30

ON A BLOCK FLOATING POINT IMPLEMENTATION OF AN INTRUSION-DETECTION ALGORITHM

by

ROBERT JOSEPH FOGLER

B.S., Kansas State University, 1977

A MASTER'S THESIS

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Electrical Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1979

Approved by:

Nasir Ahmed

Major Professor

To my loving parents,

without whose encouragement

this work would not have

been possible.

TABLE OF CONTENTS

APPENDICES

CHAPTER I

INTRODUCTION

The use of adaptive prediction to improve the performance of perimeter sensors for intrusion-detection was introduced in [1]. Such sensors are buried cables, typically 100 meters in length, deployed about an area containing a resource to be protected, as depicted in Fig. 1. The response of a sensor to a given stimulus depends upon the nature as well as the proximity of the source. Such sensors not only respond to intruder stimuli, but also a variety of other sources causing poor signal-to-noise ratios. Examples of such noise sources are shown in Fig. 2.

Adaptive prediction is employed since the ambient noise is non-stationary, or at best, stationary on a short-term basis. Conversely, intruder signals are transients whose spectra are broadband and relatively "white" over the passband of interest--i.e., 0-4 Hz. Hence they pass through the adaptive predictor essentially unchanged. Since the predictor strives to decorrelate the input noise, most of the correlated components are removed, resulting in a substantial reduction of noise power at the output. Thus, the signal-to-noise ratio at the output for an intruder signal embedded in noise is greatly improved.

From the above discussion it follows that the overall problem of detecting an intruder is equivalent to that of detecting a random signal in white noise. The solution to this problem is well-known for the case of stationary signals [2,3]. An approximation is employed since the signal of interest may be stationary only in the short-term. To this end, an adaptive threshold detector (ATD) is used which processes the predictor output and generates a "1" if an intruder is present, or a "0" if an intruder is not present.

The adaptive predictor is implemented as a lattice structure (ALP) because it has superior convergence properties to that of transversal filter structures, (due to the successive orthogonalization of the prediction error and the decoupling of the filter coefficients (weights) at each lattice stage [5,6].) Moreover, the lattice structure exhibits a lower sensitivity to roundoff noise [4] as is the case with limited precision implementations.

The ALP-ATD combination, heretofore referred to as the intrusion-detection algorithm, is shown in Fig. 3.

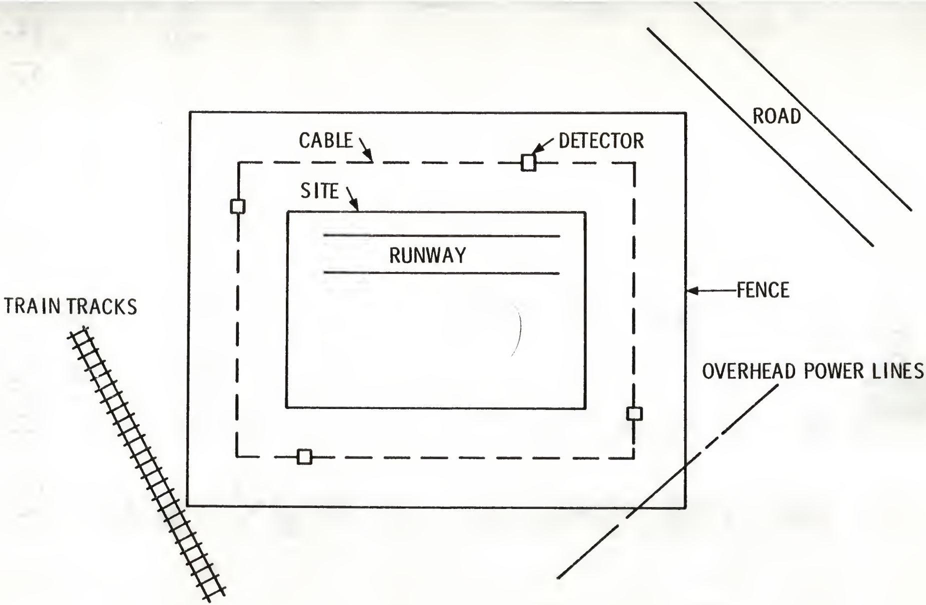

Fig. 1. Sensor deployment and some related noise sources.

CHAPTER II

ADAPTIVE LATTICE PREDICTOR

An important concept related to the lattice filter structure is the notion of "forward" and "backward" prediction [6].

Given the input sequence x(n-1), x(n-2),...x(n-N), forward prediction implies that the current input sample x(n) is to be predicted. If $\hat{x}(n)$ denotes a linear estimate of x(n), then

$$\hat{x}(n) = -[d_{1,N}x(n-1) + d_{2,N}x(n-2) + \ldots + d_{N,N}x(n-N)] \tag{1}$$

where the $d_{i,N}$ are the forward prediction coefficients of an N-weight linear predictor. The corresponding prediction error is given by

$$e_N(n) = x(n) + d_N^T x_n. \tag{2}$$

The $d_{i,N}$ are computed so as to minimize the mean-squared error from

$$\nabla_{d_{i,N}} E[e_N^2(n)] = 0, \tag{3}$$

which leads to

$$E[x(n)x_n] = E[x_n x_n^T]d_N. \tag{4}$$

If the process is wide-sense stationary one can denote $E[x(n)x_n]$ by r_N and $E[x_n x_n^T]$ by R_N and obtain

$$r_N = -R_N d_N \tag{5}$$

where

$$r_N^T = [R_{xx}(1)\ R_{xx}(2)\ R_{xx}(3)\ .\ .\ .\ R_{xx}(N)] \tag{6}$$

and

$$R_N = \begin{bmatrix} R_{xx}(0) & R_{xx}(1) & R_{xx}(2) & \dots & R_{xx}(N-1) \\ R_{xx}(1) & R_{xx}(0) & R_{xx}(1) & \dots & R_{xx}(N-2) \\ & \vdots & & & \\ R_{xx}(N-1) & R_{xx}(N-2) & & \dots & R_{xx}(0) \end{bmatrix}. \tag{7}$$

It is observed that the autocorrelation matrix R_N is Toeplitz.

In backward prediction, given the same input sequence, a past input sample $x(n-N-1)$ is to be predicted. If $\hat{x}(n-N-1)$ denotes an estimate of that sample, then

$$\hat{x}(n-N-1) = -[c_{1,N}x(n-1) + c_{2,N}x(n-2) + \dots + c_{N,N}x(n-N)] \tag{8}$$

where the $c_{i,N}$ are the backward prediction coefficients of an N-weight linear predictor. The corresponding prediction error is given by

$$w_N(n) = x(n-N-1) + c_N^T x_n \tag{9}$$

Again, computing the $c_{i,N}$ so as to minimize the mean-squared error leads to

$$s_N = -R_N c_N \tag{10}$$

where

$$s_N^T = [R_{xx}(N)\ R_{xx}(N-1)\ R_{xx}(N-2)\ \dots\ R_{xx}(1)] \tag{11}$$

and R_N is given by (7).

Observing that s_N and r_N are related by

$$s_{i,N} = r_{N+1-i,N}\,, \tag{12}$$

from (5) and (10) it is apparent that

$$c_{i,N} = d_{N+1-i,N}\,,\ i=1,2,\ldots N. \tag{13}$$

From the above discussion it follows that the forward prediction coefficients for an N+1 weight predictor can be obtained from

$$r_{N+1} = -R_{N+1}d_{N+1}. \tag{14}$$

The above matrices can be partitioned [9] as follows:

$$\left[\begin{array}{c} R_{xx}(1) \\ R_{xx}(2) \\ \vdots \\ R_{xx}(N) \\ \hline R_{xx}(N+1) \end{array}\right] = -\left[\begin{array}{ccccc|c} R_{xx}(0) & R_{xx}(1) & R_{xx}(2) & \ldots & R_{xx}(N-1) & R_{xx}(N) \\ R_{xx}(1) & R_{xx}(0) & R_{xx}(1) & \ldots & R_{xx}(N-2) & R_{xx}(N-1) \\ & \vdots & & & & \vdots \\ R_{xx}(N-1) & R_{xx}(N-2) & R_{xx}(N-3) & \ldots & R_{xx}(0) & R_{xx}(1) \\ \hline R_{xx}(N) & R_{xx}(N-1) & R_{xx}(N-2) & \ldots & R_{xx}(1) & R_{xx}(0) \end{array}\right] d_{N+1}$$

It is apparent that the above expression can be written as

$$\left[\begin{array}{c} r_N \\ \hline R_{xx}(N+1) \end{array}\right] = -\left[\begin{array}{c|c} R_N & s_N \\ \hline s_N^T & R_{xx}(0) \end{array}\right] d_{N+1}. \tag{15}$$

From matrix bordering [10,11], the following can be obtained:

$$d_{N+1} = -\left[\begin{array}{c|c} \beta_{11} & \beta_{12} \\ \hline \beta_{21} & \beta_{22} \end{array}\right]\left[\begin{array}{c} r_N \\ \hline R_{xx}(N+1) \end{array}\right] \tag{16}$$

where

$$\beta_{11} = R_N^{-1} + \frac{R_N^{-1}\, s_N\, s_N^T\, R_N^{-1}}{\xi},$$

$$\beta_{12} = -\frac{R_N^{-1} s_N}{\xi},$$

$$\beta_{21} = -\frac{s_N^T\, R_N^{-1}}{\xi},$$

$$\beta_{22} = \frac{1}{\xi},$$

and $\xi = Rxx(0) - s_N^T\, R_N^{-1}\, s_N$.

Now, (16) can be expressed as

$$d_{N+1} = -\left[\begin{array}{c|c} R_N^{-1} & 0 \\ \hline 0 & 0 \end{array}\right]\left[\begin{array}{c} r_N \\ \hline R_{xx}(N+1) \end{array}\right] - \frac{1}{\xi}\left[\begin{array}{c|c} R_N^{-1}\, s_N\, s_N^T\, R_N^{-1} & -R_N^{-1} s_N \\ \hline -s_N^T\, R_N^{-1} & 1 \end{array}\right]\left[\begin{array}{c} r_N \\ \hline R_{xx}(N+1) \end{array}\right]$$

which leads to

$$d_{N+1} = -\left[\begin{array}{c} R_N^{-1}\, r_N \\ \hline 0 \end{array}\right] - \frac{1}{\xi}\left[\begin{array}{c} R_N^{-1}\, s_N\, s_N^T\, R_N^{-1}\, r_N - R_N^{-1}\, s_N\, R_{xx}(N+1) \\ \hline -s_N^T\, R_N^{-1}\, r_N + R_{xx}(N+1) \end{array}\right] . \tag{17}$$

From (5) and (10), the above expression reduces to

$$d_{N+1} = \left[\begin{array}{c} d_N \\ \hline 0 \end{array}\right] - \frac{1}{\xi}\left[\begin{array}{c} -c_N\, s_N^T\, d_N + c_N\, R_{xx}(N+1) \\ \hline -s_N^T\, d_N + R_{xx}(N+1) \end{array}\right] \tag{18}$$

where $R_{xx}(N+1) - s_N^T\, d_N$ is a scalar. Letting

$$K_{N+1} = \frac{R_{xx}(N+1) - s_N^T\, d_N}{\xi},$$

(18) reduces to

$$d_{N+1} = \begin{bmatrix} d_N \\ -\bar{0} \end{bmatrix} - K_{N+1} \begin{bmatrix} c_N \\ -\bar{1} \end{bmatrix} \tag{19}$$

Thus, it follows that

$$d_{i,N+1} = d_{i,N} - K_{N+1}\, c_{i,N} \quad , \quad i=1,2,\ldots N \tag{20}$$

and from (13),

$$d_{i,N+1} = d_{i,N} - K_{N+1}\, d_{N+1-i,N} \, , \quad i=1,2,\ldots N \tag{21}$$

In the z-transform domain,

$$D_{N+1}(z) = D_N(z) - K_{N+1} C_N(z) \tag{22}$$

and $$D_{N+1}(z^{-1}) = D_N(z^{-1}) - K_{N+1} z^{N+1} D_N(z) \tag{23}$$

where $$D_N(z) = \sum_{i=0}^{N} d_{i,N} z^{-i} \quad \text{with } d_{0,N} = 1 \tag{24}$$

and $$C_N(z) = \sum_{i=1}^{N+1} c_{i,N} z^{-i} \quad \text{with } c_{N+1,N} = 1. \tag{25}$$

Again, from (13) we have

$$C_N(z) = z^{-(N+1)} D_N(z^{-1}). \tag{26}$$

The forward and backward prediction errors are respectively,

$$E_N(z) = X(z) D_N(z) \tag{27}$$

and $$W_N(z) = X(z)\, C_N(z). \tag{28}$$

Substitution of (27) into (22) yields

$$E_{N+1}(z) = E_N(z) - K_{N+1} W_N(z)$$

or $$e_{N+1}(n) = e_N(n) - K_{N+1}w_N(n). \tag{29}$$

Now, from (28) and (26),

$$W_{N+1}(z) = X(z)[z^{-(N+2)}D_{N+1}(z^{-1})] \tag{30}$$

Substitution of (30) into (23) results in

$$W_{N+1}(z) = z^{-(N+2)}[D_N(z^{-1}) - K_{N+1}z^{(N+1)}D_N(z)]X(z)$$

$$= z^{-1}[C_N(z) - K_{N+1}D_N(z)]X(z)$$

which leads to

$$W_{N+1}(z) = z^{-1}[W_N(z) - K_{N+1}E_N(z)]$$

or $$w_{N+1}(n+1) = w_N(n) - K_{N+1}e_N(n). \tag{31}$$

For notational convenience an auxiliary backward prediction error $\hat{w}_{N+1}(n)$ is defined as

$$\hat{w}_{N+1}(n) = \hat{w}_{N+1}(n-1) - K_{N+1}e_n(n). \tag{32}$$

From (29) and (32) the following difference equations describe the lattice predictor shown in Fig. 4.

$$x(n) = e_o(n) = \hat{w}_o(n)$$

$$e_\ell(n) = e_{\ell-1}(n) - K_\ell\hat{w}_{\ell-1}(n-1)$$

and $$\hat{w}_\ell(n) = \hat{w}_{\ell-1}(n-1) - K_\ell e_{\ell-1}(n) \;, \quad \ell = 1,2,\ldots N. \tag{33}$$

The above lattice structure can be made adaptive using several strategies [5-8] which yield time-varying methods for computing the lattice

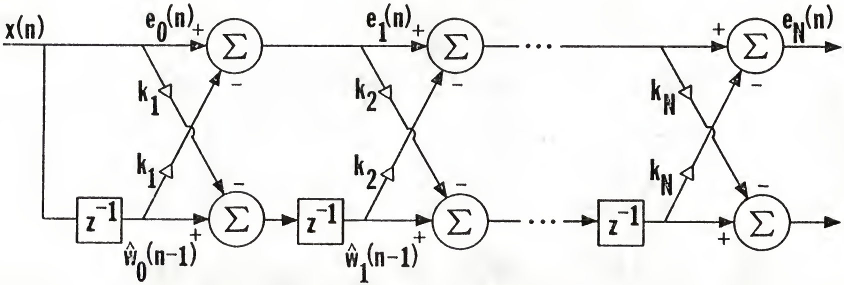

Fig. 4. One-step delay lattice predictor.

weights K_ℓ by minimizing the mean-squared prediction error. The method of steepest descent [5,6] is employed here since it involves only scalar operations and therefore keeps the computational burden to a minimum.

If the total prediction error is denoted by

$$s_\ell^2(n) = e_\ell^2(n) + \hat{w}_\ell^2(n) \ , \tag{34}$$

the lattice weights K_ℓ are updated by

$$K_\ell(n+1) = K_\ell(n) - \hat{\mu}\frac{\partial s_\ell^2(n)}{\partial K_\ell(n)} \tag{35}$$

where $K_\ell(n)$ denotes the value of K_ℓ at time n, and $\hat{\mu}$ is a convergence parameter. From (34)

$$\frac{\partial s_\ell^2(n)}{\partial K_\ell(n)} = 2e_\ell(n)\frac{\partial e_\ell(n)}{\partial K_\ell(n)} + 2\hat{w}_\ell(n)\frac{\partial \hat{w}_\ell(n)}{\partial K_\ell(n)} \ . \tag{36}$$

Additionally, (33) implies that

$$\frac{\partial e_\ell(n)}{\partial K_\ell(n)} = -\hat{w}_{\ell-1}(n-1)$$

and $$\frac{\partial w_\ell(n)}{\partial K_\ell(n)} = -e_{\ell-1}(n). \tag{37}$$

Substitution of (36) and (37) in (35) leads to

$$K_\ell(n+1) = K_\ell(n) + 2\hat{\mu}[e_\ell(n)\hat{w}_{\ell-1}(n-1) + \hat{w}_\ell(n)e_{\ell-1}(n)] \tag{38}$$

As discussed in [6], due to the successive orthogonalization and decoupling properties of the lattice structure, the convergence parameter $\hat{\mu}$ can be computed independently at each lattice stage. Moreover, the power in the forward and backward prediction error sequences decreases with each successive stage. Thus, if σ_ℓ^2 denotes the power estimate at the ℓ-th stage, it can be updated [5] using the relation

$$\sigma_\ell^2(n) = \beta\sigma_\ell^2(n-1) + (1-\beta)\ [e_\ell^2(n) + \hat{w}_\ell^2(n-1)] \tag{39}$$

where $|\beta| < 1$ is a smoothing parameter. Thus the normalized convergence parameter $\hat{\mu}$ assumes the form $\frac{\alpha}{\sigma^2(n)}$ where α is a constant, and the equation for updating the lattice coefficients becomes

$$K_\ell(n+1) = K_\ell(n) + \frac{\alpha}{\sigma_\ell^2(n)}\ [e_\ell(n)\ \hat{w}_{\ell-1}(n-1) + \hat{w}_\ell(n)e_{\ell-1}(n)]. \tag{40}$$

In practice, the above relation must be slightly modified to account for two problems. The first concerns the case when the power estimate σ_ℓ^2 is very small. Thus, division by σ_ℓ^2 in (40) could cause the algorithm to become unstable. This condition can be avoided by not updating the lattice coefficients if

$$\sigma_\ell^2(n) < \varepsilon \tag{41}$$

where ε is a small positive constant.

A second problem involves a desensitization of the predictor over the long-term as demonstrated for Widrow's LMS (least-mean-square) predictor in [12,13]. This effect is referred to as the no-pass phenomenon. It occurs when the predictor with sufficient number of coefficients first adapts to higher levels in the input, decorrelating it as much as possible. It then adapts to very low-level signal components. As such it tends to create an overall transfer function which is close to zero over a significant portion of the passband. It is this condition which eliminates noise as well as intruder stimuli. Solutions to this problem are given in [13], one of which involves a slightly modified form of the LMS algorithm. A corresponding modification can be made to the lattice predictor as follows:

$$K_\ell(n+1) = uK_\ell(n) + \frac{\alpha}{\sigma_\ell^2(n)} [e_\ell(n)\ \hat{w}_{\ell-1}(n-1) + \hat{w}_\ell(n)\ e_{\ell-1}(n)] \quad (42)$$

where u is a constant arbitrarily close to 1.

In summary, to account for both the problems cited above, (40) can be expressed as

$$K_\ell(n+1) = uK_\ell(n) + \frac{\alpha\delta}{\sigma_\ell^2(n)} [e_\ell(n)\ \hat{w}_{\ell-1}(n-1) + \hat{w}_\ell(n) e_{\ell-1}(n)] \quad (43)$$

where $\delta = 0$ if $\sigma_\ell^2(n) < \varepsilon$

and $\delta = 1$ if $\sigma_\ell^2(n) \geq \varepsilon$

CHAPTER III

ADAPTIVE THRESHOLD DETECTOR (ATD)

Since the ALP tends to remove the correlated components from input noise while passing the broadband and relatively "white" intruder signals, the ATD need only be capable of detecting intruder signals in essentially white noise.

Thus, we make the following assumptions:

1) Noise and intruder sequences have zero mean.

2) Both sequences have Gaussian distributions.

3) Successive noise and intruder samples are uncorrelated.

Then, an optimum decision rule [2] can be obtained.

Let σ_s^2 and σ_n^2 denote intruder and noise variances respectively, and e_j denote the predictor output at time j. Then the conditional probability density function given that no intruder is present, and the conditional probability density function given that an intruder is present, are respectively

$$f(e_j|0) = \frac{1}{\sqrt{2\pi}\sigma_n} e^{-e_j^2/2\sigma_n^2}, \quad e_j = n_j$$

and $$f(e_j|1) = \frac{1}{\sqrt{2\pi}\sigma} e^{-e_j^2/2\sigma^2}, \quad e_j = n_j + s_j \tag{1}$$

where $\sigma^2 = \sigma_s^2 + \sigma_n^2$.

Using a likelihood ratio approach, the decision that an intruder is present in M samples is given by

$$\sum_{j=1}^{M} \ln \frac{f(e_j|1)}{f(e_j|0)} \geq K_1 \tag{2}$$

where K_1 is a constant. Substitution of (1) in (2) leads to

$$M\ln\left\{\frac{\sigma_n}{\sigma}\right\} + \frac{1}{2}\left[\frac{1}{\sigma_n^2} - \frac{1}{\sigma^2}\right]\sum_{j=1}^{M} e_j^2 \geq K_1. \tag{3}$$

Since $\sigma^2 = \sigma_s^2 + \sigma_n^2$, (3) can be rewritten as

$$\sum_{j=1}^{M} e_j^2 \geq 2\,\sigma_n^2\left[1 + \frac{\sigma_n^2}{\sigma_s^2}\right]\left[K_1 + \frac{M}{2}\ln\frac{\sigma_s^2}{\sigma_n^2}\right]. \tag{4}$$

Note that σ_n^2/σ_s^2 is the noise-to-signal ratio at the output of the predictor. If σ_n^2/σ_s^2 is assumed to be much less than 1, then (4) becomes

$$\sum_{j=1}^{M} e_j^2 \geq 2\sigma_n^2\left[K_1 + \frac{M}{2}\ln\left\{\frac{\sigma_s^2}{\sigma_n^2}\right\}\right] \tag{5}$$

or,

$$\frac{1}{M}\sum_{j=1}^{M} e_j^2 \geq K_2\,\sigma_n^2 \tag{6}$$

where

$$K_2 = \frac{2K_1}{M} + \ln\left\{\frac{\sigma_s^2}{\sigma_n^2}\right\}.$$

Thus from (6), the optimum decision rule is to declare that an intruder is present if the variance of the predictor output sequence over a M-sample interval is greater than or equal to a fraction of the noise variance. In practice, however, the noise may only be stationary on a short-term basis. Moreover, the assumptions given above may only be approximately correct. Thus, a suboptimum decision rule is adopted, declaring that an intruder is present if

$$\frac{1}{M}\sum_{i=1}^{M} e_{j-i+1}^2 \geq \frac{K}{L}\sum_{i=1}^{L} e_{j-i-D}^2 + \Theta \qquad (7)$$

where K and Θ are constants,

$\frac{1}{M}\sum_{i=1}^{M} e_{j-i+1}^2$ is an estimate of the ALP output at time j,

and $\frac{1}{L}\sum_{j=1}^{L} e_{j-i-D}^2$ denotes the corresponding noise variance which is estimated D samples in the past, (see Fig. 5). The delay term D is introduced to minimize the error in the noise variance estimate due to the possible presence of an intruder signal. The above decision rule is referred to as the ATD algorithm.

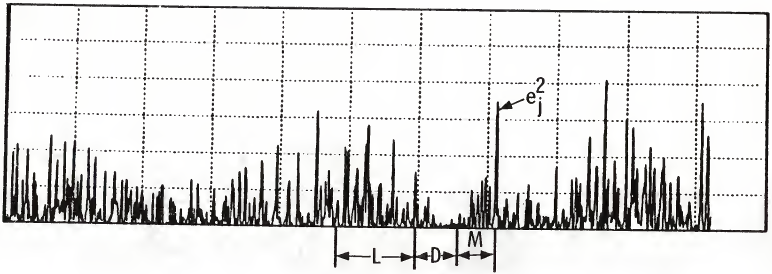

Fig. 5. Pertaining to the ATD algorithm.

CHAPTER IV

BLOCK FLOATING POINT NOTATION

Intrusion-detection algorithms have been implemented in the past [1,14] using a block floating point notation similar to that described in [15]. The basic idea is to represent numbers in the form (P/Q), where P indicates the number of integer bits, including sign, to the left of the binary point; Q indicates the number of bits of fraction to the right of the binary point such that P + Q always equals the word length. This construct allows one to keep track of the position of the binary point through arithmetic operations via a simple set of rules [14,15].

This notation has three basic limitations. First, it does not describe and therefore cannot avert the condition of arithmetic overflow. Secondly, it does not describe the degree of resolution obtainable from an arithmetic operation on two numbers which have fewer significant bits than the word length. This again implies that an underflow can not be described. Finally, it cannot conveniently represent numbers of magnitude greater than $2^{N-1}-1$ or less than 2^{-N+1} where N is the word length.

In an attempt to improve upon the limitations discussed above, an alternate block floating point notation has been developed [16].

A. Notational Definition

Numbers are represented in the form $\pm(S/I/F)E$, where:

S is the number of sign bits; I is the number of (integer) significant bits to the left of the binary point; F is the number of (fraction) bits to the right of the binary point, and E is the power-of-two exponent.

Additionally, since one may have a priori knowledge of the sign of a number, the following convention is adopted: If a number is positive, a "+" is prefixed to the above representation. Similarly, if a number is known to be negative, a "-" is prefixed. If the sign of a number is not known, there is no prefix. Further, N is defined to be the word length upon which an operator acts, typically equal to or a multiple of the machine word length. The above notation is referred to as the block floating-point (BFP) format.

A number is said to have a valid format if the following conditions are satisfied:

1) The number of sign bits S must be in the range $1 \leq S \leq N-1$.
2) The number of integer bits I must be in the range $0 \leq I \leq N-1$.
3) The number of fraction bits F must be in the range $0 \leq F \leq N-1$.
4) The power-of-two exponent E must be an integer.
5) S + I + F must be $\leq$ N.
6) I + F must be $\geq$ 1.

Some examples of valid formats are given in Table 1 for a 16 bit word length. Note that if the exponent is zero it is omitted.

TABLE 1. Examples of 16 bit representations.

Format	Representation			
(1/0/15)	S.FFF	FFFF	FFFF	FFFF
+(2/3/8)	00II	I.FFF	FFFF	F000
-(6/3/0)	1111	11II	I.000	0000
(4/0/7)-3	S.SSS	FFFF	FFF0	0000

Table 2 contains examples of invalid formats and gives the rule which is violated, for the case N = 16.

TABLE 2. Examples of rule violations.

Format	Rule violated
(5/-2/6)	I must be in the range $(0 \leq I \leq N-1)$
+(1/5/12)	$S + I + F$ must be $\leq N$
(0/2/8)	S must be in the range $(1 \leq S \leq N-1)$
(3/0/0)	$I + F$ must be > = 1

Note that the third entry in Table 2 describes the condition of arithmetic overflow, and the fourth entry describes the condition of arithmetic underflow.

B. Equivalent Formats and the Normalized Form

In the format description given above, a number can have more than one representation. For example, the format +(1/0/15) is equivalent to +(1/2/13)-2, since the binary point is located one bit from the left in the binary word in both instances, and the number of significant bits is the same. Moreover, the sign is known to be positive in both cases from the "+" prefix.

Thus, if X denotes the sign prefix, which may be "+", "-" or 0, where 0 indicates that the sign of the number is unknown, two formats are said to be equivalent if

$X1 = X2$

$S1 = S2$

$I1 + F1 = I2 + F2$

$S1 + I1 + Ei = S2 + I2 + E2$

Equivalence of formats suggests a normalized form in which the binary exponent E is minimized in absolute magnitude so as to maintain a valid and equivalent format. The rules for normalizing a format are as follows:

CASE: E1 = 0 (Format is already normalized)

CASE: E1>0

X2 = X1
S2 = S1
F2 = MAX (0,F1 - E1)
I2 = I1 + F1 - F2
E2 = E1 + I1 - I2

CASE: E1<0

X2 = X1
S2 = S1
I2 = MAX (0, I1 + E1)
F2 + F1 + I1 - I2
E2 + E1 + I1 - I2

where the function MAX selects the largest member of the function list, e.g. MAX (-3,1,2) = 2.

Examples of the normalized form for the case N = 16 are given in Table 3.

TABLE 3. Normalized form examples.

Equivalent format	Normalized form
(3/4/5)-3	(3/1/8)
(3/2/6)-3	(3/0/8)-1
(2/4/7) 3	(2/7/4)
(3/4/2) 3	(3/6/0) 1

C. Aligned Formats

In order to perform addition or subtraction of blocked floating point numbers, the binary points of the operands must be aligned within the machine word; i.e.,

$$S1 + I1 + E1 = S2 + I2 + E2.$$

Note that equivalent formats are in fact aligned but include an additional constraint in that the precision must be the same. Thus aligned formats are not necessarily equivalent.

Formats are aligned by shifting operands arithmetically left or right where, in general, a right arithmetic shift propagates copies of the sign bit into the most-significant bit of the high-order machine word. On the other hand, left arithmetic shift propagates zeroes into the least-significant bit of the low-order word. The choice of shifting operands left or right for the purpose of alignment of the binary point can be answered by the following queries:

- Will a left shift cause overflow ($S = 0$)?
- Will a right shift cause the loss of a significant bit, or even underflow ($F = 0$, $I = 0$)?

D. Arithmetic Operations

1. <u>Clipping</u>. It is sometimes useful to limit the magnitude of a number to a maximum value 2^L-1, L an integer, setting that number equal to the limit if exceeded. This allows one to perform operations on the number such as addition or subtraction without concern for overflow. In terms of the BFP format, a number is said to be clipped by M bits if the result is limited in amplitude so as to have a format which contains at least $M + 1$ sign bits. M is restricted to the range $0 \le M \le S+I+F-1$ to prevent underflow. The rules for format clipping are the following.

$$
\begin{aligned}
X2 &= X1 \\
S2 &= \mathrm{MAX}(S1,\ M+1) \\
I2 &= \mathrm{MAX}(0, I1 + S1 - S2) \\
F2 &= F1 + S1 + I1 - S2 - I2 \\
E2 &= E1 + F2 - F1
\end{aligned}
$$

Some examples are given in Table 5.

TABLE 5. Some format clipping examples.

M	Operand	Result
3	-(5/0/10)	-(5/0/10)
3	(1/5/9)-6	(4/2/9)-6
4	(1/2/8)	(5/0/6)-2

We note that the results are not necessarily in normal form.

2. Arithmetic Shifts. Such shifts can serve two functions. They can be used to align formats for subsequent arithmetic operations, or they can be used to multiply or divide numbers by integer powers of two. This last case is discussed in a later section.

For notational convenience, left and right arithmetic shifts are treated separately.

The number of left arithmetic shifts M is restricted to the range $0 \leq M \leq S-1$ where S is the number of sign bits in the operand, in order to avoid the condition of arithmetic overflow. The rules for left arithmetic shifts are the following.

X2 = X1
S2 = S1-M
I2 = I1
F2 = F1
E2 = E1

For the right arithmetic shifts, the number of shifts M is restricted to the range $0 \leq M \leq N-S1-1$, where N is the word length. The rules for right arithmetic shifts are the following

X2 = X1
S2 = S1 + M
F2 + MAX (0, MIN(N-M-S1-I1, F1))
I2 = MIN (I1, N-M-S1)
E2 = E1 + I1 - I2.

Some examples for N = 16 are shown in Table 5.

TABLE 5. Examples of right arithmetic shifts.

Direction	M	Operand	Result
L	2	(3/2/7)4	(1/2/7)4
L	1	(2/0/14)	(1/0/14)
R	3	(1/5/1)	(4/5/1)
R	3	(1/1/13)	(4/1/11)
R	3	(1/13/1)	(4/12/0)1

Note that the resulting format is not necessarily in normal form.

3. Addition and Subtraction. In order for addition or subtraction to be performed on two operands, the binary points must be aligned; i.e.,

S1 + I1 + E1 = S2 + I2 + E2.

There is a further restriction in that for addition, if the operands for addition and subtraction are not known to have the same sign, then both operands must have at least two sign bits to avoid the condition of arithmetic overflow. The rules for addition and subtraction fall under two cases:

CASE 1 Addition: operands have opposite sign (X2 = -X1 $\neq$ 0).

Subtraction: operands have same sign (X2 = X1 $\neq$ 0).

It is convenient to compute the intermediate quantities

T1 = MAX (S1-S2,0)
T2 = MAX (S2-S1,0)
U1 = MIN (I1+F1+T1, MAX (I1+E1+T1,0))
U2 = MIN (I2+F2+T2, MAX (I2+E2+T2,0))

and obtain

```
X3 = 0
S3 = MIN (S1,S2)
I3 = MAX (U1,U2)
E3 = S1 + I1 + E1 - S3 - I3
F3 = MAX (F1 - E1 + E3, F2 - E2 + E3).
```

<u>CASE 2</u> Addition: signs of operands are unknown or have same sign.

Subtraction: signs of operands are unknown or have opposite sign.

Again, the intermediate quantities can be computed as follows:

```
T1 = MAX (S1-S2,0)
T2 = MAX (S2-S1,0)
U1 = MIN (I1 + F1 + T1, MAX (I1 + E1 + T1, 0))
U2 = MIN (I2 + F2 + T2, MAX (I2 + E2 + T2, 0))
```

These results yield

```
IF X1 = X2 THEN X3 = X1 ELSE X3 = 0
S3 = MIN (S1,S2)-1
I3 = MAX (U1,U2)
E3 = S1 + I1 + E1 - S3 - I3
F3 = MAX (F1 - E1 + E3, F2 - E2 + E3).
```

Some examples are included in Table 6.

TABLE 6. Examples related to addition and subtraction.

Addition		Subtraction	
(+)	(2/0/11) (2/0/12) (1/1/12)	(-)	(2/0/11) (2/0/12) (1/1/12)
(+)	-(2/2/6) 10 +(4/1/8) 9 (2/11/0)1	(-)	-(2/2/6) 10 +(4/1/8) 9 -(1/12/0)1
(+)	-(2/2/6)-5 (4/1/8)-6 (1/0/12)-2	(-)	-(2/2/6)-5 (4/1/8)-6 (1/0/12)-2

We note that each result is always in normal form.

4. Multiplication and Rounding. Due to variations in computer hardware, the result of a multiplication can assume several forms. The hardware configuration which appears to be the most prevalent is the one in which the product of the largest positive integer that can be stored in a N-bit word, times itself, yields a 2-N bit result which contains two sign bits to the left. Any form of multiplication which does not yield this result is considered special purpose and is not discussed here.

Rules for the multiplication of formats are quite straightforward with one exception. If both the multiplier and the multiplicand exactly equal the largest possible negative integer that can be stored in the given word length N, (e.g., 8000 hexadecimal for the case N=16), the result is totally accurate, but possesses only one sign bit. Every other possible combination of values for the multiplier and multiplicand yields a result which contains at least two sign bits. In order to maintain a consistent set of rules for the multiplication of formats, the special case given above is disallowed. Thus, the rules for multiplication are

$$S3 = S1 + S2$$
$$I3 = I1 + I2$$
$$F3 = F1 + F2$$
$$E3 = E1 + E2,$$

where the sign of the result (X3) can be obtained from Table 7.

TABLE 7. Related to rules for multiplication.

X1	X2	X3
0	0	0
0	+	0
0	-	0
+	0	0
+	+	+
+	-	-
-	0	0
-	+	-
-	-	+

Again, X = 0 indicates that the sign of the operand is not known. Some examples are as follows:

$$\begin{array}{lr} & (1/0/15) \\ (\mathrm{x}) & \underline{(1/0/15)} \\ & (2/0/30) \end{array} \qquad \begin{array}{lr} & (3/2/7)3 \\ (\mathrm{x}) & \underline{-(2/0/5)-1} \\ & -(5/2/12)2 \end{array}$$

$$\begin{array}{lr} & +(2/2/11) \\ (\mathrm{x}) & \underline{(1/0/9)-1} \\ & (3/2/20)-1 \end{array} \qquad \begin{array}{lr} & -(4/5/7)1 \\ (\mathrm{x}) & \underline{-(2/5/9)-1} \\ & +(6/10/16) \end{array}$$

Note that the resultant format is of double-word length and is not necessarily in normal form.

It is often desirable to round the double-word result of a multiplication to a N-bit format. Since a product is guaranteed to contain at least two sign bits, a double-word left arithmetic shift is typically performed before rounding to minimize any loss of precision, without concern for arithmetic overflow. One may however wish to perform arithmetic operations such as addition and subtraction on the double-word product before rounding. The choice of rounding before or after an arithmetic operation is made based upon a tradeoff between speed of execution and the desired precision. Rounding usually becomes necessary before an operand is to be used in a subsequent multiplication.

At the machine level, a 2N-bit operand is rounded to N-bits by incrementing the high-order word if the most significant bit of the low-order word is a 1. Only the high-order word is retained. This operation has two problem cases. First, if the operand is of sufficiently small magnitude that after rounding, the result contains no significant information, underflow has occurred. Secondly, if the high-order word of the operand exactly equals the largest positive integer which can be represented in N bits, (e.g., 7FFF hexadecimal for the case N = 16), and the most significant bit of the low-order word is a 1, overflow will occur. Thus, in the rounding of formats, these cases must not be allowed.

Given the above restrictions, the rules for rounding formats are the following:

X2 = X1
S2 = S1
I2 = MIN (I1, N-S1)
F2 = MAX (0, MIN (N-S1-I1, F1+I1-I2))
E2 = E1+I1-I2.

Some examples are given in Table 8 for the case 2N = 32.

TABLE 8. Examples related to rounding.

Operand	Result
+(1/1/1)	+(1/1/1)
(1/1/30)	(1/1/14)
(1/30/1)	(1/15/0)15
(15/2/7)	(15/1/0)1

5. Division. Due to variations in computer hardware, the result of a division can assume several forms. The hardware configuration which appears to be the most prevalent is one which obeys the following: a double-word length dividend, divided by a single-word divisor yields a single word quotient and remainder. The sign of the remainder is the same as that of the dividend. Arithmetic overflow occurs if the magnitude of the divisor is less than the magnitude of the high-order word of the dividend. Any hardware configuration which does not adhere to these criteria is considered special-purpose and is not discussed here.

Division is a difficult operation to perform due to the persistent problem of arithmetic overflow. One must always ensure that the divisor is of sufficiently large magnitude and that the dividend is of sufficiently small magnitude as to prevent the overflow condition. Some relief can be obtained, however, if the divisor is a known constant. In this instance, one can prevent overflow by guaranteeing that the dividend has more sign bits than the divisor. This method has the advantage that it is quite easy to implement. However, it has a disadvantage in that it disallows division with a dividend having a magnitude in the range $\{|\text{divisor}| + 1\}$ to $\{|\text{divisor}| \times 2 - 1\}$, a condition which would not actually cause overflow. The result is a wasted loss of precision caused by shifting the dividend to the right a sufficient number of bits to prevent overflow to the nearest integer power of 2. Thus, if the divisor is exactly a multiple of 2, checking the number of sign bits in the dividend is optimum. In this case, however, one may wish to perform the division with arithmetic shifts, a technique discussed in a later section.

Assuming the dividend (operand 1) has a 2-N bit format, the divisor (operand 2) has a N-bit format, and noting the above restrictions concerning overflow, the rules for format division for the quotient (operand 3) and the remainder (operand 4) are respectively:

$$
\begin{aligned}
S3 &= S1-S2 \\
I3 &= MAX\ (0,\ I1-I2) \\
F3 &= N-S3-I3 \\
E3 &- E1-E2+I1-I2-I3
\end{aligned}
$$

where the sign of the quotient can be obtained from Table 9.

TABLE 9. Related to rules for division.

X1	X2	X3
0	0	0
0	+	0
0	-	0
+	0	0
+	+	+
+	-	-
-	0	0
-	+	-
-	-	+

Again,

X4 = X1
S4 = 1
I4 = 0
F4 = N-1
E4 = S1+I1+E1-N-1.

Example:

+(3/3/16)5 ÷ -(2/0/14)6 = -(1/3/12)-1 rem. +(1/0/15)-6

Note that the remainder is in normal form but the quotient is not.

6. Multiplication and division by integer powers of 2. Arithmetic shifts are performed either to align the binary point for a subsequent operation as discussed previously, or to scale an operand by an integer power of 2. In the latter case, left arithmetic shifts could be considered as multiplications and right arithmetic shifts as division, by positive integer powers of 2. For notational convenience, multiplication and division are treated separately.

In multiplication, M is restricted to the range $0 \leq M \leq S-1$, where S is the number of sign bits in the operand format, in order to avoid overflow. The rules for power-of-two multiplication of formats are the following:

X2 = X1
S2 = S1-M
I2 = MIN (I1+F1, I1+M)
F2 = MAX (0, F1+I1-I2)
E2 = E1+I1-I2+M.

Some examples are given in Table 10.

TABLE 10. Examples of multiplication.

M	Operand	Result
2	+(3/0/10)	+(1/2/8)
3	(4/0/12)-2	(1/3/9)-2

It should be mentioned that summations of numbers having the same formats, over a block of length L (which is an integer power of 2) yield a resultant format which is identical to that of multiplication of an equivalent format by $\log_2(L)$ powers of 2.

Example:

$$+(4/0/12)\mathrm{x}2^3 = +(1/3/9) = \sum_{j=1}^{8} + (4/0/12).$$

For division, M is restricted to the range $0 \leq M \leq N-S-1$ where N is the word length and S is the number of sign bits in the operand. The rules for power-of-two division of formats are the following:

X2 = X1
S2 = S1 + M
I2 = MAX (0, I1-M)
F2 = MAX (0, MIN(F1+I1-I2, N-S2-I2))
E2 = E1+I1-I2-M.

Some examples are given in Table 11.

TABLE 11. Examples related to division.

M	Operand	Result
2	+(3/0/10)	+(5/0/10)-2
3	(4/0/12)-2	(7/0/9)-5

Note that for both multiplication and division, the resultant formats are not in normal form.

7. Implementation-dependent formats. There exist sequences of arithmetic operations in which the choice of implementation can disguise the format of the final result. For example, consider a moving window summation whose window length is a multiple of 2. Then at each iteration a summation is computed over the window, a previous value is discarded and a new value

is read. This sequence of operations can be implemented in several ways, two of which are given below in a high-level computer pseudo-language.

Implementation 1.

```
1  DECLARE X(8)
2  POINTER = 1
3  DO UNTIL ENDFILE
4      READ X(POINTER)
5      SUM = 0
6      DO I = 1 TO 8
7          SUM = SUM + X(I)
8      ENDLOOP
9      WRITE SUM
10     POINTER = MOD(POINTER,8)+1
11 ENDLOOP
12 STOP
```

Implementation 2.

```
1  DECLARE X(8)
2  DO I = 1 TO 8
3      X(I) = 0
4  ENDLOOP
5  POINTER = 1
6  SUM = 0
7  DO UNTIL ENDFILE
8      READ NEWVALUE
9      SUM = SUM-X(POINTER)+NEWVALUE
10     WRITE SUM
11     X(POINTER) = NEWVALUE
12     POINTER = MOD(POINTER,8)+1
13 ENDLOOP
```

If each element of the array X is assumed to have format (4/0/12), then from implementation 1 (statements 5 through 8), the format of SUM is (1/3/12) by inspection. This format is not obvious from implementation 2 since according to the format rules for addition and subtraction, statement 9 cannot be computed repetitively without overflow. The key to solving this dilemma lies in the knowledge that each NEWVALUE is actually subtracted from SUM after some delay as an X(I).

A proposed solution to this problem is to introduce the notion of an implementation-dependent format, denoted by $\pm$[S/I/F]E where the parentheses have been replaced by square brackets. This construct is used only as a documentation aid and should not propagate through a program listing. That is, if a number has an implementation-dependent format [S/I/F]E at a particular stage in a sequence of arithmetic operations, its format in subsequent operations should be (S/I/F)E.

CHAPTER V

MICROPROCESSOR IMPLEMENTATION

A. Hardware

A microprocessor-based intrusion-detection algorithm test system was designed and constructed using two Texas Instruments 990 series development systems which feature the TMS 9900, a 16-bit NMOS microprocessor. Both processors were mounted in separate TM990-510 four-slot card cages powered by Kepco RMT 001-A switching supplies, and operate fully independently. The two card cages and power supplies are located in a 12 inch high rack-mounted drawer shown in Fig. 6.

Each 990 system contains four main items which are summarized in a tabular form below.

Manufacturer	Item	Description
Texas Instruments	TM990-100M	CPU, memory, and I/0 board
Texas Instruments	TM990-201	Memory expansion board
Analogic	ANDS 1001	A/D converter subsystem
Analogic	ANDS 2001-4	D/A converter subsystem

CPU board. The 990-100M board can accommodate up to 512 words of RAM and 4K words of EPROM memory. It contains two interval timers, 16 bits of parallel I/0, and a serial interface for EIA or TTY operation. An operating monitor called TIBUG is also provided which allows the user to modify memory and execute programs from a terminal. An optional line-by-line

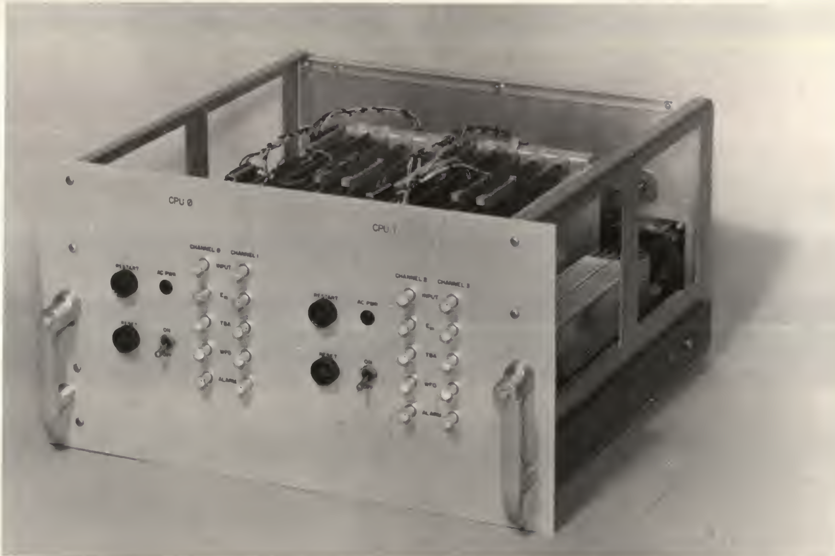

Fig. 6. Intrusion-detection algorithm test system.

assembler was incorporated in the test system to allow convenient modification of programs in the field.

System Memory. All IC sockets on the TM990-201 memory expansion board were populated which yielded 8K words of EPROM and 4K words of RAM. EPROM was mapped from memory addresses 2000 to 5FFF hexadecimal and RAM was mapped from A000 to BFFF hexadecimal.

Analog I/O. Analog-to-digital (A/D) conversion on input is performed by an Analogic ANDS 1001 subsystem which provides 16 single-ended or 8 true differential channels with up to 12 bits of resolution. This board can be configured for either sign-magnitude or two's complement representations and can operate as an I/O device or in memory-mapped mode.

Digital-to-analog (D/A) conversion on output is performed by an Analogic ANDS 2001-4 D/A subsystem which provides 4 channels with up to 12 bits of resolution. It as well can be configured for either sign-magnitude or two's complement representations and can output in several voltage ranges.

These boards were configured for the test system as summarized below.

ANDS 1001 A/D:

- $\pm$ 5 volts full scale
- 2's complement representation
- 12 bit resolution
- Memory-mapped mode
- CRU base address 03E0 hexadecimal
- Memory base address E000 hexadecimal
- Sequential channel addressing

ANDS 2001 D/A:

- $\pm$ 5 volts full scale
- 2's complement representation

- 12 bit resolution
- Memory base address E100 hexadecimal
- Sequential channel addressing

Terminal Interface. Each TI processor is interfaced to a Digital Equipment Corporation LSI-11 minicomputer which acts as a host allowing the user to communicate with any of the microprocessor systems from one terminal. Further, the LSI-11, running with floppy disks, allows TI object code programs to be loaded and stored from disk via the TIBUG paper tape load and dump routines. Finally, a processor reset feature, incorporated in the test system interface, allows the user to reset any of the processors independently under software control.

B. Software

The intrusion-detection algorithm was implemented in a dual channel configuration, that is, two algorithms per processor. Since the TI9900 is capable of executing both channels quite easily at the sampling frequency of 8 sps, modularity was stressed rather than execution speed. Further, all arithmetic operations were coded using the block floating point notation detailed in the previous chapter.

Both the adaptive lattice predictor and the adaptive threshold detector were implemented as subroutines capable of servicing two algorithm channels. This was accomplished by accessing all arrays and variables via displacements relative to a single address pointer. Thus each time a routine is invoked, a pointer is initialized which specifies the algorithm channel. A similar technique was employed for the predictor and detector initialization routines.

Input to the ADP is obtained from a subroutine which reads data samples from the A/D converter. The A/D channel number is passed as an argument which allows the same routine to service more than one algorithm. The ATD output, which is either "0" or "1", is passed to an output subroutine which pulses a hardware I/O select line corresponding to an alarm channel, if the ATD output is "1", indicating that an intruder is present. A D/A converter output subroutine is also provided to output intermediate quantities such as the ALP error for the purpose of monitoring algorithm performance in detail.

Algorithm timing is accomplished by one of the real-time clocks provided on the TM990-100M CPU board. Each clock functions as an interval timer which decrements an internal clock register at a rate of 1/64th the system clock frequency, and causes an interrupt when the register decrements to zero. Thus, with the interval timer programmed to interrupt every 0.125 seconds, the intrusion-detection algorithm operates at 8 Hertz.

The timer interrupt service routine which calls the ALP, the ATD, the attendant I/O, and the support routines, constitutes the main program shell as depicted in Fig. 7. A source listing for the intrusion-detection algorithm is given in Appendix B.

C. Experimental Results

A data sequence consisting of intruder signals embedded in noise caused by a nearby train was processed by the ALP-ATD combination. The resulting ALP and ATD outputs were recorded. The results are shown in Fig. 8.

The upper trace shows the input data sequence which contains five intruder crossings, indicated by the symbol "↑". The corresponding ALP

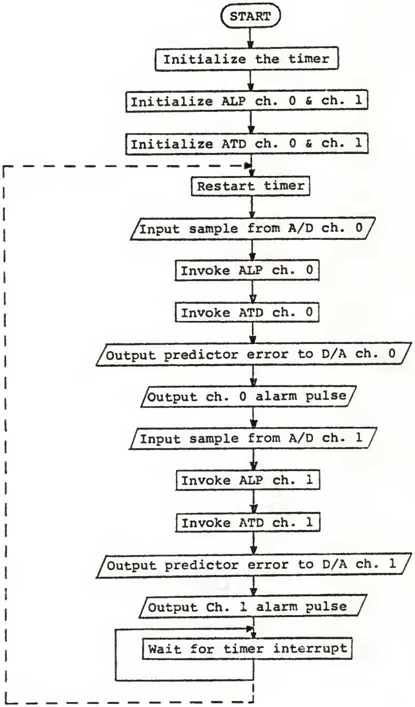

Fig. 7. Flowchart representation of intrusion-detection algorithm.

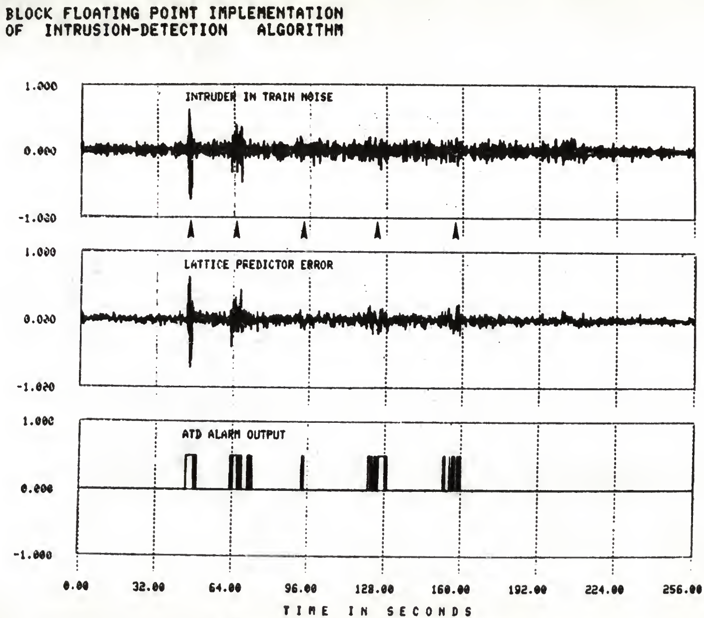

Fig. 8. Input and corresponding output.

output shown in the center trace indicates an apparent increase in the signal-to-noise ratio. More notably, a burst of noise which occurs after 200 seconds and having an amplitude at least as great as the last intruder crossing, is very effectively removed. This is reinforced by the output of the ATD which detected the five intruder crossings but did not generate a false alarm on the noise burst.

CHAPTER VI

CONCLUDING REMARKS

The feasibility of implementing an intrusion-detection algorithm using an ALP on a 16-bit microprocessor using block floating point arithmetic was demonstrated. The author feels that the ALP-ATD combination may prove useful in medical instrumentation, radar tracking, and a variety of other signal processing applications. Moreover, the block floating point notation described in Chapter IV may be a forebear to a language for digital signal processors.

Future efforts in the area of signal processing will concern experimentation with alternate lattice structures and the exploitation of frequency information in the detection algorithm.

REFERENCES

[1] N. Ahmed, R.J. Fogler, D.L. Soldan, G.R. Elliott, and N.A. Bourgeois, "On an Intrusion-Detection Approach Via Adaptive Prediction," IEEE Trans. Aerospace and Electronic Systems, May 1979, pp. 430-436.

[2] M. Schwartz and L. Shaw, Signal Processing: Discrete Spectral Analysis, Detection, and Estimation, McGraw-Hill, 1975, pp. 260-263.

[3] A. Papoulis, Signal Analysis, McGraw-Hill, New York, N.Y., 1977.

[4] J.D. Markel and A.H. Gray, Jr., "Roundoff noise characteristics of a class of orthogonal polynomial structures," IEEE Trans. Acoust., Speech, and Sig. Proc., ASSP-23, Oct. 1975, pp. 473-486.

[5] L.J. Griffiths, "An adaptive lattice structure for noise-cancelling applications," Proc. ICASSP, Apr. 1978, Tulsa, OK., pp. 87-90.

[6] L.J. Griffiths, "A continuously-adaptive filter implemented as a lattice structure," Proc. ICASSP, May 1977, Hartford, CT., pp. 683-686.

[7] J. Makhoul, "A class of all-zero lattice digital filters: Properties and applications," IEEE Trans. Acoust., Speech, and Sig. Proc., ASSP-26, Aug. 1978, pp. 304-314.

[8] J. Makhoul, "Stable and efficient lattice methods for linear prediction," IEEE Trans. Acoust., Speech, and Sig. Proc., ASSP-25, pp. 423-428.

[9] N. Ahmed and R.J. Fogler, "On an adaptive lattice predictor and a related application," to be published in IEEE Circuits and Systems Magazine, Dec. 1979.

[10] V.N. Faddeeva, Computational Methods of Linear Algebra, Dover Publications, 1959, pp. 111-117.

[11] D.K. Faddeev and V.N. Faddeeva, Computational Methods of Linear Algebra, W.H. Freeman and Co., 1963, pp. 168-171.

[12] N. Ahmed, G.R. Elliot, and S.D. Stearns, "Long-term instability problems in adaptive noise cancellers," Sandia Laboratories, Technical Report No. SAND 78-1032, Aug. 1978; available from the National Technical Information Service, 5285 Port Royal Road, Springfield, VA. 22161.

[13] N. Ahmed, G.R. Elliot, and S.D. Stearns, "Long-term stability considerations of adaptive predictors," Proc. 12th Asilomar Conf. on Circuits and Systems, Asilomar, CA., Nov. 1978, pp. 180-183.

[14] N.A. Bourgeois, Jr., "Data acquisition and test system software," Technical Report - No. SAND 78-1972, March 1979, pp. 21-36.

[15] D.D. McCracken, Digital Computer Programming, John Wiley and Sons, New York, N.Y., March 1963.

[16] J.E. Simpson and R.J. Fogler, "A block floating notation for digital signal processes," to be published in a Sandia Laboratories technical report.

ACKNOWLEDGMENTS

The author would like to thank the Sandia Laboratories, Albuquerque, N.M. for providing the financial support, data, and computing facilities used in connection with this thesis. The assistance and encouragement given by Jim Simpson who originated the Block Floating Point notation, Glenn Elliot, Nick Bourgeois, and Dick Wayne of Sandia Laboratories, is gratefully acknowledged.

The author would also like to express his appreciation to Dr. Donald R. Hummels and Dr. J. David Logan for serving as graduate committee members and for sharing their knowledge and experience. A special thanks is extended to my major advisor, Dr. Nasir Ahmed whose teachings both in and out of the classroom have been of immeasurable value.

APPENDIX A

A Block Floating Point Format

Tutorial Program

```
      PROGRAM FORMAT
C
C BLOCK FLOATING POINT TUTORIAL PROGRAM
C
C JOE FOGLER    08/21/79
C REV. 01.00    08/23/79
C
      LOGICAL*1 INSTR(81),SPACE
      DATA INSTR(81),SPACE /0,' '/
      EQUIVALENCE (INSTR(1),RSTR)
      INTEGER X,S,I,F,E
      COMMON /WSIZE/ NW,NBW,N
    1 FORMAT(' ENTER MACHINE WORD SIZE: ',$)
    2 FORMAT('0COMMAND: ',$)
    3 FORMAT(80A1)
    4 FORMAT('0LIST OF COMMANDS:'//
     *'  HELP   TYPES OUT THE LIST OF COMMANDS'/
     *'  QUIT   TERMINATES THE FORMAT PROGRAM'/
     *'  SIZE   CHANGES THE MACHINE WORD SIZE'/
     *'  EQU    DETERMINES IF FORMATS ARE EQUIVALENT'/
     *'  NORM   COMPUTES NORMALIZED FORMAT'/
     *'  ALIG   DETERMINES IF TWO FORMATS ARE ALIGNED'/
     *'  CLIP   CLIPS A FORMAT'/
     *'  ASHL   LEFT ARITHMETIC SHIFT'/
     *'  ASHR   RIGHT ARITHMETIC SHIFT'/
     *'  ADD    ADDITION OF FORMATS'/
     *'  SUB    SUBTRACTION OF FORMATS'/
     *'  MPY    MULTIPLICATION OF FORMATS'/
     *'  DIV    DIVISION OF FORMATS'/
     *'  MPY2   MULTIPLICATION BY POWERS OF 2'/
     *'  DIV2   DIVISION BY POWERS OF 2'/
     *'  RND    ROUND A FORMAT'/)
      TYPE 4
  100 TYPE 1
      ACCEPT *,NBW
      NW = 1
      N = NW*NBW
  110 TYPE 2
      ACCEPT 3,(INSTR(J),J=1,80)
      IF (RSTR.EQ.'ASHL') CALL ASHL
      IF (RSTR.EQ.'ASHR') CALL ASHR
      IF (RSTR.EQ.'MPY2') CALL MPY2
      IF (RSTR.EQ.'DIV2') CALL DIV2
      IF (INSTR(4).NE.'2') INSTR(4) = SPACE
      IF (RSTR.EQ.'HEL ') TYPE 4
      IF (RSTR.EQ.'QUI ') GO TO 120
      IF (RSTR.EQ.'SIZ ') GO TO 100
      IF (RSTR.EQ.'EQU ') CALL EQV
      IF (RSTR.EQ.'NOR ') CALL NORM
      IF (RSTR.EQ.'ALI ') CALL ALIGN
      IF (RSTR.EQ.'CLI ') CALL CLIP
      IF (RSTR.EQ.'ADD ') CALL ADD
      IF (RSTR.EQ.'SUB ') CALL SUB
      IF (RSTR.EQ.'MPY ') CALL MPY
      IF (RSTR.EQ.'DIV ') CALL DIV
      IF (RSTR.EQ.'RND ') CALL ROUND
      GO TO 110
  120 STOP 'PROGRAM TERMINATED'
      END
```

```
      SUBROUTINE EQV
      INTEGER X1,S1,I1,F1,E1
      INTEGER X2,S2,I2,F2,E2
C
C ROUTINE TO TEST EQUIVALENCE OF FORMATS
C
C JOE FOGLER    08/22/79
C REV. 01.00    08/23/79
C
    1 FORMAT(' EQU: ENTER THE FORMATS:')
    2 FORMAT(' FORMATS ARE EQUIVALENT')
    3 FORMAT(' FORMATS ARE NOT EQUIVALENT')
C
      TYPE 1
      CALL GET(X1,S1,I1,F1,E1)
      CALL GET(X2,S2,I2,F2,E2)
C
      IF (S1.NE.S2) GO TO 100
      IF (I1+F1.NE.I2+F2) GO TO 100
      IF (S1+I1+E1.NE.S2+I2+E2) GO TO 100
      IF (X1.NE.X2) GO TO 100
      TYPE 2
      GO TO 110
  100 TYPE 3
  110 RETURN
      END
```

```
      SUBROUTINE NORM
      INTEGER X1,S1,I1,F1,E1
      INTEGER X2,S2,I2,F2,E2
C
C ROUTINE TO NORMALIZE TWO FORMATS
C
C JOE FOGLER   08/22/79
C REV. 01.00   08/23/79
C
    1 FORMAT(' NORM: ENTER THE FORMAT')
C
      TYPE 1
      CALL GET(X1,S1,I1,F1,E1)
C
      CALL NORML(X1,S1,I1,F1,E1)
C
      CALL PUT(X1,S1,I1,F1,E1)
      RETURN
      END
```

```
      SUBROUTINE NORML(X1,S1,I1,F1,E1)
      INTEGER X1,S1,I1,F1,E1
      INTEGER X2,S2,I2,F2,E2
C
C NORMALIZATION SUPPORT SUBROUTINE
C
C JOE FOGLER    08/22/79
C REV. 01.00    08/23/79
C
C
      IF (E1.EQ.0) GO TO 110
C
      IF (E1.LT.0) GO TO 100
C
      F2 = MAX0(0,F1-E1)
      I2 = I1 + F1 - F2
      E1 = E1 + I1 - I2
      I1 = I2
      F1 = F2
      GO TO 110
C
  100 I2 = MAX0(0,I1+E1)
      F2 = I1 + F1 - I2
      E1 = E1 + I1 - I2
      I1 = I2
      F1 = F2
C
  110 RETURN
      END
```

```
      SUBROUTINE ALIGN
      INTEGER X1,S1,I1,F1,E1
      INTEGER X2,S2,I2,F2,E2
C
C ROUTINE TO NORMALIZE TWO FORMATS
C
C JOE FOGLER    08/22/79
C REV. 01.00    08/23/79
C
    1 FORMAT(' ALIG: ENTER THE FORMATS')
    2 FORMAT(' ARG ',I1,' EXCEEDS ARG ',I1,' BY ',I2,' POWERS OF 2')
    3 FORMAT(' FORMATS ARE ALIGNED')
C
      TYPE 1
      CALL GET(X1,S1,I1,F1,E1)
      CALL GET(X2,S2,I2,F2,E2)
C
      J1 = S1 + I1 + E1
      J2 = S2 + I2 + E2
      IF (J1.EQ.J2) TYPE 3
      IF (J1.GT.J2) TYPE 2,1,2,J1-J2
      IF (J1.LT.J2) TYPE 2,2,1,J2-J1
C
      RETURN
      END
```

```
      SUBROUTINE CLIP
      INTEGER X1,S1,I1,F1,E1
      INTEGER X2,S2,I2,F2,E2
C
C ROUTINE TO CLIP A FORMAT
C
C JOE FOGLER    08/22/79
C REV. 01.00    08/23/79
C
    1 FORMAT(' CLIP: HOW MANY BITS? ',$)
    2 FORMAT(' 1 <= NBITS <= ',I1)
C
      TYPE 1
      ACCEPT *,M
      CALL GET(X1,S1,I1,F1,E1)
C
      J1 = S1 + I1 + F1 - 1
      IF (M.LT.1 .OR. M.GT.J1) GO TO 100
      X2 = X1
      S2 = MAX0(S1,M+1)
      I2 = MAX0(0,I1+S1-S2)
      F2 = F1 + S1 + I1 - S2 - I2
      E2 = E1 + F2 - F1
      CALL NORML(X2,S2,I2,F2,E2)
      CALL PUT(X2,S2,I2,F2,E2)
      GO TO 120
  100 TYPE 2,J1
  120 RETURN
      END
```

```
      SUBROUTINE ASHL
      INTEGER X1,S1,I1,F1,E1
      INTEGER X2,S2,I2,F2,E2
C
C ROUTINE TO LEFT SHIFT A FORMAT
C
C JOE FOGLER   08/22/79
C REV. 01.00   08/23/79
C
    1 FORMAT(' ASHL: HOW MANY BITS? ',S)
    2 FORMAT(' ',I2,' SHIFTS WILL CAUSE OVERFLOW')
C
      TYPE 1
      ACCEPT *,M
      IF (M.LT.1) GO TO 120
      CALL GET(X1,S1,I1,F1,E1)
C
      IF (M.GT.S1-1) GO TO 100
      X2 = X1
      S2 = S1 - M
      I2 = I1
      F2 = F1
      E2 = E1
      CALL NORML(X2,S2,I2,F2,E2)
      CALL PUT(X2,S2,I2,F2,E2)
      GO TO 120
  100 TYPE 2,M
  120 RETURN
      END
```

```
      SUBROUTINE ASHR
      COMMON /WSIZE/ NW,NBW,N
      INTEGER X1,S1,I1,F1,E1
      INTEGER X2,S2,I2,F2,E2
C
C ROUTINE TO RIGHT SHIFT A FORMAT
C
C JOE FUGLER   08/22/79
C REV. 01.00   08/23/79
C
    1 FORMAT(' ASHR: HOW MANY BITS? ',$)
    2 FORMAT(' ',I2,' SHIFTS WILL CAUSE UNDERFLOW')
C
      TYPE 1
      ACCEPT *,M
      IF (M.LT.1) GO TO 120
      CALL GET(X1,S1,I1,F1,E1)
C
      IF (M.GE.N-S1) GO TO 100
      X2 = X1
      S2 = S1 + M
      F2 = MAX0(0,MIN0(N-M-S1-I1,F1))
      I2 = MIN0(I1,N-M-S1)
      E2 = E1 + I1 - I2
C
      CALL NORML(X2,S2,I2,F2,E2)
      CALL PUT(X2,S2,I2,F2,E2)
      GO TO 120
  100 TYPE 2,M
  120 RETURN
      END
```

```
      SUBROUTINE ADD
      INTEGER X1,S1,I1,F1,E1,T1,U1
      INTEGER X2,S2,I2,F2,E2,T2,U2
      INTEGER X3,S3,I3,F3,E3
C
C ROUTINE TO ADD TWO FORMATS
C
C JOE FOGLER    08/22/79
C REV. 01.00    08/23/79
C
    1 FORMAT(' ADD: COMPUTES ARG1 + ARG2')
    2 FORMAT(' FORMATS ARE NOT ALIGNED')
    3 FORMAT(' OVERFLOW POSSIBLE')
C
      TYPE 1
      CALL GET(X1,S1,I1,F1,E1)
      CALL GET(X2,S2,I2,F2,E2)
      IF (S1+I1+E1 .NE. S2+I2+E2) GO TO 120
      T1 = MAX0(S1-S2,0)
      T2 = MAX0(S2-S1,0)
C
      IF (X1.EQ.0) GO TO 100
      IF (X1.NE.-X2) GO TO 100
C
C ARGUMENTS HAVE OPPOSITE EXPLICIT SIGNS
C
      IF (S1.LT.1 .OR. S2.LT.1) GO TO 130
C
      U1 = MIN0(I1+F1+T1,MAX0(I1+E1+T1,0))
      U2 = MIN0(I2+F2+T2,MAX0(I2+E2+T2,0))
      X3 = 0
      S3 = MIN0(S1,S2)
      I3 = MAX0(U1,U2)
      E3 = S1 + I1 + E1 - S3 - I3
      F3 = MAX0(F1 - E1 + E3,F2 - E2 + E3)
      GO TO 110
C
C ARGUMENTS DO NOT HAVE OPPOSITE EXPLICIT SIGNS
C
  100 IF (S1.LT.2 .OR. S2.LT.2) GO TO 130
C
      U1 = MIN0(I1+F1+T1+1,MAX0(I1+E1+T1+1,0))
      U2 = MIN0(I2+F2+T2+1,MAX0(I2+E2+T2+1,0))
      X3 = 0
      IF (X1.EQ.X2) X3 = X1
      S3 = MIN0(S1,S2) - 1
      I3 = MAX0(U1,U2)
      E3 = S1 + I1 + E1 - S3 - I3
      F3 = MAX0(F1-E1+E3,F2-E2+E3)
C
  110 CALL PUT(X3,S3,I3,F3,E3)
      GO TO 140
  120 TYPE 2
      GO TO 140
  130 TYPE 3
  140 RETURN
      END
```

```
      SUBROUTINE SUB
      INTEGER X1,S1,I1,F1,E1,T1,U1
      INTEGER X2,S2,I2,F2,E2,T2,U2
      INTEGER X3,S3,I3,F3,E3
C
C ROUTINE TO SUB TWO FORMATS
C
C JOE FOGLER    08/22/79
C REV. 01.00    08/23/79
C
    1 FORMAT(' SUB: COMPUTES ARG1 - ARG2')
    2 FORMAT(' FORMATS ARE NOT ALIGNED')
    3 FORMAT(' OVERFLOW POSSIBLE')
C
      TYPE 1
      CALL GET(X1,S1,I1,F1,E1)
      CALL GET(X2,S2,I2,F2,E2)
      IF (S1+I1+E1 .NE. S2+I2+E2) GO TO 120
      T1 = MAX0(S1-S2,0)
      T2 = MAX0(S2-S1,0)
C
      IF (X1.EQ.0) GO TO 100
      IF (X1.NE.X2) GO TO 100
C
C ARGUMENTS HAVE SAME EXPLICIT SIGNS
C
      IF (S1.LT.1 .OR. S2.LT.1) GO TO 130
      U1 = MIN0(I1+F1+T1,MAX0(I1+E1+T1,0))
      U2 = MIN0(I2+F2+T2,MAX0(I2+E2+T2,0))
C
      X3 = 0
      S3 = MIN0(S1,S2)
      I3 = MAX0(U1,U2)
      E3 = S1 + I1 + E1 - S3 - I3
      F3 = MAX0(F1 - E1 + E3,F2 - E2 + E3)
      GO TO 110
C
C ARGUMENTS DO NOT HAVE SAME EXPLICIT SIGNS
C
  100 IF (S1.LT.2 .OR. S2.LT.2) GO TO 130
      U1 = MIN0(I1+F1+T1+1,MAX0(I1+E1+T1+1,0))
      U2 = MIN0(I2+F2+T2+1,MAX0(I2+E2+T2+1,0))
      X3 = 0
      IF (X1.EQ.X2) X3 = X1
      S3 = MIN0(S1,S2) - 1
      I3 = MAX0(U1,U2)
      E3 = S1 + I1 + E1 - S3 - I3
      F3 = MAX0(F1-E1+E3,F2-E2+E3)
C
  110 CALL PUT(X3,S3,I3,F3,E3)
      GO TO 140
  120 TYPE 2
      GO TO 140
  130 TYPE 3
  140 RETURN
      END
```

```
      SUBROUTINE MPY
      COMMON /WSIZE/ NW,NBW,N
      INTEGER X1,S1,I1,F1,E1,T1,U1
      INTEGER X2,S2,I2,F2,E2,T2,U2
      INTEGER X3,S3,I3,F3,E3
C
C ROUTINE TO MULTIPLY TWO FORMATS
C
C JOE FOGLER    08/22/79
C REV. 01.00    08/23/79
C
    1 FORMAT(' MPY: COMPUTES ARG1 X ARG2')
    2 FORMAT(' WORD SIZE IS NOW ',I3,' BITS')
    3 FORMAT(' WARNING -- BOTH CANNOT BE LARGEST NEGATIVE NUMBER')
C
      TYPE 1
      CALL GET(X1,S1,I1,F1,E1)
      CALL GET(X2,S2,I2,F2,E2)
C
      X3 = 0
      IF (X1.EQ.0 .OR. X2.EQ.0) GO TO 100
      IF (X1.EQ.X2) X3 = 1
      IF (X1.NE.X2) X3 = -1
C
  100 S3 = S1 + S2
      I3 = I1 + I2
      F3 = F1 + F2
      E3 = E1 + E2
      CALL NORML(X3,S3,I3,F3,E3)
      CALL PUT(X3,S3,I3,F3,E3)
C
      IF (S1.EQ.1.AND.S2.EQ.1.AND.X1.NE.1.AND.X2.NE.1) TYPE 3
C
      NW = 2
      N = NBW*NW
      TYPE 2,N
C
      RETURN
      END
```

```
      SUBROUTINE DIV
      COMMON /NSIZE/ NW,NBW,N
      INTEGER X1,S1,I1,F1,E1,T1,U1
      INTEGER X2,S2,I2,F2,E2,T2,U2
      INTEGER X3,S3,I3,F3,E3
C
C ROUTINE TO DIVIDE TWO FORMATS
C
C JOE FOGLER    08/22/79
C REV. 01.00    08/23/79
C
    1 FORMAT(' DIV: COMPUTES ARG1 / ARG2')
    2 FORMAT(' OVERFLOW POSSIBLE')
    3 FORMAT(' WORD SIZE IS NOW ',I3,' BITS')
    4 FORMAT(' WARNING - OVERFLOW CONDITIONS MUST BE CHECKED')
C
      TYPE 1
      N = 2*NBW
      CALL GET(X1,S1,I1,F1,E1)
      NW = 1
      N = NW*NBW
      CALL GET(X2,S2,I2,F2,E2)
C
      IF (S1.LE.S2) GO TO 120
C
      X3 = 0
      IF (X1.EQ.0 .OR. X2.EQ.0) GO TO 100
      IF (X1.EQ.X2) X3 = 1
      IF (X1.NE.X2) X3 = -1
C
  100 S3 = S1 - S2
      I3 = MAX0(0,I1-I2)
      F3 = N - S3 - I3
      E3 = E1 - E2 + I1 - I2 - I3
      CALL NORML(X3,S3,I3,F3,E3)
      CALL PUT(X3,S3,I3,F3,E3)
      TYPE 4
      TYPE 3,N
      GO TO 130
C
  120 TYPE 2
C
  130 RETURN
      END
```

```
      SUBROUTINE MPY2
      COMMON /WSIZE/ NW,NBW,N
      INTEGER X1,S1,I1,F1,E1
      INTEGER X2,S2,I2,F2,E2
C
C ROUTINE TO MULTIPLY A FORMAT
C BY AN INTEGER POWER OF 2.
C
C JOE FOGLER    08/22/79
C REV. 01.00    08/23/79
C
    1 FORMAT(' MPY2: HOW MANY POWERS OF 2? ',$)
    2 FORMAT(' ARG X 2**',I2,' WILL CAUSE OVERFLOW')
C
      TYPE 1
      ACCEPT *,M
      IF (M.LT.1) GO TO 120
      CALL GET(X1,S1,I1,F1,E1)
C
      IF (M.GT.S1-1) GO TO 100
      X2 = X1
      S2 = S1 - M
      I2 = MIN0(I1+F1,I1+M)
      F2 = MAX0(0,F1+I1-I2)
      E2 = E1 + I1 - I2 + M
C
      CALL NORML(X2,S2,I2,F2,E2)
      CALL PUT(X2,S2,I2,F2,E2)
      GO TO 120
C
  100 TYPE 2,M
  120 RETURN
      END
```

```
      SUBROUTINE DIV2
      COMMON /WSIZE/ NW,NBW,N
      INTEGER X1,S1,I1,F1,E1
      INTEGER X2,S2,I2,F2,E2
C
C ROUTINE TO DIVIDE A FORMAT
C BY AN INTEGER POWER OF 2.
C
C JOE FOGLER    08/22/79
C REV. 01.00    08/23/79
C
    1 FORMAT(' DIV2: HOW MANY POWERS OF 2? ',$)
    2 FORMAT(' ARG / 2**',I2,' WILL CAUSE UNDERFLOW')
C
      TYPE 1
      ACCEPT *,M
      IF (M.LT.1) GO TO 120
      CALL GET(X1,S1,I1,F1,E1)
C
      IF (M.GE.N-S1) GO TO 100
      X2 = X1
      S2 = S1 + M
      I2 = MAX0(0,I1-M)
      F2 = MAX0(0,MIN0(F1+I1-I2,N-S2-I2))
      E2 = E1 + I1 - I2 - M
C
      CALL NORML(X2,S2,I2,F2,E2)
      CALL PUT(X2,S2,I2,F2,E2)
      GO TO 120
C
  100 TYPE 2,M
  120 RETURN
      END
```

```
      SUBROUTINE ROUND
      COMMON /WSIZE/ NW,NBW,N
      INTEGER X1,S1,I1,F1,E1
      INTEGER X2,S2,I2,F2,E2
C
C ROUTINE TO ROUND A DOUBLE LENGTH
C OPERAND TO SINGLE WORD LENGTH
C
C JOE FOGLER    08/22/79
C REV. 01.00    08/23/79
C
    1 FORMAT(' RND: ENTER THE FORMAT')
    2 FORMAT(' WARNING -- FORMAT OVERFLOW MUST BE CHECKED')
    3 FORMAT(' WORD SIZE IS NOW ',I3,' BITS')
C
      NW = 2
      N = NW*NBW
C
      TYPE 1
      CALL GET(X1,S1,I1,F1,E1)
C
      NW = 1
      N = NW*NBW
C
      X2 = X1
      S2 = S1
      I2 = MIN0(I1,N-S1)
      F2 = MAX0(0,MIN0(N-S1-I1,F1+I1-I2))
      E2 = E1 + I1 - I2
C
      CALL PUT(X2,S2,I2,F2,E2)
      IF (X1.NE.-1) TYPE 2
      TYPE 3,N
C
      RETURN
      END
```

```
      SUBROUTINE PUT(X,S,I,F,E)
C
C SUBROUTINE TO WRITE A FORMAT
C
C JOE FOGLER    08/21/79
C REV. 01.00    08/23/79
C
      INTEGER X,S,I,F,E
      LOGICAL*1 STR1(4)
      DATA STR1 /'-',' ','+',' '/
C
    1 FORMAT(' =',1A1,'(',I2,'/',I2,'/',I2,')',I3)
    2 FORMAT(' =',1A1,'(',I2,'/',I2,'/',I2,')')
C
      IF (E.NE.0) TYPE 1,STR1(X+2),S,I,F,E
      IF (E.EQ.0) TYPE 2,STR1(X+2),S,I,F
      RETURN
      END
```

```
      SUBROUTINE GET(X,S,I,F,E)
C
C GET FORMAT SUBROUTINE
C
C JOE FOGLER    08/21/79
C REV. 01.00    08/23/79
C
      INTEGER X,S,I,F,E,ES
      LOGICAL*1 STR(81)
      LOGICAL*1 SPACE,DELIM
      DATA SPACE /' '/
      COMMON /WSIZE/ NW,NBW,N
C
    1 FORMAT(' >',S)
    2 FORMAT(80A1)
    3 FORMAT(' ')
    4 FORMAT(1H+,1A1,S)
    5 FORMAT(' PRECISION EXCEEDS',I3)
    6 FORMAT(' PRECISION UNDEFINED')
    7 FORMAT(' NOT ENOUGH SIGN BITS')
    8 FORMAT(' INVALID FORMAT')
C
   90 E = 0
      X = 0
      DO 95 I=1,81
         STR(I) = 0
   95 CONTINUE
      TYPE 1
      ACCEPT 2,(STR(J),J=1,80)
      INDX = 1
  100 IF (STR(INDX).NE.SPACE) GO TO 110
      INDX = INDX + 1
      IF (INDX.GT.80) GO TO 206
      GO TO 100
C
  110 IF (STR(INDX).NE.'+') GO TO 120
      X = 1
      INDX = INDX + 1
      GO TO 130
  120 IF (STR(INDX).NE.'-') GO TO 130
      X = -1
      INDX = INDX + 1
  130 IF (STR(INDX).NE.SPACE) GO TO 135
      INDX = INDX + 1
      IF (INDX.GT.80) GO TO 206
      GO TO 130
  135 IF (STR(INDX).NE.'(') GO TO 200
      INDX = INDX + 1
      CALL CONV(STR,INDX,'/',S,IERR)
      IF (IERR.NE.0) GO TO 200
      INDX = INDX + 1
      CALL CONV(STR,INDX,'/',I,IERR)
      IF (IERR.NE.0) GO TO 200
      INDX = INDX + 1
      CALL CONV(STR,INDX,')',F,IERR)
      IF (IERR.NE.0) GO TO 200
      INDX = INDX + 1
```

```
C
  150 IF (STR(INDX).NE.SPACE) GO TO 160
      INDX = INDX + 1
      IF (INDX.GT.80) GO TO 185
      GO TO 150
C
  160 ES = 0
      IF (STR(INDX).EQ.'+') GO TO 170
      IF (STR(INDX).NE.'-') GO TO 180
      ES = 1
  170 INDX = INDX + 1
  180 CALL CONV(STR,INDX,SPACE,E,IERR)
      IF (IERR.NE.0) E = 0
      IF (ES.NE.0) E = -E
C
  185 IF (I.EQ.0 .AND. F.EQ.0) GO TO 220
      IF (S.LT.1) GO TO 230
      IF (I.GT.N-1) GO TO 210
      IF (F.GT.N-1) GO TO 210
      IF (I+F.LT.1) GO TO 230
      IF (S+I+F.GT.N) GO TO 210
      RETURN
C
  200 IF (INDX.LE.0 .OR. INDX.GT.80) GO TO 206
      TYPE 3
      DO 205 K=1,INDX
         TYPE 4,' '
  205 CONTINUE
      TYPE 4,'^'
      TYPE 3
  206 TYPE 8
      GO TO 90
  210 TYPE 5,N
      GO TO 90
  220 TYPE 6
      GO TO 90
  230 TYPE 7
      GO TO 90
      END
```

```
      SUBROUTINE CONV(STR,INDX,DELIM,IVAL,IERR)
C
C ROUTINE TO CONVERT AN ASCII STRING TO AN INTEGER
C
C JOE FOGLER    08/21/79
C REV. 01.00    08/23/79
C
      LOGICAL*1 STR(81),DELIM,TABL(10),SPACE,DUMMY
      DATA TABL /'0','1','2','3','4','5','6','7','8','9'/
      DATA SPACE /' '/
C
      IERR = 1
      IVAL = 0
  100 IF (STR(INDX).NE.SPACE) GO TO 110
      INDX = INDX + 1
      IF (INDX.GT.80) GO TO 140
      GO TO 100
C
  110 DO 120 I=1,10
      J = I - 1
      IF (STR(INDX).EQ.TABL(I)) GO TO 130
  120 CONTINUE
  125 IF (STR(INDX).NE.SPACE) GO TO 135
      INDX = INDX + 1
      IF (INDX.GT.80) GO TO 135
      GO TO 125
C
  130 IERR = 0
      IVAL = 10*IVAL + J
      INDX = INDX + 1
      IF (INDX.LE.80) GO TO 110
      IERR = 1
      GO TO 140
C
  135 IF (STR(INDX).NE.DELIM.AND.DELIM.NE.SPACE) IERR = 1
  140 RETURN
      END
```

APPENDIX B

Intrusion-Detection Algorithm

Source Listing

```
        IDT     'LMSALP'
*
* LEAST MEAN SQUARE ADAPTIVE LATTICE PREDICTOR
*      WITH ADAPTIVE THRESHOLD DETECTION
*
* JOE FOGLER    08/13/79
* REV. 02.03    09/05/79
*
* DELCARATIONS
*
ADC0    EQU     >E000           ;A/D CONVERTER CHANNEL 0
DAC0    EQU     >E100           ;D/A CONVERTER CHANNEL 0
ADCCRU  EQU     >03E0           ;A/D CRU BASE ADDRESS
ADMODE  EQU     >000C           ;A/D MEMORY-MAP MODE BIT #

TMRCRU  EQU     >0100           ;TIMER CRU BASE ADDRESS
TMRCNT  EQU     >17C1           ;TIMER COUNT FOR 8HZ INTERRUPT
TMRENB  EQU     >0003           ;TIMER ENABLE BIT #
TMMODE  EQU     >0000           ;TIMER INTERRUPT MODE BIT #
TMRVEC  EQU     >FF88           ;TIMER VECTOR ADDRESS
TMRMSK  EQU     >0003           ;TIMER INTERRUPT MASK
BRANCH  EQU     >0460           ;BRANCH INSTRUCTION

SELCRU  EQU     >0000           ;SEL CRU BASE ADDRESS
SEL1    EQU     >0000           ;SEL1 DISPLACEMENT
SEL2    EQU     >0020           ;SEL2 DISPLACEMENT

WSP0    EQU     >A000           ;WORKSPACE 0
WSP1    EQU     >A020           ;WORKSPACE 1
WSP2    EQU     >A040           ;WORKSPACE 2
WSP3    EQU     >A400           ;WORKSPACE 3
WSP4    EQU     >A420           ;WORKSPACE 4
WSP5    EQU     >A440           ;WORKSPACE 5

LTPTR0  EQU     >A060           ;LATTICE 0 POINTER
LTPTR1  EQU     >A460           ;LATTICE 1 POINTER

F       EQU     >0000           ;F(M,L) DISPLACEMENT
G       EQU     >0020           ;G(M,L) DISPLACEMENT
G1      EQU     >0040           ;G(M-1,L) DISPLACEMENT
B       EQU     >0060           ;B(M,L) DISPLACEMENT
V       EQU     >0080           ;V(M,L) DISPLACEMENT

U       EQU     >7FFC           ;0.999878                    +(1/0/15)
ALPHA   EQU     >028F           ;0.1998901                   +(6/0/10)-5
BETA    EQU     >7D70           ;0.9799804                   +(1/0/15)
GAMMA   EQU     >028F           ;0.1998901   (1 - BETA)      +(6/0/10)-5
EPSLON  EQU     >0001           ;0.000122                    +(1/2/13)
N       EQU     >0008           ;NUMBER OF LATTICE STAGES
```

```
ATPTR0   EQU     >A100                ;ATD 0 POINTER
ATPTR1   EQU     >A500                ;ATD 1 POINTER

M        EQU     >0020                ;2*16 BYTE QA WINDOW LENGTH
L        EQU     >0080                ;2*64 BYTE QB WINDOW LENGTH
D        EQU     >0020                ;2*DELAY LENGTH
K        EQU     >0002                ;ATD CONSTANT  LOG2(L/M)
THETAH   EQU     >0000                ;THETA = 0.0029297        +(27/0/5)-8
THETAL   EQU     >0018                ;

EA       EQU     D+L+2                ;EA POINTER ADDRESS DISPLACEMENT
EB       EQU     EA+2                 ;EB POINTER ADDRESS DISPLACEMENT
EC       EQU     EB+2                 ;EC POINTER ADDRESS DISPLACEMENT

QA       EQU     EC+2                 ;QA ADDRESS DISPLACEMENT
QB       EQU     QA+4                 ;QB ADDRESS DISPLACEMENT

POSMAX   EQU     >7FFF                ;LARGEST POSITIVE NUMBER
NEGMAX   EQU     >8000                ;LARGEST NEGATIVE NUMBER
LAST     EQU     >A7FF                ;LAST BYTE OF RAM USED

*
* ADAPTIVE LATTICE PREDICTOR SHELL
*

         AORG    >A800

START    LWPI    WSP0                 ;DEFINE SHELL WORKSPACE
         BL      @TMINIT              ;INITIALIZE THE TIMER
         BL      @CLRRAM              ;CLEAR RAM
         LI      R9,ATPTR0            ;LOAD ATD 0 POINTER
         BL      @ATINIT              ;INITIALIZE ATD 0
         LI      R9,ATPTR1            ;LOAD ATD 1 POINTER
         BL      @ATINIT              ;INITIALIZE ATD 1
```

```
*
* SHELL MAIN LOOP (TIMER INTERRUPT SERVICE)
*

TMRSRV  LWPI    WSP0            ;DEFINE SHELL WORKSPACE
        LI      R12,TMRCRU      ;LOAD TIMER CRU-BASE ADDRESS
        SBO     TMRENB          ;ENABLE THE TIMER

        LI      R1,>0000        ;LOAD A/D CHANNEL NUMBER
        BL      @READ           ;READ A/D CHANNEL 0
        LI      R9,LTPTR0       ;LOAD LATTICE 0 POINTER
        BL      @ALP            ;INVOKE LATTICE PREDICTOR
        LI      R9,ATPTR0       ;LOAD ATD 0 POINTER
        BL      @ATD            ;INVOKE ADAPTIVE THRESHOLD DETECTOR
        MOV     R1,R0           ;GET E(M)**2
        LI      R1,>0000        ;LOAD D/A CHANNEL NUMBER
        BL      @WRITE          ;OUTPUT E(M)**2 TO D/A CH 0
        MOV     R4,R0           ;GET QA(M)
        LI      R1,>0002        ;LOAD CHANNEL 2
        BL      @WRITE          ;WRITE QA(M) TO D/A CH 2
        MOV     R7,R0           ;GET ALARM(M)
        LI      R1,>0000        ;LOAD CHANNEL 0
        BL      @PUT            ;WRITE ALARM TO OUTPUT 0

        LI      R1,>0001        ;LOAD A/D CHANNEL NUMBER
        BL      @READ           ;READ A/D CHANNEL 1
        LI      R9,LTPTR1       ;LOAD LATTICE 1 POINTER
        BL      @ALP            ;INVOKE LATTICE PREDICTOR
        LI      R9,ATPTR1       ;LOAD ATD 1 POINTER
        BL      @ATD            ;INVOKE ADAPTIVE THRESHOLD DETECTOR
        MOV     R1,R0           ;GET E(M)**2
        LI      R1,>0001        ;LOAD D/A CHANNEL NUMBER
        BL      @WRITE          ;OUTPUT E(M)**2 TO D/A CH 1
        MOV     R4,R0           ;GET QA(M)
        LI      R1,>0003        ;LOAD CHANNEL 3
        BL      @WRITE          ;WRITE QA(M) TO D/A CH 3
        MOV     R7,R0           ;GET ALARM(M)
        LI      R1,>0001        ;LOAD OUTPUT CHANNEL NUMBER
        BL      @PUT            ;OUTPUT ALARM TO OUTPUT CHANNEL 0

        LIMI    TMRMSK          ;ENABLE TIMER INTERRUPT
        JMP     $+0             ;WAIT FOR TIMER INTERRUPT
```

```
*
* TIMER INITIALIZATION ROUTINE
*
* REGISTER USAGE: R0,R11,R12
*
TMINIT  LI      R12,ADCCRU       ;LOAD A/D CRU-BASE ADDRESS
        SBO     ADMODE           ;A/D MEMORY-MAPPED MODE
        LI      R12,BRANCH       ;LOAD BRANCH INSTRUCTION
        MOV     R12,@TMRVEC      ;STORE AT TIMER VECTOR ADDRESS
        LI      R12,TMRSRV       ;LOAD TIMER SERVICE ADDRESS
        MOV     R12,@TMRVEC+2    ;STORE AT TIMER VECTOR + 2

        LI      R12,TMRCRU       ;LOAD TIMER CRU-BASE ADDRESS
        LI      R0,TMRCNT        ;LOAD TIMER COUNT
        LDCR    R0,0             ;SET UP TIMER
        SBZ     TMMODE           ;TIMER INTERRUPT MODE
        B       *R11             ;RETURN TO CALLER

*
* CLEAR RAM ROUTINE
*
* REGISTER USAGE: R0,R11
*
CLRRAM  LI      R0,WSP1          ;LOAD ADDRESS OF WORKSPACE 1
LOOPC   CLR     *R0+             ;CLEAR RAM
        CI      R0,LAST+1        ;END OF RAM?
        JNE     LOOPC            ;LOOP IF NOT
        B       *R11             ;ELSE RETURN TO CALLER

*
* ADAPTIVE THRESHOLD DETECTOR INITIALIZATION
*
* REGISTER USAGE:
*
* R0  TEMPORARY
* R9  ATD POINTER
* R11 SUBROUTINE LINKAGE
*
ATINIT  LI      R0,L+D-M         ;LOAD POINTER DISPLACEMENT
        A       R9,R0            ;FORM EA POINTER
        MOV     R0,@EA(R9)       ;STORE THE POINTER
        LI      R0,L             ;LOAD POINTER DISPLACEMENT
        A       R9,R0            ;FORM EB POINTER
        MOV     R0,@EB(R9)       ;STORE THE POINTER
        LI      R0,>0000         ;LOAD POINTER DISPLACEMENT
        A       R9,R0            ;FORM EC POINTER
        MOV     R0,@EC(R9)       ;STORE THE POINTER

        CLR     @QA(R9)          ;CLEAR QA HI
        CLR     @QA+2(R9)        ;CLEAR QA LO
        CLR     @QB(R9)          ;CLEAR QB HI
        CLR     @QB+2(R9)        ;CLEAR QB LO

        B       *R11             ;RETURN TO CALLER
```

```
*
* A/D READ SUBROUTINE
*
* REGISTER USAGE:
*
* R0    RETURNS SAMPLE FROM A/D IN (2/0/11) FORMAT
* R1    CONTAINS A/D CHANNEL #
* R11   SUBROUTINE LINKAGE REGISTER
*
READ    SLA     R1,1            ;FORM A/D CHANNEL DISPLACEMENT
        AI      R1,ADC0         ;FORM A/D CHANNEL ADDRESS
        MOV     *R1,R0          ;READ FROM A/D                  (1/0/11)
        CI      R0,NEGMAX       ;DISALLOW LARGEST NEGATIVE #
        JNE     $+4
        INC     R0
        SRA     R0,1            ;REFORMAT TO                    (2/0/11)
        B       *R11            ;RETURN TO CALLER

*
* D/A WRITE SUBROUTINE
*
* REGISTER USAGE:
*
* R0    CONTAINS DATA TO BE OUTPUT (LEFT JUSTIFIED)
* R1    CONTAINS D/A CHANNEL #
* R11   SUBROUTINE LINKAGE REGISTER
*
WRITE   SLA     R1,1            ;FORM D/A CHANNEL DISPLACEMENT
        AI      R1,DAC0         ;FORM D/A CHANNEL ADDRESS
        MOV     R0,*R1          ;OUTPUT DATA TO D/A CONVERTER
        B       *R11            ;RETURN TO CALLER

*
* PUT SUBROUTINE
*
* REGISTER USAGE:
*
* R0    INDICATES ALARM TRUE IF NONZERO
* R1    OUTPUT CHANNEL NUMBER (0 OR 1)
* R11   SUBROUTINE LINKAGE REGISTER
*
* NOTE: OUTPUT FALLS FOR APPROX. .66 MICROSECONDS
*
PUT     MOV     R0,R0           ;ALARM TRUE?
        JEQ     RETP            ;NO, DON'T OUTPUT
        LI      R12,SELCRU      ;LOAD SEL CRU BASE ADDRESS
        CI      R1,>0000        ;OUTPUT CHANNEL 0?
        JEQ     PUT0            ;YES
        CI      R1,>0001        ;OUTPUT CHANNEL 1?
        JEQ     PUT1            ;YES
        JMP     RETP
PUT0    TB      SEL1            ;TWIDDLE SEL1 LINE
        JMP     RETP
PUT1    TB      SEL2            ;TWIDDLE SEL2 LINE
RETP    B       *R11            ;RETURN TO CALLER
```

```
*
* LEAST MEAN SQUARE ADAPTIVE LATTICE PREDICTOR
*
* REGISTER USAGE:
*
* R0    RETURNS PREDICTOR ERROR
* R1    THRU R8 USED
* R9    POINTER
* R10   SAVES RETURN LINKAGE
* R11   LOCAL SUBROUTINE RETURN LINKAGE
*
ALP     MOV     R11,R10          ;SAVE RETURN LINKAGE
        MOV     R0,@F(R9)        ;F(M,1) = X(M)                  (2/0/11)
        MOV     R0,@G(R9)        ;G(M,1) = X(M)                  (2/0/11)
        LI      R8,N             ;LOAD # OF LATTICE STAGES

LOOPL   MOV     @B(R9),R1        ;LOAD B(M,L)                    (1/0/15)
        MOV     @G1(R9),R2       ;LOAD G(M-1,L)                  (1/1/14)
        BL      @MULT            ;B(M,L)*G(M-1,L)                (2/1/29)
        MOV     R2,R4            ;COPY ARGUMENT
        MOV     R3,R5            ;
        MOV     @F(R9),R2        ;LOAD F(M,L)                    (1/1/14)
        CLR     R3               ;EXTEND PRECISION
        BL      @ASHR            ;REFORMAT F(M,L)                (2/1/14)
        S       R4,R2            ;F(M,L) - B(M,L)*G(M-1,L)       (1/2/29)
        S       R5,R3            ;
        JOC     $+4              ;
        DEC     R2               ;
        BL      @ASHL            ;CLIP TO                        (1/1/29)
        BL      @EDIT            ;ROUND TO                       (1/1/14)
        MOV     R2,@F+2(R9)      ;F(M,L+1) =
*                                ; F(M,L) - B(M,L)*G(M-1,L)      (1/1/14)
        MOV     @B(R9),R1        ;LOAD B(M,L)                    (1/0/15)
        MOV     @F(R9),R2        ;LOAD F(M,L)                    (1/1/14)
        BL      @MULT            ;B(M,L)*F(M,L)                  (2/1/29)
        MOV     R2,R4            ;COPY ARGUMENT
        MOV     R3,R5            ;
        MOV     @G1(R9),R2       ;LOAD G(M-1,L)                  (1/1/14)
        CLR     R3               ;EXTEND PRECISION
        BL      @ASHR            ;REFORMAT G(M-1,L)              (2/1/14)
        S       R4,R2            ;G(M-1,L) - B(M,L)*F(M,L)       (1/2/29)
        S       R5,R3            ;
        JOC     $+4              ;
        DEC     R2               ;
        BL      @ASHL            ;CLIP TO                        (1/1/29)
        BL      @EDIT            ;ROUND TO                       (1/1/14)
        MOV     R2,@G+2(R9)      ;G(M,L+1) =
*                                ; G(M-1,L) - B(M,L)*F(M,L)      (1/1/14)
```

```
MOV     @F(R9),R2       ;LOAD F(M,L)                     (1/1/14)
ABS     R2              ;ABS(F(M,L))                    +(1/1/14)
MPY     R2,R2           ;F(M,L)**2                      +(2/2/28)
MOV     @G1(R9),R4      ;LOAD G(M-1,L)                   (1/1/14)
ABS     R4              ;ABS(G(M-1,L))                  +(1/1/14)
MPY     R4,R4           ;G(M-1,L)**2                    +(2/2/28)
A       R4,R2           ;F(M,L)**2 + G(M-1,L)**2        +(1/3/28)
A       R5,R3           ;
JNC     $+4             ;
INC     R2              ;
BL      @ASHL           ;CLIP TO                        +(1/2/28)
BL      @EDIT           ;ROUND TO                       +(1/2/13)
LI      R1,GAMMA        ;GAMMA = 1 - BETA               +(6/0/10)-5
MPY     R1,R2           ;GAMMA*(F(M,L)**2 + G(M-1,L)**2)
MOV     R2,R4           ;COPY ARGUMENT                  +(7/2/23)-5
MOV     R3,R5           ;

LI      R1,BETA         ;LOAD BETA                      +(1/0/15)
MOV     @V(R9),R2       ;LOAD V(M-1,L)                  +(1/2/13)
MPY     R1,R2           ;BETA*V(M-1,L)                  +(2/2/28)

A       R4,R2           ;BETA*V(M-1,L) +
A       R5,R3           ; GAMMA*(F(M,L)**2 + G(M-1,L)**2)
JNC     $+4             ;                               +(1/3/28)
INC     R2              ;
BL      @ASHL           ;CLIP TO                        +(1/2/28)
BL      @EDIT           ;ROUND TO                       +(1/2/13)
MOV     R2,@V(R9)       ;V(M,L) = BETA*V(M-1,L) +
MOV     R2,R7           ; GAMMA*(F(M,L)**2 + G(M-1,L)**2)
*                       ;                               +(1/2/13)

CLR     R6              ;T = 0
CI      R2,EPSLON       ;V(M,L) < EPSLON?
JLT     BYPASS          ;YES, BYPASS COMPUTATION OF I

MOV     @F+2(R9),R1     ;LOAD F(M,L+1)                   (1/1/14)
MOV     @G1(R9),R2      ;LOAD G(M-1,L)                   (1/1/14)
BL      @MULT           ;F(M,L+1)*G(M-1,L)               (2/2/28)
MOV     R2,R4           ;COPY ARGUMENT
MOV     R3,R5           ;
MOV     @F(R9),R1       ;LOAD F(M,L)                     (1/1/14)
MOV     @G+2(R9),R2     ;LOAD G(M,L+1)                   (1/1/14)
BL      @MULT           ;F(M,L)*G(M,L+1)                 (2/2/28)
A       R4,R2           ;F(M,L+1)*G(M-1,L) + F(M,L)*G(M,L+1)
A       R5,R3           ;                                (1/3/28)
JNC     $+4             ;
INC     R2              ;
BL      @ASHL           ;CLIP TO                         (1/2/28)
BL      @EDIT           ;ROUND TO                        (1/2/13)
MOV     R2,R6           ;SAVE SIGN INFO
ABS     R2              ;ABS(    )                      +(1/2/13)
SRA     R2,1            ;REFORMAT                       +(2/2/12)
LI      R1,ALPHA        ;LOAD ALPHA                     +(6/0/10)-5
MPY     R1,R2           ;ALPHA*ABS(          )          +(8/2/23)-5
DIV     R7,R2           ;ALPHA*ABS(          )/V(M,L)   +(7/0/9)-5
INV     R6              ;TEST SIGN
JLT     $+4             ;RESTORE SIGN
NEG     R2              ;T = ALPHA*(F(M,L+1)*G(M-1,L) +
MOV     R2,R6           ; F(M,L)*G(M,L+1))/V(M,L)        (7/0/9)-5
```

```
BYPASS  LI      R1,U              ;LOAD U                        +(1/0/15)
        MOV     @B(R9),R2         ;LOAD B(M,L)                    (1/0/15)
        BL      @MULT             ;U*B(M,L)                       (2/0/30)

        A       R6,R2             ;B(M+1,L) = U*B(M,L) + T        (1/1/30)
        BL      @ASHL             ;CLIP TO                        (1/0/30)
        BL      @EDIT             ;ROUND TO                       (1/0/15)
        MOV     R2,@B(R9)         ;STORE B(M+1,L)                 (1/0/15)

        MOV     @G(R9),@G1(R9)    ;G(M-1,L) = G(M,L)              (1/1/14)
        INCT    R9                ;BUMP POINTER
        DEC     R8                ;COUNT = COUNT - 1
        JEQ     ENDLOP            ;QUIT IF COUNT=0
        B       @LOOPL            ;ELSE PROCESS NEXT STAGE

ENDLOP  MOV     @G(R9),@G1(R9)    ;G(M-1,N+1) = G(M,N+1)          (1/1/14)
        MOV     @F(R9),R0         ;LOAD PREDICTOR ERROR           (1/1/14)

        MOV     R0,R2             ;COPY E(M)                      (1/1/14)
        ABS     R2                ;ABS(E(M))                     +(1/1/14)
        MPY     R2,R2             ;E(M)**2                       +(2/2/28)
        BL      @ASHL             ;CLIP TO                       +(1/2/28)
        BL      @EDIT             ;ROUND TO                      +(1/2/13)
        MOV     R2,R1             ;COPY E(M) FOR OUTPUT          +(1/2/13)

        SRA     R0,4              ;FORMAT E(M) FOR OUTPUT         (5/1/10)
        SRA     R1,4              ;FORMAT E(M)^2 FOR OUTPUT      +(5/2/9)

        B       *R10              ;RETURN TO CALLER
```

```
*
* ADAPTIVE THRESHOLD DETECTOR ROUTINE
*
ATD     MOV     R11,R10          ;SAVE RETURN POINTER
        MOV     @EA(R9),R6       ;LOAD EA POINTER
        MOV     @EB(R9),R7       ;LOAD EB POINTER
        MOV     @EC(R9),R8       ;LOAD EC POINTER

        S       *R8,@QB+2(R9)    ;QB(M) = QB(M-1) ...
        JOC     $+4              ; - E(M-L-D)^2 ...
        DEC     @QB(R9)          ; + E(M-D)^2           [11/8/13]
        A       *R7,@QB+2(R9)    ;
        JNC     $+4              ;
        INC     @QB(R9)          ;

        S       *R6,@QA+2(R9)    ;QA(M) = QA(M-1) ...
        JOC     $+4              ; - E(M-M)^2 ...
        DEC     @QA(R9)          ; + E(M)^2             [13/6/13]
        A       R2,@QA+2(R9)     ;
        JNC     $+4              ;
        INC     @QA(R9)          ;

        MOV     R2,*R8           ;E(M-L-D)^2 = E(M)^2       +(1/2/13)

        LI      R5,D+L+2         ;LOAD MAXIMUM BUFFER DISPLACEMENT
        A       R9,R5            ;FORM ABSOLUTE ADDRESS
        INCT    R6               ;ADVANCE EA POINTER
        C       R5,R6            ;TIME TO CIRCULATE?
        JNE     $+4              ;
        MOV     R9,R6            ;YES
        MOV     R6,@EA(R9)       ;STORE THE POINTER
        INCT    R7               ;ADVANCE EB POINTER
        C       R5,R7            ;TIME TO CIRCULATE?
        JNE     $+4              ;
        MOV     R9,R7            ;YES
        MOV     R7,@EB(R9)       ;STORE THE POINTER
        INCT    R8               ;ADVANCE EC POINTER
        C       R5,R8            ;TIME TO CIRCULATE?
        JNE     $+4              ;
        MOV     R9,R8            ;YES
        MOV     R8,@EC(R9)       ;STORE THE POINTER
```

```
        MOV     @QA(R9),R2          ;LOAD QA(M)                    (13/6/13)
        MOV     @QA+2(R9),R3        ;
        LI      R8,K                ;LOAD ATD CONSTANT
LP0     BL      @ASHL               ;RESCALE QA(M)                 (11/8/13)
        DEC     R8                  ;
        JNE     LP0                 ;
        MOV     R2,R12              ;SAVE QA(M) FOR OUTPUT
        MOV     R3,R13              ;
        S       @QB(R9),R2          ;QA(M) - QB(M)                 (10/9/13)
        S       @QB+2(R9),R3        ;
        JOC     $+4                 ;
        DEC     R2                  ;

        MOV     R2,R4               ;SAVE
        MOV     R3,R5               ;
        LI      R7,THETAH           ;LOAD THETA                 +(27/0/5)-8
        LI      R8,THETAL           ;
        S       R7,R4               ;(QA(M)-QB(M)) - THETA         (9/10/13)
        S       R8,R5               ;
        JOC     $+4                 ;
        DEC     R4                  ;

        LI      R7,>07FF            ;ALARM(M) = .TRUE.
        MOV     R4,R4               ;(QA(M)-QB(M)) - THETA > 0?
        JGT     $+6                 ;
        JEQ     $+4                 ;
        CLR     R7                  ;NO, ALARM(M) = .FALSE.

        LI      R8,>0004            ;LOAD OUTPUT SHIFT COUNT
LP1     BL      @ASHR               ;REFORMAT
        DEC     R8                  ;BUMP SHIFT COUNT
        JNE     LP1                 ;
        MOV     R3,R6               ;QA-QB FOR OUTPUT

        MOV     R12,R2              ;LOAD QA(M)
        MOV     R13,R3              ;
        LI      R8,>0004            ;LOAD OUTPUT SHIFT COUNT
LP2     BL      @ASHR               ;REFORMAT
        DEC     R8                  ;BUMP SHIFT COUNT
        JNE     LP2                 ;
        MOV     R3,R4               ;QA(M) IN R4 FOR OUTPUT

        MOV     @QB(R9),R2          ;LOAD QB(M)
        MOV     @QB+2(R9),R3        ;
        LI      R8,>0004            ;LOAD OUTPUT SHIFT COUNT
LP3     BL      @ASHR               ;REFORMAT
        DEC     R8                  ;BUMP SHIFT COUNT
        JNE     LP3                 ;
        MOV     R3,R5               ;QB(M) IN R5 FOR OUTPUT

        B       *R10                ;RETURN TO CALLER
```

```
*
* 2'S COMPLEMENT SIGNED MULTIPLY ROUTINE
*
* R2:R3 <-- R1 * R2
* R0  IS MODIFIED
* R11 IS USED FOR RETURN LINKAGE
*
MULT    CLR     R0              ;SIGN FIX = 0
        MOV     R1,R1           ;TEST SIGN OF R1
        JGT     $+6             ;
        JEQ     $+4             ;
        MOV     R2,R0           ;SIGN FIX = (R2)
        MOV     R2,R2           ;TEST SIGN OF R2
        JGT     $+6             ;
        JEQ     $+4             ;
        A       R1,R0           ;SIGN FIX = SIGN FIX + (R1)
        MPY     R1,R2           ;R2:R3 <-- R1 * R2
        S       R0,R2           ;FIX SIGN OF RESULT
        B       *R11            ;RETURN TO CALLER

*
* EDIT (ROUNDUP) ROUTINE
*
* R2 <-- EDIT(R2:R3)
* R1 IS USED FOR SUBROUTINE LINKAGE
*
EDIT    MOV     R3,R3           ;TEST MSB OF LOW ORDER WORD
        JGT     EDT             ;DON'T INCREMENT IF ZERO
        JEQ     EDT             ;
        INC     R2              ;INCREMENT HIGH-ORDER WORD
        JNO     EDT             ;SKIP IF NO OVERFLOW
        LI      R2,POSMAX       ;LOAD LARGEST POSITIVE NUMBER
EDT     CI      R2,NEGMAX       ;DISALLOW LARGEST NEGATIVE NO.
        JNE     $+4             ;
        INC     R2              ;
        B       *R11            ;RETURN TO CALLER
```

```
*
* DOUBLE PRECISION RIGHT ARITHMETIC SHIFT
*
* R2:R3 <-- ASHR(R2:R3)
* R11 IS USED FOR SUBROUTINE LINKAGE
*
ASHR    SRL     R3,1            ;SHIFT LOWER WORD RIGHT
        SRA     R2,1            ;SHIFT HIGHER WORD RIGHT
        JNC     $+6             ;SKIP IF LSB WAS ZERO
        AI      R3,NEGMAX       ;SET MSB OF LOWER WORD TO ONE
        B       *R11            ;RETURN TO CALLER

*
* DOUBLE PRECISION LEFT ARITHMETIC
* SHIFT WITH OVERFLOW CHECK
*
* R2:R3 <-- ASHL(R2:R3)
* R11 IS USED FOR SUBROUTINE LINKAGE
*
ASHL    SLA     R2,1            ;SHIFT HIGHER WORD LEFT
        JNO     ASL             ;OVERFLOW?
        JNC     PASL            ;WAS IT POSITIVE?
        LI      R2,NEGMAX       ;NO, LOAD MAX NEG. VALUE
        CLR     R3              ;
        JMP     RASL            ;RETURN TO CALLER
PASL    LI      R2,POSMAX       ;YES, LOAD MAX POS. VALUE
        SETO    R3              ;
        JMP     RASL            ;RETURN TO CALLER
ASL     SLA     R3,1            ;SHIFT LOWER WORD LEFT
        JNC     $+4             ;SKIP IF MSB WAS ZERO
        INC     R2              ;SET LSB OF HIGHER ORDER WORD
RASL    B       *R11            ;RETURN TO CALLER

        END     START
```

ON A BLOCK FLOATING POINT IMPLEMENTATION OF AN INTRUSION-DETECTION ALGORITHM

by

ROBERT JOSEPH FOGLER

B.S., Kansas State University, 1977

AN ABSTRACT OF A MASTER'S THESIS

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Electrical Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1979

The main objective of this paper is to present various aspects of implementing a specific intrusion-detection algorithm on a microprocessor using block floating point arithmetic. In particular a TI 9900 based test system is considered.

The proposed algorithm is able to detect intruder stimuli which are broadband and transient in nature, while rejecting correlated noise which may be present with the intruder signal. The algorithm consists of two main functional blocks: an adaptive lattice predictor (ALP) and an adaptive threshold detector (ATD).

The ALP is used to remove correlated noise hence reduces the number of false alarms, while improving the signal-to-noise ratio when intruder stimuli are present, thereby simplifying the task of the ATD.

The ATD uses a variance estimate of a noise segment, and a signal plus noise segment from the ALP output sequence. It then compares a function of these estimates with a fixed threshold.

Experimental results demonstrating the performance of the intrusion-detection algorithm using data obtained from an actual test site are included.