle through the loop represents one increment of simulated time.

Figures 12—17 and 19—20 were cut from the object browser in the Symantec compiler. These figures show each function associated with each object.

Ways in which the simulation could be modified include (but of course are not limited to) the following:

- Changing the architecture of the TVF modulator or demodulator. This could include changing the number of layers or transforms, or changing the coefficients of the transforms.
- Changing the characteristics of the channel; for example, changing noise to some non-Gaussian type, or changing the multipath configuration.
- Incorporation of a more realistic synchronization scheme.

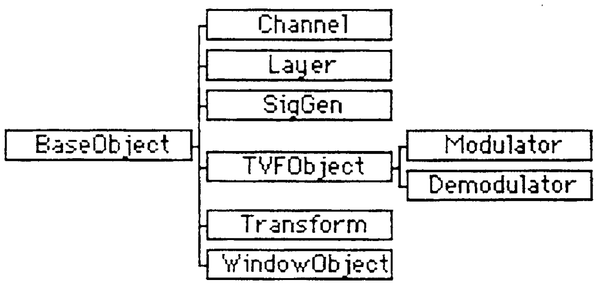
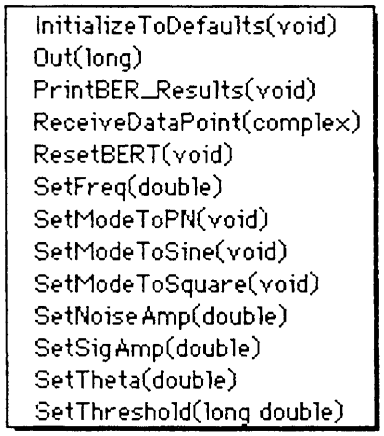


Figure 12. Objects and their relationships in the TVF simulation. All objects in the simulation are derived from (i.e., are special cases of) the BaseObject. The modulator and demodulator are derived from the TVFObject.

2.4.1 Baseband Signal Generator/BER Tester Object

As shown in figure 1, the Baseband Signal Generator/BER Tester object consists of the baseband signal generator, a detector, and code for comparing the transmitted baseband signal against the received baseband signal and for keeping a running count of the bit error-rate statistics.

The baseband signal generator generates waveforms as a function of time; that is, sending the message Out (t to the signal generator returns the signal value at that value of t. The sine wave is generated with a standard sine function, and the square wave is generated by applying a threshold to the sine wave. The PN sequence is generated with a 29-stage shift-register generator; the generating polynomial is $\chi^{29} + \chi^2 + 1$ (i.e., the taps are 29 and 2). This configuration was chosen because it provides a long sequence ($2^{29} - 1 = 536{,}870{,}911$ chips before repeating), but only requires two feedback taps, making implementation somewhat simpler than other choices.



```
InitializeToDefaults(void)
Out(long)
PrintBER_Results(void)
ReceiveDataPoint(complex)
ResetBERT(void)
SetFreq(double)
SetModeToPN(void)
SetModeToSine(void)
SetModeToSquare(void)
SetNoiseAmp(double)
SetSigAmp(double)
SetTheta(double)
SetThreshold(long double)
```

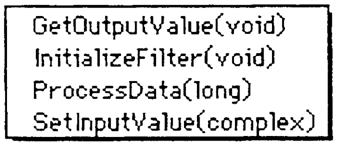
Figure 13. Messages that can be sent to a Signal Generator/BER Tester object. These messages allow the characteristics of the signal generator to be changed (signal amplitude, type, etc.); they also allow a sample to be obtained from the signal generator, or a sample of the demodulated signal to be received for BER measurement. The function SetNoiseAmp is used to introduce noise into the baseband signal prior to the channel and is not used in this simulation.

Detection of the received signals consists of feeding the demodulated signal into an integrate and dump, and comparing the output of the integrator to a threshold. As mentioned previously, the transmitted data consists of +1 and –1 values; thus, the threshold applied to the output of the integrate and dump is set to zero.

2.4.2 Modulator and Demodulator Objects

The time-varying filters that make up the modulator and demodulator are both constructed from layer objects and transform objects. The modulator and demodulator are identical except for the setting of the transformation coefficients in the transforms connecting the two layers.

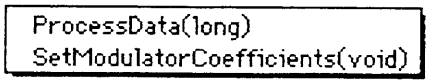
Because of the similarity between the modulator and demodulator, both of these objects are special cases of—that is, are descendents of—an object called a TVFObject. Thus, certain functions that are the same for the modulator and demodulator can be implemented as TVFObject functions. For example, the messages GetOutputValue and SetInputValue can be sent to either a modulator or a demodulator. The SetInputValue message simply shifts the samples in the input layer one register to the right and then places the input sample in the left-most register. Likewise, the GetOutputValue message returns an output sample obtained by shifting each sample in the output layer one register to the right. (See figure 14.)



```
GetOutputValue(void)
InitializeFilter(void)
ProcessData(long)
SetInputValue(complex)
```

Figure 14. Messages that can be sent to a TVFObject.

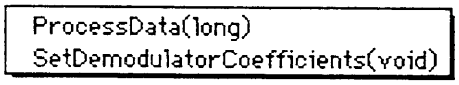
The message SetModulatorCoefficients sets the modulator coefficients to specific values; i.e., the coefficients are part of the code of the SetModulatorCoefficients membership function.



```
ProcessData(long)
SetModulatorCoefficients(void)
```

Figure 15. Messages that can be sent to a modulator object.

In a way similar to the modulator, the SetDemodulatorCoefficients message sets the demodulator coefficients to specific values. The ProcessData function simply multiplies data from the input layer through the transforms and places them in the output layer. This message is implemented separately in the modulator and demodulator, but since the action is the same in both cases, it could have been implemented as a function in the TVFObject.

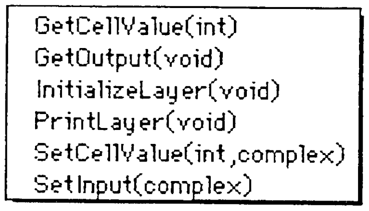


```
ProcessData(long)
SetDemodulatorCoefficients(void)
```

Figure 16. Messages that can be sent to a demodulator object.

2.4.3 Layer Object

A layer object is a shift register with provisions for shifting complex valued data in from the left (for an input layer) and for shifting stored data out from the right (for an output layer).

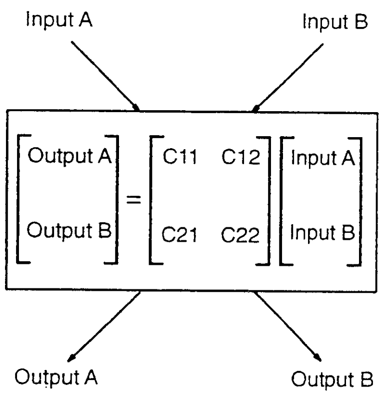


```
GetCellValue(int)
GetOutput(void)
InitializeLayer(void)
PrintLayer(void)
SetCellValue(int,complex)
SetInput(complex)
```

Figure 17. Messages that can be sent to a layer object. A shift operation occurs automatically when either SetInput or GetOutput are called. Any cell can be set directly with the SetCellValue message.

2.4.4 Transform Object

A transform object is a 2-by-2 matrix with complex coefficients. It transforms a two-dimensional complex vector into another two-dimensional complex vector (figure 18).

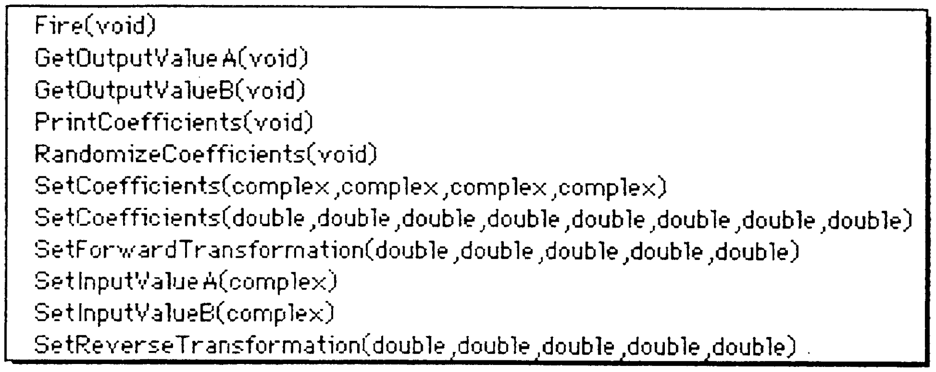


$$\begin{bmatrix} \text{Output A} \\ \text{Output B} \end{bmatrix} = \begin{bmatrix} C11 & C12 \\ C21 & C22 \end{bmatrix} \begin{bmatrix} \text{Input A} \\ \text{Input B} \end{bmatrix}$$

Figure 18. Structure of a unitary transformation object. Coefficients are set according to the constraints given in Appendix A of Dyckman (1995).

As shown in figure 2, both the TVF modulator and demodulator incorporate transform objects in their structure. The coefficients of each of these transforms are constrained so that the transformation is unitary (see Dyckman, 1995, Appendix A, for a more detailed description of unitary transformations).

Having the modulator generate its coefficients automatically would have been convenient. However, the coefficients of the demodulator are dependent on those of the modulator. Thus, if the modulator were to in some way set its coefficients automatically, it would have to communicate this information to the demodulator. In the interest of keeping the modulator and the demodulator as totally separate entities, we chose a different approach. The SetForwardTransformation function, which is called with five arguments, is used to set the coefficients for the modulator transformation. The function SetReverseTransformation is called with the same arguments as the SetForwardTransformation; however, it sets the inverse transformation for the demodulator. This allows the arguments from the demodulator to be simply copied and pasted from the modulator. Thus, the modulator and demodulator communicate only by way of the data transmitted through the channel.

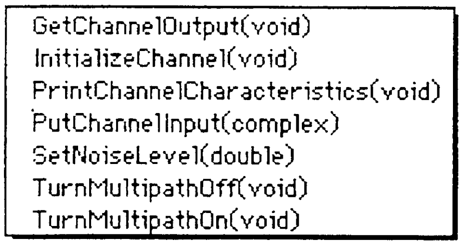


```
Fire(void)
GetOutputValueA(void)
GetOutputValueB(void)
PrintCoefficients(void)
RandomizeCoefficients(void)
SetCoefficients(complex,complex,complex,complex)
SetCoefficients(double,double,double,double,double,double,double,double)
SetForwardTransformation(double,double,double,double,double)
SetInputValueA(complex)
SetInputValueB(complex)
SetReverseTransformation(double,double,double,double,double)
```

Figure 19. Messages that can be sent to a transform object. These messages essentially consist of ways of setting the coefficients of the transformation, putting data into it, carrying out the matrix multiplication, and then the getting data out.

2.4.5 Channel Object

As previously mentioned, the channel is essentially a shift register through which waveform samples pass. Noise can be added to a sample as it leaves the shift register, and multipath can be simulated by forming an output that consists of the sum of two shift-register elements. Messages that can be sent to the channel are shown in figure 20.



```
GetChannelOutput(void)
InitializeChannel(void)
PrintChannelCharacteristics(void)
PutChannelInput(complex)
SetNoiseLevel(double)
TurnMultipathOff(void)
TurnMultipathOn(void)
```

Figure 20. Messages that can be sent to a channel object. These messages provide ways of putting data into the channel, getting data out of the channel, setting the noise level, and turning multipath on and off.

Though the Macintosh Toolbox provides a built-in function for generating uniformly distributed random numbers, no routines exist for generating random variables with a Gaussian distribution. Thus, Gaussian noise in the channel is generated by adding together several uniformly distributed variables in accord with the Central Limit Theorem. Relevant calculations and considerations are summarized as follows (Hogg & Craig, 1978, p. 192):

We have as definitions

$$\sigma^2 \equiv E\left\{(X-\mu)^2\right\} = \mathrm{E}\left\{X^2 - 2X\mu + \mu^2\right\} = \mathrm{E}\left\{X^2\right\} - \mu^2, \qquad E\{u(X)\} \equiv \int_{-\infty}^{\infty} u(x)\cdot f(x)\, dx,$$

where $f(x)$ is the probability density function of the random variable χ, and $\mu = \int_{-\infty}^{\infty} x \cdot f(x)dx$. Say we have a random variable uniformly distributed between 0 and 1; then

$$\sigma^2 = E\{X^2\} - \mu^2 = \int_0^1 x^2 f(x)dx - \left(\frac{1}{2}\right)^2 = \int_0^1 x^2 dx - \left(\frac{1}{4}\right) = \frac{1}{3}x^3\Big]_0^1 - \left(\frac{1}{4}\right) = \frac{1}{3} - \frac{1}{4} = \frac{1}{12}$$

The central limit theorem says that if we let x_1, x_2 x11 denote the items of a random sample from a distribution that has mean and positive variance 2, then the random variable

$$Y_n = \left(\sum_i^n X_i - n\mu\right) \Big/ \sigma\sqrt{n}$$

has a limiting distribution that is normal, with mean zero and variance 1. Thus, if we take 12 random variables uniformly distributed between 0 and 1, we can get a random variable that approximates a Gaussian random variable with mean 0 and $\sigma^2 = 1$ by forming

$$Y_n = \left(\sum_i^{12} X_i - 12\cdot\left(\tfrac{1}{2}\right)\right) \Big/ \left(\sqrt{\tfrac{1}{12}}\right)\sqrt{12} = \sum_i^{12} X_i - 6$$

That is, we simply add together the 12 random variables and subtract 6 to get an approximate Gaussian random variable with $\mu = 0$ and $\sigma = 1$.

Say we have a random variable uniformly distributed between –1 and 1. (Such a random variable can be easily generated with the Macintosh toolbox call `Random ()`, which returns a pseudorandom integer uniformly distributed between –32767 and 32767.) Calculation shows that for such a random variable, the variance is

$$\sigma^2 = E\{X^2\} - \mu^2 = \int_{-1}^1 x^2 f(x)dx - 0 = \frac{1}{2}\int_{-1}^1 x^2 dx = \frac{1}{2}\left[\frac{1}{3}x^3\right]_{-1}^1 = \frac{1}{2}\left[\frac{1}{3} + \frac{1}{3}\right] = \frac{1}{3}$$

Using the above formula for Yn, we see that if we add together 12 of these and divide by

$$\sigma\sqrt{n} = \sqrt{\frac{1}{3}}\cdot\sqrt{12} = \sqrt{4} = 2 \ ,$$

we get a Gaussian distribution with mean 0 and 2 = 1. This approach is a little better than the previous one because it saves the computational trouble of taking the absolute value of the result returned by the call Random (

To verify that the channel noise is indeed Gaussian, a histogram (figure 21) of the values of the channel noise can be generated in Matlab. Comparison of this histogram against histograms of Gaussian noise generated in Matlab indicates that the TVF noise is close to Gaussian. The Matlab program used to generate the plot is shown below.

```
»TVF_Data
»hist(real(TVF),25)
»title('Histogram of Simulated Noise')
»ylabel('Number of Samples in Interval')
»xlabel('Value of Real Component')
```

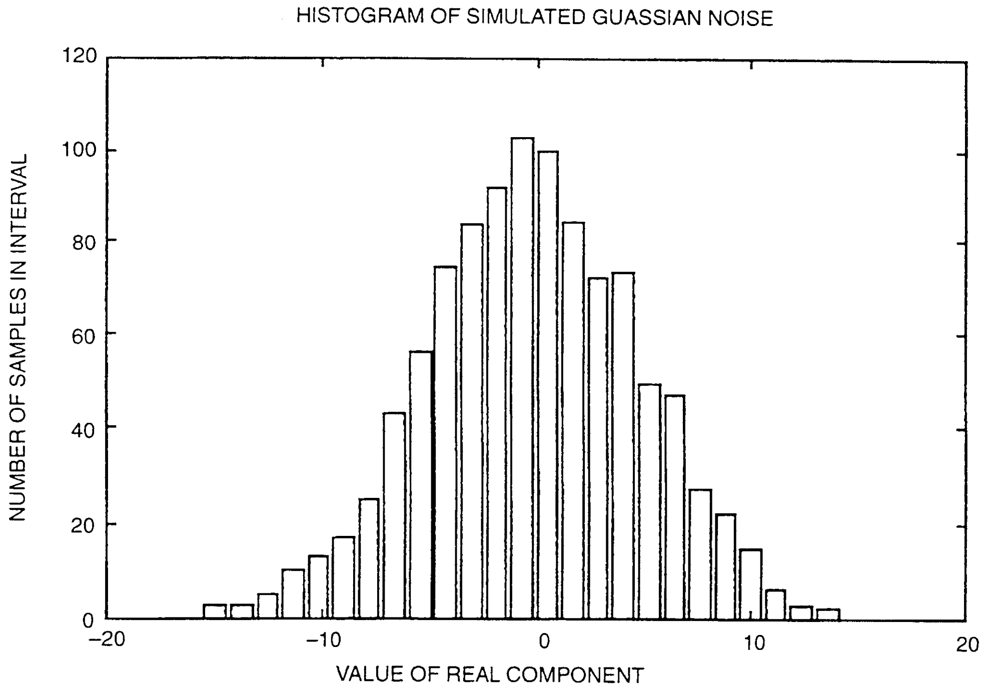


Figure 21. Histogram of real component of simulated Gaussian noise. Data were generated by setting the signal amplitude to zero and the noise standard deviation to 5. Record length was 1024 samples. Histograms of imaginary components are similar in appearance. This histogram and others generated by the same method are reasonably similar to those representing Matlab-generated Gaussian noise.

Multipath effects are generated by combining the last element of the shift register with some other element of the shift register. Both samples can be scaled by a complex constant. Though multipath effects can be turned on and off as the simulation is running, changing the time delay or the complex scale factors requires modification of the GetChannelOutput function.

2.4.6 Window Object

The Window object (figure 22) allows the graphical display of a waveform (data distributed in time). As such, it could be thought of as an oscilloscope object. Scrolling takes place automatically when a data point is sent to a window; scrolling is effected by the toolbox call ScrollRect (The text window is provided by the ANSI library in the Symantec compiler; in that documentation, it is referred to as the console window.

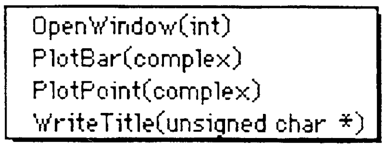

```
OpenWindow(int)
PlotBar(complex)
PlotPoint(complex)
WriteTitle(unsigned char *)
```

Figure 22. Messages that can be sent to a window object. These messages provide ways of opening a window and of sending data to it.

2.5 AREAS FOR IMPROVEMENT OF MACINTOSH SIMULATIONS

Because the Macintosh simulations were written with the highest priority placed on achieving correct results, some efficiency has been sacrificed to make the logical flow of the program as simple and as clear as possible. The most notable example is that the layer object and the channel object are each implemented as a one-dimensional array. When a value has to be shifted into or out of either of these objects, all entries in the array must be shifted. A much more efficient technique would involve using a circular buffer, in which new values simply overwrite old ones; and all that need be updated is the index of the starting point. As indicated previously, other areas for improvement include development of more sophisticated TVF architectures, a more sophisticated channel, and incorporation of a more sophisticated synchronization scheme.

Only one bug is known to exist in the Macintosh TVF simulation: the program incorrectly overwrites data files. Specifically, if data are to be saved to a file with the same name as a data file that already exists, the tail end of the old file becomes a part of the new file. To work around this problem, give all new files names that are distinct from existing files.

3.0 SIMULATIONS OF TIME-VARYING FILTERS UNDER SPW ON THE SUN

As the software on the Macintosh was nearing completion, Comdisco's Signal Processing Work-system (SPW) became available to our group. SPW is a simulation environment for digital signal processing and communication systems. It provides a large collection of blocks from which full systems can be built hierarchically. No code has to be written; instead, systems are built up in terms of graphical block diagrams. Since blocks can be constructed in a hierarchical manner, special purpose blocks (such as a TVF) can be constructed and tested with standard channel models.

Several TVF-based systems of various complexity have been developed within the SPW environment. Figure 23 shows a simple but complete communication system based on TVFs.

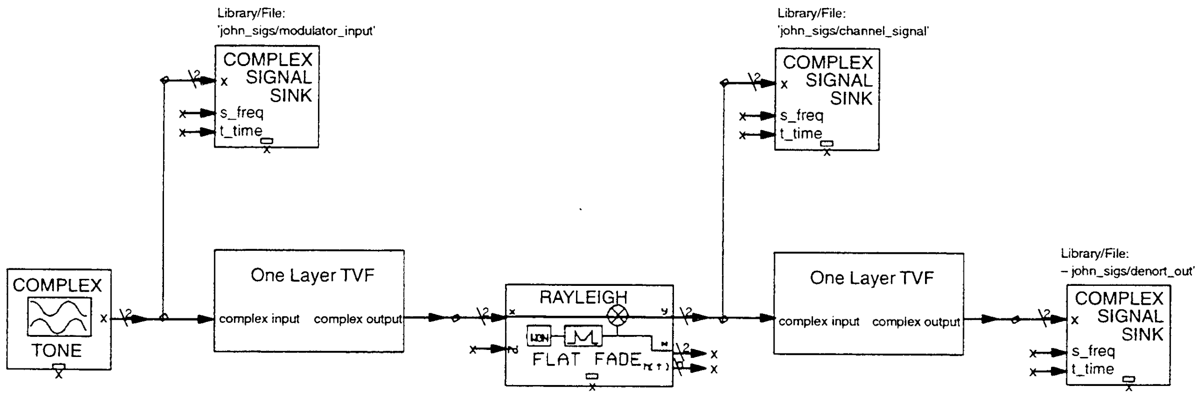


Figure 23. Communication system based on time-varying filters in SPW. The input to the system is from a file (shown at left), and waveforms at various points in the system are saved to files for later analysis. The coefficients for the forward and inverse transformations were generated by the program shown in figure 24 and then entered by hand. The channel model is a Rayleigh model.

The TVF shown in figure 23 operates by collecting a block of 128-input samples and then breaking this block into 4 subblocks. Each of these subblocks of 32 samples is sent through a separate array of unitary transformations with an effective structure like that shown in figure 2. The outputs of the unitary transformations are then concatenated into an output stream for transmission. The demodulator after the channel has the same structure as the forward transformation, except that the coefficients are set for the inverse transformation. The structure of the TVF blocks in the simulation (figure 23) should enable us to construct multilayer TVFs simply by stringing several blocks together; however, we invariably get "out-of-memory" type errors when this is attempted. We have worked with Comdisco's technical support to track down the exact cause of this problem.

The coefficients for the TVFs shown in figure 23 are hard-coded into the block. The coefficients for both the forward and inverse transformations were generated by the program shown in figure 24.

```
generateUnitaryXform:=Module[
  {alpha,beta,gamma,delta,theta,c,c11,c12,c21,c22,d},
  theta=Random[Real,{0, N[2Pi]}];
  alpha=Random[Real,{0,1}];
  beta=Random[Real,{0,1}];
  gamma=Random[Real,{0,1}];
  delta=N[2Pi] - alpha + beta + gamma;
  c11= Exp[I alpha]Cos[theta];
  c12=-Exp[I beta ]Sin[theta];
  c21= Exp[I gamma]Sin[theta];
  c22= Exp[I delta]Cos[theta];
  Print["Coefficients for forward unitary transform are:"];
  Print[" ",c11,"          ",c12];
  Print[" ",c21,"          ",c22];
  Print[""];
  c={{c11,c12},{c21,c22}};
  d=Inverse[c];
  Print["Coefficients for the inverse transform are:"];
  Print[" ",d[[1,1]],"          ",d[[1,2]]];
  Print[" ",d[[2,1]],"          ",d[[2,2]]];
  Print[" "];
]
```

Figure 24. A *Mathematical* program for generating the coefficients of random unitary transformations and their inverses. The inverse of a unitary transformation is its complex-conjugate transpose, as can be verified by the outputs of this program. This program was used to generate the coefficients used in the simulation of figure 23.

A more sophisticated TVF modulator is shown in figure 25. The input signal is read in at the upper left of the figure. Blocks of the input signal are duplicated and are fed into the four modules in the center. The four components in the center of the figure each represent an array of unitary transformations. Some of the coefficients for the transformations are read in from a file of random numbers; the circuitry for this is shown at the lower left. The rest of the coefficients are generated internally, so that the resulting transformations are unitary. Outputs from the four modules are concatenated to form the output of the modulator.

One advantage of this architecture is that the coefficients of the TVFs can be changed as the simulation is running. Another advantage is that the processing gain can be extended arbitrarily by simply placing more “modules” in the system. The only practical limitation is “out-of-memory” type errors. Time spreading is fixed by the architecture of the individual modules.

One of the most important benefits of the SPW environment for this project is the availability of a wide variety of channel models and other blocks, some of which are included with SPW, and some of which have been developed by outside parties. These will be valuable tools in testing TVF spread spectrum waveforms. The following is a list of channel models provided with the SPW system; each description is a shortened version of the description given in the on-line documentation.

- *Am-Am/Am-Pm* This block implements an input-amplitude-to—output—amplitude (AM to AM) and an input-amplitude-to-output-phase nonlinearity.
- *Indoor (Rappaport)* This block implements a statistical model for an indoor radio multipath channel.
- *Indoor (Saleh)* This block implements a statistical model for an indoor radio multipath channel.

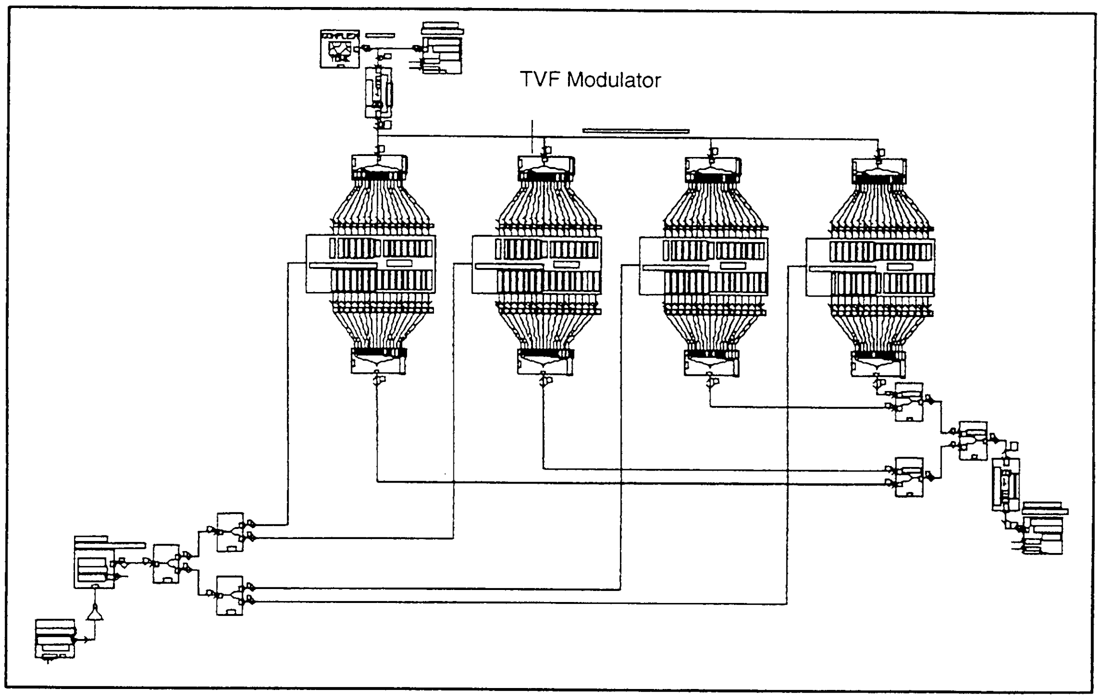


Figure 25. A time-varying filter modulator developed in SPW. The input waveform is read in from a file; the blocks associated with this are near the top, to the left of center. The random coefficients are read in from a file at the lower left of the diagram. The output waveform is written to a file at the lower right. Each of the four modules in the center of the figure takes a block of the input waveform and passes it through an array of unitary transformations. The structure of this modulator allows a demodulator to be constructed using the same coefficient file, with only minor modifications to the architecture.

- *Multipath (Rummler)* This block implements Rummler's model of a multipath signal.
- *Rayleigh Flat* This block implements a flat (or single-ray) Rayleigh fading-channel model.
- *Rayleigh Selective* This block implements a selective (or two-ray) Rayleigh fading-channel model. Rayleigh fading-models are typically used in mobile communication systems to model the effects of multiple-point scatterers in the neighborhood of a moving receiver.
- *Tabular Non-Linear* This block implements a mapping from input to output according to a data file.
- *Timing Drift* This block simulates asynchronous transmitter and receiver clocks by resampling the input signal to produce an output signal that slowly drifts from the input signal. A 'jitter" input can also change the sampling time, allowing you to simulate sample jitter. Both real and complex versions of this channel are available.
- *Traveling-Wave Tube Amp* This block implements the AM-to-AM and AM-to-PM characteristics for a typical Traveling-Wave Tube (TWT) amplifier.

In addition to the modules provided with the SPW system, we have obtained a variety of modules developed by the University of Kansas for Rome Labs, which can be used to assess system vulnerability in terms of AJ and LPI performance. The following is a list of jammers, radiometers, and signal generators obtained from Rome Labs; each description is a shortened version of the description given in the "SPW Communications Vulnerability (CVA)."

- *Noise -Loaded AM Jammer* This is a jamming tone AM modulated with noise.
- *Noise-Loaded FM Jammer* This is a jamming tone FM modulated with noise.
- *Random-Tone Jammer* This module generates up to three tones which are randomly distributed in the specified bandwidth. At the start of each hop interval, new tones with random frequencies are generated. This module can be used for partial band jamming.
- *Sweep Jammer* This block generates a jammer tone that sweeps over the desired bandwidth.
- *Tone Jammer* This block implements a CW tone jammer. It generates up to three CW tones with frequencies specified by the user.
- *Channelized Radiometer* Measures the signal power in each channel (frequency band) by appropriately integrating the power spectral density over the frequencies.
- *Wideband Total-Power Radiometer* Measures the signal power in the entire RF spectrum by appropriately integrating the power spectral density.
- *Scanning Radiometer* This module sweeps the reception bandwidth in steps and measures the energy intercepted at each dwell position in the sweep. Various detector modules, including the BMWD and the OR-BMWD, are designed to work in conjunction with the intercept receiver modules.
- *Modules to Generate the JTIDS Waveform*
- *Modules to Generate the SINCGARS Waveform*
- *Generic Frequency Hopper* The generation of the hop frequencies can be governed by a PN sequence generator, a uniform random number generator, an arbitrary PMF random number generator, or a user-designed code-generator module.
- *Generic Time Hopper* The generation of the time slots, where the signal is hopped, can be governed by a PN sequence generator, a uniform random-number generator, an arbitrary PMF random-number generator, or a user-designed code-generator module.
- *Direct Sequence Spreader* This module generates a PN sequence waveform that should be multiplied with the source signal for implementing direct sequence spreading.

Various controllers are available to drive the generic time hopper and the generic frequency hopper.

4.0 CONCLUSIONS AND PLANS FOR FUTURE WORK

Because of the different natures of the SPW simulations and the Macintosh simulations, each can present advantages and disadvantages for a given simulation task. One of the fundamental differences between the two approaches is that the various signals within SPW (i.e., input signals and output signals) are represented as files, whereas the Macintosh simulation operates on each point of a waveform as it is generated, similar to the way real equipment operates in real time. Another difference is that simulations that employ multirate signals (i.e., signals with different sampling rates) are handled more conveniently in SPW. Also, for a given number of input waveform sample points, SPW executes more quickly. However, some level of expertise is required to use SPW effectively; this difficulty is compounded by the fact that we tend to have "out-of-memory" type errors with large simulations on our Sun LX. Also, the user appears somewhat restricted in analysis and data display options. Taken together, the SPW simulations and the Macintosh simulation provide a valuable adjunct to mathematical analysis in the study of TVFs for spread spectrum modulation.

A specific area in which we expect to use these simulations is the study of synchronization. As explained in Dyckman (1 995), for a general TVF spread spectrum system, the TVF modulator and demodulator do not commute with a frequency shift. For this reason, in general, a two-dimensional synchronization search appears to be necessary. These simulations will be useful for evaluating candidate synchronization schemes.

Another potential project involves verifying that the processing gain determined by simulation matches what we calculate analytically. We can easily calculate the processing gain for a particular TVF modulator. To verify these figures through simulation, we could transmit a nonspread (i.e., baseband) waveform over the channel and measure bit error rate vs. E_b/N_o for a range of white-noise powers N_o. We will then repeat the same measurements using a waveform spread with a TVF-based modulator/demodulator. By comparing the BER vs. E_b/N_o curves for these two cases, we will be able to experimentally verify the value of processing gain for a particular filter.

Another area of interest involves using the TVF waveform as a featureless waveform for LPI communication. A BPSK waveform spread with a PN sequence will show "features" corresponding to the chip rate, when passed through a delay and multiply receiver. The simulations presented here could be useful, for example, in studying how a delay and multiply receiver responds to a TVF-modulated waveform.

5.0 REFERENCES

Dyckman, H. L. 1995. “Spread Spectrum Modulation by Means of Time-Varying Linear Filtering.” NRaD TD 2733 (March). Naval Command, Control and Ocean Surveillance Center, RDT&E Division (NRaD), San Diego, CA.

Hogg, R. V., and A. T. Craig. 1978. *Introduction to Mathematical Statistics*. Macmillan Publishing Co., Inc., New York, NY.

Jeruchim, M. C., P. Balaban, and S. K. Shanmugan. 1992. *Simulation of Communication Systems*. Plenum Press, New York, NY.

6.0 BIBLIOGRAPHY

Dixon, R. C. 1976. *Spread Spectrum Systems.* Wiley, New York, NY.

“Signal Processing Worksystem (SPW) Manual Set.” Comdisco, Inc.

“SPW Communications Vulnerability Assessment (CVA) Modules.” 1993. Final Technical Report. Contract #F30602-93-C-0109, 15 April 1993. Prepared for: Rome Laboratories, RL/C3BB, Griffiss AFB, New York 13441. Prepared by: CECASE Rapid Prototyping Lab, University of Kansas, 2291 Irving Hill Drive, Lawrence, Kansas 66045.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE June 1995	3. REPORT TYPE AND DATES COVERED Final: FY 93 – FY 94

4. TITLE AND SUBTITLE SIMULATION OF TIME-VARYING FILTERS FOR SPREAD SPECTRUM COMMUNICATION	5. FUNDING NUMBERS PE: 0602232N PR: RC32C13 WU: 30–NNB101 ACCESS NO.: DN081123
6. AUTHOR(S) John Custy	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) NRaD Warminster Det. Naval Command, Control and Ocean Surveillance Center (NCCOSC) RDT&E Division, Block Programs 53560 Hull Street San Diego, CA 92152–5001	8. PERFORMING ORGANIZATION REPORT NUMBER TD 2739
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) ONR Technology Program NCCOSC RDT&E Division San Diego, CA 92152–5001	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This report describes two computer simulations which have been developed to aid in the study of time-varying filters for spread spectrum communication (Dyckman, 1995). One simulation, written to run on a Macintosh computer, allows bit-error-rate measurements to be carried out, and also allows the TVF-waveform data to be saved for analysis in Matlab. The other simulation, developed udner Comdisco's Signal Processing Worksystem, allows a variety of jammers and intercept receivers to be applied to the TVF-generated waveform.

14. SUBJECT TERMS time-varying filters spread spectrum communication computer simulation	15. NUMBER OF PAGES 33
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT SAME AS REPORT

UNCLASSIFIED

21a. NAME OF RESPONSIBLE INDIVIDUAL	21b. TELEPHONE *(include Area Code)*	21c. OFFICE SYMBOL
John Custy	(619) 553–2599	Code 342

INITIAL DISTRIBUTION

Code 0012	Patent Counsel	(1)
Code 0271	Archive/Stock	(6)
Code 0274	Library	(2)
Code 30	W. J. Schoppe	(1)
Code 34	P. J. Finnegan	(1)
Code 342	J. Custy	(15)
Code 804	R. D. Peterson	(1)

Defense Technical Information Center Fort Belvoir, VA 22060–6218	(2)
NCCOSC Washington Liaison Office Washington, DC 20363–5100	
Center for Naval Analyses Alexandria, VA 22302–0268	
Navy Acquisition, Research & Development Information Center (NARDIC) Arlington, VA 22244–5114	
GIDEP Operations Center Corona, VA 91718–8000	
Office of Naval Research Arlington, VA 22217–5660	(2)
Naval Air Warfare Center Warminster, PA 18974–0591	(2)