Fall 1-31-1998

# A simple neural agent communicating through sets

James P. Stanski
*New Jersey Institute of Technology*

# Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

## ABSTRACT

### A SIMPLE NEURAL AGENT COMMUNICATING THROUGH SETS

by
James P. Stanski

Networked agents of the simplest kind offer the power of cooperative problem solving through parallel operation along with tight packaging potential. Such agents are self-contained analog machines capable of only a few primitive intelligent operations. In this thesis, a design will be developed for a simple agent capable of sending, receiving, and processing information in a environment where agents are coupled together. This environment imposes unorchestrated simultaneous input while expecting a useful timely response. Successful collaboration in these conditions is accomplished through sets encoded within pulse ensembles. The simplicity of the set definition is an inviting candidate for message communication and processing. Although its use is restricted to spatial pattern recognition, a predicted side effect of set communication in a multilayer network configuration is the ability to reintroduce the output back into the input layers for further processing.

# A SIMPLE NEURAL AGENT COMMUNICATING THROUGH SETS

by
James P. Stanski

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Science

Department of Computer and Information Science

January 1998

# APPROVAL PAGE

## A SIMPLE NEURAL AGENT COMMUNICATING THROUGH SETS

**James P. Stanski**

Dr. Franz Kurfess, Thesis Advisor — Date
Associate Professor of Computer and Information Science, NJIT

Dr. Qianhong Liu, Committee Member — Date
Asistant Professor of Computer and Information Science, NJIT

Dr. Ajaz Rana, Committee Member — Date
Assistant Professor of Computer and Information Science, NJIT

## BIOGRAPHICAL SKETCH

**Author:** James P. Stanski

**Degree:** Master of Science

**Date:** January 1998

**Undergraduate and Graduate Education:**

- Master of Science in Computer Science,
  New Jersey Institute of Technology, Newark, NJ, 1998

- Bachelor of Science in Electrical Engineering
  New Jersey Institute of Technology, Newark, NJ, 1990

**Major:** Computer Science

This thesis is dedicated to my Mother and Father

## ACKNOWLEDGMENT

I would like to express my deepest appreciation to Dr. Franz Kurfess for his wisdom, guidance, and support as my research supervisor. His confidence, patience, and encouragement enabled me to persevere during the times when no simple solution was readily available. Special thanks to committe member Dr. Qianhong Liu for her genuine interest and leadership. Committee member Dr. Ajaz Rana has been an inspirational model of academic excellence since the beginning of my graduate studies.

I appreciate the help from Dr. Robert Allen at Bellcore for his resources and insight during my final year.

## TABLE OF CONTENTS

## TABLE OF CONTENTS
## (Continued)

**Chapter** **Page**

## TABLE OF CONTENTS
## (Continued)

**Chapter** **Page**

## TABLE OF CONTENTS
## (Continued)

## TABLE OF CONTENTS
(Continued)

**Chapter** **Page**

## LIST OF TABLES

**Table** **Page**

## LIST OF FIGURES

## LIST OF FIGURES
## (Continued)

**Figure** **Page**

## LIST OF FIGURES
## (Continued)

**Figure** **Page**

# CHAPTER 1

# INTRODUCTION

## 1.1 Objective

The objective of this thesis is to present the issues involved in considering the design for a simple analog agent. The simplicity of an agent is determined by its footprint in anticipation of computer chip packaging. The footprint will be minimized by scaling size according to the complexity of the information being processed instead of the number of parallel inputs. Other efforts to minimize agent size are to employ special materials instead of discrete components as well as choosing a simple means of communication between agents.

Once considering the environment an agent will be placed in, the focus will be placed on communicating using sets. This design will be subdivided into three parts; receiver, processor, and transmitter of which the first two will be analyzed. Although the exact physical design will not be presented, an abstract layout and description of the materials needed will be established. Once considering the design described in this thesis, other designs are possible following the guidelines presented.

## 1.2 Background Information

In order to realize the parallel processing benefits of neural networks, efforts are made to place these networks on silicon chips. Development efforts can be divided into several categories such as analog designs, digital designs, hybrid designs, artifical neuron approximation, and biological neuron approximation. Biological designs such as the

hippocampus [7] require several microcomponents due to a digital design and a two dimensional processing section. Another effort is to understand and duplicate biological neurons [2,18]. This includes simulating molecular functions of a biological neuron with silicon counterparts according to the physics of field effect transistors [3]. The results are a higher utilization of silicon area due to creative design. The design presented in this thesis is a combination of implementing an artificial neuron using the same creative physical designs.

The design includes using a material to propagate pulses slowly similar to a previous design of an artificial Cochlea [12], but is inspired from the novel processing possibilities provided from slow diffusion ion channels found in biological neurons [5]. Slow diffusion is useful in sequential processing [6] and will also be shown useful in communicating with sets.

Research exists in asychronous communication between neural assemblies [13] and will be extended to the level of individual agents.

### 1.3 Deviations from Previous Models

The agent described here contradicts behaviour witnessed in biological neurons. There are no synaptic weights in the design. All inputs are treated equally and the usefullness of each signal is determined by how well it correlates with other signals. It has been found that biological neurons have the natural ability to interpret ensemble synchrony codes [8], however, the current agent design allows for just the opposite in detecting asynchronous pattern codes.

# CHAPTER 2

# AGENT ENVIRONMENT

This chapter will describe how the agent will be used in order to provide a better understanding when designing the agent components. The term agent is used in the place of the term neuron because neurons are usually described in terms of a mathematical equation while an agent is described in terms of goals. The goal of the agent presented here is to recognize and present input patterns. The agent has the same anatomy of a neuron containing several inputs and one output, but uses an advanced communication medium to enhance its processing power.

## 2.1 Environment Description

The environment of the agent will be similar to that of multilayer feedforward neural networks [14]. Chapter 5 describes how the communication medium chosen allows for the last stage and hidden stages to feed backwards to previous stages, forming loops. These loops do not induce oscillation, but allow one agent to process at several levels of comprehension when the network is recognizing a complex pattern. Regenerative agents are efficient for utilizing the full potential of an agent as well as correlating events patterns across levels of comprehension.

When operating within the network, agents will be presented with several simultaneous inputs. As discussed in Chapter 3, the option the agent faces is to either expect these inputs to be synchronized and process these as ensemble synchrony codes, or to accept and understand these inputs inputs as unsynchronized.

The environment will also demand an appropriate response within a reasonable time. If the input pattern is changing, then the detection rate must be able to sample the input at a high enough rate to provide sufficient results. The requirement is local processing.

Each agent is self contained and has no access to global information. The learning algorithms must be local.

Since patterns will be recognized at each level in the hierarchy and presented to the next for further processing, these patterns must be stable throughout the learning process.

# CHAPTER 3

# AGENT COMMUNICATION MEDIUM

This chapter applies the ideas presented in Chapter 2 to concrete solutions after considering what communication attributes are desired and what is possible to achive working with these attributes. These attributes include what type of information is to be passed, what it represents, and under what conditions it will be sent. Lower level concerns will also be addressed such as vocabulary and message size as well as physical interconnections and intended network architecture. The results of these considerations will converge on the suggestion to communicate by sending sets of numbers where each number represents another set of numbers or multiple sets sharing several members. The discussion of processing the information is continued in Chapter 5 where limits of this medium will be presented.

## 3.1 Expected Input/Output Conditions

The first condition imposed on a design of a communication medium is the physical interconnection between agents. Inspiration comes from biological nerual networks with the serial unidirectional connections between neurons. Information is either sent with a single pulse or through a pulse ensemble. This serial connection is superior over a parallel connection in the savings of interconnecting only one wire instead of several. The speed of communication suffers in a serial connection, but this may not be a factor in slow moving environments while serial communication offers the ability to vary the size of the message being sent. Pulse ensembles are favored over a single pulse since the only

information that can be represented in a single pulse is its presence and its relative position. Pulse ensembles, however, can be used to carry much more information within the relative positions of the pulses. The pulse ensemble offers the advantage of having its pulses relative to others within the same burst. When receiving a single pulse, the time of the pulse's arrival must be relative to something else external, such as other incoming pulses. The remainder of this chapter will concentrate on communicating through pulse ensembles.

Another feature of biological networks is the mandatory broadcasting of messages to all other neurons physically linked to the output axon. There is no routing of messages along the axon. The design in this thesis will copy the broadcast architecture for its simplicity. Therefore, as in biological neural networks, an agent can receive from several agents, and broadcast its output to all agents connected to its output. All connected agents receive the exact same message.

Since pulse ensembles can be treated relative to themselves, they need not synchronize precisely with other ensembles. It is best to avoid synchronizing problems since one agent may have to synchronize with many other agents at the same time. Occasionally, it will be impossible to do this unless all of the agents are synchronized together. This is not an impossibility but implies complete interconnection among agents. If one agent must synchronize with two others, then the two must be synchronized with each other. This would assume some form of synchronizing information being passed between the two. This should not be confused with synchronizing processes in an operating system where one process waits for another to finish. Figure 2.1 represents this

synchronization problem where A1 must synchronize with A2 and A3, but there is no guarantee A2 and A3 are synchronized with each other.

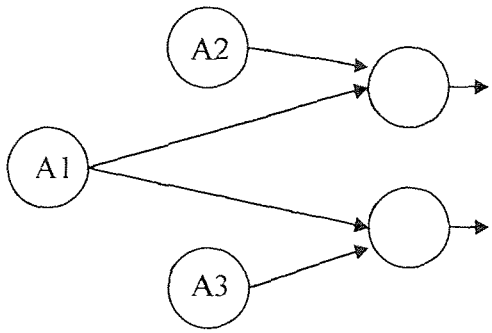



**Figure 3.1** Asynchronous Communication Dilemma

To solve the problem, A2 and A3 must synchronize with A1. However, A2 may have other commitments and may not be as flexible. To avoid this problem, there will be no ensemble synchronizing at a micro scale. At a macro scale, the ensemble must fit within a specified window for its information to be considered relevant to the current problem being solved.

Just as asynchronous communication relaxes constraints on the transmission end of communication, it can also impede the design of the receiver. Multiple synchronous signals are simpler to receive since multiple signals are designed to work together rather than interfere with the other. A research decision was made to investigate asynchronous communication at the cost of solving the problems that follow.

The main problem with asynchronous communication is the simultaneous reception of several signals which are not considerate of the other. An example of this problem is to consider the problems of listening in a crowded room or listening to the echo from

multiple speakers of a PA system. Either way, the infomation received is either completely different or exactly the same but out of phase. The agent receiving such messages must be able to listen to all messages at the same time. It may be a simpler alternative to have an agent choose which message to listen to and reject the others as humans try to do, but then there would be no guarantee that a message sent is a message received. What would be required then is a confirmation message. The problems that would arise are twofold. First, this requires bidirectional communication between agents, second, it would require the ability to direct individual messages to targets. Messages will also have to include the identification of the target agent to receive the message. Without this mechanism, the sender would have to broadcast again to everybody even if one receiver missed the message. Asynchronous, simultaneous communication is not difficult, if the messages sent can all be combined into one large message and interpreted as such. When all messages are lumped together, the uniqueness of the message and sender is lost. Gains can be made in this dilemma by haveing each sender use its own signature.

Consider messages sent encoded with a unique signature. Like fingerprints, there is no guarantee each is unique, but likely. A discriminating receiver can recognize different message signatures and be able to separate messages encoded by comparing signatures. A less complicated use for signatures is to prevent any incoming messages from sounding alike. If two sources give the exact same message and these two messages are lumped together, the receiver will only see one and will never know two of the exact same were sent. This is critical if the senders are employing two different languages which sound the same but have different meanings. For example, if Agent A1 means 'Yes'

when saying 'No' and agent A2 means 'No' when saying 'No', then when they both say 'No', the receiving agent will get just one 'No' and never confirm whether agent A1 really meant 'Yes' or agent A2 meant 'No' or both. Since the word is shared, the same confusion exists even if only one agent sends 'No' and the other is quiet. In this example, a unique signature means not using the same words as other agents since the same words can have different meanings.

A crippling problem with this arrangement will be a shortage of available words to communicate with in large networks containing several agents. In order to discuss this problem, it should be understood that each communication link communicates with its own vocabulary understood by only the sender and connected receivers. For each link, the language will be different. There is no global language requiring all members to use different words in that language. Local languages allow local problems to be solved. There criteria for determining whether or not signatures are unique in a local language network is to prove each vocabulary sent to a given agent is unique from the others being sent to it. If an agent receives from eight other agents, then these eight senders must possess their own vocabularies.

### 3.2 Message Format

The low level message format is paramount in determining the flexibility and power of expression for a communication link. Although each link in Section 3.1 operates within its own language, all links obey a common message format. This section will discuss four

such formats, evaluating these according to power of expression, capability to implement, and ease of processing.

### 3.2.1 The Language format of the McCulloch-Pitts Neuron

The McCulloch-Pitts neuron [1] is a early type of artificial neuron designed to respond to one pattern presented at its inputs when trained properly. The neuron issues a single output which is a real value between zero and one. A low output value is interpreted to mean that the current input pattern matches the internal reference pattern poorly. A high output value implies a close match. The transition from low to high can be either linear or nonlinear. The above interpretation is the message format of the McCulloch-Pitts neuron. It has been chosen as a first example due to its simplicity and legacy althouth it no longer plays a part in computational neuroscience [9]. This type of neuron is an agent which can give an opinion about one thing. The message it sends exclusively contains the strength of that opinion. Since inputs are additive, there are no problems with multiple reception.

The McCulloch-Pitts agent uses a three stage processor. The first determines the value of each input. Unlike the statement in Section 3.2, the McCulloch-Pitts neuron identifies a message with a sender and the messages do not carry a unique signature. The second stage adds the inputs, while the third applies a nonlinear function to the result. In terms of networking, the problem with this neuron is its poor expressive power. Although simple to implement, the options for processing are limited. Even if more powerful processing was employed such as finding subpatterns, it would be impossible to communicate these findings since the output is restricted to 'Yes', 'No', and 'Maybe'.

### 3.2.2 Word Sequences

Information can be expressed in the sequential order of a limited vocabulary. Much like natural language, these sequences are equivalent to a sentence of variable length. Such a language format is ideal for agents working towards a common goal since the power of a written expression could be equal to that of humans. Realistically, an application for this language format would be in a cooperative image recognition grid shown in Figure 2.2.

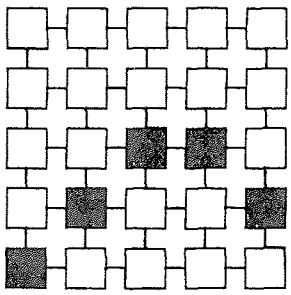


**Figure 3.2** Grid of Cooperative Agents for Image Recognition

Given one agent per square, these are capable of discussing who is black and who is not. The image may even be disconnected. From information that is relayed through the network, all of the grey agents can classify the image impressed upon them. Another option is to have one agent recognize this pattern based on all of the information it receives. The key to the success of this architecture is the language format supporting message relays [11]. The expressive power of word sequences allows for this.

Since what is expressed is determined by the words used and the order they are placed, a wrong word or a misplaced word may create an entirely different meaning. When communicating in pulse ensembles, a simple encoding method is to represent

different words using a unique timing between pulses. A sequence of pulses is a sequence of timings representing a sequence of words. If noise enters the system, added pulses can destroy the entire meaning of the sentence if more robust modulation methods are not used.

The processing model of McCulloch-Pitts neuron is simple and could be split into two basic parts. The main processing section adds and applies a nonlinear function. The second half is dedicated to serve each input. These are the weights. Figure 2.3 illustrates this model.

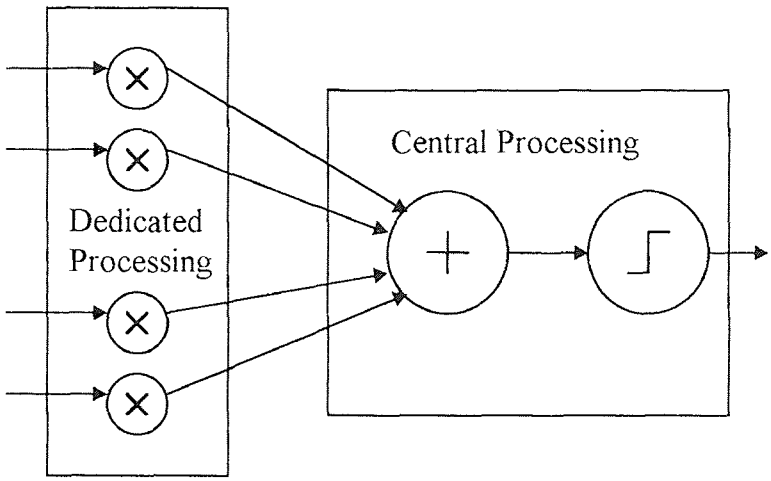



**Figure 3.3** Processing Sections of a McCulloch-Pitts Neuron

Since the dedicated part of the processing is a simple multiplier, this agent can afford to dedicate one for every input. When choosing a research path, it was concluded that a processor capable of interpreting word sequences probably would not be simple enough to afford dedicating processing power to each input. Even if some processing function could be dedicated, there would still be a centralized processor struggling to process several simultaneous messages. The risk is that the power of the language format may lead to the

inability to receive simultaneous inputs and process each equally. A simpler solution is needed that is between the limited expressive power of a McCulloch-Pitts neuron and the complex solutions surrounding word sequences.

### 3.2.3 Space Vectors

The space vector is an inviting alternative to word sequences. Space vectors are limited in expression according to the number of dimensions desired, thus simpler to understand and reason with. Also, space vectors provide interesting avenues for processing functions such as addition and multiplication of vectors, multiplication through a matrix, splicing of components between inputs, as well as normalization. Some simple vector processes can be dedicated to each input easing simultaneous communication difficulties. One process of interest would be to detect correlation between vectors and have each vector influence the other based on the strength of the correlation.

Agents would pass vectors through pulse ensembles. Time distances between pulses can represent the value of each component of a vector. A requirement in this transmission is to preserve the components without confusing which is which. Information containing dimension identification for each scalar value is required. It is expensive to correlate several vectored inputs. Given **n** vectors of **m** dimensions, a correlation matrix of $\mathbf{mn}^2$ is required to see a correlation of just one set of values between two vectors for each pair of vectors. Before vector correlation can become a contender as a simple networking agent, a simple method for vector correlation is needed. In general, correlation of **n** inputs should not result in an $\mathbf{n}^2$ matrix. The language format chosen

should allow for inexpensive correlation. The McCulloch-Pitts neuron can also perform a correlation operation by multiplying inputs together. Unless specific inputs are only allowed to be multiplied with specific other inputs, an $\mathbf{n}^2$ matrix is also required. Chapter 6 shows how set members can be correlated without using this expensive matrix.

### 3.2.4 Using Sets as a Language Format

Communicating through sets is another limited language format easy to understand and reason about. Expression is through the presence and absence of designated members in the set. Each member or a group of members can represent a concept such as a recognized pattern. A McCulloch-Pitts neuron can only represent one concept while an agent communicating with sets can represent several concepts. The tradeoff is that McCulloch-Pitts neurons can express a range of opinion of a concept while sets can only indicate whether a concept is present or not. It has not been discovered how to relay messages using the expressive power in sets to solve the problem in Figure 3.2.

Sets are chosen as the language format of choice for the expressive power over the McCulloch-Pitts family of artificial neurons and the ability to be received simultaneously and asynchronously. With allowable degradation of performance, correlation can be performed with a cost proportional to the average set members sent per transmission.

A non-empty set contains one or more unordered, nonrepeating members. This simplicity allows for robust reception and simple processing. When sending a set through a pulse ensemble, members are represented by the time distances between pulses. Since order is not important, no extra information is needed. If the set is sent more than once,

then the transmission is resistant to noise. This observation is the foundation to asynchronous simultaneous communication. If order preservation or special value identification was required during transmission, reception of superimposed signals would require message processing schemes more complicated than those presented in Chapters 4 and 6.

## 3.3 Sending and Receiving Sets

This section addresses how asynchronous simultaneous communication is possible at the abstract level. The discussion will begin with simple single input reception and graduate to the difficulties presented with several superimposed signals. The essential component used for both reception and processing is the matched filter, capable of extracting signals from noise.

### 3.3.1 Single Signal Communication

The members of a set are represented by time distances between pulses in a pulse ensemble. Figure 3.4 is an example of a pulse ensemble encoding the set {11,20,23,28}.

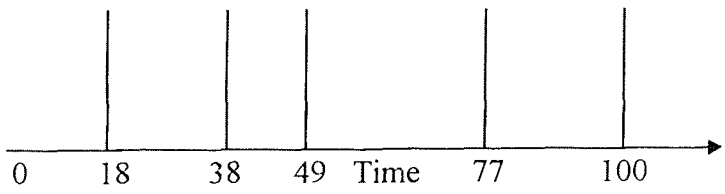



**Figure 3.4** An Encoded Set

Note the indifference to the order members are presented as well as the indifference to the arrival time of the ensemble. Members of a set can be determined as the ensemble is received one pulse at a time. There may have been a pulse received 143 time units ago before this ensemble appeared. This would make 143 a member of the set also which introduces the first restriction when communicating with sets. Only positive integers less than a predetermined value are allowed to exist in a set. Subsection 3.3.2 further tightens this constraint. Such constraints prevent spurious members from being accidentally included in the set.

### 3.3.2 Multiple Signal Communication

The model presented does not employ dedicated receivers for each input, therefore, all input is superimposed and sent to one receiver and processor. Figure 3.5 adds a second signal to Figure 3.4 representing the set {15,24,33,8}. When the two are superimposed using the dashed lines, the resulting interpretation is disturbing.

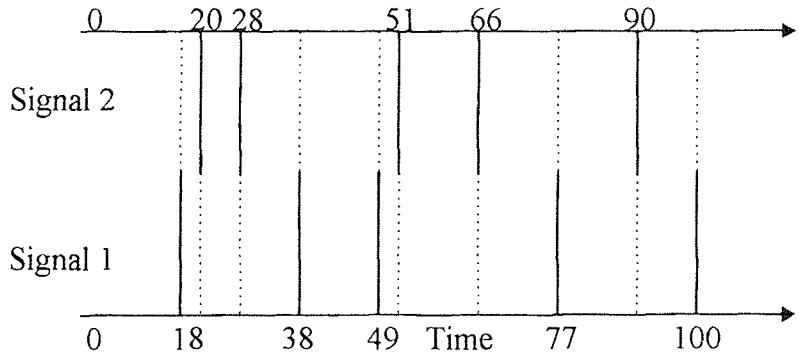



**Figure 3.5** Superimposed Encoded Sets

The resulting set is {2,8,10,11,15,13}. Although some intended set members were counted for, the amount of error will grow as more and more sets become superimposed. Smaller numbers are likely to be added while larger numbers are more likely to be cut by an interfering pulse. The corruption of the received set is in two directions. A way to eliminate this kind of of discrimination to larger set members is to consider all of the distances between all of the pulses. Using this method, the pulse ensemble in Figure 3.4 now yields a larger set of {10,31,59,82,11,39,62,28,51,23}. This set contains the intended members plus extras. The resulting set for Figure 3.5 contains the union of the two sets {2.10,20,31,33,48,59,72,82,8,18,29,46,57,70,80,21,23,38,49,62,72,11,13} and {28,39,52,17,41,51,15,26,24,34}. Again, all of the intended set members are present with extras. It is unacceptable to see the ensemble in Figure 3.4 generating so many extra members. Once the method of counting distances betwen all pulses can be applied to a single signal without spurious results, it can be used for multiple signals. Again, the advantage of this method is the ability to still capture all intended set members.

The way to prevent spurious set members from forming when interpreting the ensemble in Figure 3.4 is to further restrict the set member boundary to one octave in values. For example, if the lowest member of the set is 8, then the highest member can be 15. When these kinds of sets are encoded, decoding returns the same set since all of the additional numbers will be above one octave boundary. When applied to multiple signal input, the values above and below the octave limit are cut, but the extra members within the octave are counted. The matched filter described in the following Subsection will eliminate these extra members.

### 3.3.3 Using a Matched Filter

A matched filter is special kind of filter that expects what the input signal will appear like and pull it out of the noise [10]. If a matched filter is looking for the number 34, it will reject all other numbers except for 34. In terms of pulse ensembles, the matched filter looks for a certain time distance between pulses. The plan for properly using this filter is to find which time distance to tune the filter to and record the output. The previous sentence sounds awkward at first since if the distance is already known, it should just be added to the set bypassing the filter. This is correct except that there is a filter for each possible member in the set octave and the filter tunes it self by becoming more sensitive to a distance each time it occurs. The object is to send the set several times within one ensemble. As the intended distances are repeated, the filters tuned to these distances become more sensitive and allow these numbers to enter into the received set. The unintended pulse distances are less likely to repeat as often as the intended distances, therefore, these filters will be less sensitive, and will reject the noise. This depends on the spurious distances ocurring less frequently.

### 3.3.4 Problems with the Matched Filter

Proper operation of the matched filter approach requires the spurious time distances to occurr less frequently than the intended distances. This will not be the case if the ensemble combination in Figure 3.5 is repeated exactly the same each time. At this rate the noise can not be separated from the intended signals. It is best to eliminate any periodic patterns between simultaneous sets. The time it takes for a set to be represented

once in an ensemble is the sum of all the members. If one set repeats every 100 time units and another set repeats every 80 time units, then the pulse relations will be exactly the same every 500 time units. At this rate, the first set sends its intended signals 4 times before the exact same spurious distance squences repeat twice. In the case where the set sums are identical, there needs to be more effort to keep spurious distances from repeating.

If each time the set is sent through an ensemble the order of the members is changed, then the chance for repetition is decreased even more. Tests show that the combination of long periods and random member selection sufficiently scatters the spurious pulse distances. The worst case would be several sets with few members and short repetition periods. Figure 3.5 show the results of such a test. Here, the acceptable octave is from 100 to 199. The height of the line represents the strength of the presence of the set member it represents.

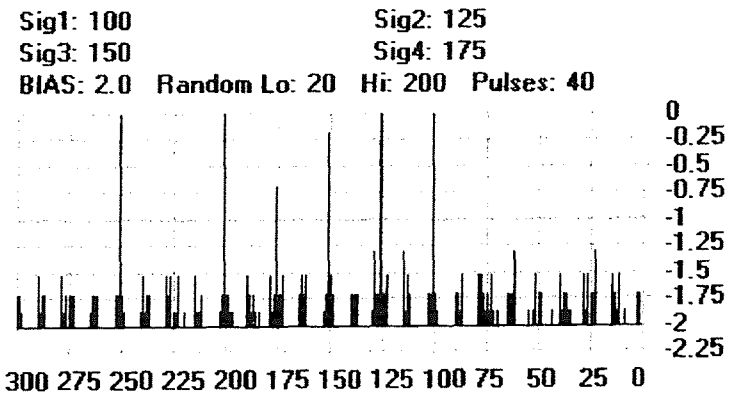



**Figure 3.6** Worst Case Scenario for the Matced Filter

The sets are {100},{125},{150}, and {175} being the best candidates for periodic behaviour. Note on how the noise is grouped together and additive unlike other samples in Appendix A. The intended signals still dominate over the noise, but according to the detection thresholds set in Chapter 4 as -1.71 or -1.33, several noise products will also be interpreted as acceptable signals.

### 3.4 Conclusions

When encoded and decoded properly, communication through sets appears to be an effective alternative to the other possibilities presented. The key concept is bringing the set members out of the noise by repetition.

# CHAPTER 4

# RECEIVER DESIGN

Given the input medium is a pulse ensemble where distances between neighboring pulses represent numbers in a set, this chapter will design a receiver capable of interpreting such input. Other criteria set in Chapter 3 are the abilities to handle simultaneous signals asynchronously. Following the guidelines of the thesis objective, the design will be analog in nature. See Chapter 8 for a suggested physical implementation.

## 4.1 Nominal Design

The nominal design concentrates on basic architecture and algorithms to ensure the above criteria can be accomplished. Section 4.2 is concerned with achieving maximum performance. It would be ordinarily be difficult to detect pattern codes [8], but the matched filter presented in the design below simplifies the design.

### 4.1.1 Basic Receiver

The basic problem to be solved from Chapter 3 is how to measure the distance between two pulses when one or several interfering pulses are positioned in between. In the case of simultaneous input, there will be several ongoing pulse width calculations of which each input must be received through the interference caused by the others. The digital solution would be to use an array for all integers in an octave. A linked list is used to store all possible pulse widths less than or within the above octave. As a new pulse arrives, a new element is formed in the linked list containing the distance in time from the current pulse

to the previous pulse. This distance is also added to each link in the list. If the distance in the link becomes greater than the highest accepted value in the octave, then the link is removed. In this way, all possible pulse distances can be remembered. The matched filter can be employed to pull the intended distances from the noise. The problem with this design is the complexity required to operate the linked list.

All pulse widths can be memorized dynamically by allowing them to travel slowly through a special material. This slow material is the heart of the receiver and processor design. The next three figures show the basic concept of determining all possible pulse distances as a new pulse arrives. The first step is to start with the first pulse. This enters at the left and travels through the fast material. A simplification in the design is to treat the fast material as instantaneous in pulse propagation relative to the slow material. This pulse instantaneously travels throught the fast material and ends at the far right. The pulse will now continue its travel to the left along the slow material.

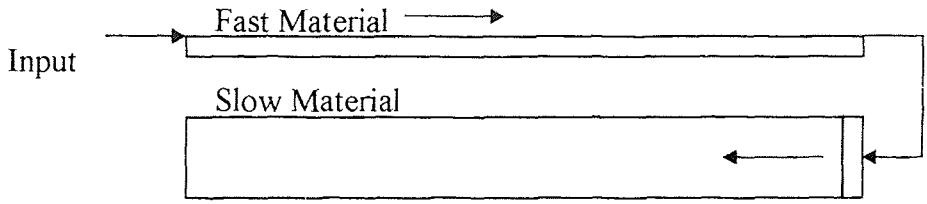



**Figure 4.1** Initial Signal into Slow Material

On the far right of the slow material is a pulse propagating to the left. A second pulse will now arrive on the input as shown in Figure 4.2. Before this though, some time has passed and the first pulse has been allowed to propagate along the slow material. Now, the distance in time between the first pulse and the second pulse is represented in the slow

material as a distance in length. Since this distance needs to be entered into the matched filter, a third material is introduced into Figure 4.2. This material has analog static memory along its length. When a pulse travels throught the fast material, it intersects all pulses traveling in the slow material. Where the fast pulse meets the slow pulse is where this pulse distance is recorded on the memory material. The position of the projection relative to the right edge of the memory material marks the value of the received signal.

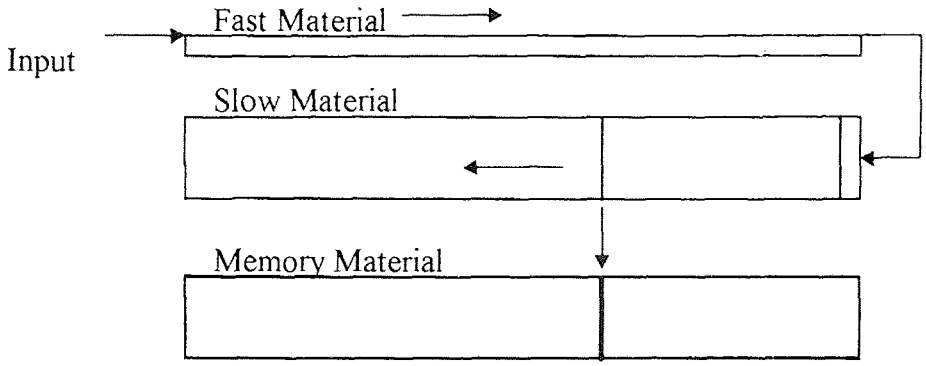



**Figure 4.2** Second Signal Projects First onto Memory Material

A third example, shown in Figure 4.3, demonstrates how all pulse widths are remembered by the memory material. When the third pulse arrives, the first two have propagated accordingly throuth the slow material. Again, the third pulse will intersect with the pulses in the slow material and project this intersection upon the memory material. In this Figure, the octave bounds have been added. It can be seen that the time distance between the first and third pulses is beyond the upper range of the octave. Out of range pulse widths are shown to be projected onto the Memory material, however, actual implementations need not do this. The simulation program does not clip these projections

in order to show the kinds of spurious signals that can occur, and to demonstrate the need for the octave boundaries.

If a fourth pulse were to arrive, it will be treated much like the first. When enough pulses arrive in order to send each member of a set more than once, the travelling pulse along the slow material will be projected onto the same place more than once. This activity makes the memory material more sensitive in this area. As discussed in Chapter 3, the amount of added projection onto the memory material is exponentially equal to the amount of projection already present at the moment.

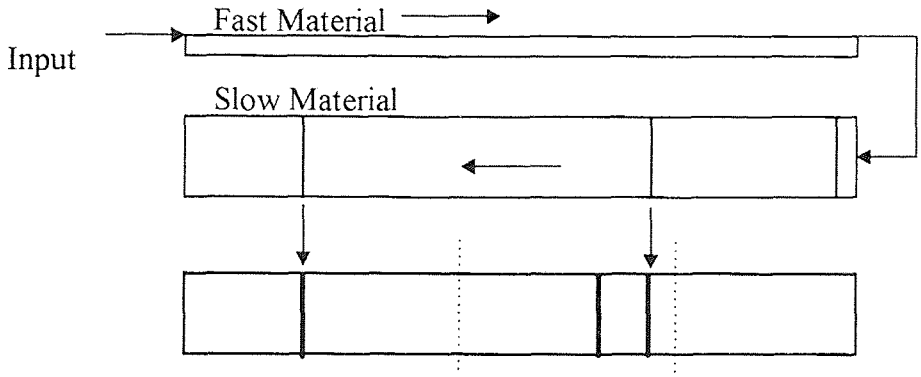



**Figure 4.3** Third Signal Projects Previous onto Memory Material

This method for recording all meaningful pulse widths is demonstrated in all of the figures in Appendix A. The values from 0 to 300 mark the distance along the memory material as in the above figures. In each figure, on the top right of the graph is a measurement called **PULSES**. This indicates the total number of pulses input to the receiver. Figure A.10 shows exponential growth on the memory material. As discussed in Chapter 3, this exponential growth is the matched filter adjusting itself to either accept the intended signal that repeats frequently or to reject the noise which shouldn't.

### 4.1.2 The Matched Filter and Noise

At this point, the matched filter self-adjusts to exponentially accept repeating signals. However, if a long enough period persists, the unwanted pulse widths will also repeat and begin to project increasingly stronger onto the memory material. Figure 4.4 is similar to Figure A.3 except that it is sensitive to repetitive noise. Another relationship between signal and noise needs to be exploited more fully is that the intended signal will statistically repeat itself more often than the noise at a given slot. The receiver can take advantage of this relationship by having competition among projections. This Subsection intends to study three methods for managing competition. All methods of competition are a form of normalization upon the memory material. Eventually, how to normalize will best be determined when attempting to physically implement the receiver in Chapter8. In general, any normalization is better than none.

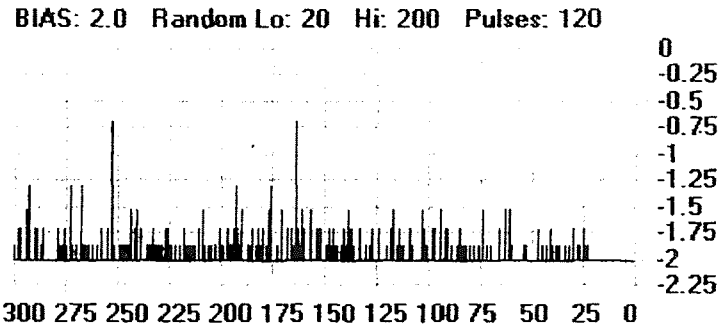



**Figure 4.4** Receiver Output for Random Input, No Noise Suppression

**4.1.2.1 Unbiased Normalization:** This concept is the simplist to visualize. Given there are 100 individual slots in the memory material, any increase in a given slot will cause

equally distributed decrease to all its neighbors. If the increase is by .25 then all the other slots will be decreased by .25/99.

**4.1.2.2 Compete with Neighbors:** This concept treats the memory material like a transmission line. As a pulse is projected, the increase of the slot potential in the memory material causes a negative wave to the left and right of the projection. Each wave begins at half the value of the initial increase. As the wave travels, a percentage of the wave is absorbed at each slot and the remaining part of the wave continues to the next slot. If the edge of the material is reached, then the wave is reflected back as in an open circuit reflection. As the percentage absorbed approaches zero, this type of normalization becomes unbiased as in 4.1.2.1. This type of competition favors slot potentials to be spread out evenly across the octave. If intended set members are close in value, then they will compete with each other rather than the noise. Section 4.3 discusses expected set member distribution.

**4.1.2.3 Compete with Neighbors, Biased:** The method descriped here is the same as above except that instead of absorbing a small percentage that is predetermined and fair to all slots, an unfair amount is absorbed according to the current value in the slot. All data in Appendix A use this method. The amount absorbed is shown below where $\mathbf{A_n}$ is the amount absorbed in the slot n and $\mathbf{V_n}$ is the total potential for the memory slot.

$$\mathbf{A_n} = \mathbf{e}^{\mathbf{V_n}} \tag{4.1}$$

What is not absorbed continues on in the wave. This method favrs higher memory potentials over lower ones.

**4.1.2.4 Methods that do not Normalize:** Perhaps a hybrid between the previous two methods can be achieved by applyng the negative potential across all slots at once, but each slot absorbs according to its current potential. This method does not normalize, but also does not target neighbors unfairly. A failure to normalize can result in unbalancing the sensitivity of the receiver. No competition causes an unbalanced state where the sensitivity of the receiver is greater than it should be for a given slot. Too much competition will hurl the receiver in the other direction where intended signals will not be detected due to poor sensitivity. An iterating algorithm can be employed to ensure normalization does occur, however, this will be complicated and will go against the goals of building simple analog circuits.

### 4.1.3 Properties of the Memory Material

The memory material is for short term use only. Once the agent has sent its signal, the memory material will be reset for another round of input. As to exactly what potential to set this material depends on a compromise between signal to noise ratio and response time. This subject is discussed in detail in Section 4.2.

### 4.1.4 Making a Hypothesis

Given the mechanisms to separate the intended signal from the noise, there needs to be a final hypothesis as to whether or not a slot contains an intended signal or just noise. Figure A.7 shows a progression of snapshots of the receiver's memory material. Given the octave is from 85 to 170, one can see there is no noise present. After 25 pulses have been received, the intended set is clearly identified. Figure A.11 shows a progression that is not so clear. After 75 pulses, frame (c) clearly detects the set {103,148}, but it is less clear on detecting the complete set {165,115,85}. Note that for each input, which member of a set to send in the next pulse is determined by a standard random number generator. Apparently, the number 85 has not been sent as many times as its brothers in the set. This fairness issue is discussed in Chapter 7. Another factor is the set size for each input. The first input has 3 members to represent with a fixed amount of pulses, while the second input needs to only represent 2 members given the same number of pulses. This results in superior detection of the second set over the first. The task is to recognize both sets equally.

One simple method is to employ a detection threshold. Any slot in the memory material having a potential greater than a given limit will be considered as an intended signal, a set member. Those being less are considered as noise. For Figure A.11, a possible threshold can be a potential of -2. Figure A.13 (d) shows a complex graph with four input sets with about two members each. A threshold of -1.5 will work well in this situation. A general equation can be used to determine an acceptable threshold:

$$\mathbf{T} = \ln\left((\mathbf{B} + \mathbf{e}^{\mathbf{B}}) + \mathbf{e}^{(\mathbf{B} + \mathbf{e}^{\mathbf{B}})}\right) \qquad (4.2)$$

**T** is the threshold and **B** is the initial bias on the memory material. The equation places the threshold threshold to the value of a slot after it has been projected upon twice. Three times will be necessary to exceed the threshold. Note that if competition as described in Subsection 4.1.2 is employed, then the slot may have to be projected upon by more than tree times. Applying this equation to Figure A.11, the threshold should be 2.33 which is fine given no noise. Applying this equation to Figure A.13 yields a threshold of 1.71 which also works well with multiple input sets.

Another bias equation is to avoid exponents and look for a signal to rise a certain percentage of the bias determined by **K**.

$$\mathbf{T} = (\mathbf{K}\text{-}1)\mathbf{B} \tag{4.3}$$

The general rule is to increase the threshold to make a more certain hypothesis while decreasing the threshold will produce a quicker response.

### 4.1.5 Handling Aging Sets

A set ages after it is detected resulting in a diminished presence in the receiver memory. If competition is employed, then a set sent previously will be erased by more recent input. The examples given so far have been worst case where all input sets occur superimposed. A more probable scenario are the sets arriving in streams of about 20 pulses, where these streams may partially overlap, or not overlap at all. The acceptable window to receive a stream may be relatively large in comparison to the stream itself. Speculations on the average case will be discussed in Section 4.3. The question is whether to forget aging sets or to remember them.

Assume a set was received early and although it has been received, the members of the set are not sufficient to cause output. See Chapter 7 for criteria to send output. After this isolated event, nothing happens for a long time. This makes the initial input set stale. It is likely that this set is no longer relevant. This set should be forgotten.

Assume a set was received early as above. Another set has been received either on the overlapping end of the first set or immediately after. In this case, the first set should be remembered. A decay should be employed upon the slot potentials of the memory material. This decay should be adjusted accordingly as to not ruin relevant set members or to preserve stale set members. Given the decay function will approach the original bias from either side, it can be written as:

$$\mathbf{V_n} = (\mathbf{V_n} - \mathbf{B})e^{-t/T} \quad (4.4)$$

$$\mathbf{T} = \mathbf{KMS/O} \quad (4.5)$$

Where **T** is the decay rate, **M** is the length in time to send a set, **S** is the number of sets expected, and **O** is the expected set overlap. **K** is a constant to control the rate of decay. The actual values of these parameters depend on constructing actual networks which is beyond the scope of this thesis.

## 4.2 Maximizing the Parameters

This section is dedicated to study the response of a receiver constructed in Section 4.1 using the competition method described in 4.1.2.3. There are three general areas of performance: pure noise response, single set response, and multiple set response. The key issue is where to set the bias for the memory material. This section also serves as an

analysis of the design presented in Section 4.1. After studying the graphs in Appendix A, one can get a feel for the capabilities and limitations of this receiver.

### 4.2.1 Performance in Pure Noise

The pure noise tests are a prelude to Subsection 4.2.3 where multiple inputs will interfere with each other. The goal of this section is to understand how setting the bias affects noise performance in the presence of competition. In these tests, the noise is generated by a simple random number generator supplied with the Sun Java 1.0.2 compiler.

Figure A.1 shows the results of applying just five pulses of pure noise to the receiver input. No input is allowed to exceed a potential of zero. If this were not the case, then the slot potential will continue to grow exponentially and kill all competitors. Also, when attempting to construct a physical circuit, there needs to be a physical cutoff limit. From Figure A.1, it can be seen that only one pulse is required to push the slot potential to the limit. When observing the base of each active slot, the negative impact the slot has upon its neighbors can be seen in negative skirts. With the bias level at just -0.5, these skirts are too narrow. If the skirts are too narrow, then the affect of the skirt is local only, and at a distance has the effect of no competition at all.

Figure A.2 shows slightly improved results over the previous figure. Here, the bias has been set to -1.0. Two hits on the same slot are required to push the slot potential to its limit. The competition skirts are wider and noticable in frame (c). The intention of the competition skirts is to reduce the slot potential of neighbors. If the receiver is exposed to enough noise, some slots will be hit more than once. The object is to reduce the potential

of the slot before the second hit is due. Given a good random distribution, several neighbor slots will be hit at least once before the same slot is hit twice. As the neighbors are hit, thier competition skirts will reduce the potential on the central slot.

Figure A.3 (d) shows the intended application of the skirt. Compared to the previous figures, the Bias of -1.5 will be the maximum acceptable for noise rejection. According to the detection threshold set in Subsection 4.1.4, a slot potential of -1.0 is enough to be judged as an intended signal. In this example, 15 to 25 random pulses are required to produce a false signal.

Figure A.4 shows the skirt in a different mode where it begins to appear flat as more random signals are input. Eventually, the shape of the skirt seen in (c) is lost in (e). From 30 to 60 random pulses are required to produce a false result. At this point, the bias is at -2.0

Increasing the bias past -2.0 does not increase the noise rejection significantly farther. Setting the bias to -2.5 produces a false signal at around 60 pulses and setting the bias to -3.0 produces a false signal at around 60 pulses also. If the second equation of Subsection 4.1.4 is employed, then noise rejection increases steadily as bias decreases. Table 4.1 displays the compiled results.

**Table 4.1** Receiver Noise Immunity

| Bias | Threshold Eq. 1 | Threshold Eq 2 (33%) |
|---|---|---|
| -0.5 | <1 pulse | <1 pulse |
| -1.0 | 15 pulses | <1 pulse |
| -1.5 | 15 to 25 | 15 to 25 |
| -2.0 | 30 to 60 | 120 to 200 |
| -2.5 | 60 pulses | > 500 pulses |
| -3.0 | 60 pulses | > 500 pulses |

As mentioned in Section 4.3, there will be no situation of random noise only. The purpose of this test is to predict performance in Subsection 4.2.3. From the results given here, the best bias to use is from -1.5 and below.

### 4.2.2 Single Input Set Performance

The objective of this Subsection is to determine the response time of the receiver without any noise input. This time should be about 10 pulses regardless of the bias given according to the threshold calculation given in Subsection 4.1.4 first equation. If the other threshold equation is employed, with **K**=0.33, then the slot potential must exceed one third the distance from the bias to the top.

Figure A.7 is a trial with a bias of -1.5. This is considered the minimum acceptable by in the last Subsection. Here, 10 pulses meets the threshold for either equation. As an improvement, Figure A.8 is a rceiver with a bias of -2.0. Here, 10 pulses push the appropriate slots past the first equation threshold as expected. With the second equation, about 15 to 20 are required. As the bias decreases, more and more pulses are required as shown in table 4.2.

**Table 4.2** Single Signal Reception

| Bias | Threshold Eq. 1 | Threshold Eq 2 (33%) |
|---|---|---|
| -0.5 | <1 pulse | <1 pulse |
| -1.0 | 15 pulses | <1 pulse |
| -1.5 | 15 to 25 | 15 to 25 |
| -2.0 | 30 to 60 | 120 to 200 |
| -2.5 | 60 pulses | > 500 pulses |
| -3.0 | 60 pulses | > 500 pulses |

### 4.2.3 Multiple Input Set Performance

The true test for this receiver is to input multiple sets where each set has at least two members. As discussed in Chapter 3, when two or more inputs are simultaneous, spurious pulse widths will be generated and will be the limiting factor in these tests. The first test will be using two simultanous sets, while the second test will double this amount to four. The objective is to determine an appropriate operating bias of the memory material as well as to measure how well the receiver performs at these levels of difficulty.

**4.2.3.1 Two Simultaneous Inputs:** Figure A.11 shows the reception of two sets of numbers. The first set is {165,115,85} and the second set is {103,148}. The operating bias is -1.5. For either threshold calculation, it takes about 30 pulses to detect the intended signal. This comes to 15 pulses from each source. Spurious noise is present and will eventually punch throught the threshold as well. At this bias level, noise is noticable, but not a factor after 60 pulses.

Decreasing the bias to -2.0 requires more input pulses to produce output results, but performs better with noise. In Figure A.12, the first threshold equation finds all five numbers after about 30 pulses. The second threshold equation needs about 60 pulses.

Lowering the bias even more produces the results in Figure A.13. It takes 35 pulses for the first equation and 75 pulses for the second. Note how the noise is practically eliminated.

From the compiled results in Table 4.3, the best choices at this time are a bias of -1.5 and -2.0. These are the critical points for receiver sensitivity. A higher bias allows too much noise, while a lower bias requires too many input pulses.

**Table 4.3** Double Signal Reception

| Bias | Threshold Eq. 1 | Threshold Eq 2 (33%) | Noise |
|---|---|---|---|
| -1.5 | 15 pulses each | 15 pulses each | Noticeable |
| -2.0 | 15 pulses each | 30 pulses each | Low |
| -2.5 | 18 pulses each | 38 pulses each | None |

**4.2.3.2 Four Simultaneous Inputs:** This is the most difficult test for this receiver. Only the two recommended bias values from the previous tests will be used in this study. The noise factor is expected to corrupt results as well as competition among intentional signals, requiring more pulses input to meet detection threshold. The four input sets are {165,115,85},{103,148},{155,93}, and {132,121}.

The first test is at a bias of -1.5. Figure A.14 shows that after 39 pulses, 8 out of 9 numbers have been detected and one spurious result between 125 and 150. After 58

pulses, all 9 numbers are detected with one spurious result. After 78 pulses, two spurious numbers are showing. The receiver at this bias level is able to detect the intended signals, but unable to reject the noise.

The second test is at a bias of -2.0. Figure A.15 show that after 90 pulses, all intended numbers are detected. This is at about 22 pulses per input source. After about 180 pulses did spurious results show. Using the second threshold equation at 33% of bias, all pulses were not detected until about 130 pulses have passed. However, no spurious signals were detected even after 180 pulses have been present.

The conclusion of the test is that a bias of -2.0 is recommended for operation while the detection threshold method is not a critical factor in receiver sensitivity. The first bias equation is recommended over the second due to improved response time. It will be shown in Chapter 5 that these spurious results have a low chance in affecting agent output.

## 4.3 Expected Operating Conditions

This section discusses what environment the receiver is expected to perform in. It can also be interpreted as specifications for which operation is possible. Two main catagories of specification are the expected output specifications of an agent and specifications on how inputs can overlap. These specifications are a result of receiver design criteria and test results.

### 4.3.1 Agent Output Specifications

Each agent outputs one set containing one or more numbers. The number of pulses sent must be proportional to the number of members in the set. For single signal reception, only 3 pulses per member are needed but this number increases to 7 in noisy environments. Members in a set are expected to be distributed evenly. See Chapter 6 for more details on how to accomplish this. None of the output pulses are spurious. Members of a set will be transmitted randomly in order to randomize spurious pulse widths when combined with other output signals. Members of a set are represented fairly in the output stream. Negligence towards one member in a set will result in failure to detect.

### 4.3.2 Input Combination Specifications

The tests performed previously are worst case scenarios where all sets completely overlap each other. The case of single signal reception is also a best case scenario which will not occur every time. There is no experimental information to suggest the true operating environment these receivers will be subjected to. The purpose of this section is to suggest one. This section affects the aging decay half life introduced in Subsection 4.1.5. Below is a table suggesting operating preferences of the receiver:

**Table 4.4** Specifications for Multiple Inputs

| | |
|---|---|
| Shortest output pulse width | 85t |
| Longest output pulse width | 170t |
| Avg. pulse width | 127.5t |
| Avg. number of pulses per input | 15 |
| Average length of set stream | 1912.5t |
| Avg. number of sets to receive | 8 |
| % overlap between sets | 60% |
| Min. number of simultaneous inputs | 2 |
| Max.. number of simultaneous inputs | 3 |
| Avg. % stream shared with none | 21% |
| Avg. % stream shared with one | 42% |
| Avg. % stream shared with two | 37% |
| Avg. length of combined stream | 7267.5t |

# CHAPTER 5

## MESSAGE PROCESSING

The goal of the processing unit is to transform the input message intelligently to an output message. The intelligent aspect of the transform may not be readibly comprehendable for a stand alone agent, and may be realized when within a group of agents networked together. Two basic processing functions are available; message relaying, and message correlation. This chapter will present the possibilities of both using sets.

In general, the objective of the processing stage is to combine all inputs into one consistent output. The consistent output can be interpreted as one voice, instead of several voices sharing one output path. All agents connected to the output are only concerned with interpreting the one voice instead of several. If the output appears to be several languages overlapped, then agents not only learn the language of the previous stage, but of the stages before. The one voice is comparable to the output signature presented in Chapter 3. The single voice initiative is a concept fitting for an agent having several inputs and only one output.

Given an agent with several simultaneous inputs carrying large quantities of information, it will be impossible to fit all of these inputs into one output, and as mentioned above, undesirable since the output will consist of several different voices. When combining the inputs, compression is required to eliminate redundant messages. Two policies can be applied to this method. The first is to output only the messages that can be combined, and the second is to output as much as possible. The second policy is

more complicated since it requires decision making about messages that can not be combined with others.

## 5.1 Message Relay

A message relay mechanism within an agent is useful for recognizing variance in input patterns. Regardless of where the pattern is placed on the input, the pattern will be sent to the proper agents for processing. The alternative to message relaying is to have each agent in a network trained for every possible variance.

The primary goal is to combine inputs that can be combined in order to produce an output of one voice. The inputs that can not be combined can either be omitted from the output process, or can be sent in addition to the combined output according to the message bandwidth left over. These additional messages are essentially relayed. As mentioned in the introduction, decisions need to be made on what messages can get relayed and which will not.

The above policy does not guarantee sending the proper messages to a destination where the information will be useful. Along with a relayed message, a destination tag is required. This tag can be interpreted by all agents along the way. An individual agent will be responsible for reading the tag and deciding either to relay or not to whether or not one of its output paths brings the message close to the destination. This requires the agent to know its position relative to all destinations. Even if one of the output paths leads to the destination, the message will get relayed to all agents connected to the output. These agents may or may not retransmit the message. According to Chapter 2, agents will be

networked in a mesh resulting in an indirect path from the output of the agent back into the input. These closed loops can be dangerous causing unwanted oscallation of relayed messages. Unless the relay process is handled properly, it will cause more harm than good. Currently, these issues do not implement well using the set communication medium and will not be considered. Focus on the processing section will be spent combining several messages into one based on correlation. Messages not correlating are omitted.

## 5.2 Message Correlation

Chosen as the only processing function, there are several issues concerning message correlation including how to define and measure correlation according to what information is suppplied and what is needed. In set communication, the presence of a number is a positive indication of an event or a set of events. The absence of a number indicates no event by default. Only the positive presence of events will be measured, ignoring absent events. Other issues are policies on representing complex patterns with hierarchical structures.

Two messages correlate when two or more numbers from the union of the sets occur often enough to be noticed in competition with all other message combinations. Only a top percentage of message combinations resulting in the highest correlation detection will be accepted as a correlation event. Every time this happens, the agent will output a single number to represent the number pair or group. When communicating with sets, each number either represents itself or another set of numbers. When considering complex patterns, a number in a set may represent more than one set.

### 5.2.1 Correlation Detection Method

Correlation may be measured loosely or tightly. Consider two coins being tossed at the same time. The results of the one coin do not track the other, therefore it can be said the results of the coin toss do not correlate. The other approach is to maintain there are four popular patterns, {H,H},{H,T},{T,H}, and {T,T}. The second approach is more useful in categorizing the coin toss. The equations for both methods are presented below:

$$\mathbf{C(AB) = P(AB) + P(\overline{A}\overline{B}) - P(\overline{A}B) - P(A\overline{B})} \quad (5.1)$$

$$\mathbf{C(AB) = P(AB)} \quad (5.2)$$

**C** is the correlation function while **P** is either a probability or a frequency function. Equation 5.1 measures how often both coins are the same in relation to how often both are different. The second equation measures how often both coins are {H,H} in relation to no other combination frequencies. To measure the correlation of all the other combinations, three more equations like Equation 5.2 are required measure the frequencies of {H,T}, {T,H}, and {T,T}. Equation 5.2 has advantages over 5.1 in being easier to calucluate and not being able to track when an event does not happen. The results are more plentiful and preference is given to message sets occurring frequenly over those that are rare. Two numbers that track each other very well but occur seldom will be measured strongly by 5.1 and ignored by 5.2. Likewise, two numbers that never occur simultaneously are also recognized by 5.1 and ignored by 5.2. When communicating with sets, the presence of a number represents an event or a set of events. It would be useless to measure how often two events do not occur, therefore Equation 5.2 detects useless correlations. Given the design approach in Chapter 6, only equation 5.2 is possible. In this case, several patterns will be recognized, requiring competition for the strongest. In

order to make 5.2 behave closer to 5.1, numbers ocurring at low frequencies must have an equal chance to be recognized as those occurring often. Equation 5.3 provides an equalizer.

$$C(AB) = \frac{P(AB)}{P(A) + P(B)} \tag{5.3}$$

### 5.2.2 Representing Patterns

This Subsection provides possible solutions to representing the correlation of complex patterns. The policies for representing these patterns will be determined by the actual processing algorithm. Desirable solutions, however, will be presented here.

The simplest pattern recognizable in a set message is two numbers. When these two numbers occur, a pattern is recognized. This process can be expanded to three numbers or more. Each time, the number assigned to the pattern represents the numbers within the pattern. Complex patterns are hierarchichal in nature and are not readily representable with flat sets. In this process, some information will be lost, either the hierarchical relationship or the difference between layers in the hierarchy. Both pieces of information can be preserved by assigning one number to represent what is common and one number to represent what is different. Discussions will be limited to two levels within the hierarchy.

**5.2.2.1 Subpatterns:** A subpattern is a relationship where one pattern is contained completely within another. For example, {7,2} is a subpattern of {10,7,2,18}. If the parent patterns occurs, then it is simple to assign a number to this pattern. In the event of

the subpattern, there are three choices. The first is to maintain the relationship in the hierarchy and represent both patterns as the same number, the other is to maintain the difference by representing both by individual numbers. A third option is to keep both by having one number represent the hierarchy and the other represent a modification within it. In this example, the set {22} represents the subpattern, and the set (22, 25} represents both the subpattern and the parent patttern. It is unclear what the set {25} represents. If subpatterns are treated as a type of shared pattern, where both share {7,2}, then {25} represents {10,18}.

**5.2.2.2 Shared Patterns:** A shared pattern is a combination of overlaping patterns such at {12,15,8} and {5,12,15}. Again, there are three ways to represent these types of patterns. The first is to preserve the relationship by assigning one number to represent both patterns. The definition of the pattern becomes $(12 \cap 15) \cap (5 \cup 8)$. The second interpretation preserves the difference by assigning separate set numbers for each pattern. The third approach attempts to preserve both by assigning one number for $(12 \cap 15)$, a second number for (5) and a third number for (8). The second and third numbers are not allowed to appear without the first.

**5.2.2.3 Evaluating Each Approach:** The third approach presents the most information at the cost of using more numbers. The worst case is a group of two shared patterns such as {1,5} and {1,3}. According to the first method, one number is required to represent the input pattern of {1,3,5}. The second method requires two, and the third requires three

numbers. From this example, the third method has performed no compression of the information. The best way to understand how each method will perform is to evaluate what will happen at the following stages. Since the third method did not perform any true processing for the worst case, it will be propagated through all stages in an endless loop. This is not acceptable. Even without the worst case at first, the worst case will eventually be derived and propagated. The first method will output one number for either pattern matched. This number will go to the next layer of agents. If this number correlates, strongly, then it will be represented as part of another pattern. If it does not correlate, then the influence of the original patterns stops at these agents. With the first method, information that two patterns have been presented is forgotten and is treated as if only one pattern is present. Either this relationship is forgotten, or another agent sharing similar inputs will detect one of the two patterns also. The second method is a compromize between the first two. Two overlapping patterns are represented as two numbers, the following stage can easily correlate these two numbers and represent the two sets as one number. The second stage can also use each pattern number to combine with others. This option is the most flexible and is encouraged.

### 5.3 Regenerative Agents

An advantage of communicating and processing sets is the ability to create regenerative agents. When an object is regenerative, its output or a function of the output can be routed back into the input for the positive effect of further processing. These agents operate at several levels in the hiererchy, one leavel for each cycle in the loop. This avoids

the linear approach to cascading agents as in Figure 5.1. Instead of a strait line, the output of the last layer can be wrapped back around to the first layer again. Here, the first layer processes low level input information as well as higher levels.

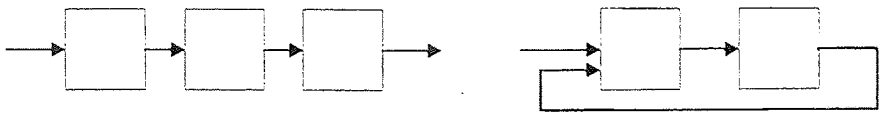


**Figure 5.1** Linear Vs. Regenerative Architecture

The advantages of a regenerative architecture come from two areas. First, the feedback path allows complex information to develop without ending. The complexity of the results depends on the number of layers in the linear architecture. A drawback is in the difficulty of extracting the result, since it can be anywhere along the length. The same can be said for the linear design, since simple patterns will be recognized early. The second advantage area is the inherent method for which input is forwarded upstream. In the case where a complex pattern higher in the processing chain requires low level input to continue, several forwarding paths are required to accomplish this. The regenerative design provides this path naturally.

With an agent comnumicating with sets, all set numbers are combined together providing anonymity to the sender. Therefore the sender can pretend to be several virtual agents providing several sets, one for each level in the hierarchy, to the target agent. The target agent will see correlations among several inputs for each level of communication as well as correlations between levels.

# CHAPTER 6

# PROCESSOR DESIGN

This chapter will present a design for a simple processing unit to recognize frequent sets. Finding a correlation between several input sets would normally require an **n** by **n** array to compare each possible number with every other. This approach becomes three dimensional when attempting to correlate sets of three numbers. With receiver specifications set to handle several simultaneous sets, a more efficient design is required. The scheme presented below performs a similar operation using an **n** element one dimensional array. As expected, there is a compromise in performance, requiring iterations of tuning for acceptable results. This design is substantially more complicated than the receiver not only in algorithm form shown in Appendix D, but also in the physical design approach presented in Chapter 8.

A similarity exists with visual pattern recognition where sets can be mapped out in a grid as in Figure B.1. Here, the members allowed to be in a set belong to a pool of sixteen numbers. These numbers are chosen randomly, but are consistent from set to set.

## 6.1 Nominal Design

The nominal design concentrates on basic architecture and algorithms. Since this design presents a compromise in performance, it will be necessary to utilize the scheme presented below with the upmost efficiency in both the careful selection of performance goals as well as proper algorithms to achieve these goals. Given the current constraints of a simple physical design, the performance of this current design falls short of expectation. This

nominal design should be interpreted as a starting place rather than the end. This section provides several selections and guidelines for design, while Section 6.4 suggests improvements on the design based on the test results.

### 6.1.1 Basic Processor

The goals of this Subsection is to present the paradigm to solve for correlation in one dimension. The design paradigm and materials are the same as used in Chapter 4. As discussed in Chapter 5, the exact method for correlation will not be achieved but rather a measurement for the frequency of coincidence. A side effect of this measurement is that a recognized pair of numbers may just be appearing at a high rate and are not related in any way. This method was found to be acceptable and even more desirable over exact correlation since even though a pair or group of events may not correlate, frequent combinations will be noticed. For reasons of simplicity, Equation 5.3 will not be implemented.

Before presenting the single-dimensional design, the two-dimensional design is presented first as shown in Figure 6.1. Each axis contains the complete pool of possible members that can belong to any imaginable set detected by the receiver. Each square on the grid represents a correlation measurement between two possible numbers. The blackened squares present a meaningless measurement since each event correlates with itself automatically. This two dimensional grid requires a wide physical area and is not suitable for the complex examples presented in Chapter 5. Strict correlation can be measured here since the presence of one number and the absence of another can be noted.

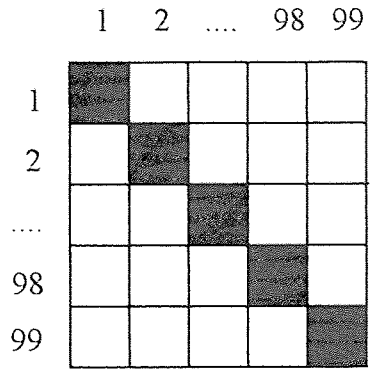



**Figure 6.1** Two Dimensional Processor Solution

A method called autocorrelation is used to find patterns in numbers similar to the matched filter from Chapter 4. Figure 6.2 shows the pairing of the receiver to the processor. The receiver has currently detected three numbers represented by vertical lines. When it is time to process, these lines become pulses traveling to the right out of the receiver and into the processor, along the same slow material without interruption.

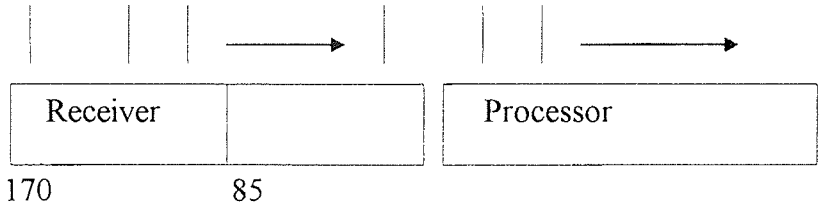



**Figure 6.2** Receiver to Processor Transfer

These pulses represent detected signals and are not to be confused with the receiver input pulses from Chapter 4. Once in the processor section, the pulse train continues to flow, disappearing off the right edge. The processor section consists of a long term memory material marking input patterns the same way the matched filter

marked intended signals in the receiver. The long term memory also serves as a matched filter not to detect incoming pulse widths, but incoming distance pattterns between detected signals. As the detected pulse train passes over the preset memory material, the matched filter will detect the pattern if one fits. The matched filter acts as an unfinished puzzle, detecting when the last remaining pieces can fit. Figure 6.3 demonstrates the matched filter in operation. There are two important observations to mention. First, the detected signal consists of three numbers, not just two. It could have been more or less than this. Second, the impressed pattern on the long term memory material is more than what is needed to match to the three propagating pulses. The three matching filter values are embedded within the group of memorized patterns. This memory material serves for all patterns recognized by the agent causing patterns to overlap.

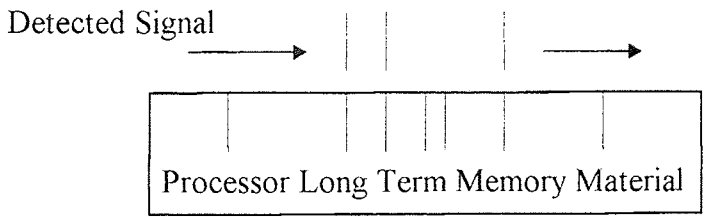



**Figure 6.3** Detected Signal Propagating over Long Term Memory

A drawback to this approach is the increased probability for a partial match between the detected signal and another pattern stored in the long term memory. Careful planing is required to render this method of processing accurate. A second drawback is the reliability upon distinct and exclusive number spacings in order to avoid {90,120} and {135,165} from matching in the same place. An agent receiving several numbers will end up with many short spacings between pairs of numbers. The chances of these number

spacings duplicating increases to the point where it can not be ignored. This may require a double constraint upon set transmission stating that multiple inputs should have noninterfering relative spacings as well as separate numbers. This would be impossible to expect since numbers spacings are between all input sets. Fortunately, this is not a problem due to the output scheme to be introduced in Chapter 7. Duplicate spacings between sets of three numbers or more are less likely and can be handled by the output algorithm.

The final step is to present the match as output. For now, the responsibility of the prcessor is to match each independent pattern at an independent position along the long term memory material. Chapter 7 discusses the means for accessing this position for broadcast. It is the job of the processor to indicate a match and its unique position.

These patterns within the Long Term Memory (LTM) do not come prepackaged and will require a learning algorithm to grow patterns within memory. Proper growth of patterns ensures proper detection. The reverse is also true. When serveral patterns have to be shared in a fixed length of LTM, the chance of interference between patterns increases. The length of the LTM can be increased in order to detect more patterns. This feature is sensible since the size of the design becomes proportional to the amount of processing power expected. The size of the two dimensional array must account for all possible number pairs possible, not for the just the minimum amount of matching patterns expected. Another approach to differentiating the two approaches is to view the two-dimensional array as a static implementation and the one-dimensional array as a dynamic one. Patterns grow dynamically in the LTM instead of in predetermined static positions.

### 6.1.2 Design Performance Goals

This subsection presents the goals[16] to be accomplished from the basic design presented in the previous subsection. Design decisions will be applied to these goals first since these goals ensure a foundation for proper operation. Any remaining goals introduced in Subsection 6.1.3 such as policies towards complex patterns will be evaluated with less importance. When attempting to satisfy all goals it will be discovered later that some compromises must be made. The two major design variables are the threshold calculation and the learning algorithm to apply to the matched filter.

Simplicity is the pervasive design goal of this thesis. The basic approach for the processing section matched filter is taken from the receiver section. The solutions for detection and learning in the processor should also be just as physically simple. It is tempting to abandon this goal in the presence of other goals demanding performance.

Only frequent coincidence of two or more number patterns are suitable for learning and detection. The occurrence of a single pulse should not be considered as meaningful.

Frequent patterns are expected to be recognized quickly. Quick learning is also expected when learning multiple patterns at once.

It is expected for patterns to compete with each other such that the resulting effect is the memorization of the most frequent patterns over those less frequent. Pattens no longer in use should eventually fade away.

A pattern already established should not prevent another pattern from growing, if the second pattern is competitive in frequency. This occurs when mature patterns are

allowed to compete unfairly when detection alone causes an extra amount of competitive power.

A mature pattern in the LTM should be stable in the face of other growing patterns. There should not be a temporary absence of detection in a learning environment. A strong pattern just learned may be a candidate for unlearning.

A simple pattern should be represented by one unique position along the LTM. Multiple detection can form from two scenarios. The first being the the same pattern is growing in two separate places in the LTM. Each is not aware of the other. The second is the case where one slot in the LTM is too sensititive and will cause a pattern match with just one pulse matched. In this case, a receiver's detected set of three numbers wil cause three pulses to travel across the LTM. Each pulse will trigger a match as it passes over the sensitive slot in the LTM. The result is three detections for this and multiple detection of every other receiver output presented. Two approaches for accomplishing this goal is to either prevent a single slot from being over sensitive by modifying the learning algorithm, or to modify the detection criteria to accept matches of two pulses or more.

Two unrelated patterns should be represented by a unique matching position along the LTM. It seems natural for this goal to be obeyed since unrelated patterns cannot occupy the same position in the LTM..

The position of each matched pattern must not only have its own unique location along the LTM, but these patterns must also be equally distributed in order to ensure the least chance of the output set for this agent does not collide with the output sets of others. This random distribution will be the partial result of fairness along the LTM for a pattern

to be learned. Improper design will create edge effects, resulting in pattterns gouping along the egdes, or the middle, the end, or the beginning of the LTM. Random distribution can also be denied if a new unrelated pattern chooses to form itself attached to an existing memory pattern. This causes unwanted grouping of patterns. Figure 6.4 describes this effect. Here, the middle slot of the new and growing pattern shares the left slot of an extablished pattern. The basic choice a new memory pattern has for forming is to either grow at a random position or to grow in a place that advances its ability to grow and be established quickly. Even if this grouping occurs, the output algorithm can separate patterns that partially and fully overlap. The general theme is to prevent grouping of patterns to provide a uniform distribution.


**Figure 6.4** A New Pattern Choosing to Grow from an Existing Pattern

### 6.1.3 Design Performance Issues

This Subsection is distinct from the previous in the uncertainty of either the correctness or the importance of the goals presented here.

Patterns of two numbers should be detected as easily as patterns of three numbers or more. This goal conflicts with the goal of having having only one number being meaningless. There will need to be extra rules and extra physical design complexity in place to account for this irregularity. In contrast, the strength of a pattern can be

porportional to its size. The problem with this is that a 20 pulse pattern from the receiver may match poorly on the LTM, but still command strength to detected.

Subpatterns of a larger pattern may grow within the parent pattern or as a duplicate unattached. Growing a subpattern within a main pattern results in the possible inability of the processor to distinguish the difference between when a subpattern is presented alone or within the parent pattern. This may be possible if the detection algorithm is clever enough. The alternative is growing the subpattern in a separate section of the LTM. This requires extra overhead, and is not considered an efficient use of the LTM. Common patterns and subpatterns should grow attached to one another and share common slots for sections of the pattern that is shared. Multiple patterns with shared numbers should have these numbers sharing the same slots in the LTM. Since these patterns are at equal level and one is not inside the other, detection should be straight-forward. As two patterns share more and more of each other, it will become increasingly difficult to detect the two separately.

### 6.1.4 Defining the Detection Algorithm

Determining how to recognize a match from the received pattern to a pattern in the LTM is critical in order to correctly identify pattern matches when present and to avoid false detections. Discerning detection alglorithms can go farther by detetecting separate subpatterns withing patterns if desirable. Below are some methods available for detection, all of which share some basic traits. As the received pattern passes over the LTM, if a pulse from the receiver passes over a sensitive slot of the LTM, this slot generates a

potential, this potential combined with other potentials across the LTM to detect a match. The suggestions presented here are in agreement with simple physical design. More complex methods may be possible.

A second application of the detection method is to give a heuristic indication to the learning algorithm. This application of the algorithm even when the result is below the threshold helps the patterns growing in the LTM to mature by strengthening the correct slots. It would make poor sense to detect patterns according to one measurement and grow them according to another.

The limiting situation with using this design is when the receiver sends several pulses representing several numbers. These numbers will correspond to one or more patterns in the LTM. As the pulse train propagates, some of the pulses will unintentionally pass over sensitive slots, causing spurious readings. The detection algorithm decides a true match from these spurious readings. The reliance is on a fully matched pattern dominating over noise expected in pulse train propagation. Detection problems increase as the number or receiver output values increase in proportion to the size of the patterns stored in the LTM. This will increase the noise element over the intended potential.

The first method is to total all of the potentials across the LTM at any instant. If the instantaneous potential at this point is greater than a given threshold, then mark the detection of a valid pattern at this point in time. This gives preference to more pules in the receiver pulse train than less. Large receiver input to the processor may cause false detection due to the addition of many small numbers. This method can differentiate between parent patterns and subpatterns by rejecting subpatterns since there are not

enough slots matching to generate a cumulative result over the threshold. The main difficulty is the wide ranges of output values due to the number of pulses propagating and the number matching. The resultant potential of a detected pattern spans too great to determine a standard threshold to apply in all situations. For this reason, detection by addition is not acceptable.

Another detection method is to average all of the slot potentials of LTM slots currently projected upon by the receiver pulse train. Consequently, it has the opposite effect of being unfair to receiver pulse trains of several pulses. The receiver output may contain several unrelated patterns to be detected by the LTM in different areas of the LTM. At a given moment, only one of those patterns will be matched. If the number of pulses in the receiver pulse train is 8 pulses, representing 8 numbers, and the current matched pattern uses only two of these numbers while the other 6 remain in slots that may or may not be sensitive. The worst case average will be (0+0+0+0+0+0+1+1) /8 = 0.25. This result will be too low even for the most lenient threshold. The best case will be when all receiver pulses match to a pattern in the LTM.

Another shortcoming of the averaging method is the failure to prevent a receiver output of only one pulse from triggering when matched with a sensitive LTM slot. This is possible since the average will be just the value of the single pulse.

A third detection method is to first calculate the average of all the slot potentials, then calculate the total deviation from this average. The lower the deviation, the stronger the match. This method is presented for variety only and lends itself to the same troubles as the averaging method.

In conclusion, no current detection algorithm will guarantee acceptable results. The averaging method is chosen over the addition algorithm since its detection threshold is limited to values between zero and one. As mentioned above, the detection result is used a a heuristic for the learning algorithm. It follows that poor detection will lead to poor pattern growth in the LTM.

### 6.1.5 Defining the Learning Algorithm

The learing algorithm will be presented relative to the detection algorithm. The amount of learning to be applied at any instant will depend on the detection value given at that instant The goal of the learning algorithm is to use the detection results wisely in order to grow patterns spread uniformly across the LTM. This Subsection presents positive and negative learning in order to keep the LTM in equalibrium. Methods using other heuristics than detection will be discussed last.

The positive algorithm grows patterns in the LTM. As the receiver's pulse train propagates over the LTM the detector calculates a value for each position. For a given position of the pulse train, the slots containing pulses will project into the LTM according to Equation 6.1.

$$\mathbf{P_n} = \mathbf{P_n} + \mathbf{D_t}(1 - \mathbf{e}^{\mathbf{P_n}})\mathbf{L} \qquad (6.1)$$

$$\mathbf{P_n} = \mathbf{P_n} + \mathbf{D_t}(\mathbf{e}^{\mathbf{P_n}})\mathbf{L} \qquad (6.2)$$

Where $\mathbf{P_n}$ is the potential of LTM slot **n** and $\mathbf{D_t}$ is the result of the detection algorithm at time **t**. Equation 6.1 ensures quick growth at the beginning, slowing as the slot potentials apporach the upper limit. A variation of 6.1 is 6.2 which speeds the apporach to the limit.

In this case $\mathbf{P_n}$ must be limited to a maximum value of zero. **L** is a learning constant designed to keep events from happening to quickly resulting in unstable pattern categories.

Figure B.3 shows the results of positive learning. The valid LTM detection area is between 100 and 200. The buffer zones on the left and right are used to help eliminate edge effects described later when positive and negative learning combine. The result is a steady increase in slot potentials across the band. This is to be expected since the propagating pulses pass over and potentiate all slots. All slots are adjusting to match the pattern without any competition. The least that can be done is to give an equal chance for the pattern to end up anywhere withing the 100 to 200 band. There is a bow across the band suggesting the pattern will definitely use a slot in the middle. Ideally, there should be no bow. This is due to edge effects.

Negative learning is needed to allow the incoming patterns to match only once. The idea is to let the winning pattern in the race suppress the rest. The winning pattern is the one which is detected as a valid pattern by the detection algorithm. Suppression is the result of the negative learning algorithm which accomplishes the opposite of the positive algorithm. A negative pulse train is sent across the LTM and any matches besides the winning match are suppressed. This allows only the winning pattern to thrive. The rest of the slots will only grow when adjusting to other patterns. The negative learning equation is shown below having strong relation to Equation 6.1:

$$\mathbf{P_n = P_n - D_t(1 - e^{P_n})S} \qquad (6.3)$$

The negative learning coefficient **S** determines the amount of punishment. Figure B.4 shows the result of the negative learning algorithm once a winner is selected. All of the

competing slots are reduced to a low state. Figure B.5 shows a second pattern being introduced. This second pattern is expected to climb to detection levels, but unfortunately, this is as high at the second pattern could get when combined with the first. This is a major setback in the design of the detection and learning algorithms.

A brief note is required about the edge effects. Since the detection algorithm depends on an average value of all pulses in the propagating pulse train, the most accurate calulation of the learning heuristic will be when all of these pulses are being accounted for. Two edge conditions occur when the pulse train is just entering the LTM area and when the pulse train is just leaving. At these times, the average is not as accurate as desired. To avoid this problem is to restart over and avoid the basic design in Subsection 6.1. The alternative becomes another goal for the detection algorithm. The algorithm must handle edge effects without distorting general trends of slot potentials away from or towards the edges. The region between 0 and 100 and the region between 200 and 300 are buffer zones where all slot potentials are a high negative value. These zones will not be discussed in detail since it is considered by the author to be a temporary solution. For details, see the source code of the applet in Appendix D.

## 6.2 Expected Results

This section describes expected results based on the criteria described in Subsections 6.1.2 and 6.1.3. The expected results will be compared with the actual results described in Section 6.3. The comparison will determine how well understood the issues and algorithms truly are. The minumum result is the ability to recognize at least one pattern

which has already been accomplished. Following this come expectations for recognizing multiple patterns. During these tests, it is expected that each unrelated pattern maps to a unique detection position along the LTM. The following are other performance expectations.

1. It is expected that only the most frequent pattern will be detected and will suppress all others.

2. The mature pattern is stable since it kills all competition. The pattern, however, will continue to grow to a condition where single pulses can be triggered at very sensitive slots.

3. The slot growth is not distributed completely evenly, resulting in the new pattern being formed about the center.

4. Subpatterns will be detected but identified the same as the parent pattern.

6. Shared patterns will be detected and identified with uniquely.

### 6.3 Actual Results

The actual results can be viewed in Appendix B. There are two basic graphs to be seen for each snapshot. The first graph describes the state of the LTM. Experimental parameters are also included such as learning coefficients *Plearn*, *Nlearn* for positive and negative learning. The coefficient *Ntr* is the negative learning cutoff threshold. During the negative learning cycle, any pattern matching below this threshold is not applicable to negative learning. Other information includes the current pattern being applied, the patterns in the current learning set, and the number of learning trials performed. The second line displays the position of the detection if one should occur. This line will occur farther down from where the pattern matched as explained in Chapter 7.

### 6.3.1 Unfair Competition

When the training set contains two unrelated patterns appearing separately but chosen randomly in the training phase, one will grow in the LTM and one will not. This behavior can be seen in Figures B.6 and B.7. Initially, the LTM slots begin in relatively sensitive positions as shown in Figure B.2 in order to generate a quick match. Figure B.6(a) shows pattern 1 being recognized after only 14 trials and represented by the slot on the lower axis. More or less trials are expected according to the setting of the positive learning coefficient *Plearn*. Pattern 0 does not become recognized until trial 66 in Figure B.7(b). Both patterns are set to appear an equal amount of times.

Figure B.7 shows the LTM attempting to learn three independent patterns presented separate and randomly. Figure B.7(a) shows pattern 2 being recognized after 17 trials. After 27 trials, pattern 0 is recognized but at the same position as pattern 2. This violates the performance constraint demanding a unique detection value for each unrelated pattern. Figure B.7(c) shows a continuing failure to detect pattern 1 even after 200 trials.

It is possible to learn multiple patterns when the learning process is modified into the following instructions. The first step is to present multiple patterns randomly until one is learned as shown in Figure B.9(a) where pattern 4 has been learned first. The second step is to stop presenting the patterns randomly and present any of the others not yet learned until it is also learned as shown in Figure B.9(b) where pattern 5 is finally learned. Since the first pattern has just been learned, it may not be as stable. Presenting both

patterns again randomly will allow both to mature as in Figures B.9(c) and (d). Multiple pattern learning is possible, however, requiring performance feedback.

The above examples have been involving at most 3 patterns. When more patterns are input, the LTM fails to detect a pattern in only one position, but in many simultaneous positions as in Figure B.11. This behavior makes the LTM useless at this level of difficulty.

It is expected that a more frequent pattern can be learned more quickly than one which appears less frequent. This the case given the current design. Figure B.8 shows pattern 1 set to appear twice as often as patterns 0 and 2 by specifying the patterns to randomly input as 0,1,1,2. As expected, pattern 1 is learned first. This is generally the case.

### 6.3.2 Pattern Stability and Growth

Pattern stability was demonstrated in the last subsection when it was shown how the LTM can learn multiple patterns if these are presented one at a time. The issue with pattern stability may be the trend to continue growing the pattern in the LTM until it is over sensitive. In Figure B.10 pattern 5 is allowed to grow with LTM sensitivity exceeding that of earlier figures. Since the firing threshold is determined by the average match across the LTM, these inflated slot sensitivities may cause cases of matching patterns where none exist. This case, however, was never seen yet. As mentioned, a single received number will cause a match in three places, but as mentioned in Chapter 5, single pulses are not allowed to match by themselves.

### 6.3.3 Uneven Distribution of Patterns

A general sign of uneven distribution of patterns is a bow in the LTM band between 100 and 200 in the figures presented in Appendix B. Figure B.6(b) is a good example of a bow in LTM band, while Figure B.7(c) shows no bow. Another factor determining pattern distribution along the LTM band is whether or not new patterns grow attached to existing patterns. According to the detection and learning algorithms, patterns will grow attached to existing or other growing patterns, resulting in poor pattern distribution.

This uneven distribution can be seen from the figures already studied, there is a trend for multiple patterns to be identified in clusters at the far right of the LTM band as in Figure B.7. Figure B.9 shows another cluster between 150 and 175 on the LTM band. A pattern has yet to be identified out of these regions.

### 6.3.4 Detection of Subpatterns

Due to the general design, received subpatterns will easily match with parent patterns in the LTM. It is expected that the subpattern will be identified the same as the parent pattern based on the averaging property of the detection algorithm. A received subpattern and a received parent pattern will produce the same detection threshold in the same position at the LTM parent pattern. This can be seen in Figure B.12 where pattern 6 is the parent pattern of pattern 5. Both patterns match at about position 155 along the LTM.

### 6.3.5 Detection of Shared Patterns

It is expected for shared patterns to share the common slots in the LTM. What makes each pattern unique is determined by the slot patterns to the left and right of the shared

slots. Each pattern should match in one unique place. These expectations were not realized as in Figure B13 where patterns 0 and 3 share common numbers. Here, pattern 0 is detected appropriately, but when pattern 3 is received, it matches the same as pattern 0 as well as in another place within the LTM. This is possible if the shared numbers of the two patterns do not share the same slots within the LTM.

## 6.4 Conclusions

Before any more testing can be performed on this design, some of the most basic pattern detection rules must first be obeyed. Most important, a received pattern should match in only one place within the LTM, not in two places or more. Secondly, two unrelated patterns should not match in the same place within the LTM. These two basic rules have not been observed. After this point, there needs to be a pattern growing policy which forces shared patterns to share slots in the LTM, and forces unrelated patterns to not share any slots in the LTM. The design approach presented in this chapter has potential and has proven adequate to detect a single pattern, but the detection and learning algorithm require another approach to meet the basic requirements to be an acceptable pattern matching process.

# CHAPTER 7

# TRANSMITTER DESIGN

The agent transmitter is the least important aspect of the design since it should not be considered difficult to convert set numbers into pulse widths. However, in the light of maintaining a simple analog design, building a transmitter can be challenging. This chapter is presented for the sake of completing the agent design. Special attention is allocated to output timing as well as providing output acceptable to the receiver input specifications proposed in Chapter 4.

## 7.1 Output Timing

In review, the receiver will gradually detect the intended numbers. Once it is determined all of the numbers have been received, the numbers are processed and the results are output in the form of a pulse ensemble. The questionable part of this chain of sequences is when to judge the receiver has obtained all intended input. This determination is the responsibility of the output section. There are two basic models to determine when to process. The first is to determine a threshold of sufficient input or correlation and the second is to continuously send output based on processor correlations at the moment.

Output threshold can be determined by the amount of information already present such as the number of input members, the number of input members multiplied by an importance factor, the amount of correlation, or the amount of correlation multiplied by an importance factor. The objective is to start sending output when it is determined all of the inputs have been detected by the receiver. If there is a sufficient amount of detected

signals or correlated patterns, the agent will output without waiting for more signals. The alternative is to begin processing when the change in receiver detection rate decreases below a threshold. As the last signal is received, the receiver stops potentaiting slots that haven't exceeded the receiver detection threshold. This will cause a drop in the detection rate, signaling the agent to process and output. A worst case scenario is when the input is continuously changing, causing the reciever to detect new numbers at a steady rate. In this case, the detection rate stays constant and the agent sends no output. The limitation to the severity of the case is the limitation of the set size to one octave. In the experiments, the octave is from 85 to 170. Eventually, all of the numbers will be detected leaving none left. This will cause a drop in the detection rate, firing the agent. The message the agent sends will be all possible pattern correlations. To avoid this problem, the detection time for an input must be faster than the rate of new information. The receiver requires more pulses per set member as the number of members per set increases. This translates to a general statement that messages of small sets are detected quicker than messages contained in large sets. Since all numbers are combined at the receiver input, multiple inputs containing multiple numbers are detected slower than a single input only.

The second policy for output is to continuously send information about whatever is correlating at the moment. This allows sufficient tracking of changing input but can not delete old input within a reasonable time. In order to keep up to date, as the neuron fires, it is desirable to clear out the short term memory of the receiver and start with only current signals. This function needs to be omitted when there is continuous output since the short term memory will be continuously cleared if the function is allowed. The

advantage of clearing the receiver is to purge old input set members instantly. Without this, old set members must age according to an exponential decay rate in equation 4.4. Before a previous set member can decay to a point below the detection threshold, it is still considered as a valid signal and may produce output. The result is recent continuous output based on input that is not always recent. Increasing the slot potential decay rate will ensure only the most recent input patterns will be transmitted.

## 7.2 Output Specifications

Most of the output specifications are determined by the receiver input specifications proposed in Chapter 4. Again, if more than one member is in the output set, the set members should be sent in non repeating orders to minimize spurious results at the receiver end. Also, if the output set contains several members, then these members should be repeated more often within the pulse ensemble than if only one member belonged to the set.

Chapter 5 mentions how the output circuit will be able to differentiate aliasing within the processor. An example of processor aliasing is the pattern {10, 33} and the pattern {24, 47}. Each has the same distance between members. The processor LTM will not differentiate between these two patterns since only the distance between numbers is used to determine correlation. The following section explains how this problem is naturally avoided.

## 7.3 A Model for Achieving Output

The output process must translate the processor results into a pulse ensemble wile obeying the above criteria. The same slow and fast materials can be used to perform this operation. Using the first method for determining output threshold, the general sequencing is as follows:

1. Receive input pulses. Wait until the output threshold is reached.

2. Send the first pulse while simultaneously starting the process sequence. The process sequence begins by projecting detected values from the receiver STM onto the slow material. This pulse train travels from the receiver towards the processor LTM. See Figure 6.2.

3. As the pulse train passes over the LTM, some patterns may match. When a match is detected, the agent fires another pulse to the output. This pulse position is the value given to the pattern stored in the LTM. Whenever the agent fires, send another train of pulses from the receiver as in step 2.

4. Currently, the first train of pulses is passing over the LTM while another train of pulses has just begun to propagate towards the LTM. See Figure 7.1. If the first train of pulses finds a second match shortly after the first, then fire another pulse out. This will create a third train of pulses propagating towards the LTM.

5. Repeat step 4 until it is determined that a sufficient number of pulses has been sent.

6. Clear the receiver STM for another cycle starting at step 1.

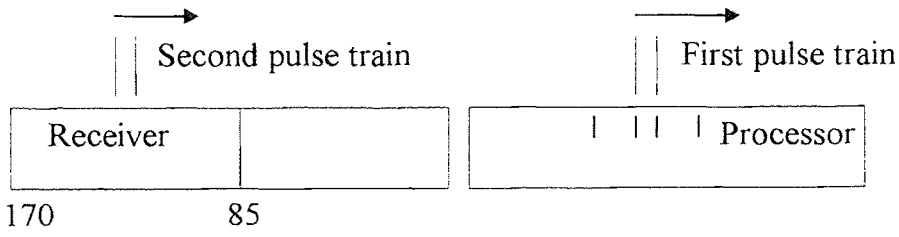



**Figure 7.1** Simultaneous Pulse Trains

These six steps outline the output cycle as well as how matches in the LTM are converted into pulse ensembles. If the detected signals from the receiver produce only one match, then the time spacing between output pulses will be the time required for the pulse train to travel across the slow material. If the receiver's detected signals match to the LTM in several places, then the output sequence will become more complicated. Shown in Figure 7.2, the second and third output pulses will be presented close in time relative to the first pulse. The output values represented by the first three pulses are the distances from the first to the second represented by a red line and from the first to the third represented by a blue line. The distance from the second to the third is only noise. Since the agent will send output twice in a short time span, two pulse trains will be propagating from the receiver closely together. Each pulse train will cause the agent to fire twice, resulting in four pulses fired for two pulse trains. Actualy, only three pulses are fired due to overlap. The next round will contain three pulse trains resulting in four pulses fired. If this pattern continues, then the output will saturate.

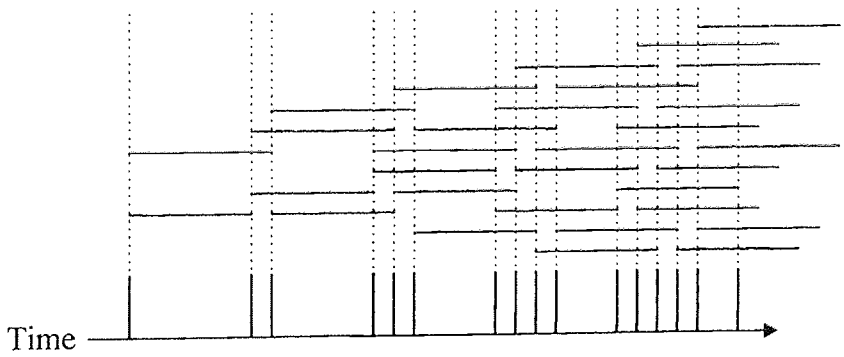



**Figure 7.2** Output Ensemble for Two Recognized Patterns

It can be seen that a single output signal can generate a substantial amount of noise due to the short output pulse spacings. As the output apporaches saturation, the noise will increase to intolerable levels. What is needed is a pulse rate limiter in the form of an exponential decay and a threshold. Figure 7.3 shows the decay from zero to a steady state value. When the agent fires, the limiter value is set to zero and the agent can not fire again until the value is above the permission threshold. The maximum firing rate is determined by a combination of the decay rate and the threshold value. The firing permission scheme will also help mix up the order in which multiple patterns are represented by the output pulse ensemble.

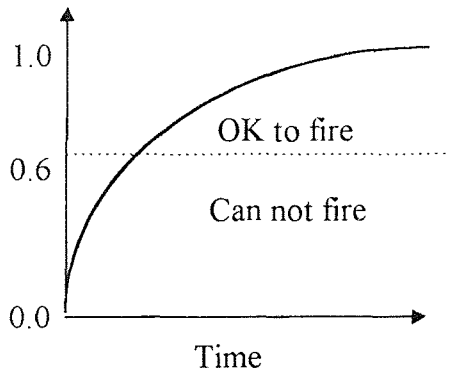



**Figure 7.3** Output Rate Limiting through Decay

Step 5 requires another value to to determine when enough pulses have been transmitted by the agent. As stated in Chapter 4, the number of total pulses sent should be proportional to the size of the set being sent. This can be achieved by using an integration function tied to the output limiting function. If a pulse is allowed to fire, then the current value in the limiting decay function is applied to ensemble stop integrator. If two or more patterns will be matched, then the decay value will be close to the threshold. This will

allow only small values to be added in the integrator. Once the integrator exceeds the stop threshold, the output cycle stops. The integrator is reset to zero until the output cycle begins.

### 7.3.1 Preventing Aliasing

The aliasing problem stated in section 7.2 is naturally bypassed since the two pairs of values detected by the receiver will match in with LTM at different times. The only way to match at the same time is to have both vaues the same. If the pair {10,33} is detected at 90 units of time from the receiver slots to the LTM slots, then the pair {24,47} will be detected in the same LTM slot but later at 90+14 units of time. As shown in Figure 7.1, the value assigned to the pattern match is the distance in time from the receiver slots to the matching LTM slots.

### 7.3.2 Staying within the Octave Boundary

The output set range is confined to the same octave boundary range as determined by the receiver. For example, if the receiver is designed to receive values from 85 to 170, the output values must also be within this range. According to the current output method, there is no guarantee of this. As mentioned before, the value of the match is the time required for the pattern to travel from the receiver slots to the pattern slots in the LTM. This timing is determined by both the receiver slot position and the LTM slot position. Since both can vary by the amount of 85 time units, the total span is actually 170 and not the 85 required by the receiver in the next stage. Part of the solution is to fix the output rate limiter to stop the agent from firing within 85 time units. The other part of the

solution is to place the LTM closer to the receiver. Figure 8.3 shows the processor far away for the receiver for the sake of simplicity. Actually, the processor must be contained within the first half of the receiver where values 0 to 84 would be detected. There would be nothing to the right of the fast material. This solution would require solving problems concerning edge effects within the processor learning algorithm.

# CHAPTER 8

# PHYSICAL DESIGN APPROACH

## 8.1 Materials

The main materials include a fast material, slow material, short term memory material, long term memory material, negating material, resistive material, and an averaging material. The speed at which pulses travel in the slow material is several times slower than that of the fast material.

$$V_f >> V_s \tag{8.1}$$

Both the fast and slow material require proper terminations at the ends to prevent pulses from reflecting backwards. When the pulse reaches the far end of the material, the energy stored within the pulse is dissapated within the terminator. The value of the terminator must matc the impedance of the materials. Consider these materials as transmission lines.

Both memory materials are long, thin slices able to maintain analog potentials along their lengh. Consider the memory materials similar to magnetic tape found in an audio cassette. The long term memory material (LTM) is static in nature with no memory decay, while the short term memory material (STM) is dynamic in nature with decay life determined by Equations 4.4 and 4.5. The STM material is potentiated when a pulse from the fast material intersects a pulse on the slow material. The amount of potentiation is determined by equations 4.1. For LTM, just the presence of pulses in the slow material causes potentiation or depotentiation according to 6.1, 6.2, 6.5. STM will be equipped with extra normalization circuitry causing depotentiation as a function of the normalization

signal. This is represented in the equation below where $\mathbf{P_n}$ is the potentiation of the STM and $\mathbf{V_n}$ is the normalization input at position **n**.

$$\mathbf{P_n} = \mathbf{P_n} - \mathbf{e}^{\mathbf{V_n}} \tag{8.2}$$

There exists an ability for the STM to project upon the slow material when commanded to do so by logic circuitry. This case will be when it has been decided to process the numbers received by the receiver. All potentials in STM above the detection threshold will be projected onto the slow material. From here the pulses can travel in both directions. The direction away from the processor will lead to a terminating resistor. The correct direction will be towards the processor.

Another special material called a negating material is needed to project negative pulses upon the slow material where positive pulses currently exist. This material performs the negative learning cycle of the LTM where negative pulses of the matched pattern are propagated along the slow material to diminish any similar patterns in the LTM. The composition of this material is unknown and becomes the first major obstacle in completing the design of the agent. Descrete components may be needed in place of continuous materials with distributed properties.

The resistive material is used for normalization of the STM in the receiver design and can come in two types depending on the type of normalization required. If an unbiased neighbor competition is desired as in 4.1.2.2, then an attenuating resistive material is required as shown in Figure 8.1. In this case, the STM draws no current from the resistive material when normalizing. When a slot in the STM becomes potentiated, a

pulse is sent to the resistive material to depotentiate its neighbors. The voltage along the attenuating material dissapates exponentially with length.

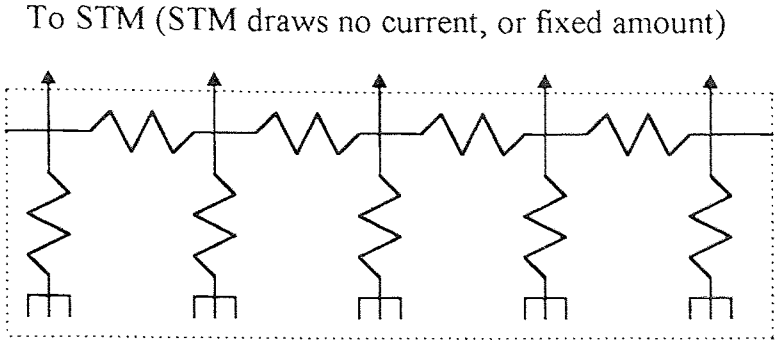



**Figure 8.1** Electrical Equivalent of the Attenuating Material

To perform the biased competition in 4.1.2.3, a resistive material is required. This material is similar to what is shown in Figure 8.1 but without the shunting resistors. The purpose of the shunting resistors was to draw a fixed amount of current to ground per unit length. This was fair and required the STM to also draw no current or a fixed amount to maintain this fairness. Since unfair competition is used, the fairness constraint is no longer needed. This time, the STM should draw current exponentially porportional to the potential at any given slot along its length as shown in Equation 8.3.

$$\mathbf{I_n = V_n * e^{Pn}} \tag{8.3}$$

The current to be drawn is $\mathbf{I_n}$ and the present voltage on the resistive material is $\mathbf{V_n}$.

A averaging material is used in conjunction with the LTM in order to determine the detection value along the LTM during the processing cycle. If the averaging material is used to calculate the average, then it is similar to Figure 8.1 but this time without the lengthwise resistors. Figure 8.2 shows the necessary modifications. The attenuating, resistive, and averaging materials are all related and consist only of resistance. The major

difference with the averaging material is that it requires discrete resistive values while the other two only require distributed resistance along the material's length.

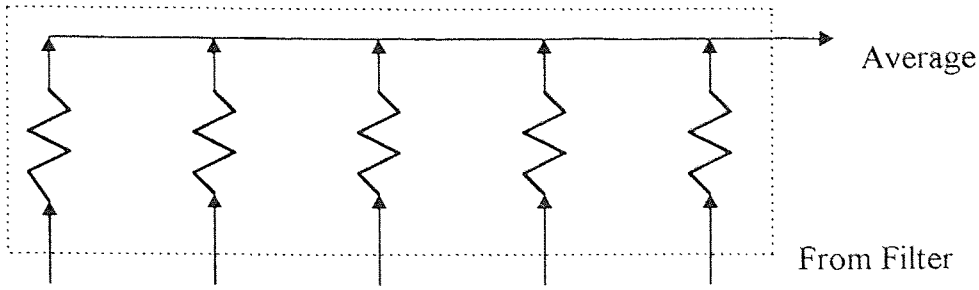



**Figure 8.2** Electrical Equivalent of the Averaging Material

## 8.2 Integrated Design

This section addresses the physical design of all components together suggesting a possible layout and providing a larger view. Some circuits such as the output circuit will require regular transistor circuitry and will not be described. Figure 8.3 shows the entire design of the agent.

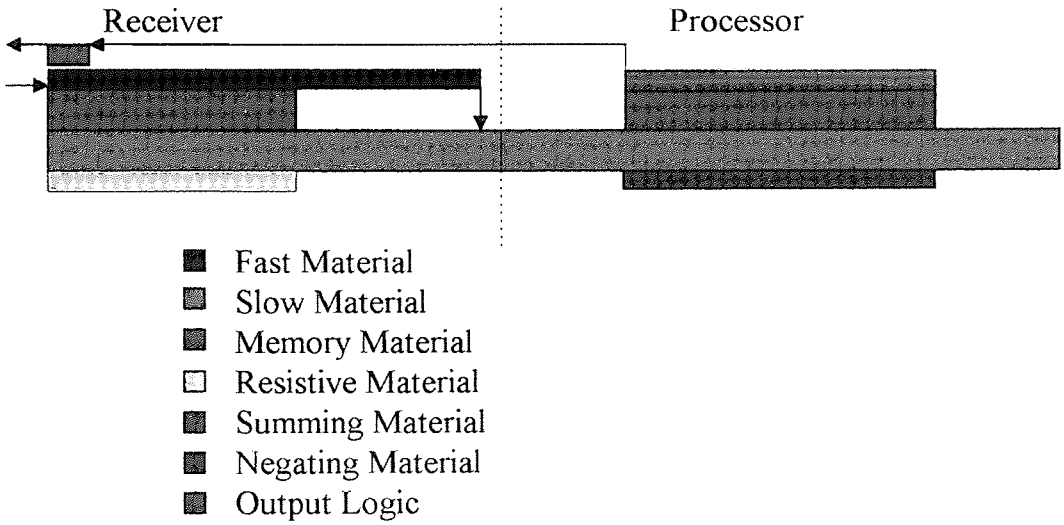



**Figure 8.3** Block Diagram of Agent Design

# CHAPTER 9

# CONCLUSIONS

## 9.1 Overall Design

The design of the agent in question was a process spanning three layers in a hierarchy. The top layer states a design using simple analog circuits. There is hardly any question there since simple analog circuits are desirable but may not be appropriate since evidence suggests the biological neuron may be a complex as as some entire artificial networks [4]. The second layer design decision was to employ communication through sets. This decision worked well for receiving but has undetermined suitability for processing. The major drawback for set communication is the inability to relay messages and cooperate with other agents in a flat society. The set processing presented here only correlates incoming sets from lower members in the hierarchy and passes results to higher members. This hierarchy can be flattened by using the regenerative properites of set processing agents. Regenerative agents may be able to actively participate at several layers within the hierarchy. The final design decision was to use the principle of propagating pulses along a slow material for all three stages. This is a wise choice for an analog design, but there are many details to be answered about the availability and feasibility of these materials.

What has been accomplished is the theoretical design for a simple agent using different materials to accomplish intelligent operations. It is an approach which should not be abandoned, but improved upon by either improving the design details of the current agent or to redesign the agent at one of the two lower layers of the design hierarchy described above.

## 9.2 Conclusions for Individual Components

Of the three components, the receiver performed the best. This may have been the result of the designing the language in terms of the receiver's performance, while ignoring the processor and the output stages. The use of sets, however, is still applicable to simple processing and output. The status of the processor section ranges from tweaking the detection and learning algorithms to considering an entirely different design approach. The key to solving the problems with the processor is with the detection method. The proper method will launch this agent to a stage where entire networks can be built and the behaviour of such networks obsevered. This is will determine the true value of this agent.

## 9.2 Future Work

Future work on this agent should almost exclusively go into refining the detection and learning algorithms of the processing unit. It is suggested to look beyond the simple slot values within the LTM for other information to help provide sharper selection of the intended pattern over the noise generated by the propagating receiver pulse train. Negative learning needs to target duplicate patterns accurately, leaving unralated patterns untouched. Both suggestions for improvements require a sharper detection of patterns requiring an improved detection algorithm. The design goals and approach given in Chapter 6 will be useful when considering future algorithms.

Other related work can be in finding ways to make combinations of materials behave intelligently with input, output, and processing stages. The intelligent behavior can range from pattern recognition to automata. Creative designs such as using slow and fast

materials extends the definition of analog computing from just using the basic adders, integrators, and multipliers[15]. Such materials will form the simplest computers, and maybe someday the most powerful.

# APPENDIX A

## RECEIVER SIMULATION RESULTS

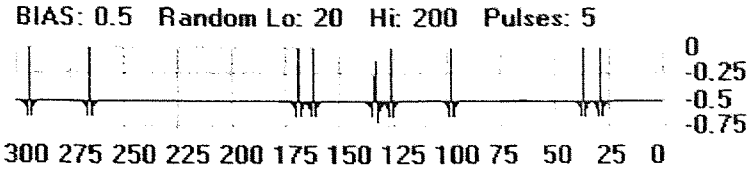
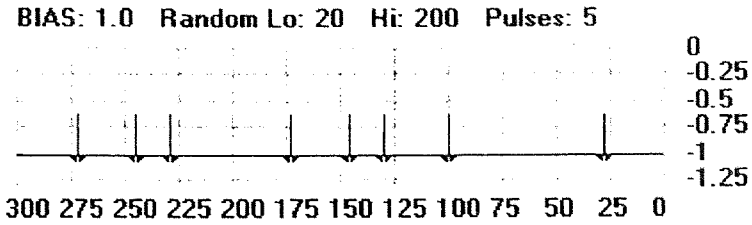
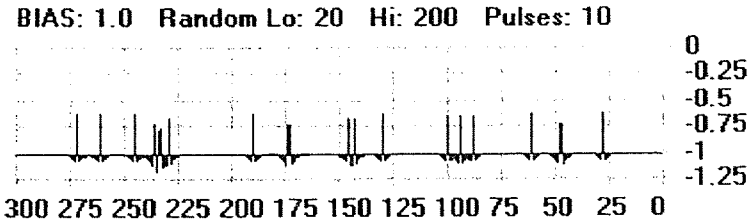


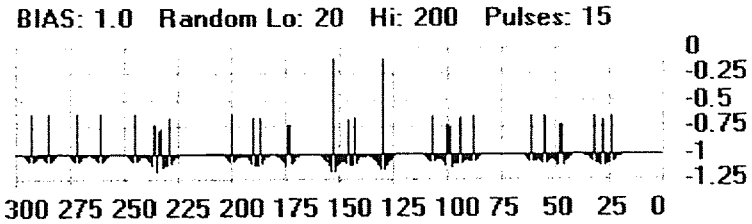

**Figure A.1** Receiver Output for Random Input, Bias = 0.5




(a)

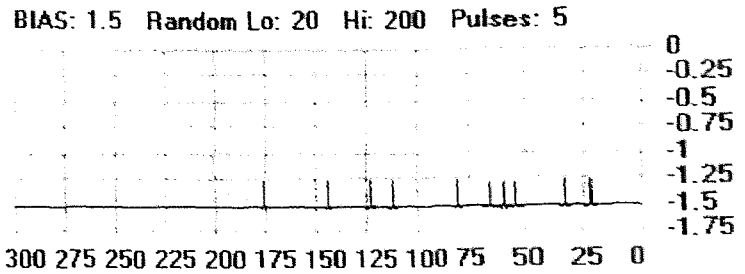



(b)

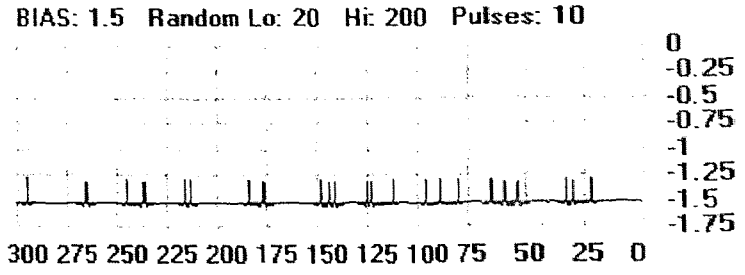



(c)

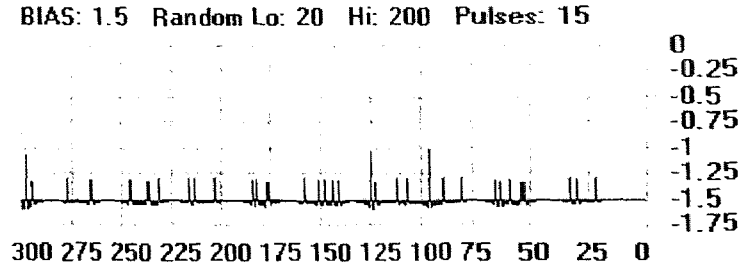
**Figure A.2** Receiver Output for Random Input, Bias = 1.0



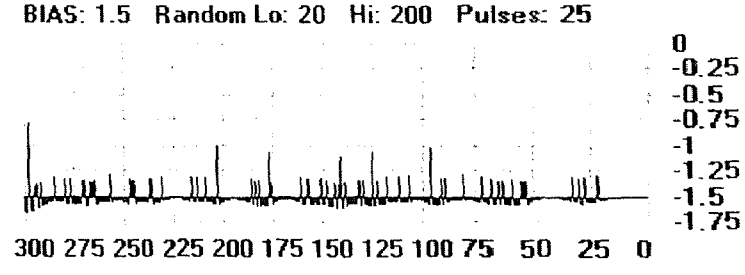

(a)




(b)

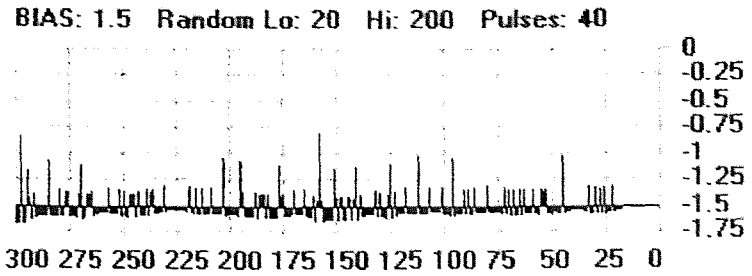



(c)

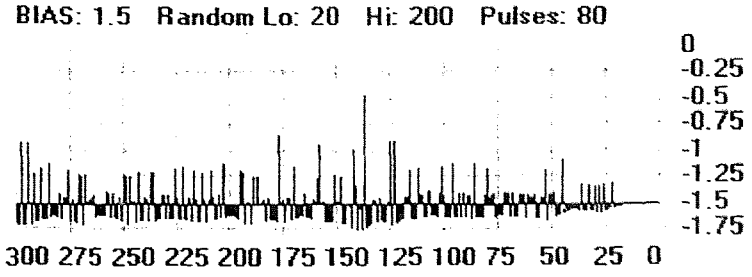



(d)

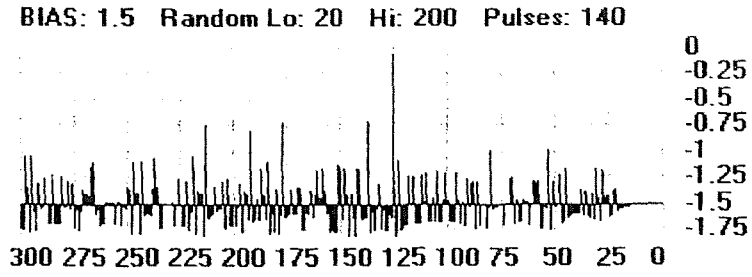
**Figure A.3** Receiver Output for Random Input, Bias = 1.5




(e)

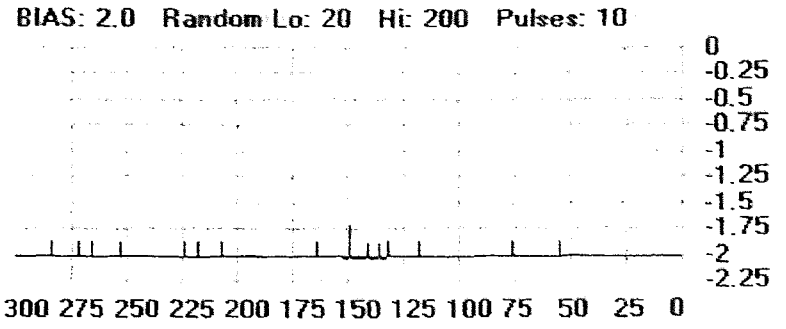



(f)

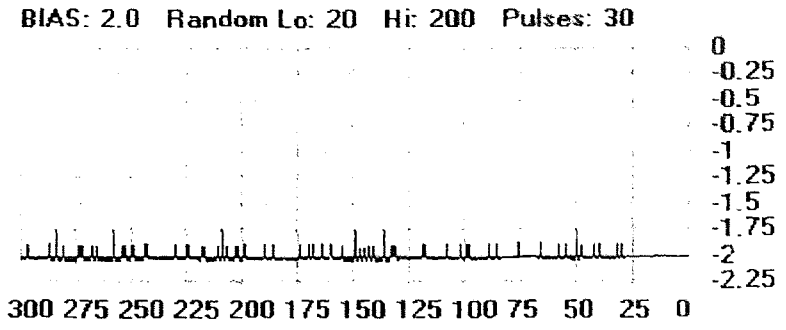



(g)

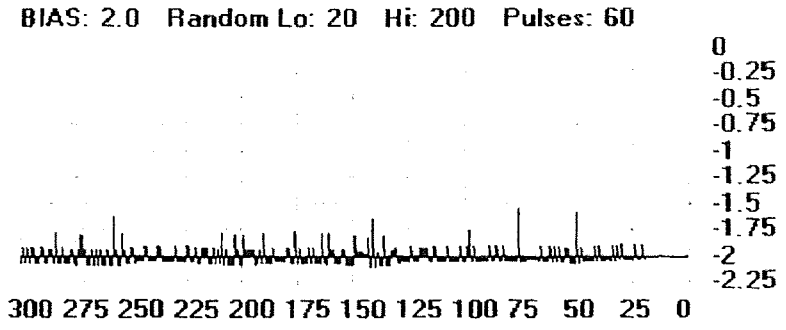
**Figure A.3 (Cont.)** Receiver Output for Random Input, Bias = 1.5



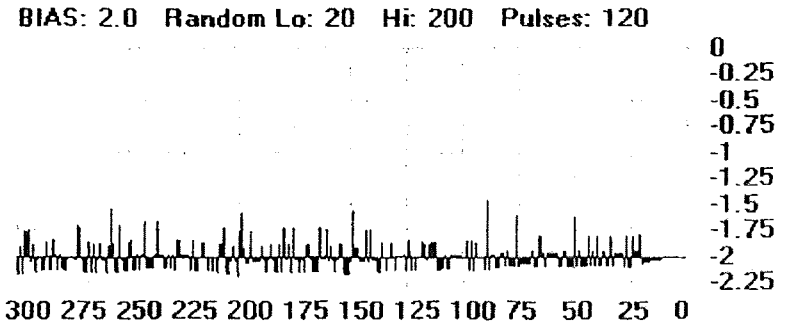

(a)




(b)

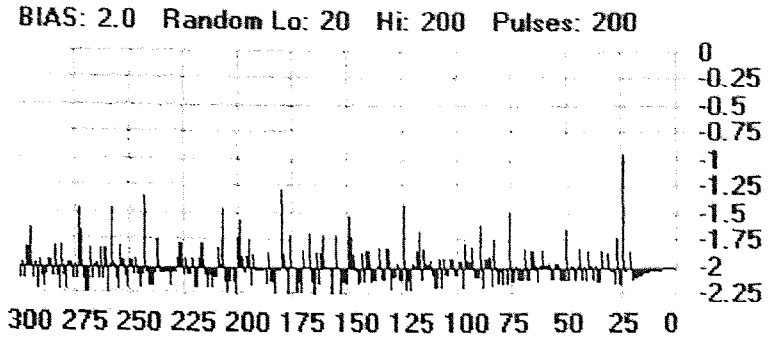



(c)

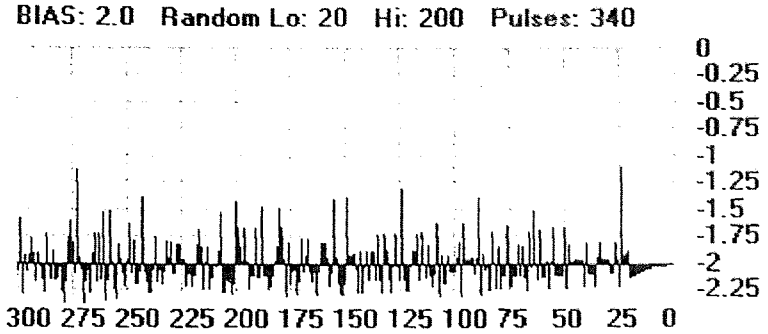



(d)

**Figure A.4** Receiver Output for Random Input, Bias = 2.0

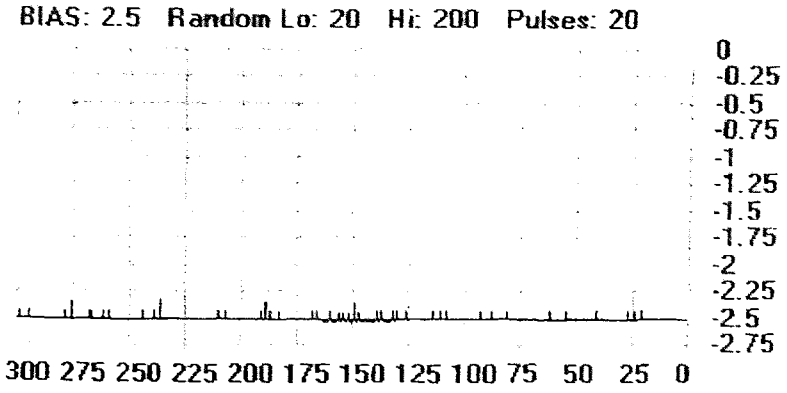



(e)

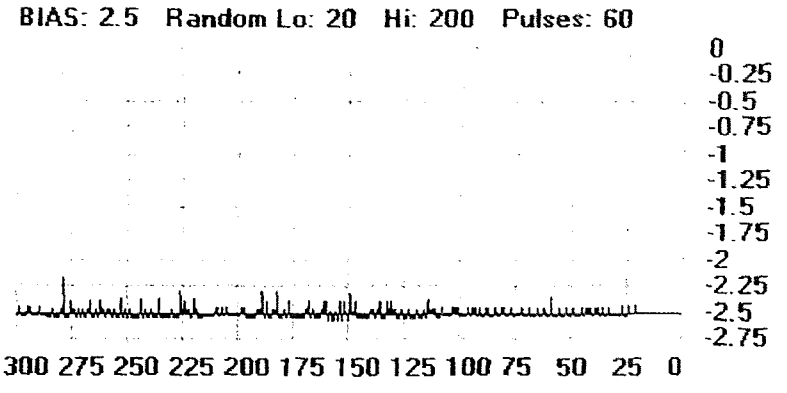



(f)

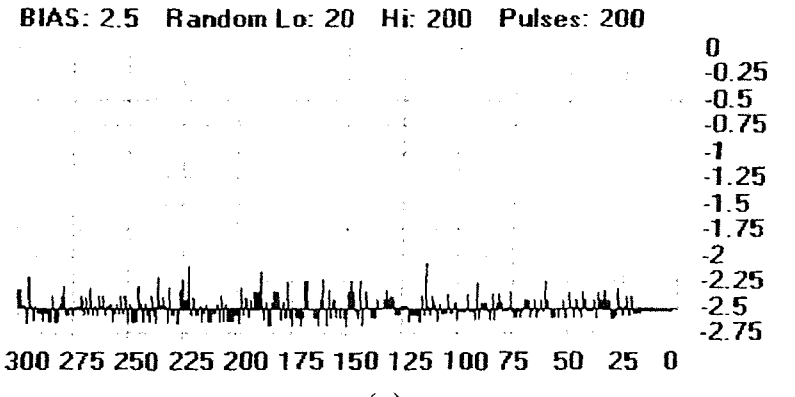
**Figure A.4 (Cont.)** Receiver Output for Random Input, Bias = 2.0




(a)

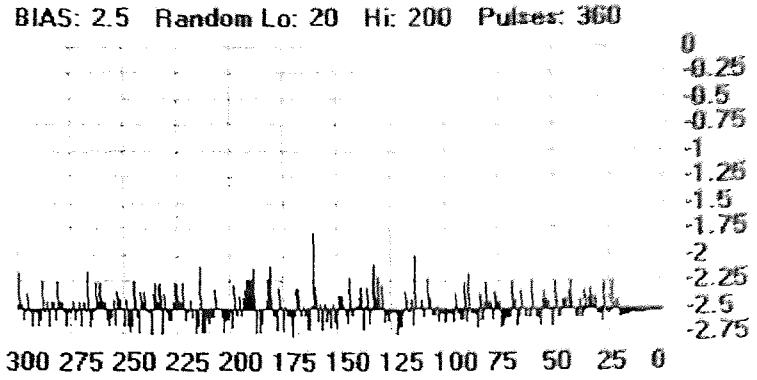



(b)

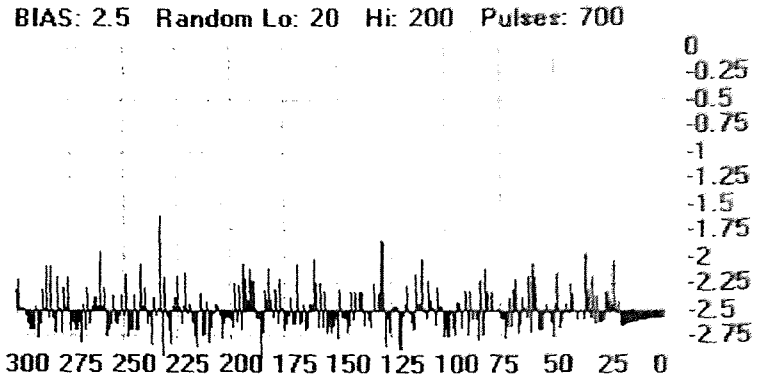



(c)

**Figure A.5** Receiver Output for Random Input, Bias = 2.5

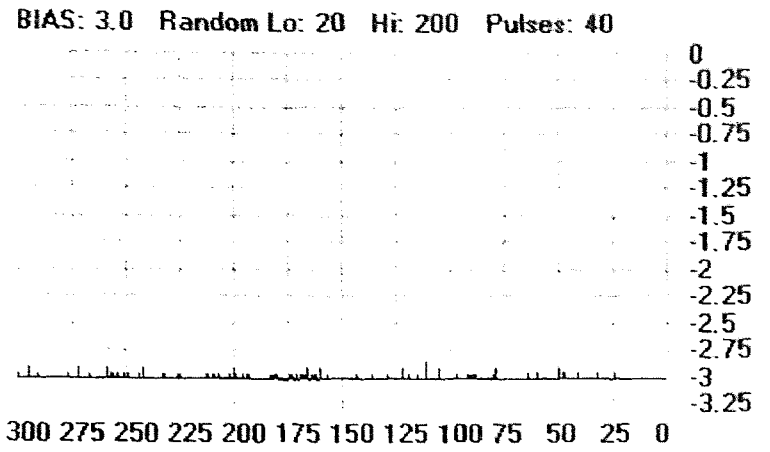



(d)

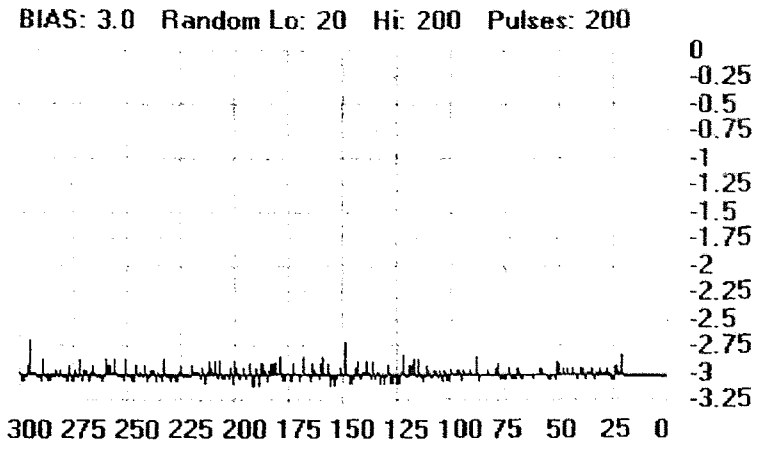



(e)

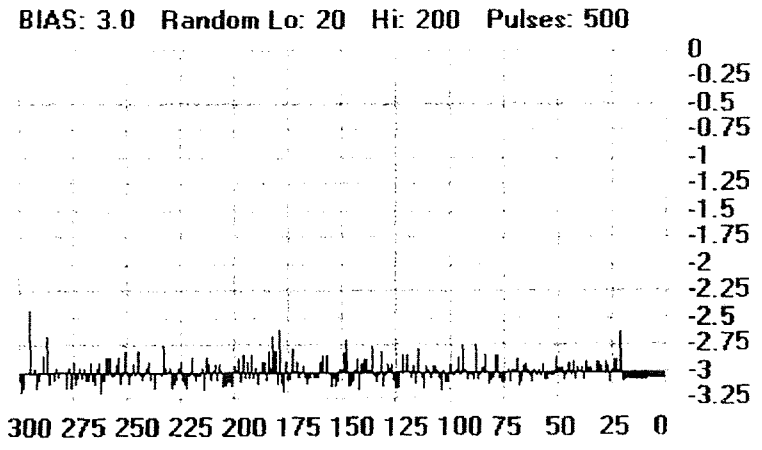
**Figure A.5 (Cont.)** Receiver Output for Random Input, Bias = 2.5




(a)

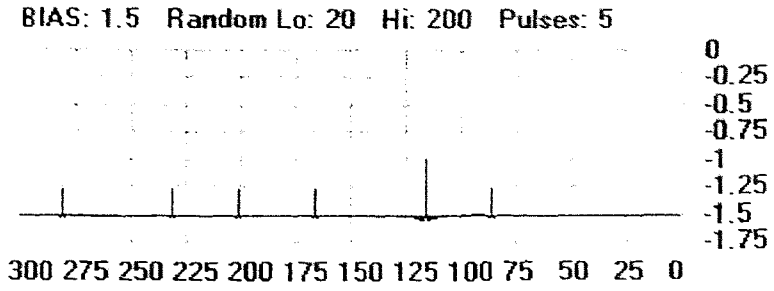



(b)

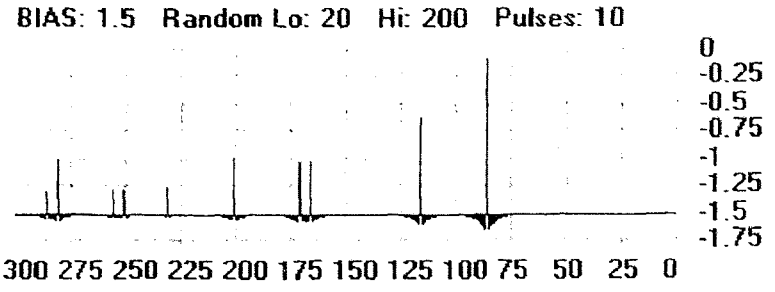



(c)

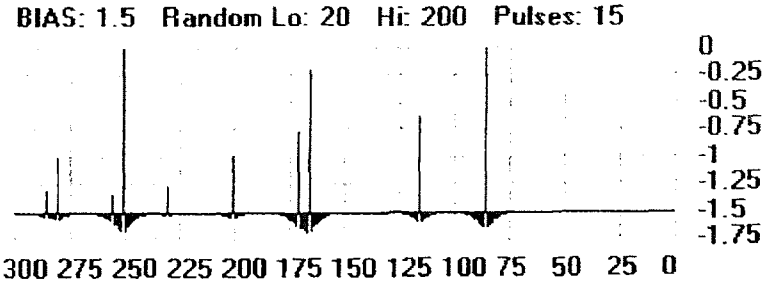
**Figure A.6** Receiver Output for Random Input, Bias = 3.0



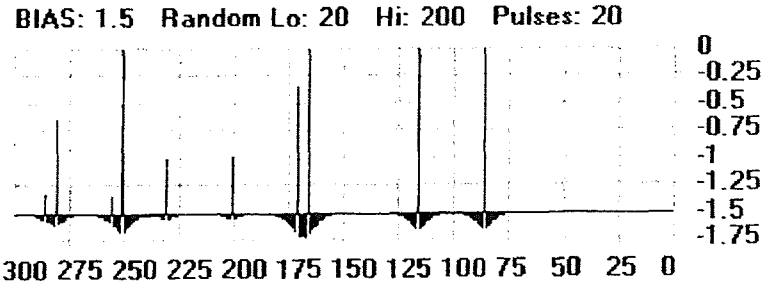

(a)




(b)

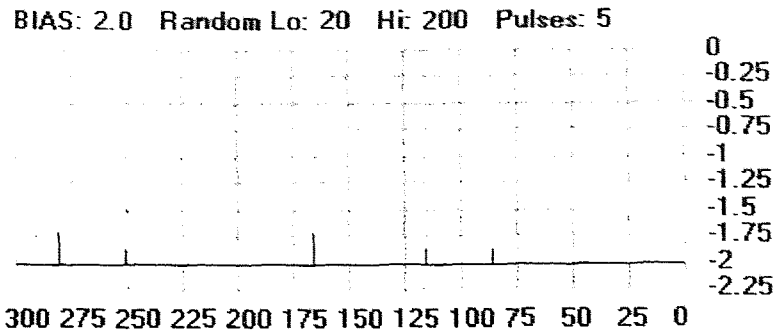



(c)

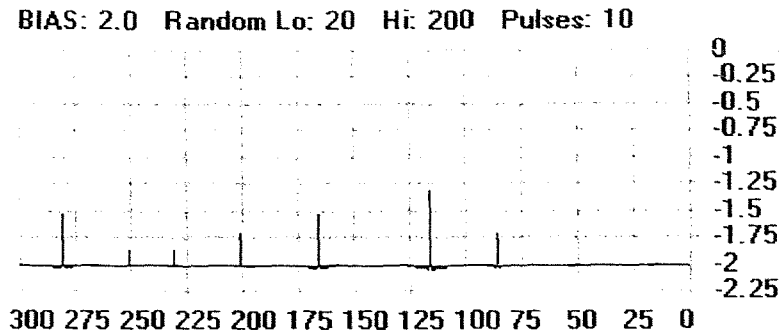



(d)

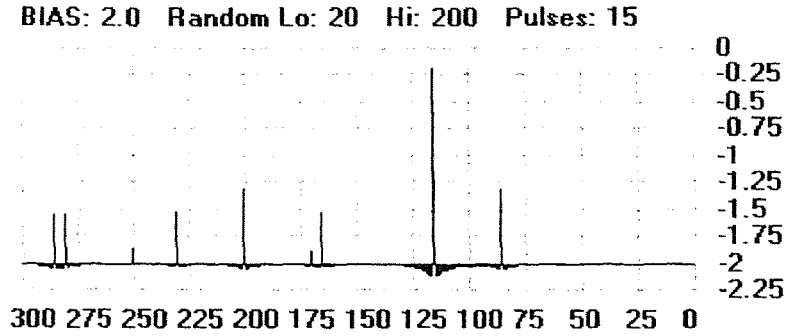
**Figure A.7** Receiver Output for 1 Set Input, Bias = 1.5




(a)

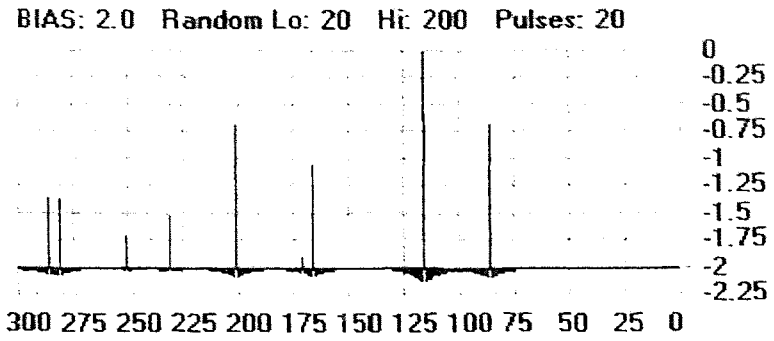



(b)

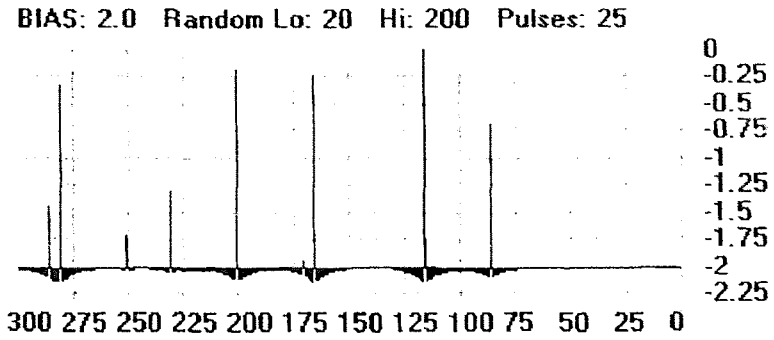



(c)

**Figure A.8** Receiver Output for 1 Set Input, Bias = 2.0

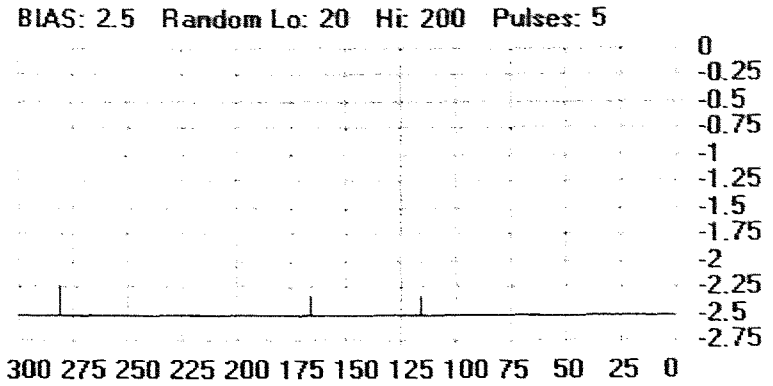



(d)

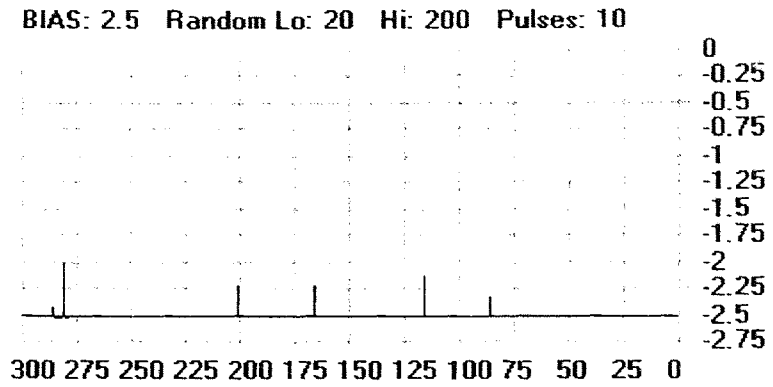



(e)

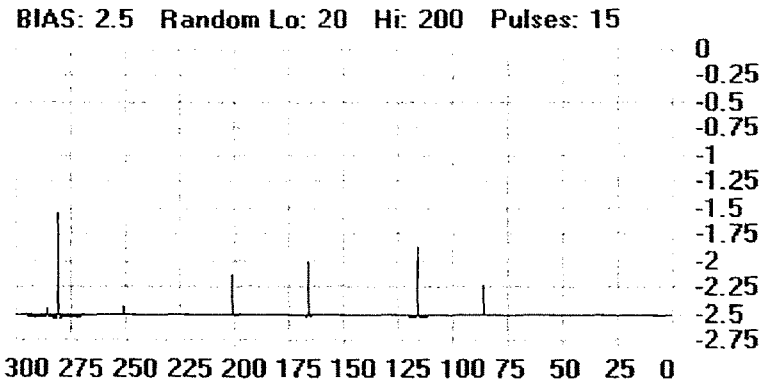
**Figure A.8 (Cont.)** Receiver Output for 1 Set Input, Bias = 2.0




(a)

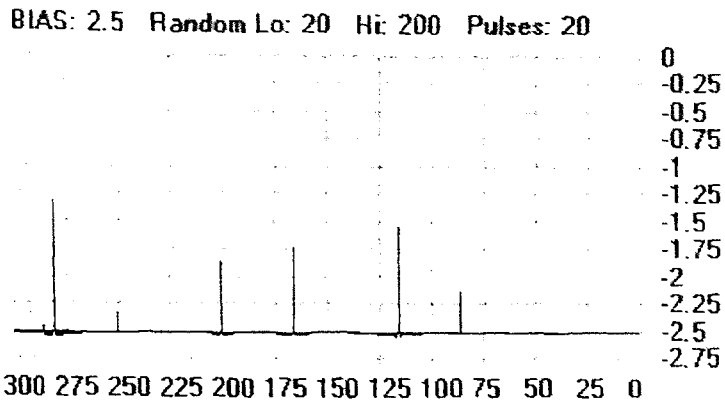



(b)

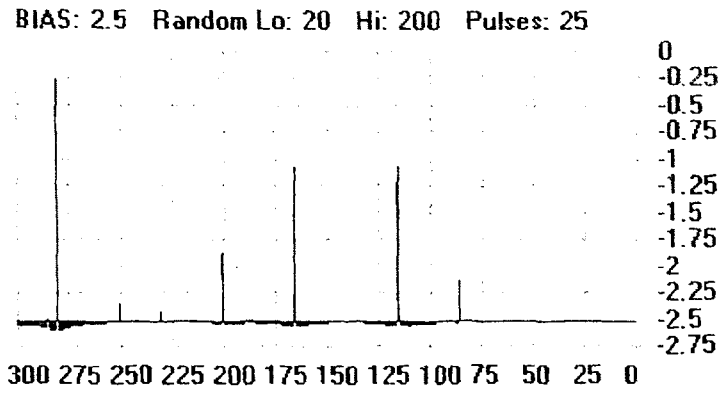



(c)

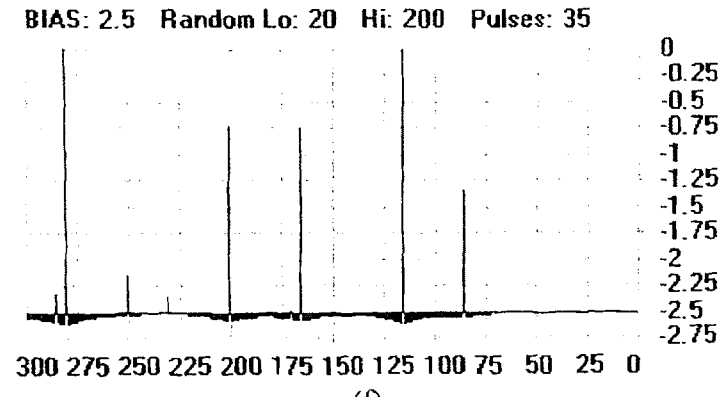
**Figure A.9** Receiver Output for 1 Set Input, Bias = 2.5




(d)

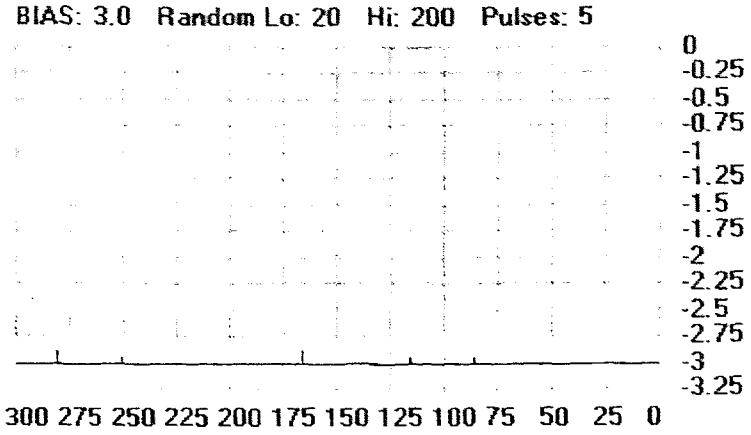



(e)

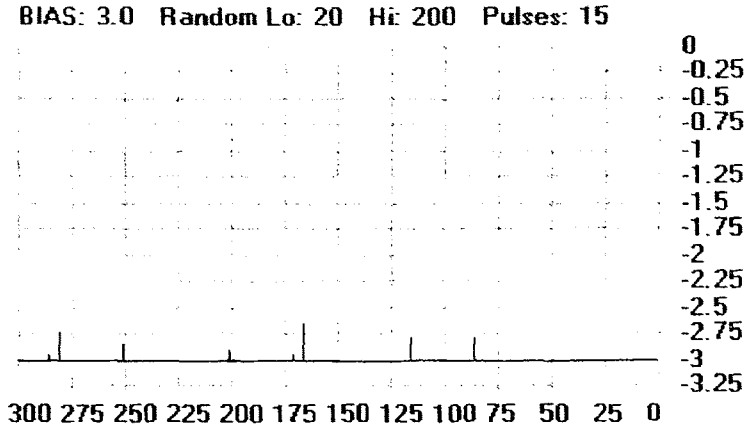



(f)

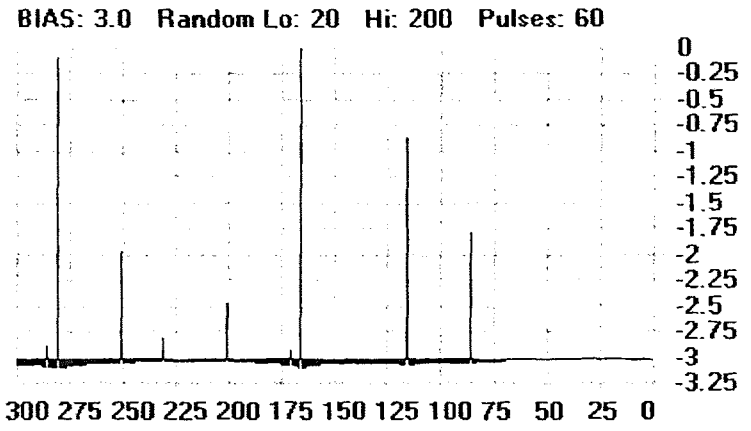
**Figure A.9 (Cont.)** Receiver Output for 1 Set Input, Bias = 2.5




(a)

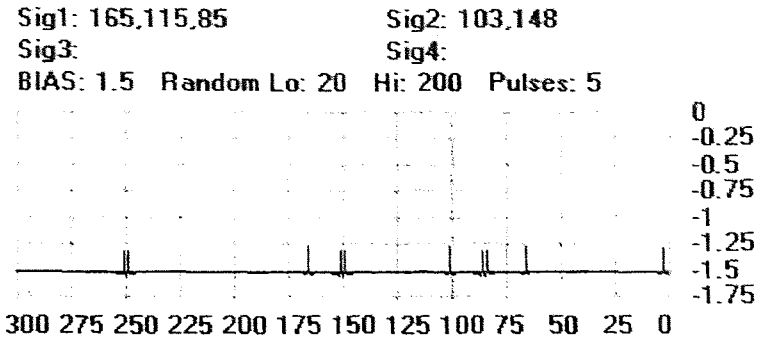



(b)

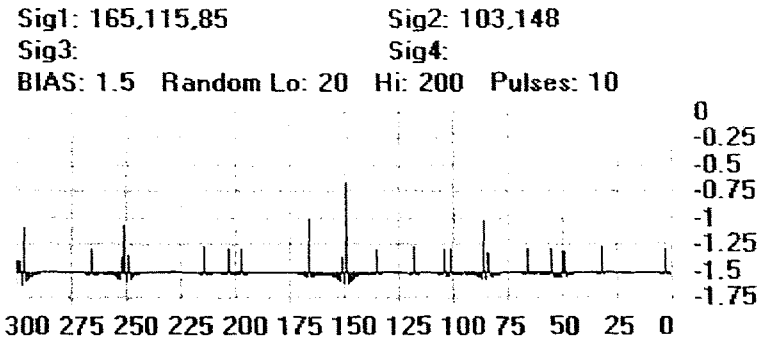



(c)

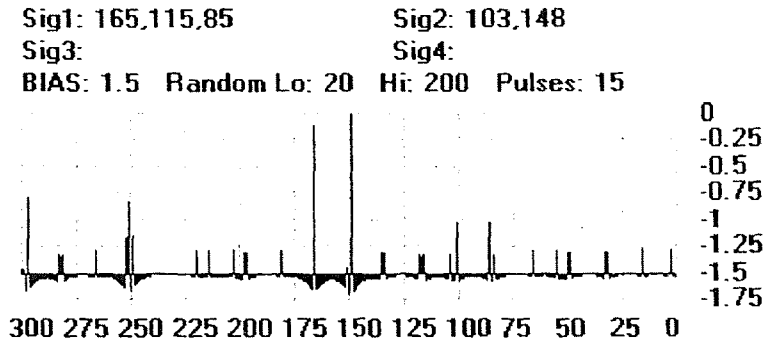
**Figure A.10** Receiver Output for 1 Set Input, Bias = 3.0




(a)

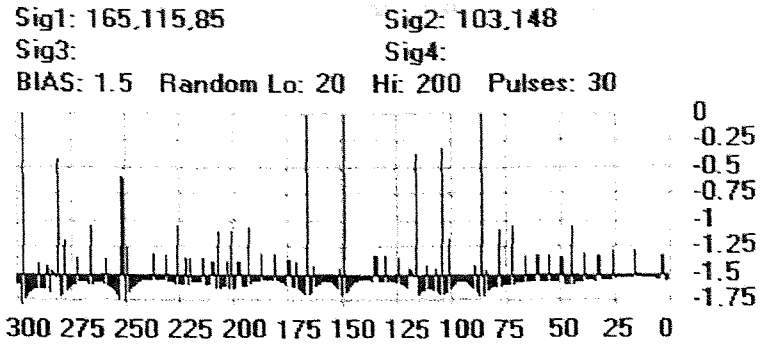



(b)

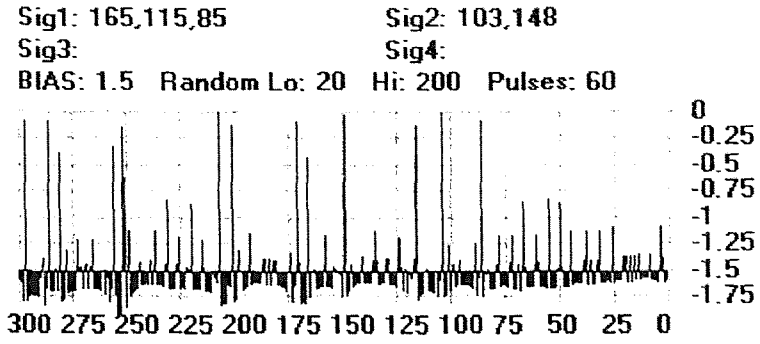



(c)

**Figure A.11** Receiver Output for 2 Sets Input, Bias = 1.5

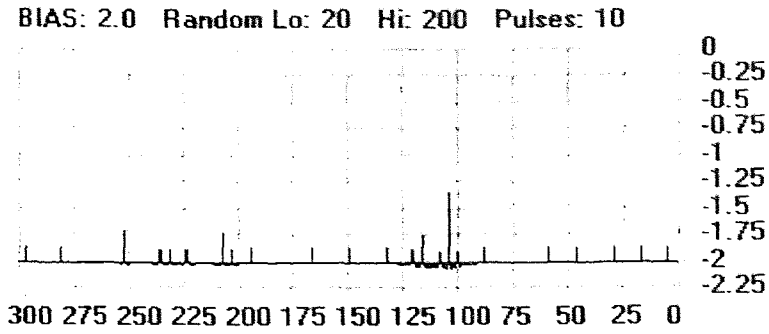



(d)

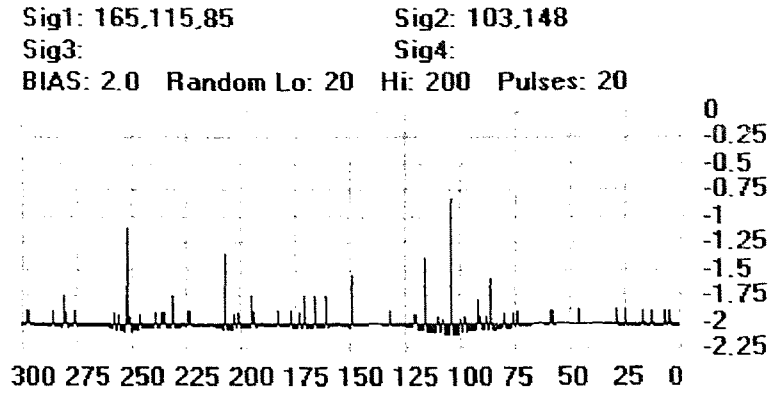



(e)

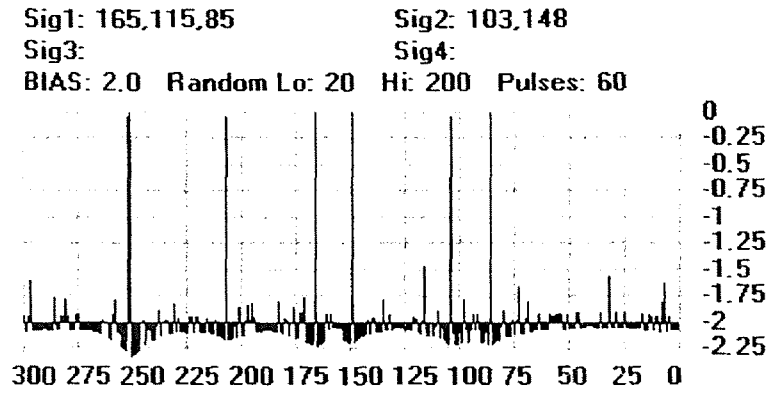
**Figure A.11 (Cont.)** Receiver Output for 2 Sets Input, Bias = 1.5




(a)

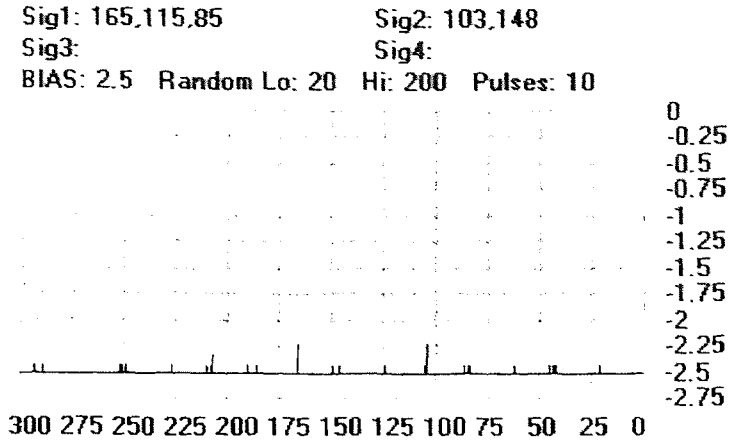



(b)

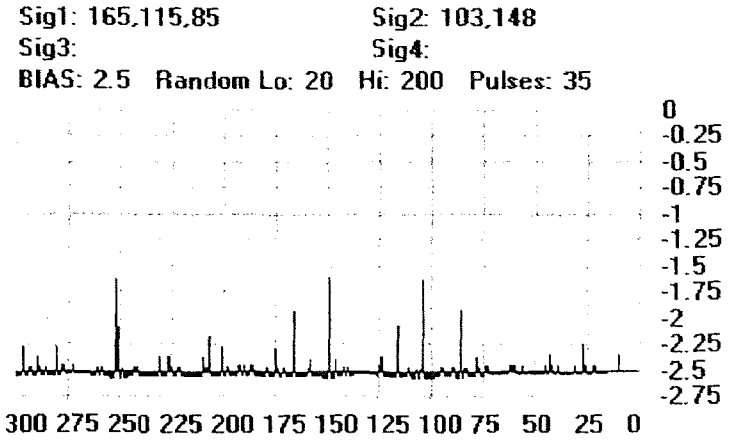



(c)

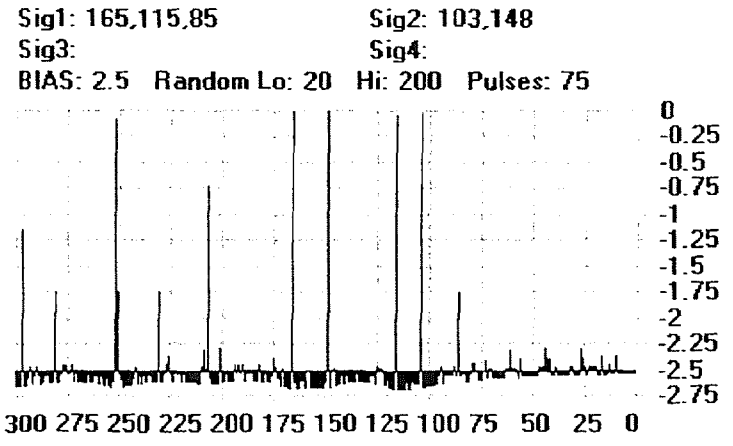
**Figure A.12** Receiver Output for 2 Sets Input, Bias = 2.0




(a)

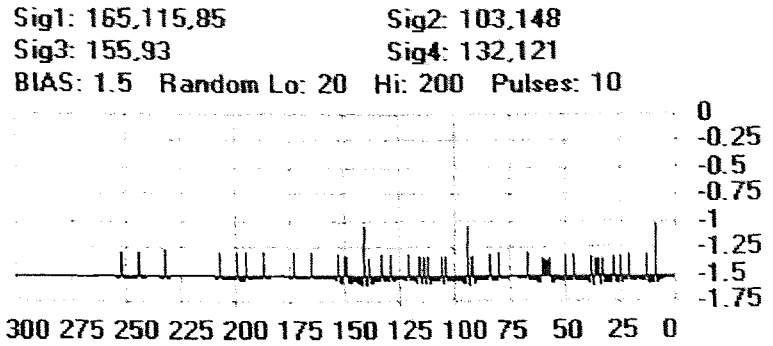



(b)

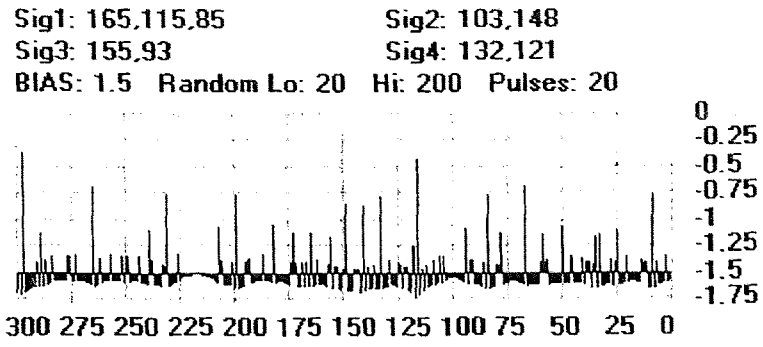



(c)

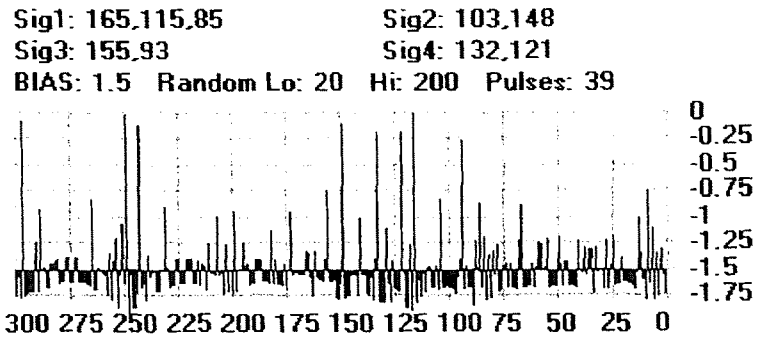
**Figure A.13** Receiver Output for 2 Sets Input, Bias = 2.5




(a)

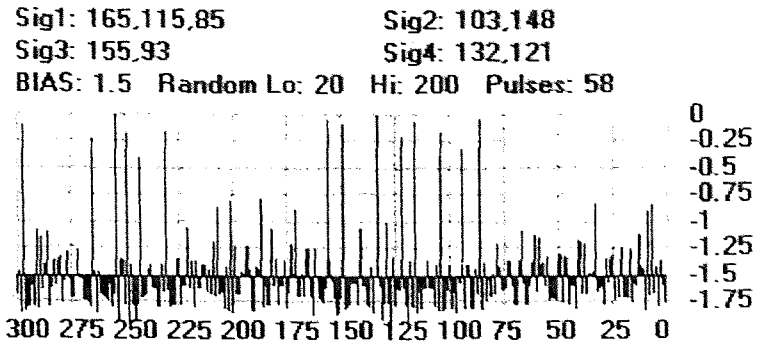



(b)

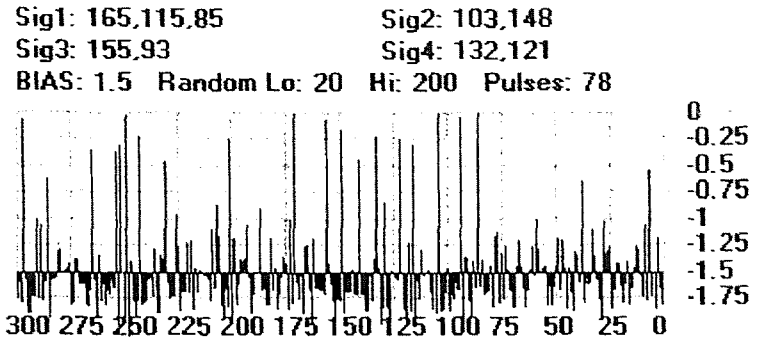



(c)

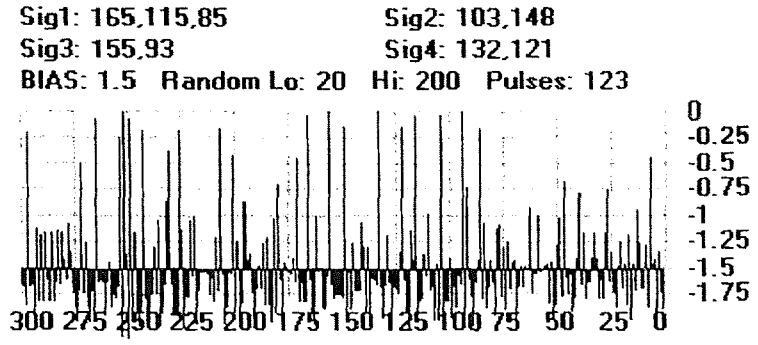
**Figure A.14** Receiver Output for 4 Sets Input, Bias = 1.5




(d)

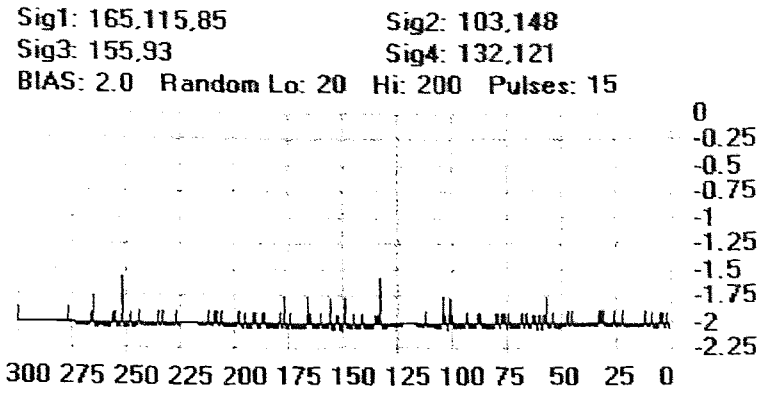



(e)

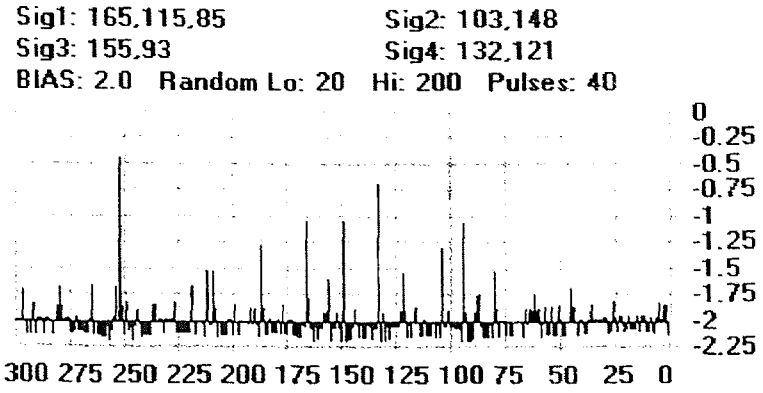



(f)

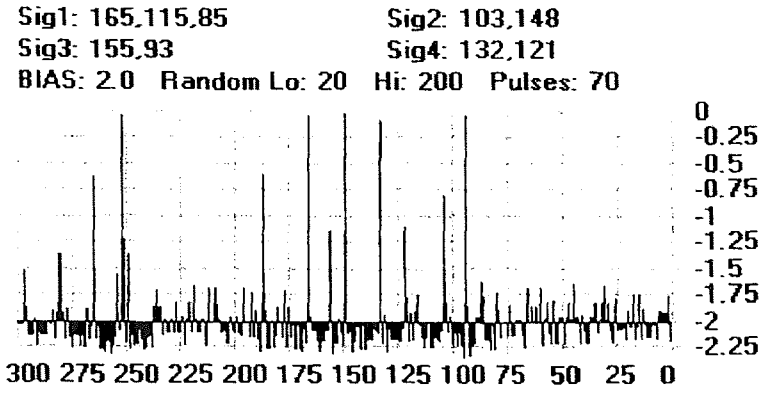
**Figure A.14 (Cont.)** Receiver Output for 4 Sets Input, Bias = 1.5




(a)

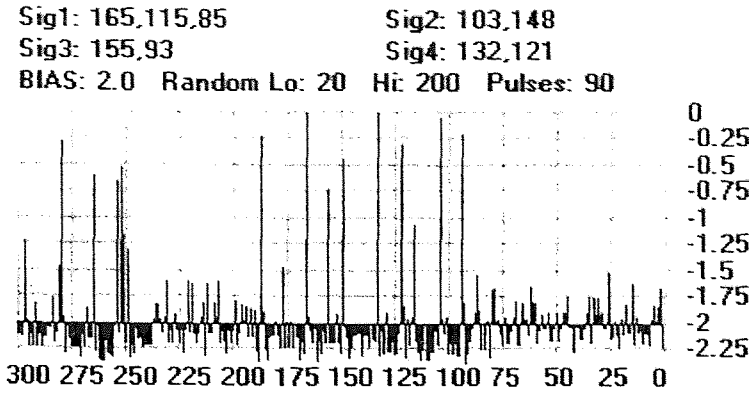



(b)

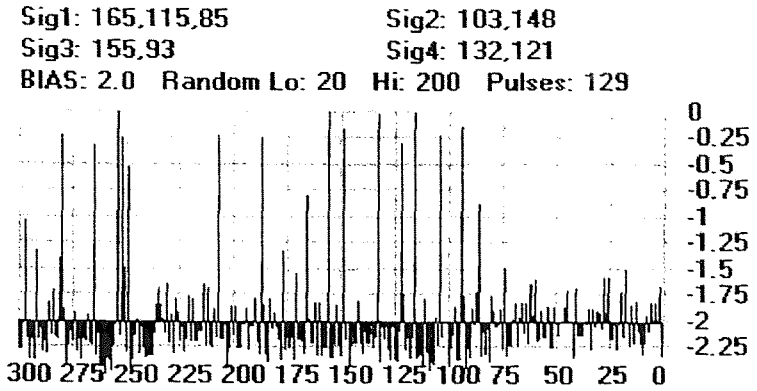



(c)

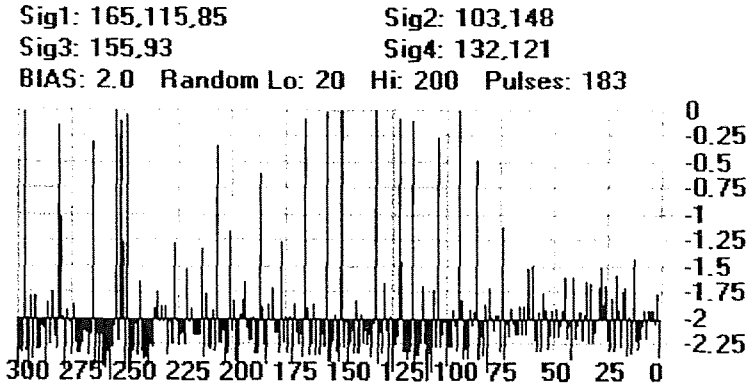
**Figure A.15** Receiver Output for 4 Sets Input, Bias = 2.0




(d)




(e)




(f)

**Figure A.15 (Cont.)** Receiver Output for 4 Sets Input, Bias = 2.0

# APPENDIX B

## PROCESSOR SIMULATION RESULTS

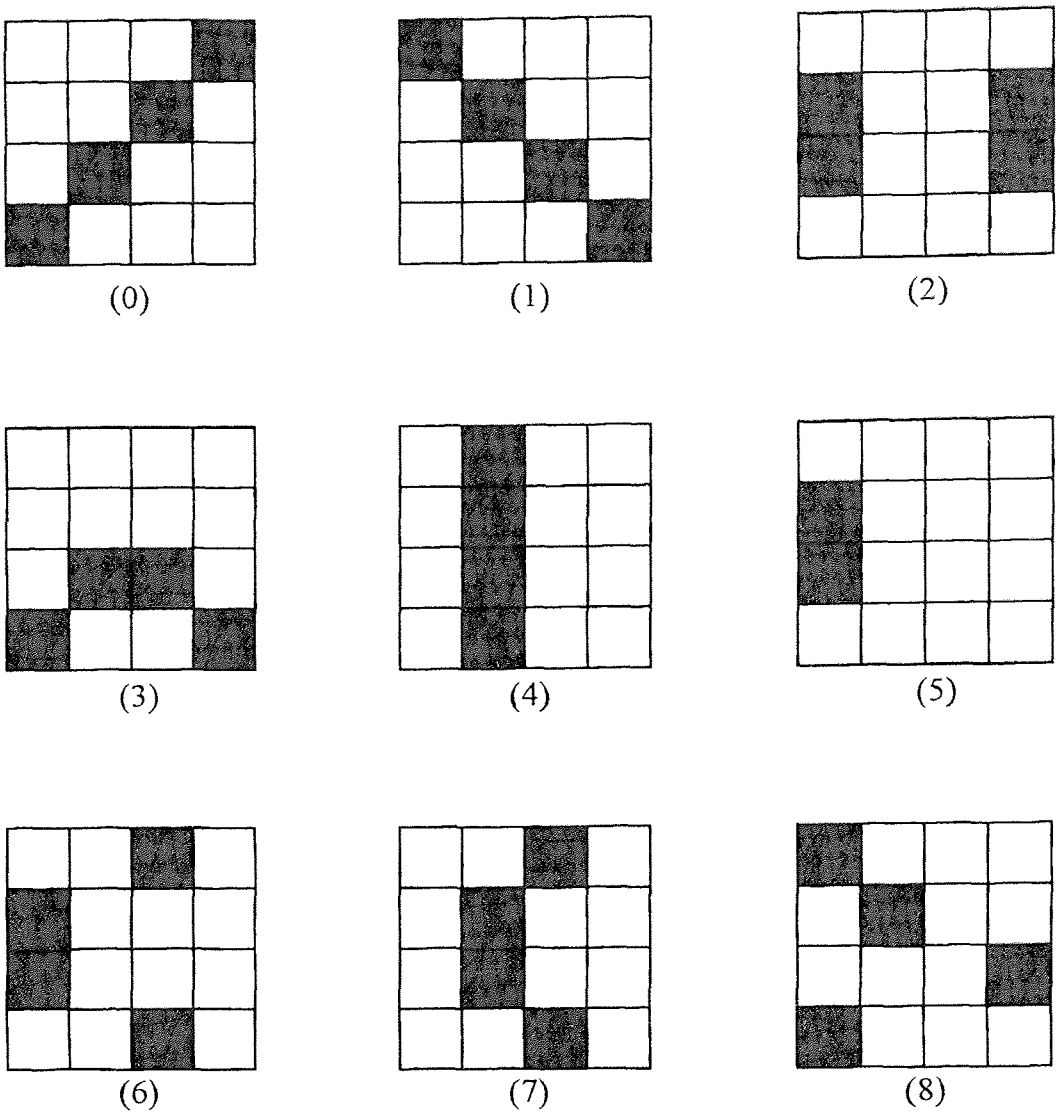



**Figure B.1** Input Test Patterns

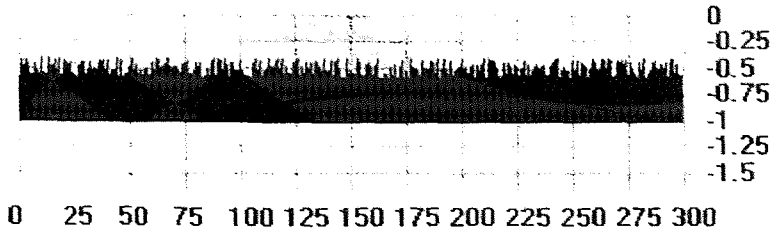
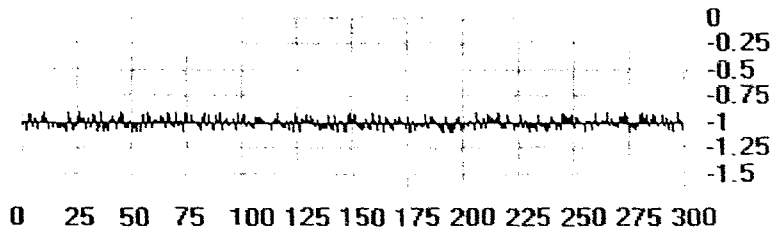
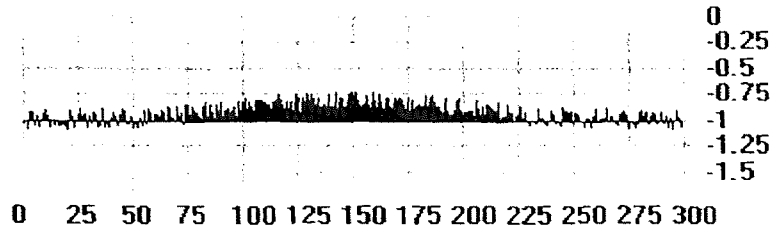


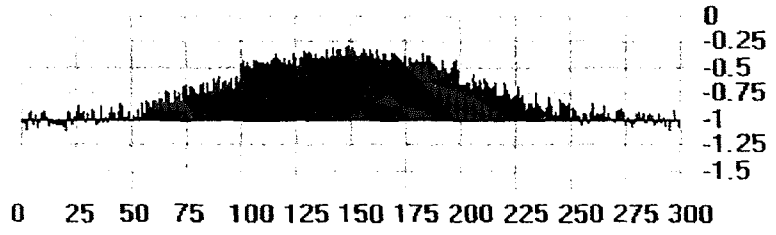

**Figure B.2** Initial Sensitive Settings




(a)

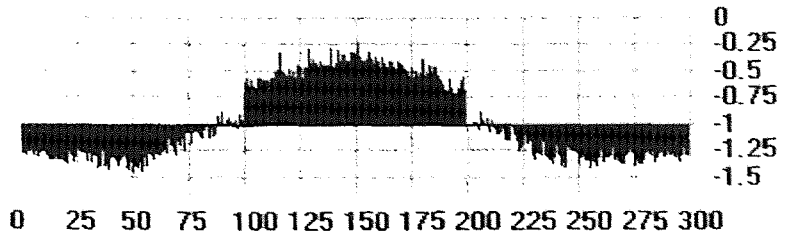



(b)

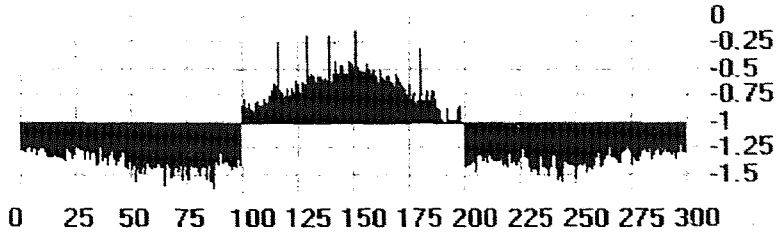



(c)

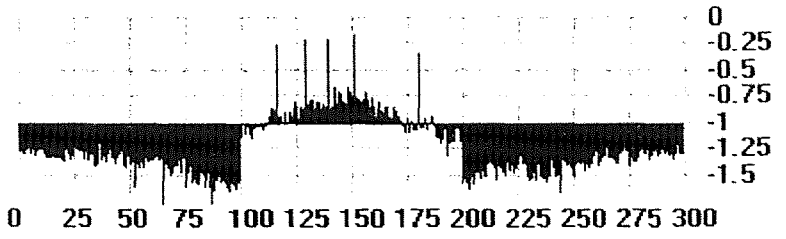
**Figure B.3** General Result of Positive Learning



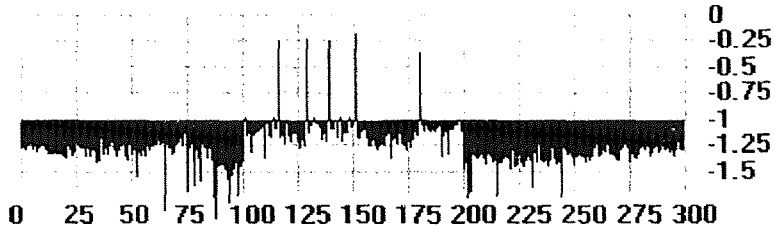

(a)




(b)




(c)




(d)

**Figure B.4** Application of Negative Learning Algorithm

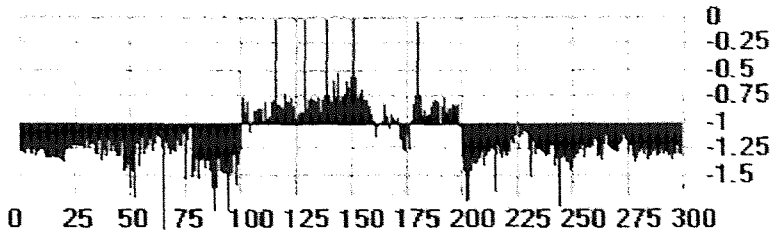
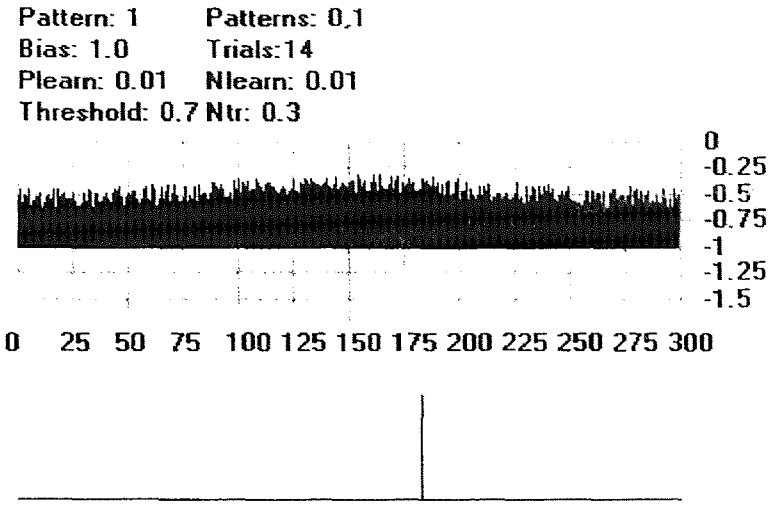
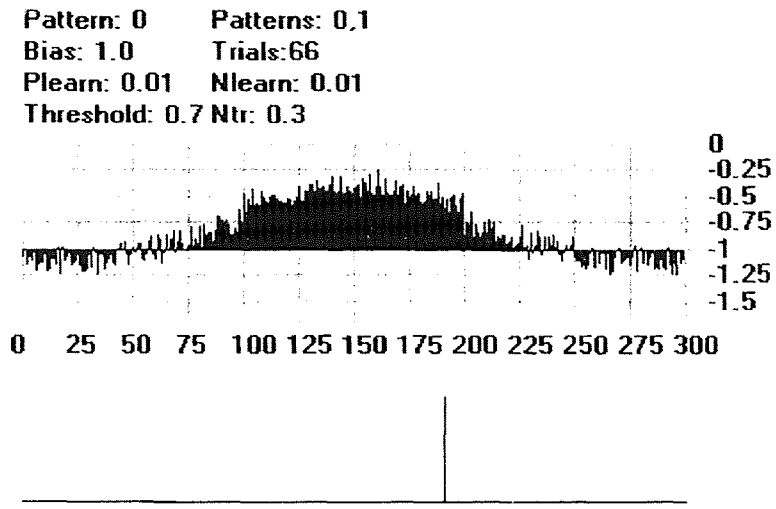



**Figure B.5** Introduction of a Second Pattern

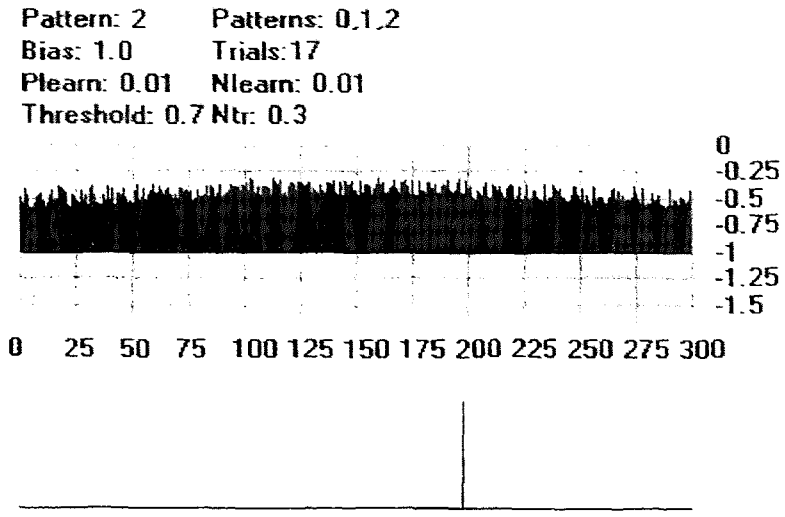



(a)

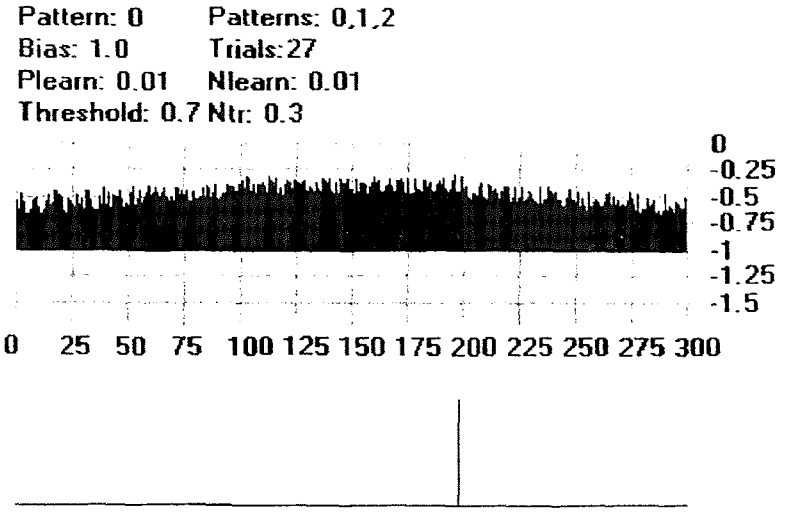



(b)

**Figure B.6** Detection of Two Patterns

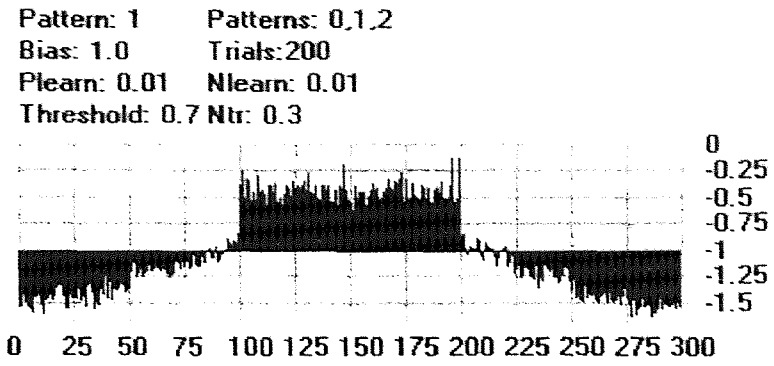



(a)




(b)

**Figure B.7** Detection of Three Patterns




(c)

**Figure B.7 (Cont.)** Detection of Three Patterns

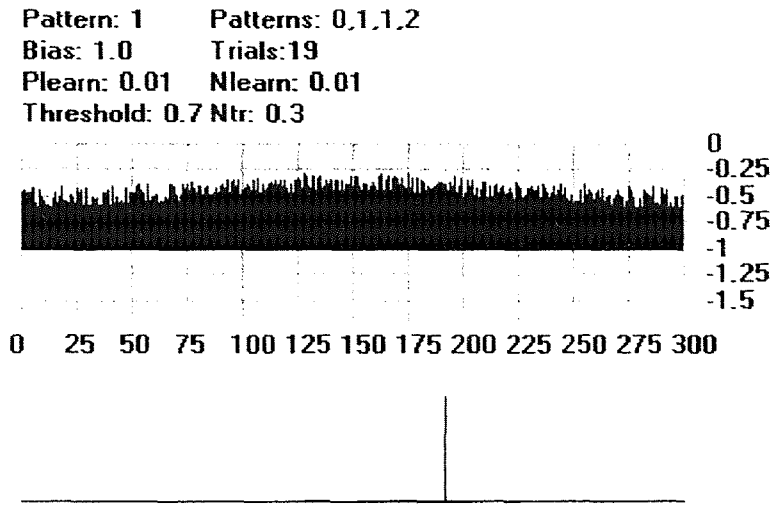
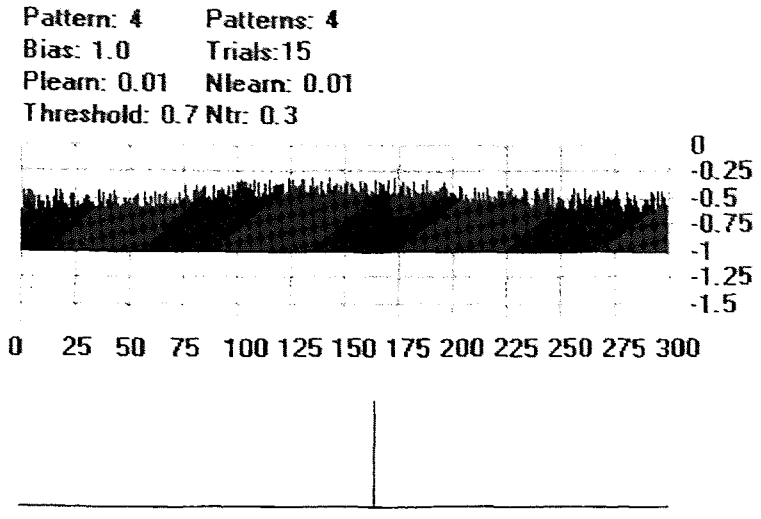
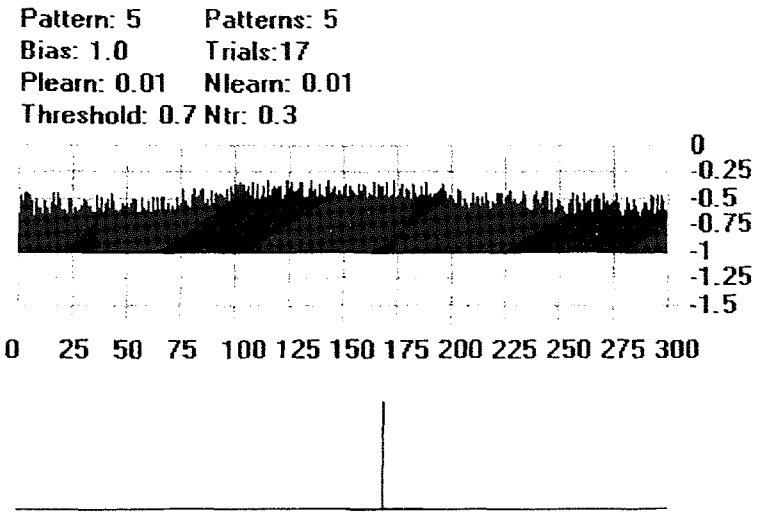



**Figure B.8** Biased Three Pattern Detection

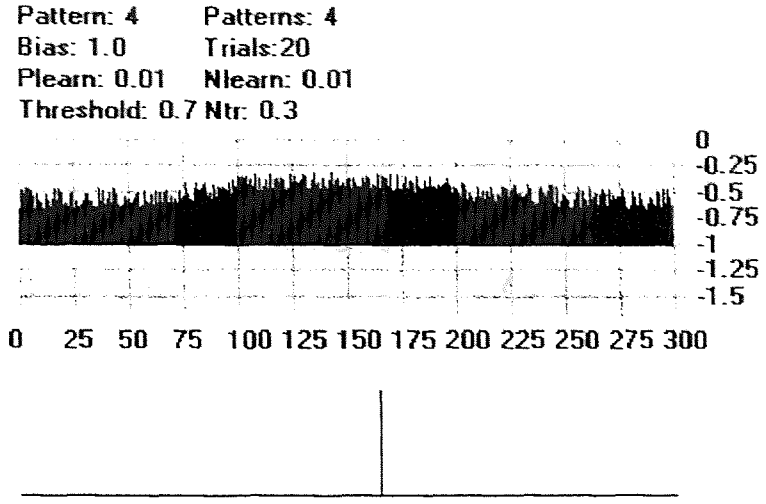



(a)

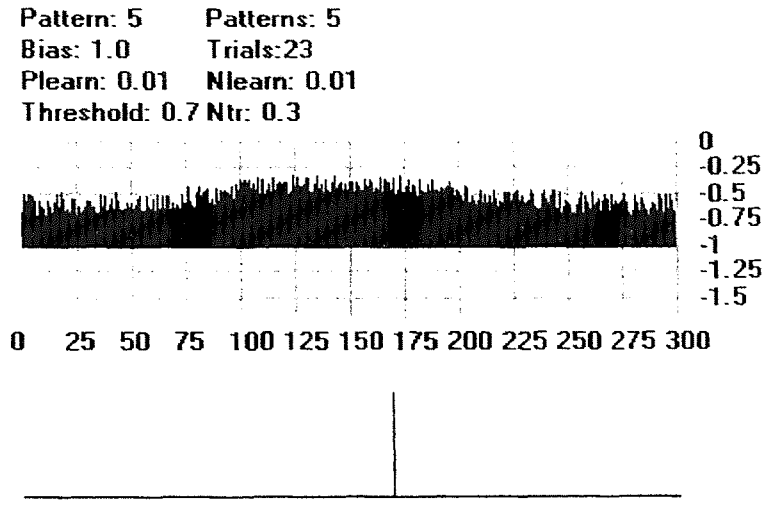



(b)

**Figure B.9** Forced Learning of Two Patterns




(c)




(d)

**Figure B.9 (Cont.)** Forced Learning of Two Patterns

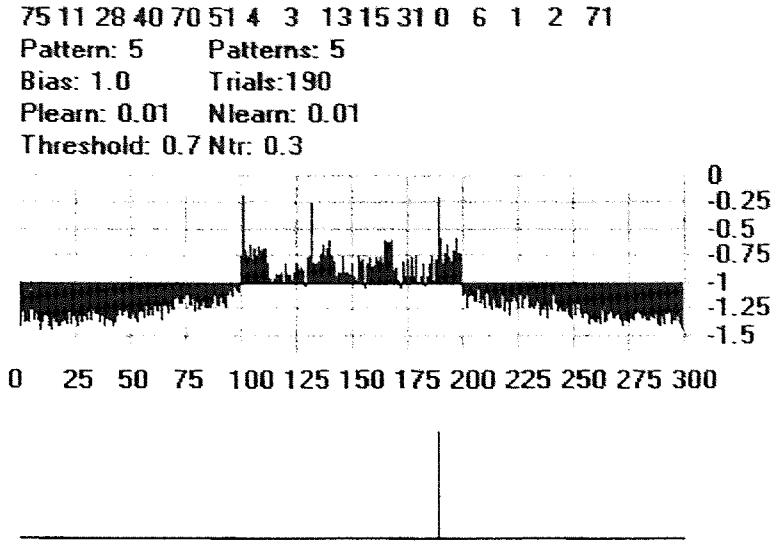



**Figure B.10** Overgrowth of a Single Pattern

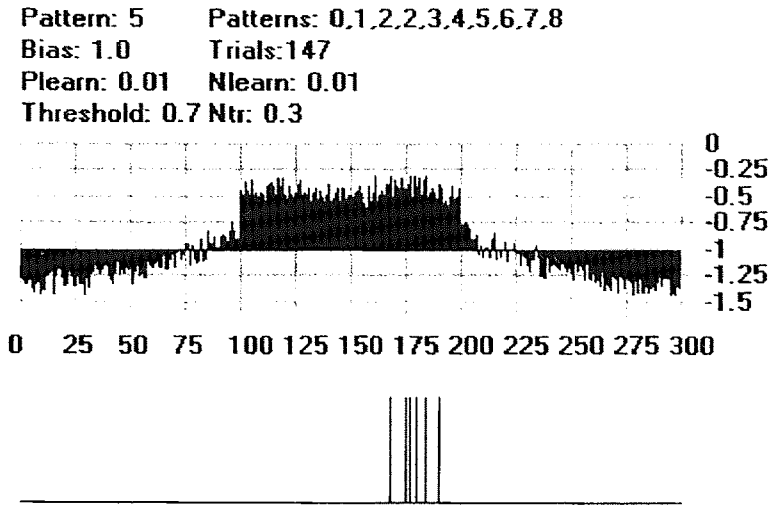
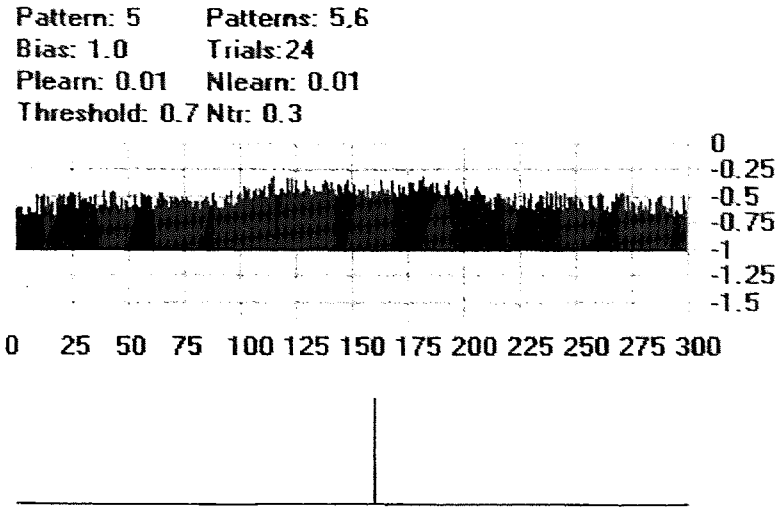
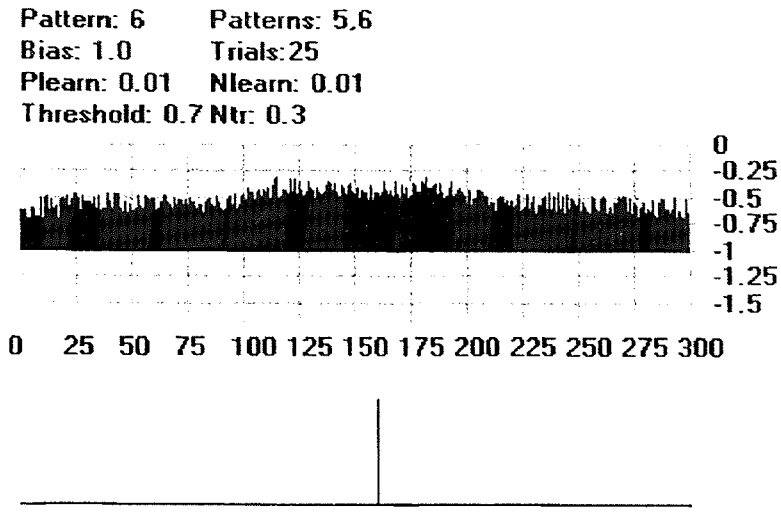



**Figure B.11** Learning Too Many Patterns

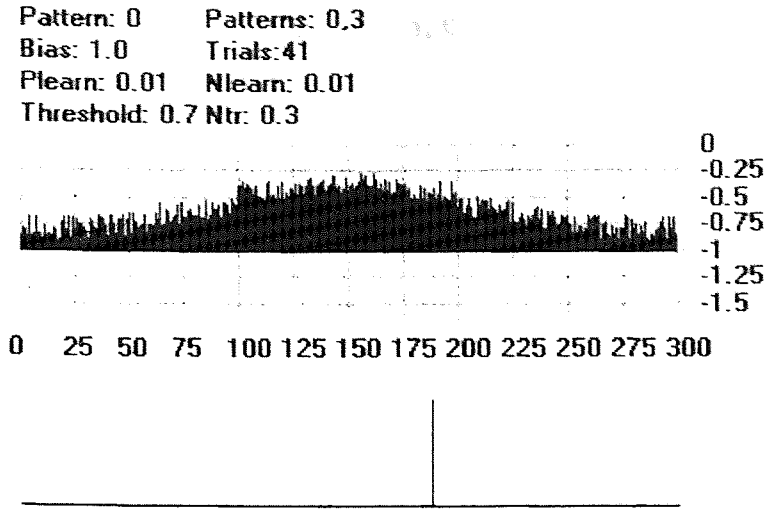



(a)

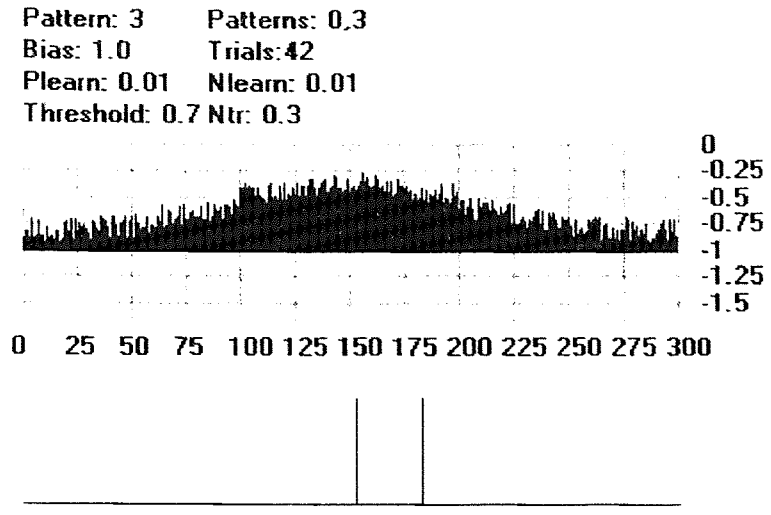



(b)

**Figure B.12** Detection of Subpatterns




(a)




(b)

**Figure B.13** Detection of Shared Patterns

# APPENDIX C

## RECEIVER JAVA APPLET

```
// Applet Javal
// by James Stanski
// 11/15/97
//
// This program simulates a receiver for interpreting input pulses
// from various sources.  Up to 4 sources may be used as well as
// several signals per source.  The accepted value range for input
// signals is 100-199.  If no signal numbers are input for the
// first input, then a random signal will be simulated instead.

import java.awt.*;
import java.applet.*;

public class javal extends java.applet.Applet {
//
// main variables for this class
//
  double gain[] = new double[300];  // Receiver output
  int pulse[] = new int[300];       // Marks present pulse in slow
                                    // material
  int input[][] = new int[10][10];  // Inputs Input source X signals per
  int next[] = new int[10];         // Holds when each source fires
                                    // again
  int count[] = new int[10];        // How many singnals per input
  boolean israndl[] = new boolean[10];  // True if a random signal is due
  int max,                          // max length of slow material
      randh, randl,                 // hi,low bounds in random gen
      pulses,                       // Current #pulses in simulation
      time;                         // Current time in simulation
  double min,                       // min value to account for
         vgain,                     // Vertical scale on out graph
         bias,                      // current bias
         oldbias;                   // Previous bias
  Button convButton;                // Runs simulation
  Label labelF, labelG, labelH, labelI, labelB, labelRH, labelRL;
  TextField textF, textG, textH, textI, textB, textRH, textRL;
```

```
//
//  Function builds entire display
//
  public void paint(java.awt.Graphics g) {
     int first = 370;                // Rcvr out X axis
     int second = 470;               // Input in X axis
     g.drawLine(50,second,max+50,second);
     g.drawLine(50,first,max+50,first);
     //
     //  Add X calibration
     //
     for (int x = 0; x <= max; x += 25){
       int y = (int)(vgain*bias);
       g.setColor(Color.lightGray);
       g.drawLine(x+50,first+15,x+50,first-y);
       g.setColor(Color.black);
       g.drawString(String.valueOf(300-x),x+44,first+30);
     }
     //
     //  Add Y calibration
     //
     for (double x = -0.25; x <= bias; x += 0.25) {
       int y = (int)(first-x*vgain);
       g.setColor(Color.lightGray);
       g.drawLine(50,y,355,y);
       g.setColor(Color.black);
       g.drawString(String.valueOf(x - bias),360,y+4);
     }
     //
     // Place information on top
     //
     int y = (int)(first-bias*vgain);
     g.drawString("BIAS: "+textB.getText()+
                  "   Random Lo: "+textRL.getText()+
                  "   Hi: "+textRH.getText()+
                  "   Pulses: "+String.valueOf(pulses),50,y-10);
     g.drawString("Sig1: "+textF.getText(),50,y-40);
     g.drawString("Sig2: "+textG.getText(),220,y-40);
     g.drawString("Sig3: "+textH.getText(),50,y-25);
     g.drawString("Sig4: "+textI.getText(),220,y-25);
     //
     // Draw graphs
     //
     for (int x = 0; x < max; x++) {
       g.drawLine(x+50,second,x+50,second-pulse[x]*50);
       g.drawLine(x+50,first,x+50,(int)(first-(gain[x]+bias)*vgain));
     }
  }
```

```
//
//  Before simulation is run
//
  public void init() {
    setBackground(Color.white);
    max = 300;      // Slow material is 300 units long
    min = 0.00005; // Ignore values below this
    vgain = 50.0;
    time = 0;

    // Set Text Fields
    labelF = new Label("Signal 1:"); add(labelF);
    textF = new TextField("165,115,85",16); add(textF);
    labelG = new Label("Signal 2:"); add(labelG);
    textG = new TextField("",16); add(textG);
    labelH = new Label("Signal 3:"); add(labelH);
    textH = new TextField("",16); add(textH);
    labelI = new Label("Signal 4:"); add(labelI);
    textI = new TextField("",16); add(textI);

    // Set Lables
    labelB = new Label("Bias:"); add(labelB);
    textB = new TextField("3.0",3); add(textB);
    labelRL = new Label("Random Low"); add(labelRL);
    textRL = new TextField("20",3); add(textRL);
    labelRH = new Label("Random Hi"); add(labelRH);
    textRH = new TextField("200",3); add(textRH);

    convButton = new Button("Run"); add(convButton);

    // Get Bias Value, set
    oldbias = bias = Double.valueOf(textB.getText()).doubleValue();
    for(int x = 0; x < max; x++) gain[x] = -bias;

    // Start all sources off randomly;
    for(int x = 0; x < 10; x++) {
      next[x] = (int)(Math.random()*180)+20;
      isrand[x] = true;
    }
  }
```

```
//
//  Function reads signal input fields and puts all input values into
//  input[][] Array according to the 'which' variable.  Function returns
//  number of input values.
//
  public int getin(TextField infield, int which) {
    String in = new String(infield.getText());
    int length = in.length();
    int x = 0; int count = 0;

    // Continue to parse inputs
    while(x < length) {
      int index2 = in.indexOf(',',x);
      // look for another ','
      if (index2 >= x) {
        input[which][count] =
            Integer.valueOf(in.substring(x,index2)).intValue();
        x = index2 + 1;
      }
      else {  // end of string
        input[which][count] =
            Integer.valueOf(in.substring(x)).intValue();
        x = length;
      }
      count++;
    }
    return count;
  }

//
//  Main function of Simulation.  Manages all four inputs and determines
//  when the next pulse will be input.  There is an option to insert
//  random pulses in betwee. But this option is currently turned off.
//
  public void getstream() {

    // If Bias changed, adjust
    bias = Double.valueOf(textB.getText()).doubleValue();
    for(int x = 0; x < max; x++) {
       gain[x] -= (bias - oldbias);
    }
    oldbias = bias;

    // Reload new random values
    randh = Integer.valueOf(textRH.getText()).intValue();
    randl = Integer.valueOf(textRL.getText()).intValue();

    // Read from all 4 input sources
    count[0] = getin(textF,0);
    count[1] = getin(textG,1);
    count[2] = getin(textH,2);
    count[3] = getin(textI,3);
```

```
    // Send five pulses each time this function is called
    // Send pulse from source which is ready to send the next pulse
    for(int x = 0; x < 5; x++) {
      int which = getlow();  // Get next source to fire

      // All signals start as random, send random pulse instead
      // of actual information.
      if (israndl[which]) {
        if (count[which] > 0) israndl[which] = false; // next is not rand
        if (next[which] > time) {
          perpulse(next[which] - time);
          time = next[which];
        }
        next[which] = time + (int)(Math.random()*(randh-randl))+randl;
      }

      // Send a pulse carrying real information
      else {
//        israndl[which] = true;  // dont send any more random pulses
        if (next[which] > time) {
          perpulse(next[which] - time);
          time = next[which];
        }
        next[which] = time + input[which][(int)(Math.random() *
                      count[which])];
      }
    }
  }

//
// Determines which input source is ready to send the next pulse
// Return source number ready
//
  public int getlow() {
    int lowest = next[0]; int z = 0;

    // Scan over all 4 sources
    for (int y = 1; y < 4; y++) {
       if ((next[y] < lowest) && (count[y] > 0)) {
         lowest = next[y]; z = y;
       }
    }
    return z;
  }
```

```
//
// This function is the core of the receiver.  Each input is a delay
// before a pulse is to be entered.  The function manages the slow
// material.  When an incoming pulse intersects with reflected slow
// pulses, the result in this position is increased.  There are options
// to normalize this increase.
//
  public void perpulse(int delay) {

    // Increment the number of total pulses sent
    pulses++;

    // forward all pulses by delay along slow material
    for (int y = 0; y < delay; y++) {
      for (int x = 0; x < max - 1; x++) {
        pulse[x] = pulse[x+1];
      }
      pulse[max-1] = 0;
    }
    // After this, assume incoming pulse came, now
    // adjust values according to input.
    for (int x = max-1; x >= 0; x--) {
      if (pulse[x] == 1) {
        // Increase exponentially
        double value = Math.exp(gain[x]);
        // If hit limit, stop
        if (value + gain[x] > 0.0) {
          value = 0.0 - gain[x];
        }
        gain[x] += value;

      // Normalize this offset, helps reject noise.
      left(value/2.0,x-1);
      right(value/2.0,x+1);
      }
    }
    // Add a reverse traveling pulse on slow material
    pulse[max-1] = 1;
    // Show results on screen.
    repaint();
  }
```

```
//
// The next two recursive functions implement the normalizing
// task.  Below, the function left normalizes to the left of the
// Receiver out increase.  The current method is to take away from
// outputs of higher value and take less from lower values
//
  public void left(double strength, int position) {

    // Check bounds
    if (position < 0) {right(strength,0);}
    else {
        double here=strength * Math.exp(gain[position]);
        double there = strength - here;
        gain[position] -= here;
        // See if value is large enough to be considered
        if (there > min) {
          left(there,position - 1);  // recursive call
        }
        else {
          gain[position] -= there;
        }
    }
  }

//
// Same as left but to the right of output increase
//
  public void right(double strength, int position) {
    if (position >= max) {left(strength,max - 1);}
    else {
        double here=strength * Math.exp(gain[position]);
        double there = strength - here;
        gain[position] -= here;
        if (there > min) {
          right(there,position + 1);
        }
        else {
          gain[position] -= there;
        }
    }
  }

//
//  Works with Java API and responds to a button press to
//  run simulation.
//
  public boolean action(Event e, Object arg) {
    if (e.target == convButton) {
      getstream();
      return true;
    }
    return false;
  }
}
```

# APPENDIX D

## PROCESSOR JAVA APPLET

```
// Applet Java2
// by James Stanski
// 11/20/97
//
// This program simulates the processing section of the agent.  The
// Input shapes are 4x4 patterns.  When running, the user can select
// which shapes will be rotated as input to the porocessor.  This
// program will identify each shape with a value.  Random values
// are assigned for each cell in the 4x4 patterns.

import java.awt.*;
import java.applet.*;

public class java2 extends java.applet.Applet {
//
//  Main variables of this class
//
  double gain[] = new double[300];     // processor LTM
  int projcnt;                         // True firing time
  double project[] = new double[620]; // Learning buffer
  int pulses[] = new int[30];          // all pulses in slow material
  int pulse[] = new int[300];          // array for propagation
  int fire[] = new int [620];          // indicates when fired
  int input[] = new int[30];           // user pattern ids
  int max,                             // max size of slow material
      trials;              // number of cycles so far
  double plearn,                       // Learn coefficient to increase
         nlearn,                       // Learn coefficient to decrease
         vgain,                        // Vertical scale on out graph
         bias,                         // Current LTM bias
         oldbias,                      // Previous LTM bias
         threshold,                    // Threshold to fire
         nthreshold;                   // Threshold to stop quence
  Button convButton;                   // Runs one cycle
  Label labelF, labelB, labelP, labelN, labelT, labelNT;
  TextField textF, textB, textP, textN, textT, textNT;
  int location[] = new int[16];        // integers for all 16 squares
  int width;                           // used to get true firing time
  int pattno;                          // pattern number to run

  int pattern[][] = {{0,0,0,1, 0,0,1,0, 0,1,0,0, 1,0,0,0},
                     {1,0,0,0, 0,1,0,0, 0,0,1,0, 0,0,0,1},
                     {0,0,0,0, 1,0,0,1, 1,0,0,1, 0,0,0,0},
                     {0,0,0,0, 0,0,0,0, 0,1,1,0, 1,0,0,1},
                     {0,1,0,0, 0,1,0,0, 0,1,0,0, 0,1,0,0},
                     {0,0,0,0, 1,0,0,0, 1,0,0,0, 0,0,0,0},
                     {0,0,1,0, 1,0,0,0, 1,0,0,0, 0,0,1,0},
                     {0,0,1,0, 0,1,0,0, 0,1,0,0, 0,0,1,0},
                     {1,0,0,0, 0,1,0,0, 0,0,0,1, 1,0,0,0},
                     {0,0,0,0, 0,1,1,0, 0,1,1,0, 0,0,0,0}};
```

```
//
// Function builds entire display
//
  public void paint(java.awt.Graphics g) {
     int first = 270;
     int second = 390;
     g.drawLine(50,second,max+50,second);  // Processor out
     g.drawLine(50,first,max+50,first);    // LTM
     //
     // Add X calibration
     //
     for (int x = 0; x <= max; x += 25){
       int y = (int)(vgain*bias);
       g.setColor(Color.lightGray);
       g.drawLine(x+50,first+35,x+50,first-y);
       g.setColor(Color.black);
       g.drawString(String.valueOf(x),x+44,first+50);
     }
     //
     //  Add Y calibration
     //
     for (double x = -0.50; x <= bias; x += 0.25) {
       int y = (int)(first-x*vgain);
       g.setColor(Color.lightGray);
       g.drawLine(50,y,max+55,y);
       g.setColor(Color.black);
       g.drawString(String.valueOf(x - bias), max+ 60,y+4);
     }
     //
     // Place information on top
     //
     int y = (int)(first-bias*vgain);
     for (int x = 0; x < 16; x++) {
       g.drawString(String.valueOf(location[x]),50 + 17*x,y-70);
     }
     g.drawString("Bias: "+textB.getText(),50,y-40);
     g.drawString("Threshold: "+textT.getText(),50, y-10);
     g.drawString("Patterns: "+textF.getText(),135,y-55);
     g.drawString("Pattern: "+String.valueOf(pattno),50,y-55);
     g.drawString("Nlearn: "+textN.getText(),135,y-25);
     g.drawString("Plearn: "+textP.getText(),50,y-25);
     g.drawString("Ntr: "+textNT.getText(),135,y-10);
     g.drawString("Trials:"+String.valueOf(trials),135, y-40);
     //
     // Draw graphs
     //
     for (int x = 0; x < max; x++) {
       g.drawLine(x+50,second,x+50,second-fire[x]*50);
       g.drawLine(x+50,first,x+50,(int)(first-(gain[x]+bias)*vgain));
     }
```

```
//
// Before asimulation is run
//
  public void init() {
    setBackground(Color.white);
    max = 300;      // Slow material is 300 units long
    vgain = 50.0;
    trials = 0;

    // Set text fields and lables
    labelT = new Label("Threshold:"); add(labelT);
    textT = new TextField("0.7",3); add(textT);
    labelB = new Label("Bias:"); add(labelB);
    textB = new TextField("1.0",3); add(textB);
    labelP = new Label("Plearn:"); add(labelP);
    textP = new TextField("0.01",5); add(textP);
    labelN = new Label("Nlearn:"); add(labelN);
    textN = new TextField("0.01",5); add(textN);
    labelNT = new Label("Nthreshold:"); add(labelNT);
    textNT = new TextField("0.3",3); add(textNT);
    labelF = new Label("Patterns:"); add(labelF);
    textF = new TextField("0,1",16); add(textF);

    convButton = new Button("Run"); add(convButton);

    // init bias
    oldbias = bias = Double.valueOf(textB.getText()).doubleValue();
    for(int x = 0; x < max; x++) gain[x] = -bias;
    for (int x = 0; x < max; x++)
           gain[x] += (Math.random()*0.2-0.1)+.50;

    // get random locations of inputs;
    for (int x = 0; x < 16; x++) {
      int temploc, found;
      do {
        found = 0;
        temploc = (int) (Math.random()*99);
        for (int y = 0; y < x; y++) {
          if(location[y] == temploc) {
            found = 1; break;
          }
        }
      } while (found == 1);
      location[x] = temploc;
    }

   // locate offset for true firing time
   int lowest = 100;
    for (int x = 0; x < 16; x++) {
       if(location[x] < lowest) {
          lowest = location[x];
       }
    }
    width = lowest;
  }
```

```
//
//  Function reads signal input fields and puts values into
//  intput[]. Function returns number of inputs in line
//
  public int getin(TextField infield) {
    String in = new String(infield.getText());
    int length = in.length();
    int x = 0; int count = 0;
    while(x < length) {
      int index2 = in.indexOf(',',x);
      if (index2 >= x) {
      input[count] = Integer.valueOf(in.substring(x,index2)).intValue();
        x = index2 + 1;
      } else {
        input[count] = Integer.valueOf(in.substring(x)).intValue();
        x = length;
      }
      count++;
    }
    return count;
  }

//
//  Main Function in simulation.  Selects an input patterns and
//  feeds this pattern to the sweep algorithm which performs the
//  autocorrelation
//
  public void getstream() {
    int x;

    // Reload all user set values
    plearn = Double.valueOf(textP.getText()).doubleValue();
    nlearn = Double.valueOf(textN.getText()).doubleValue();
    nthreshold = Double.valueOf(textNT.getText()).doubleValue();
    bias = Double.valueOf(textB.getText()).doubleValue();
    threshold = Double.valueOf(textT.getText()).doubleValue();
    for(x = 0; x < max; x++) {
       gain[x] -= (bias - oldbias);
    }
    oldbias = bias;


    trials++;
    // clear all firings, pulse array
    for(x = 0; x < max; x++) {
      fire[x] = 0;
      pulse[x] = 0;
      project[x] = 0;
    }
    projcnt = 0;
```

```
    // retrieve pattern to run
    int inputs = getin(textF);
    pattno = input[(int)(Math.random() * inputs)];
    // find lowest number
    int lowest = 101;
    for (x = 0; x < 16; x++) {
      if(pattern[pattno][x] == 1) {
        if(location[x] < lowest) {
          lowest = location[x];
        }
      }
    }

    // get next lowest number until end
    int oldlowest = lowest;
    do {
      lowest = 101;
      for(x = 0; x < 16; x++) {  // find nest lowest
        if(pattern[pattno][x] == 1) {
          if(location[x] < lowest && location[x] > oldlowest){
            lowest = location[x];
          }
        }
      }
      sweep(lowest - oldlowest);
      oldlowest = lowest;
    } while(lowest < 101);

    // flush out slow material
    sweep(max+1);
    for (x = 0; x < max; x++) {
      gain[x] += project[x];
      if (gain[x] > 0.0) gain[x] = 0.0;
    }
    repaint();
  }

//
//  This function simulates the slow material as will as the
//  correlation detection and learning algorithms.
//
  public void sweep(int delay) {
  // forward all pulses by delay.
    for (int y = 0; y < delay; y++) {
      for (int x = max-1; x > 0; x--) {
        pulse[x] = pulse[x-1];
      }
      if (y == 0) {
        pulse[0] = 1;
      }
      else {
        pulse[0] = 0;
      }
```

```
      // get a list of pulses
      int num_pulses = 0;
      for (int x = 0; x < max; x++) {
         if (pulse[x] == 1) {
           pulses[num_pulses] = x;
           num_pulses++;
         }
      }

      // get average potential on slow material
      double avg = 0.0; int n_pulses = 0;
      for (int x = 0; x < num_pulses; x++) {
        if (pulses[x] >= 100 && pulses[x] < 200) {
          avg += Math.exp(gain[pulses[x]]);
        }
      }
      avg = avg/num_pulses;


      // learning algorithm
      for (int x = 0; x < num_pulses; x++) {
//        if (avg <= threshold) {
        project[pulses[x]] +=
             avg * plearn * (1-Math.exp(gain[pulses[x]]));
//        }
      }
```

```
      //
      // If the threshold is exceeded, then fire
      //
      if (avg > threshold) { // fire, kill competition
         // put a pulse in the fired set
         fire [projcnt - width] = 1;

         // Hurt the rest having a similar pattern
         // These are the competitors of the fired pattern
         for (int x = -101; x < max; x++) {
           double navg = 0.0;
           int poffset = pulses[0];
           n_pulses = 0;

           // find average of competitors.
           if (x != poffset) { // dont kill yourself;
             for (int z = 0; z < num_pulses; z++) {
               int tpulse = (pulses[z] - poffset)+ x;
               if ((tpulse >= max) || (tpulse < 0)) continue;
               navg += Math.exp(gain[tpulse]);
               n_pulses++;
             }
             navg = navg/n_pulses;

             // Quench only if above threshold
             if (navg > nthreshold) {
               for (int z = 0; z < num_pulses; z++) {
                 int tpulse = (pulses[z] - poffset) + x;
                 if ((tpulse >= max) || (tpulse < 0)) continue;
                           project[tpulse] -= navg * nlearn * (1-
Math.exp(gain[tpulse]));
               }
             }
           } // punish all but self
         }  // punish all who attemp to copy
       } // if fired
       projcnt++;
     } // delay counter
   }

//
// Works with JAVA API and responds to a button press to
// run simulation
//
  public boolean action(Event e, Object arg) {
    if (e.target == convButton) {
      getstream();
      return true;
    }
    return false;
  }
}
```

## REFERENCES


1. W.S. McCulloch and W. H. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bull. Math. Biophys.*, vol. 5, pp.115-133, 1943.

2. M. Conrad, Molecular computing, in *Advances in Computers*, Academic Press, Boston, MA, 1990.

3. P. E. Burrows and E.G. Wilson, "The inchworm memory: A new molecular electronic device," *J. Mol. Electronics*, vol. 6, pp. 209-220, 1990.

4. W. Rall and I. Segev, Functional possibilities for synapses on dendrites and on dendric spines, in *Synaptic Function*, (G. M. Edelman., E.E. Gall., and W.M. Cowan., Eds.) Wiley, New York, NY, 1987, pp. 605-636.

5. P. F. Pinsky and J. Rinzel, "Intrinsic and network rhythmogenesis in a reduced model of CA3 neurons, *J. Computat. Neurosci.*, vol. 1, pp. 39-60, 1994.

6. H. Kargupta and Sylvian R. Ray, "Temporal sequence processing based on the biological reaction-diffusion process," *IEEE International Conference of Neural Networks*, pp. 2315-2320, 1994.

7. O. T. C. Chen, T. Berger, and B. J. Sheu, "VLSI implementation of the hippocampus on nonlinear system model," *IEEE International Conference of Neural Networks*, pp. 2009-2014, 1994

8. J. E. Dayhoff, "Temporal codes for pulsed networks," *IEEE International Conference of Neural Networks*, pp. 1273-1277, 1994.

9. M. A. Arbib, Introducing the neuron, in *The Handbook of Brain Theory and Neural Networks*, MIT Press, Cambridge, MA, 1995, pp. 4-11.

10. C. W. Helstrom, *Statistical Theory of Signal Detection*, Pergamon Press, London, 1968, pp. 102-143.

11. B. Hasslacher and M. W. Tilden, "Living machines," *NSF Workshop on Neuromorphic Analog VLSI Systems*, 1994.

12. R. F. Lyon and C.A. Mead, "An electronic cochlea," *IEEE Trans. Acoustics, Speech and Signal Processing*, vol. 36, pp. 1119-1134, 1988.

13. J Marienborg and T. Suerrelande, "Neuromorphic analog communication," *IEEE International Conference of Neural Networks*, pp. 920-925, 1995.

14. S. Haykin, *Neural Networks*, Macmillan College Publishing Company, New York, NY, 1994, pp. 18-22.

15. A. S. Jackson, *Analog Computation*, McGraw-Hill Book Company, INC., New York, NY, 1960, pp. 35-72.

16. A. Nigrin, *Neural Networks for Pattern Recognition*, The MIT Press, Cambridge, MA, 1993, pp. 20-32.

17. S. Russel and P. Norvig, *Artificial Intelligence, A Modern Approach*, Prentice Hall, Upper Saddle River, NJ, 1995, pp. 31-52.

18. T. Kanminuma and G. Matsumoto, *Biocomputers The next generation from Japan*, Chapman and Hall, London, 1991.