A SYNTACTIC METHOD FOR ANALYZING PLASMA ETCHING SIGNALS

by

Dong Wu Zhao

Memorandum No. UCB/ERL M99/41

16 July 1999

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

A Syntactic Method for Analyzing Plasma Etching Signals

by Dong Wu Zhao

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report:

Committee:

Professor Costas J. Spanos

Research Advisor

7/16/1999

(Date)

****

Professor Avideh Zakhor

Second Reader

Aug 24 1999

(Date)

Abstract

A Syntactic Method for Analyzing Plasma Etching Signals

by
Dong Wu Zhao

Electrical Engineering and Computer Sciences
University of California at Berkeley

Professor Costas J. Spanos, Advisor

Our objective is to diagnose faulty process conditions during plasma etching in semiconductor manufacturing. In this thesis we present a syntactic method for analyzing plasma etching signals. The method is used to describe the general characteristics of the etcher's real-time signals in the presence of noise and other extraneous influences. The major features of the signals are captured by a piece-wise linear approximation scheme. The sequence of line segments is described by a string of integers, where each integer is the quantized slope of each segment. The string is then compared to a set of predefined regular expressions, that define various faulty tool conditions.

Acknowledgments

I would like to thank my advisor, Professor Costas Spanos, for his careful guidance and generous support for my research. I also thank Professor Avideh Zakhor for giving me the undergraduate research opporutnities. Further, I thank my undergraduate advisor, Professor Joseph Kahn for his honest and valuable suggestions. I am grateful to the three professors for their letters of recommendation which led to my acceptance of graduate admission application to Berkeley and MIT.

I acknowledge Qi Li and Jerry Jin of Applied Materials for their insights and expertise in plasma etching. A warm "thanks" goes out to present and past members of the Berkeley Computer-Aided Manufacturing (BCAM) group for their friendships: Junwei Bao, Roawen Chen, Mark Hatzilambrou, Anna Ison, Herb Huang, Nickhil Jakatdar, Jae-Wook Lee, Jeff Lin, Greg Luurtsema, David Mudie, John Musacchio, Xinhui Niu, Manolis Terrovitis, Nikhil Vaidya, and Haolin Zhang.

This work was supported by the Semiconductor Research Corporation (SRC), the state of California Micro program, and participating companies (Advanced Micro Devices, Applied Materials, Atmel Corporation, Lam Research, National Semiconductor, Silicon Valley Group, Texas Instruments).

Chapter 1 Introduction

1.1. Motivation

As the semiconductor processing technology approaches 0.15 μm feature size and 300 mm wafer diameter, the cost of building a new fabrication plant is rising rapidly (20% per year) [26]. It is predicted that it will take about $10 billion to build a state-of-the-art facility for manufacture in 2005. To remain competitive and encounter the escalating cost, the industry has strived to improve feature size, wafer diameter, yield, and equipment utilization. However, the gains from wafer diameter and yield are disappearing. The new focus is on equipment utilization.

The key to optimize equipment utilization is through process monitoring in order to make sure that wafers are processed properly at each step. However, there are more than 100 manufacturing steps, and it is too costly and time-consuming to measure each wafer after the completion of each step. As of now, people in the industry usually measure and monitor wafers periodically, especially right after performing preventive maintenance and changing machine settings. A final test is performed on each wafer after all the steps. Thus, if an error occurs, it is very likely that many wafers are misprocessed without notice until very late. Because of the late notice, it is very difficult to trace back and locate the faulty step and diagnose the problem. Therefore, one can save considerable amount of resource by monitoring equipments on line, using their real-time signals. In this work we demonstrate that it is possible to do so, with the monitoring of plasma etch signals as an example.

The purpose of analyzing real-time etching signals is to perform fault detection and diagnosis during the manufacture of state-of-the-art integrated circuits. Plasma etching is one of the costliest steps during semiconductor processing. In addition, it is very difficult

to control, since the physical mechanism of plasma etching is not well understood. Fault detection tools determine the state of the plasma etcher by analyzing the behavior of its real-time signals. Once a fault is detected, the fault diagnosis tools will assign a cause to it, as to assist the process engineer to fix the problem. By detecting the fault early, a process engineer can prevent expensive new wafers from being fetched to the faulty etcher, and correct the fault on a timely basis. Thus, wafer yield and throughput will be enhanced. Also, preventive maintenance (PM) can be scheduled according to fault detection and diagnosis results, and down-time and mean-time-to-repair (MTTR) can be reduced.[15]

1.2. Thesis Organization

In Chapter 2 we examine the nature of plasma etching signals, and focus on the aspects that make characterizing them difficult. Spectral analysis of the plasma etching signals is presented in Chapter 3 , so that we can see the nature of the signal perturbation in the frequency domain. This discussion will show that spectral analysis is probably not very useful for characterizing plasma etching signals. In Chapter 4 a basic syntactic method is presented. In addition to describing the rough structure of the signals, we will present a supplemental method for computing some quantified attributes, such as the amplitude of various spikes. The result of characterizing a set of marathon-run data will be presented. In Chapter 5 an advanced study case of the analysis for plasma ignition waveform will be presented. We will conclude this work in Chapter 6 with a view on the nature of syntactic analysis and with some comments on how to efficiently deploy such systems.

Chapter 2 Background: Nature of Plasma Etching Signals

2.1. Real-Time Data

Plasma etching is not a very well understood process. Practical physical models for fault detection and diagnosis are not yet available. Researchers in fault detection and diagnosis so far have used empirical models. Previous works [15] involve modeling of input setting against wafer's output parameters, such as etch rate, uniformity, selectivity, and anisotropy. However, due to machine aging, maintenance and various other effects, input settings do not entirely determine the chamber state. The same settings can result in very different etching behavior. Spanos and S. Lee [14][15] show that the equipment's own electrical and mechanical signals can be modeled as time series and used effectively for fault detection and diagnosis. These real-time signals reflect the chamber state much better than the input settings; they are able to show drift in etching behavior due to machine aging and maintenance.

Usually, when the equipment is just out of control, the malfunction will first manifest itself in the real-time signals, but not much etching damage is done to the wafer yet, and the wafer is still usable if the malfunction is corrected soon enough. As a result, using real-time signals for monitoring the etching process can help prevent misprocessing costly wafers. Hundreds of real-time signals are available for computer analysis via standard communication ports, such as SECS II. An engineer can choose a few of them to monitor the etching process based on experience. Alternatively, one can find out the signals that are sensitive to faults by doing designed experiments. Some of the real-time signals proved useful are RF load, coil position RF tune vane position, peak-to-peak voltage load impedance, RF phase error, DC bias and endpoint. [16]

One of our data sets is a metal etch marathon run from an industrial vendor. The data set consists of real-time signals from more than 1400 wafers. For this analysis, we have chosen the capacitance manometer signal, which reflects the pressure level in the etcher's chamber. The waveform provided by the capacitance manometer is relatively clean, which simplifies visual verification of the analysis.

There are several steps in the etching process, including pre-etch of native oxide, main etch, and over-etch. At the beginning of each etching step, it usually takes a few seconds for the etchant gases to stabilize. We usually select the later part of the main etch step for analysis, where the waveform is relatively stable and repeatable. Figure 2.1 shows the "windowing" operation on the capacitance manometer signal. An experienced process engineer can usually tell if etching is faulty by viewing the signal's waveform. For our metal etch marathon data, the commonly seen waveforms are shown in Figure 2.2. We visually classify these signals as either "normal" or of type 1, 2, 3 and 4. Even though we do not have documented faults in this run, types 3 and 4 are most likely faulty. Notice that types 1 and 2 can be viewed as the combination of a normal signal, and a negative or a positive spike, respectively; they may be considered normal if the spike is small enough. The goal of the analysis is to correctly classify the waveforms.

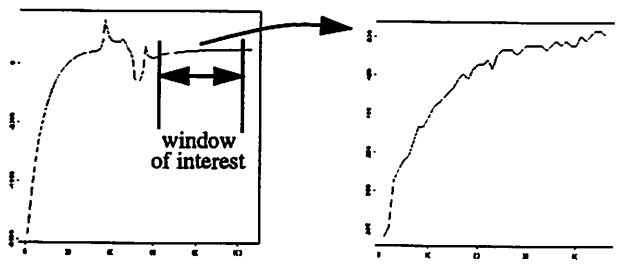


Figure 2.1. The windowing operation.

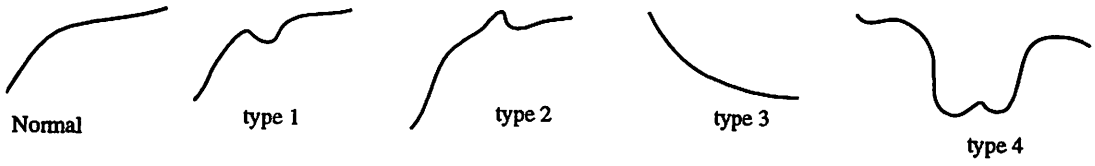


Figure 2.2. Commonly seen waveforms for capacitance manometer in a metal etch marathon run.

2.2. Analytical Difficulty for Statistical Techniques

Statistical modeling of etching signals has been difficult, due to preventive maintenance (PM), machine aging, chamber memory effects, and other influences [15]. During a maintenance cycle, residue gradually builds up in the chamber. This causes the chamber state and sensor signals to drift slowly. Notice that a drift in sensor signals does not necessarily correspond to a drift in the chamber state. For instance, as residue accumulates on the sensor window and degrades the transmittance, the intensity of the sensor signals will decrease. Yet, the operation of the equipment is far from being faulty.

If too much residue accumulates in the chamber, the process parameters will be quite different from when the chamber is clean. The aim of PM is to restore the etcher back to the original clean state. However, due to the aging of other parts of the tool, process parameters after a PM will be a little different from those at the beginning of the previous clean cycle; the average level of the signals as well as the variance may change.

One also encounters so-called "memory effects", where, for example, after a signal is unusually high indicating a fault, it will often remain relatively high for a while before returning to the normal level, even after the machine is back in control. This memory effect is very obvious when the machine first starts up. It takes a few runs before the machine reaches its steady operating state, while the signals appear to approach steady state values in an exponential fashion. (See Figure 2.3 and 2.4.)

The actual etching will be affected by the variation in the upstream processing. For instance, the photo resist thickness variation on different wafer will lead to different etching time. Also, due to the occurrence of various events, noisy spikes of different magnitude and time duration may be added to the signals.

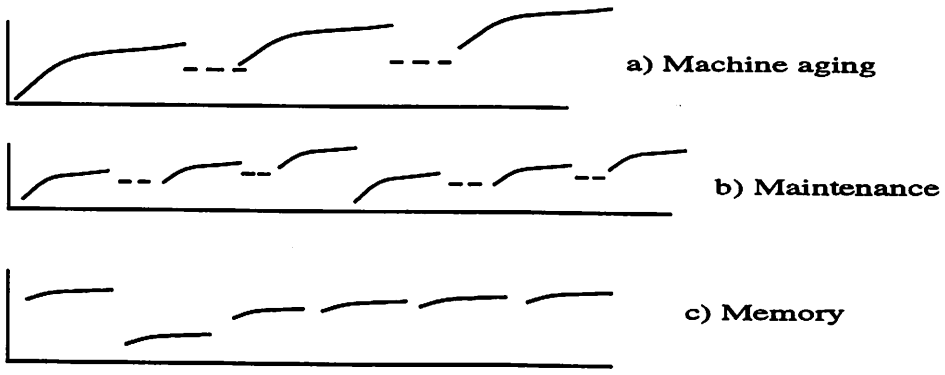


Figure 2.3. The illustrations of the nature of the plasma etching signals. a) Machine aging effect within a PM maintenance cycle. b) Maintenance effect after a PM maintenance procedure. c) Memory effect after a fault occurs.

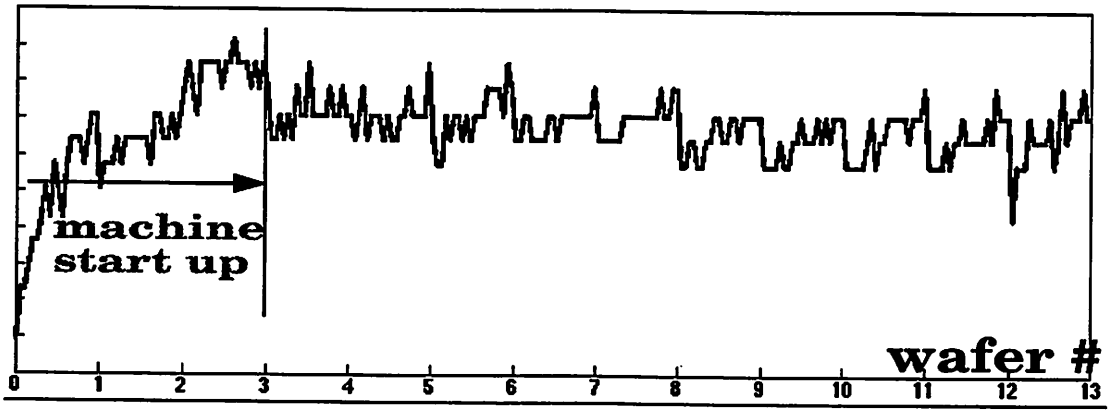


Figure 2.4. The memory effect as seen when the machine starts up.

2.3. Two Statistical Monitoring Methods

As a result of these influences, using ordinary statistical methods to monitor plasma etching processes may yield unacceptably high false alarm rates. To compensate for these effects, some researchers update certain model parameters for the models they use as the process drifts. When a sample is fed to the etcher, model parameters such as the signals' means and covariance structure are updated in an exponentially weighted fashion, with more weight given to recent samples. The following are two such examples. The first one is the Real-Time Statistical Process Control (RTSPC) project done in the Berkeley Computer Aided Manufacturing group (BCAM), which uses time series models. The covariance structure is updated during production. The means need not be updated, since the data is centered to zero. The second example is the diagnostic work done in Texas Instrument's J-88 Project, which uses principal component analysis (PCA), and updates both the mean and covariance structure for the PCA model during production. When the signal discontinues or jumps due to events such as preventive maintenance and new equipment installation, the model parameters are reset, using the data acquired after the events.

2.3.1. The RTSPC Project [15][16]

RTSPC uses time series models to capture the dynamics of real-time signals. It first learns the in-control autocorrelation structure from baseline data. Then, during production, if RTSPC detects significant deviation from the baseline model, it generates alarms.

The time series models used are ARIMA(p,d,q) models, where p is the auto-regressive order, d is the integration order, and q is the moving average order. The ARIMA models for a non-stationary time series X_t can be expressed by the following two equations,

$$w_t = -\sum_{k=1}^{p} \phi_k w_{t-k} + \sum_{k=1}^{q} \theta_k a_{t-k} \tag{2.1}$$

$$w_t = \nabla^d X_t \tag{2.2}$$

where w_t is the stationary time series after taking the dth difference on the original nonstationary series X_t, with error a_t, which is distributed as $N(0, \sigma^2)$.

With the ARIMA model, the prediction of the current stationary series is done by using past observations.

$$\hat{w}_t = -\sum_{k=1}^{p} \phi_k w_{t-k} + \sum_{k=1}^{q} \theta_k a_{t-k} \tag{2.3}$$

The actuseries is made stationary by taking the dth difference on the raw data as needed, i.e. $w_t = \nabla^d X_t$. Then the residual of the time series model is,

$$a_t = \hat{w}_t - w \tag{2.4}$$

The residual is a zero-mean IIND variable if the tool is in-control.

For some signals, the wafer-to-wafer variation is much greater than the within-wafer variation (Fig 2.5). RTSPC decomposes raw signals into long-term components (wafer-to-wafer) and short-term components (within wafer). Each component is modeled by an ARIMA model.

During production, the wafer-to-wafer averages and the within-wafer trends are filtered by their respective time series model in order to obtain the residuals $\mathbf{e}$. Then each component's residuals $\mathbf{e}$ from different signals is combined into a single score by Hotelling's statistics (Fig 2.6),

$$T^2 = \mathbf{e}^T \hat{\mathbf{S}}^{-1} \mathbf{e} \tag{2.5}$$

where $\hat{\mathbf{S}}$ is the estimated covariance matrix of the residuals, which may be computed in an exponential weighted fashion,

$$\hat{\mathbf{S}} = \sum_{i=0}^{k} \lambda^i \mathbf{e}(k-i)\mathbf{e}^T(k-i) \tag{2.6}$$

where k is the user-defined moving window length and λ is the exponential weighting factor.

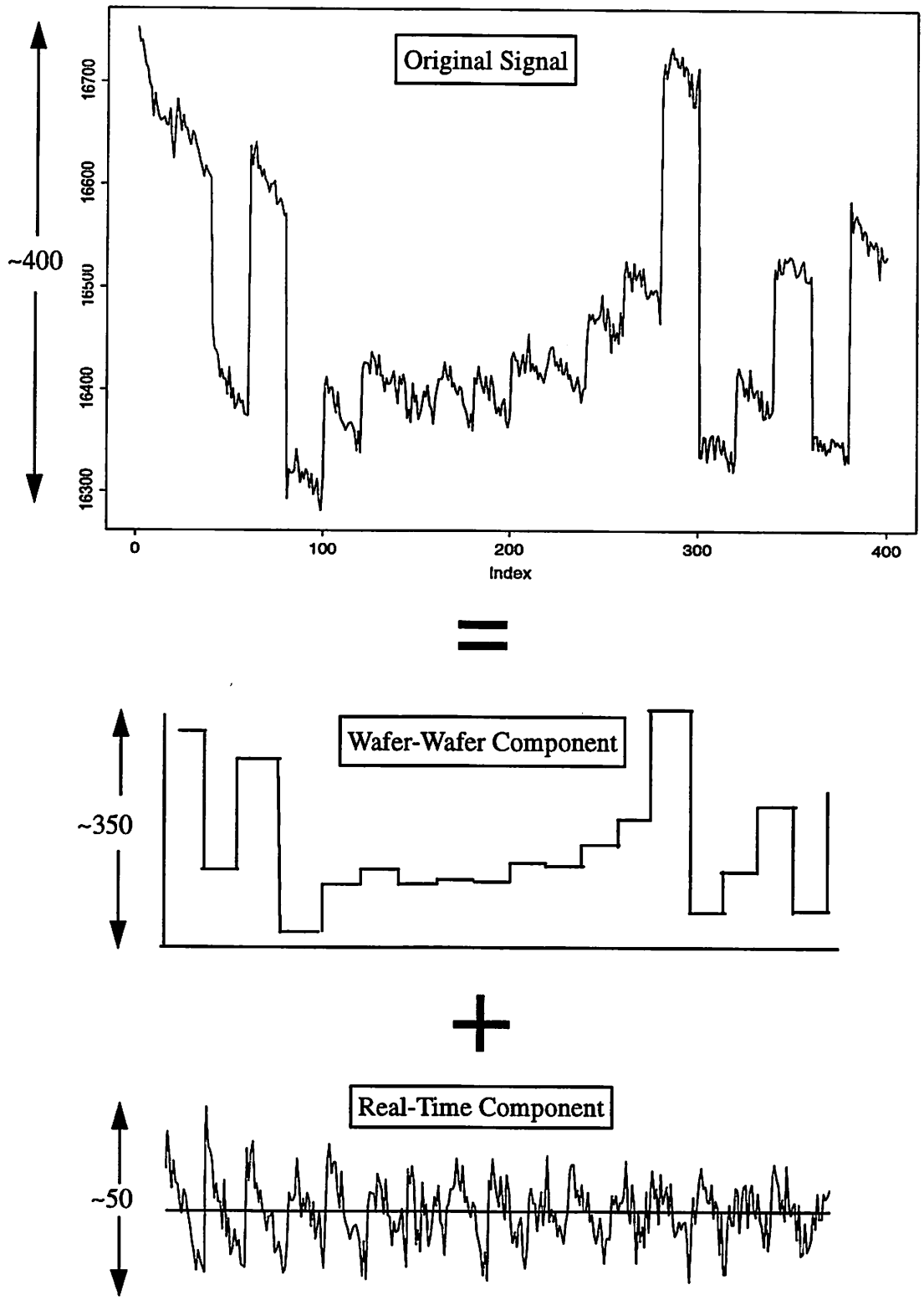


Figure 2.5.
Signal decomposition for the impedance signal.

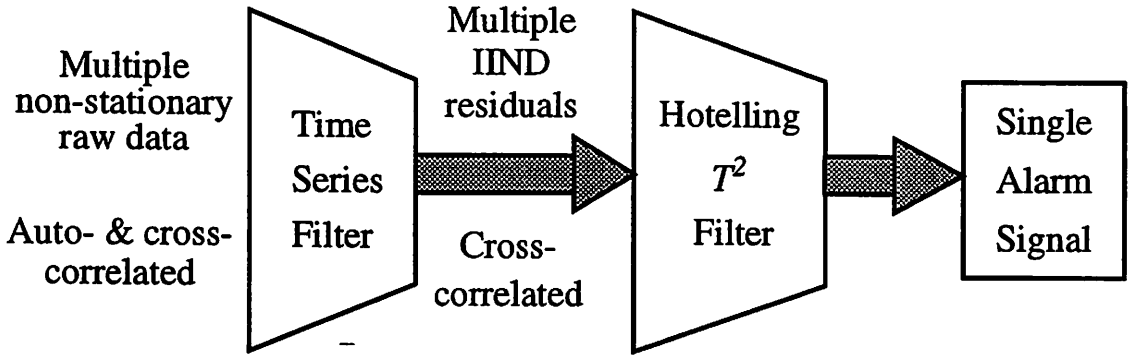


Figure 2.6.
Diagram of the real-time SPC scheme.

2.3.2. The Texas Instruments/Sematech Project (PCA analysis)[17]

During plasma etching, there is a large volume of data from the equipment. Also, signals from different sensors are highly correlated. The purpose of using principal component analysis (PCA) is to compress the data and extract relevant information. PCA splits the data matrix into systematic variation (process model) and noise (residual variance), For processing a wafer, data matrix **X** with m rows by n columns (samples by variables), can be expressed as,

$$\mathbf{X} = \mathbf{t}_1\mathbf{p}_1^T + \mathbf{t}_2\mathbf{p}_2^T + \ldots + \mathbf{t}_k\mathbf{p}_k^T + \mathbf{E} = \mathbf{T}_k\mathbf{P}_k^T + \mathbf{E} \tag{2.7}$$

Each variable in **X** has been centered by subtracting a 1 by n vector of the means of variables, and scaled by **d**, a 1 by n standard deviation vector. The $\mathbf{p}_i$ are called loading vectors, which are eigenvectors of $\mathbf{C} = \mathbf{X}^T\mathbf{X}$, the covariance matrix of **X**. They are a set of orthonormal vectors; i.e. $\mathbf{p}_i^T\mathbf{p}_j = 0$ for $i \neq j$, $\mathbf{p}_i^T\mathbf{p}_j = 1$ for $i = j$. The $\mathbf{t}_i$ are called the scores vectors, which for an individual sample, can be computed as,

$$\mathbf{t}_i = \mathbf{X}\mathbf{p}_i \tag{2.8}$$

where k is the number of principal components (PC) selected, which is less than or equal to the dimension of **X**, i.e., $k \leq min(m, n)$. For the highly correlated plasma etching real-time data, the number of PCs required to adequately capture the systematic variation of a process is far smaller than m and n.

Two statistics are used, "lack of fit" statistics Q and the Hotelling's T^2 statistics. Q is a measure of the amount of variation not captured by the PCA model.

$$Q_i = \mathbf{e}_i \mathbf{e}_i^T = \mathbf{x}_i(\mathbf{I} - \mathbf{P}_k \mathbf{P}_k^T)\mathbf{x}_i^T \tag{2.9}$$

Where $\mathbf{e}_i$ is the ith row of $\mathbf{E}$. T^2 is the measure of the variation within the PCA model,

$$T^2 = \mathbf{t}_i(\mathbf{T}_k^T \mathbf{T}_k)^{-1} \mathbf{t}_i^T \tag{2.10}$$

where $\mathbf{t}_i$ is the ith row of $\mathbf{T}_k$. Notice that $\mathbf{T}_k^T \mathbf{T}_k$ is a diagonal matrix due to the orthogonality of the $\{\mathbf{t}_i\}$ vectors. The diagonal entries of the matrix are eigenvalues of the covariance matrix of $\mathbf{X}$.

The mean vector $\mathbf{a}$, standard deviation vector $\mathbf{d}$ and the covariance matrix need to be updated in a exponential weighted way as new process data become available.

$$\mathbf{a}(j+1) = \sum_{j=1}^{J} \alpha^j \mathbf{a}'(J-j) \tag{2.11}$$

$$\mathbf{d}(j+1) = \sum_{j=1}^{J} \gamma^j \mathbf{d}'(J-j) \tag{2.12}$$

$$\mathbf{C}(j+1) = \sum_{j=1}^{J} \Gamma^j \mathbf{C}'(J-j) \tag{2.13}$$

where $\mathbf{a}'(J-j)$, $\mathbf{d}'(J-j)$, and $\mathbf{C}'(J-j)$ are the actual mean vector, standard deviation vector, and covariance matrix, respectively, for the jth measurement. α, γ, and Γ are the user-defined exponential weights. J is the window size of the past measurement. Notice that these model parameters depend only on the past observation. The PCA model is recomputed based on the covariance matrix $\mathbf{C}(j+1)$, i.e., new loading vector $\mathbf{p}_i$ and eigenvalues of the covariance matrix will be obtained. As the new process data $\mathbf{X}_{new}$ become available, it is centered with $\mathbf{a}(j+1)$ and scaled with $\mathbf{d}(j+1)$. Then new score $\mathbf{t}_{i,new}$ can be obtained by (2.8) with the new loadings. And Q, T^2 can be computed with (2.9), (2.10) by replacing $\mathbf{X}_{new}$ with $\mathbf{X}$. $\mathbf{t}_{i,new}$ with $\mathbf{t}_i$, and $\mathbf{T}_k^T \mathbf{T}_k$ with the new eigenvalue matrix.

2.4. Comment on Statistical Techniques

Statistical techniques usually rely on strict assumptions about the signal. A typical assumption is that, the signal is stationary (the mean value is unchanged; the noise is normally distributed and the variance is constant), or the drift is constant, etc. Statistical techniques cannot simultaneously deal with the many influences on the plasma etching signals. For instance, if there is a discontinuity in the etching signals due to temporary equipment downtime, the two statistical techniques will have to build the analytical models from scratch. Also, the two techniques cannot accommodate small spikes in the signals, which may lead to false alarms.

Overall, in order to analyze plasma etching signals, we need to be able to manage different influences in a flexible way. In the following chapters, we will look at two techniques that can provide this flexibility. We first investigate spectral analysis, and then syntactic analysis.

Chapter 3 An Attempt for Flexible Signal Representation Based on Spectral Analysis

3.1. Background and Motivation

The tool to estimate the spectral density is the periodogram. The periodogram is the sum of squares of the Fourier Transform of the time-domain,

$$I(\omega) = \left(\sum_{t=1}^{n} e^{-i\omega t} X(t) \right)^2 / n = \frac{1}{n} \left[\left\{ \sum_{t=1}^{n} X(t) \sin \omega t \right\}^2 + \left\{ \sum_{t=1}^{n} X(t) \cos \omega t \right\}^2 \right] \quad (3.1)$$

where X(t) is the time domain data, n is the number of samples.

Spectral analysis is mainly used to search for periodicity in the signals [18]. If there is indeed periodicity in the signals, spectral analysis is effective for diagnosing the physical mechanism responsible for generating the harmonics. Although a harmonic's time domain waveform can be easily distorted, and its magnitude can be different due to different conditions, the harmonic's power will remain conspicuous in the frequency domain. Due to this fact, frequency modulation is now commonly used in broadcasting, as to transmit data via noisy channels. Also, spectral analysis is used widely in natural science and medical science in order to analyze various phenomena, which might be very difficult to interpret in the time domain. The following are a few examples.

Michelson (1913) computes the periodogram for some sunspot data. He finds that, on the frequency domain, there is a significant peak at around 0.09 cycles/year. This corresponds to the traditional sunspot period of 11 years [19].

Nuclear magnetic resonance (NMR) is a resonance effect occurring in a particular substance when the applied magnetic field frequency matches a nuclear precession frequency of the substance. Ernst and Kaiser (1970) applied a random magnetic field, X(.), to a 2, 3-dibromothiophene (2,3-DBT) and measured the response Y(.). The periodogram of Y(.) show two doublets. Notice that the power spectrum of the random noise input X(.) is roughly constant across the measured frequency band. This shows that there are four NRM frequencies for 2,3-DBT. Scientists can make use of the NMR effect to classify different substances, since the nuclear precession frequency is an intrinsic property of a substance.

In the medical field, the physician judges the status of a patient's heart by looking at the Electrocardiogram (ECG). An ECG signal is measured with electrodes placed on the skin. For a cardiac cycle, an ECG signal consists of the following components: P wave, PQ segment, QRS complex, ST segment, T wave and TP segment. A heart rate variability plot (HRV) is constructed by extracting the T waves from the ECG signal, catenating them, and feeding them to a low pass filter with 0.5Hz cutoff frequency. Rompelman, et al., found out that in the HRV power spectrum, high frequency components are associated with respiratory fluctuations; medium frequency components are associated with blood pressure fluctuations; low frequency components are associated with body temperature fluctuations [20].

These uses of the periodogram led us to the hypothesis that perhaps by using spectral analysis, we can classify various faults based on their spectral signature. Also, perhaps, we can figure out the amount of machine aging and drift effect in the low frequency components; we might also assess the signal waveform generates by the underlying etching signal from the medium frequency components; and we could estimate the noise contribution by examining the high frequency components.

3.2. Pre-whitening of Plasma Etching Signals

In order to apply spectral analysis on plasma etching signals, we have assumed that there is periodicity in the etching signals. The task is to make the spikes of the harmonics stand out in the frequency domain. However, the power of the signal waveform generated by the physical etching mechanism is usually a lot stronger than small perturbations, such

as aging and various noisy events. In order to see the major components associated with the perturbations, we need to perform the prewhitening operation on the time series data, as to make the resulting signal approximately stationary. Here, prewhitening is the same as taking the difference on the time series data. The rationale of prewhitening is explained as follows.

Assume X(t) is of the following form,

$$X(t) = f(t) + e(t) + \sum_{j=1}^{K} R_j \cos(\omega_j t + \phi_j), \text{ t=1, ..., } n \tag{3.2}$$

where f(t) is associated with the physical etching mechanism, and can be modeled by an ARIMA(p,d,q) model. e(t) is the normal error term. The third summation term is the expression for sinusoidal perturbations, assuming there are K harmonics. The R_j, ω_j and ϕ_j are the amplitude, frequency value and phase, respectively, for different harmonics.

Take the dth difference on X(t), we have,

$$\nabla^d X(t) = \nabla^d f(t) + e(t) + \sum_{j=1}^{K} R_j \nabla^d \cos(\omega_j t + \phi_j) \tag{3.3}$$

Since f(t) is of d integrative order, by taking the dth difference, $\nabla^d f(t)$ is a stationary series. The difference operation does not change probability distribution of the error term. Taking differences on a cosinusoidal time series yields another sinusoidal (for d=2n+1) or cosinusoidal (for d=2n) time series. On the periodogram, the frequency components due to $\nabla^d f(t) + e(t)$ are independent from each other, with variance approximately $\sigma^2/(2\pi)$, and the variance does not decrease much by increasing the sample size n [18]. As for the summation term, the periodogram should show a spike of magnitude $R_j^2/2$ for each harmonic location.

Now, if $R_j \gg \sigma$, i.e., the amplitude of the cosinusoids is much greater than the standard deviation of the sum of the stationary time series and noise term, then harmonic ω_j

will stand out in the periodogram. On the other hand, if R_j is smaller or comparable to σ, the harmonics will not be able to distinguish themselves well in the periodogram.

3.3. Examination of the Technique on the Plasma Etching Signals

We assume that f(t) follows a second order integrative model, since after taking the 2nd order difference, the resulting data appears stationary. Figure 3.1 shows the waveforms, and their 1st and 2nd differences, and their respective spectra for two routine signals. The routine waveforms look very close to each other in their original time domain representation. However, in the spectral representation, they look like random noise. One cannot really tell whether they are the same or not in the frequency domain. Figure 3.2 shows the spectra for a routine run and a faulty run. We see that their original time series waveforms are very different, and we hope to see a large deviation between the waveform in the frequency domain. However, any deviation is "buried" in variance of the periodogram; we cannot really tell them apart. Figure 3.3, shows the difference between a routine run and a faulty run. We hope to find a sinusoidal perturbation due to the faulty condition. Yet, we cannot find any distinguishable harmonic in the frequency domain. To sum up after examining the spectra closely, it seems to us the assumption that the perturbation being sinusoidal is invalid. No obvious components stand out in the frequency domain. That is, we have a case where R_j is smaller or comparable to σ. Also, in the frequency domain, $var(I(\omega)) \cong I^2(\omega)$[18], that is the variance of the periodogram is approximately equal to the value square of the periodogram itself. Due to this noisy nature, it is difficult to categorize the components. From this data examination, it is suggested that spectral analysis is probably not suitable for analyzing plasma etching data.

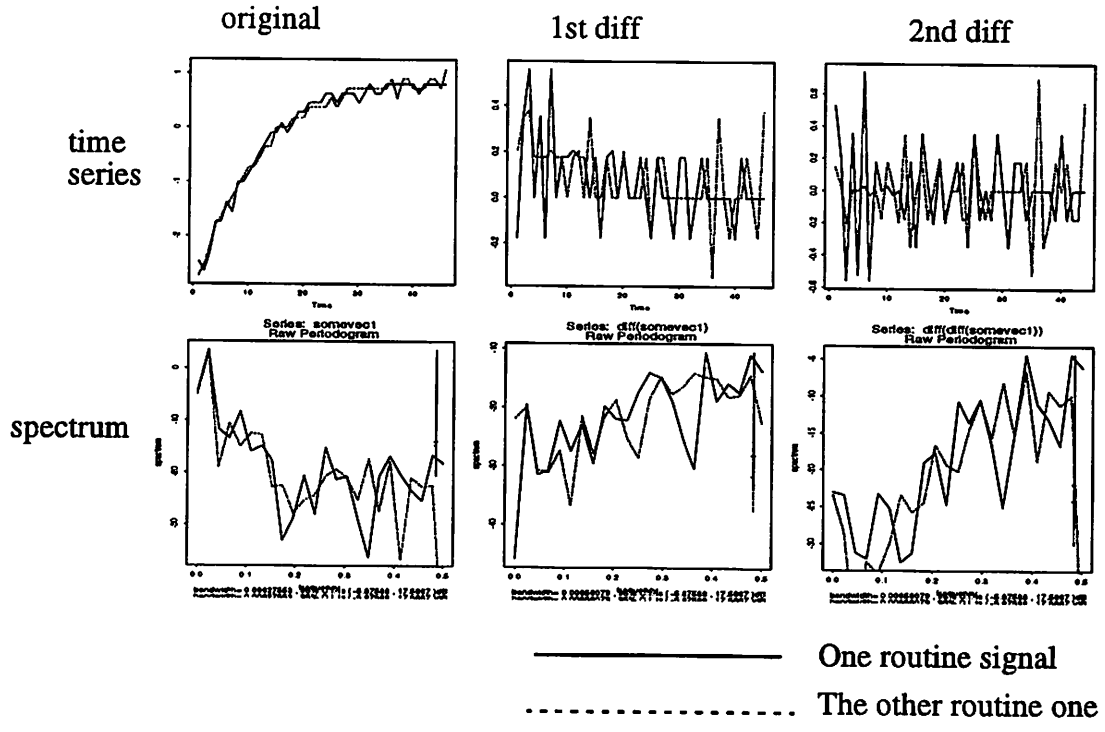


Figure 3.1. Time series and spectrum plots of two routine signals.

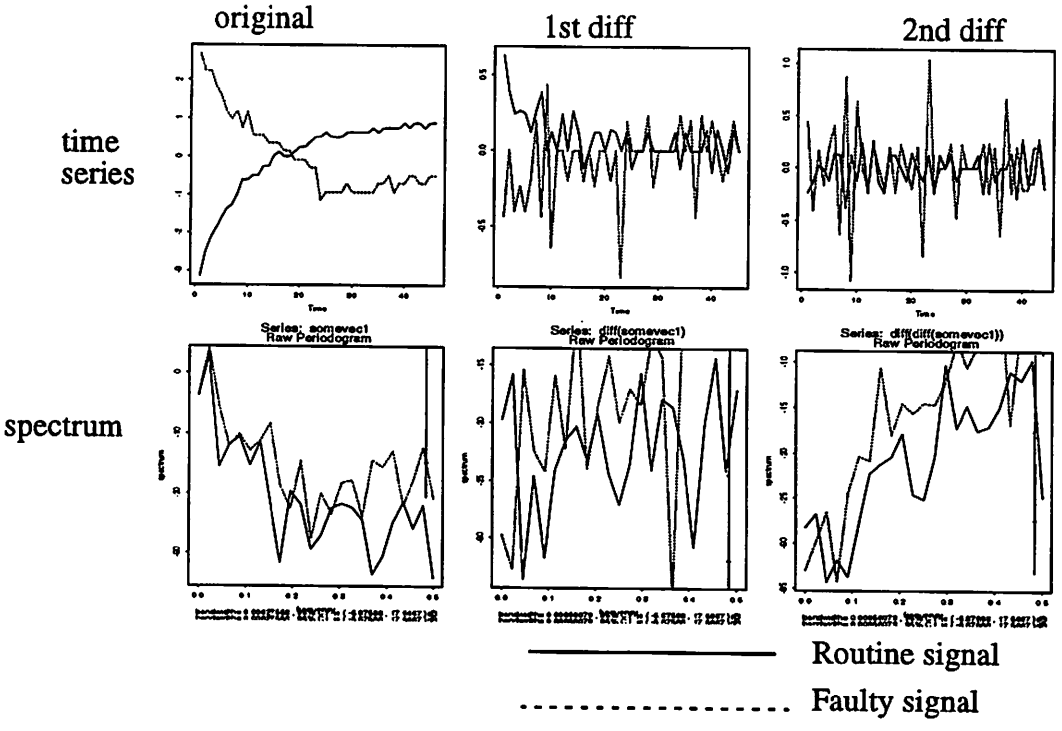


Figure 3.2. Time series and spectrum plots of a routine signal and a faulty signal.

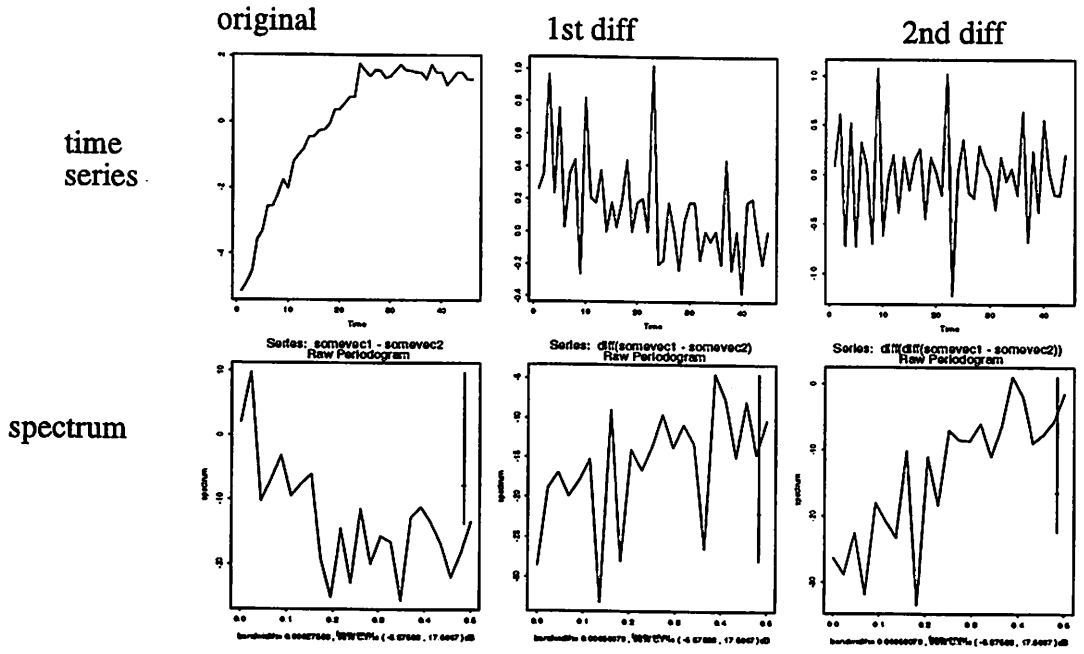


Figure 3.3. Time series and spectrum plots of the difference between a routine signal and a faulty one.

Recently (spring 1999), the author collected some plasma etching data with much higher sampling rates from the Microfabrication Lab. The new rate is 400 Hz instead of the usual 1-2 Hz in the industry. Figure 3.4 shows a time series plot of TCP load cap position and its frequency-domain representation. Some sinusoidal patterns in the waveform are clearly visible in the time domain. Some preliminary spectral analysis shows that quite a few harmonics are hidden in the time series waveform. At this point, the author is still working on coming up with better techniques for the analysis of the data. Wavelet transform and hybrid methods (combination of syntactic and statistical methods) are the two techniques under investigation.

This suggests that the reason spectral analysis cannot provide any useful information from the industrial data is that the sampling rate is too low. By increasing the sampling rate, a lot more diagnostic information about the etcher will appear in the waveforms of the

real-time signals. A setup for increasing the sampling rate has been done in the Microfabrication Lab. The result of analysis on the data will be presented in later works.

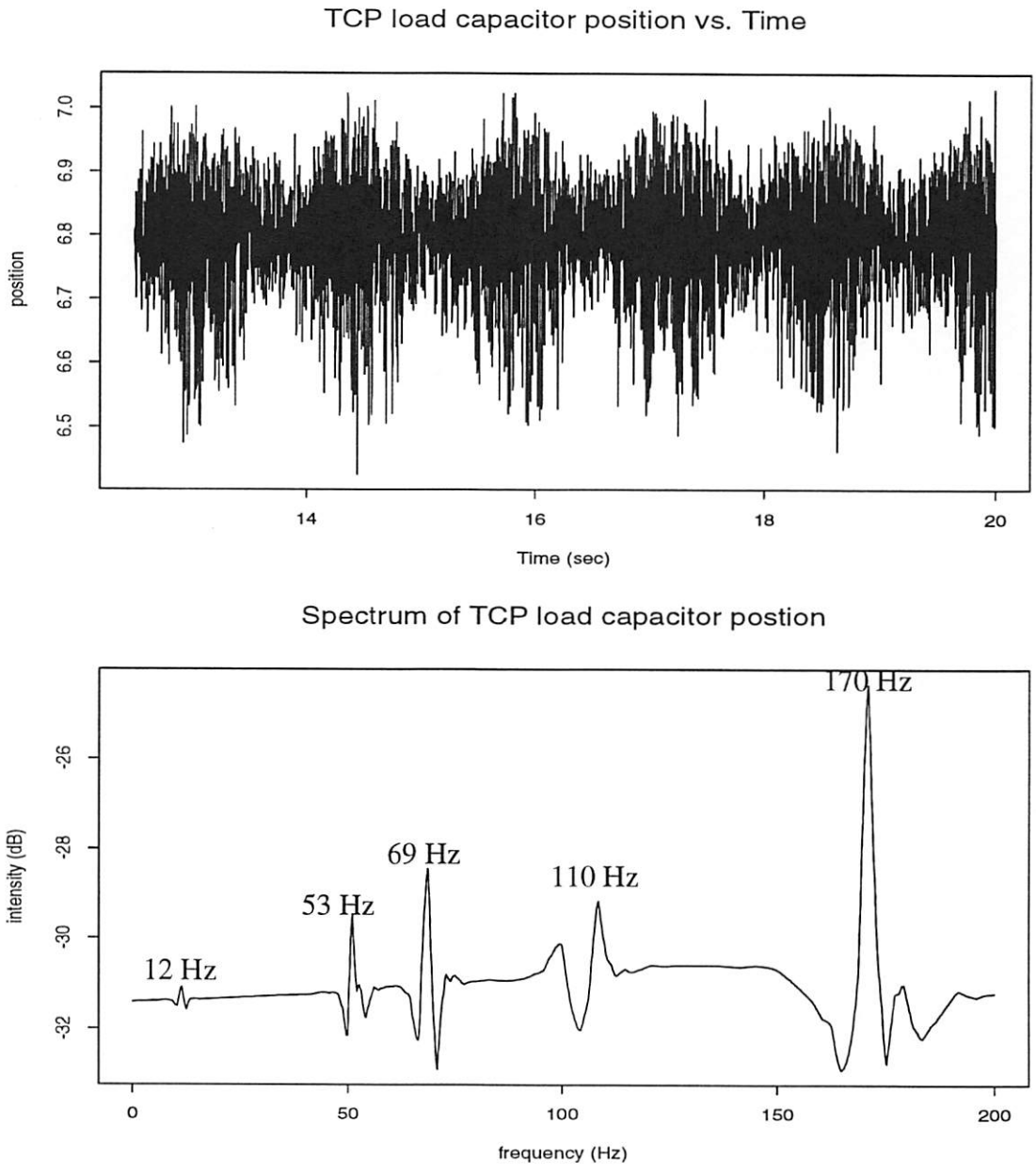


Figure 3.4. A time series of TCP load cap position at sampling rate of 400Hz, and its frequency-domain representation. A few harmonics are shown.

Chapter 4 Syntactic Analysis on Plasma Etching Signals

4.1. Introduction

Syntactic analysis refers to a general pattern recognition technique which uses formal language paradigms to describe the structure of an object. The basic approach is to decompose the object into subpatterns of primitives. By some criteria, a symbol is assigned to each primitive, and the symbols are assembled into a sentence. A grammar is a set of syntactic rules for generating sentences, which describes a class of objects. If the sentence encoded from an object is accepted by the grammar, then we consider that the object belongs to the class described by the grammar. Syntactic analysis is widely used for character recognition, especially in the Far East, where syntactic analysis-based Chinese character recognition is an active research area.

Syntactic analysis also has found some success in the medical field, for analyzing electrocardiogram (ECG) signals, in order to determine the status of a patient's heart. If done visually, the procedure is divided into two stages [10]. First, some characteristic features of ECG are recognized, such as the P wave, the PQ segment, the QRS complex, the ST segment, the T wave and the TP segment. Then, the physician measures the features' parameters, such as durations and amplitudes, and interprets these numerical values based on experience and a set of established empirical diagnostic criteria. Due to the massive amount of ECG data, there has been a great interest for computerizing the interpretation process. Many medical researchers have used syntactic pattern recognition techniques to analyze ECG signals [1-5]. Basically, they try to build an ECG processing system to imitate the physician, and to draw similar judgements about the status of the patient.

ECG signals are similar to the plasma etch signals in some respects. In addition to considerable amount of noise, their form and size can change over time [3]. Also, like etching signals differring from machine to machine, ECG signals differ considerably from person to person. Syntactic analysis is applicable since it is robust against gross change, and also appeals to intuition. Even if a signal has been "rubber stretched"(i.e. linearly transformed along the x- and y- axes), if the signal is classifiable by a human expert, then syntactic analysis can usually classify it correctly. This is the reason the author thinks syntactic analysis is a preferred choice for analyzing plasma etching signals.

4.2. The Syntactic System

A syntactic system for analyzing the etching signal of a capacitance manometer is presented here. The system attempts to discriminate among various waveform types. Figure 4.1 shows the overall block diagram. When a raw signal comes in, the waveform is preprocessed to facilitate further analysis. Then the waveform is encoded into a string of integers. The string is fed into the classifier to determine the fault category. There is also a numerical spike evaluator in the classifer. We will point out its necessity when we talk about the classification result. The major parts of this syntactic analysis system will be described next.

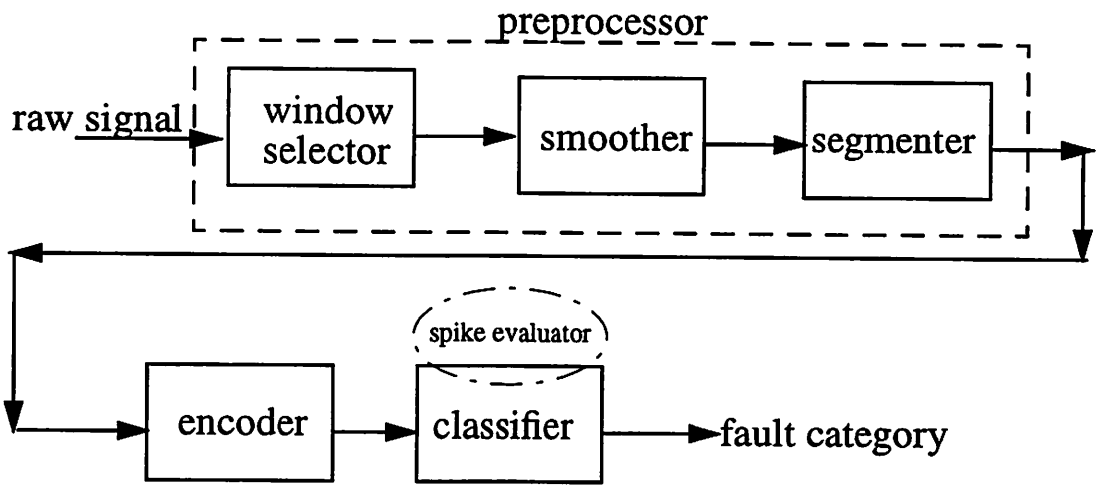


Figure 4.1. Architecture of the overall syntactic system.

4.2.1. Preprocessor

The preprocessor performs three operations: windowing, smoothing, and segmentation (Figure 4.2). Windowing refers to choosing the appropriate time interval for observation during the etch cycle of one wafer. The time window we select is usually the later part of the main etch step. For the capacitance manometer signal, there are two dominant positive spikes (as opposed to the minor ones in the stable region), one big, and one small, before the relatively stable region, so we can define a window after the small spike. Since we do not do any analysis on the random, high frequency noise, we can smooth out the noise of the windowed waveform. We use an algorithm called Locally Weighted Scatter Plot Smoothing [13]. This algorithm will try to predict each point of the signal by interpolation, by appropriately weighing the nearby raw data. The smoother lets the user specify the fraction of total data used for predicting a particular point; the larger the fraction, the smoother the fit. For the capacitance manometer, a fraction value of 0.2 is appropriate in the sense that this transformation seems to preserve the features that are analyzed later by the segmentaton algorithm and the classifier.

To divide the smoothed waveform into segments, first, a linear fit error tolerance is selected. The segment starts from the first data points. The segment keeps growing with successive points until the maximum linear fitting error is greater than the tolerance. The second segment starts with the end of the first segment, and this procedure is repeated until the entire window can be represented by linear segments.The following is the pseudo code for the segmentation algorithm.

Segmentation
Input: Time series $X=\{x_1 \ldots x_n\}$; linear fit error tolerance ε.
Output: List of line segment $L=\{s_1 \ldots s_n\}$.

```
h=1, i=1, j=1;
j=j+1;
while (j<n)
        s=linear model fit on {x_i ... x_j};
        maxerror = max {prediction error of s};
        if (maxerror > ε)
                s=linear model fit on {x_i ... x_{j-1}};
                L=append(L, s)
                i=j-1, j=i+1;
        else j=j+1;
L=append(L, s);
return L;
```

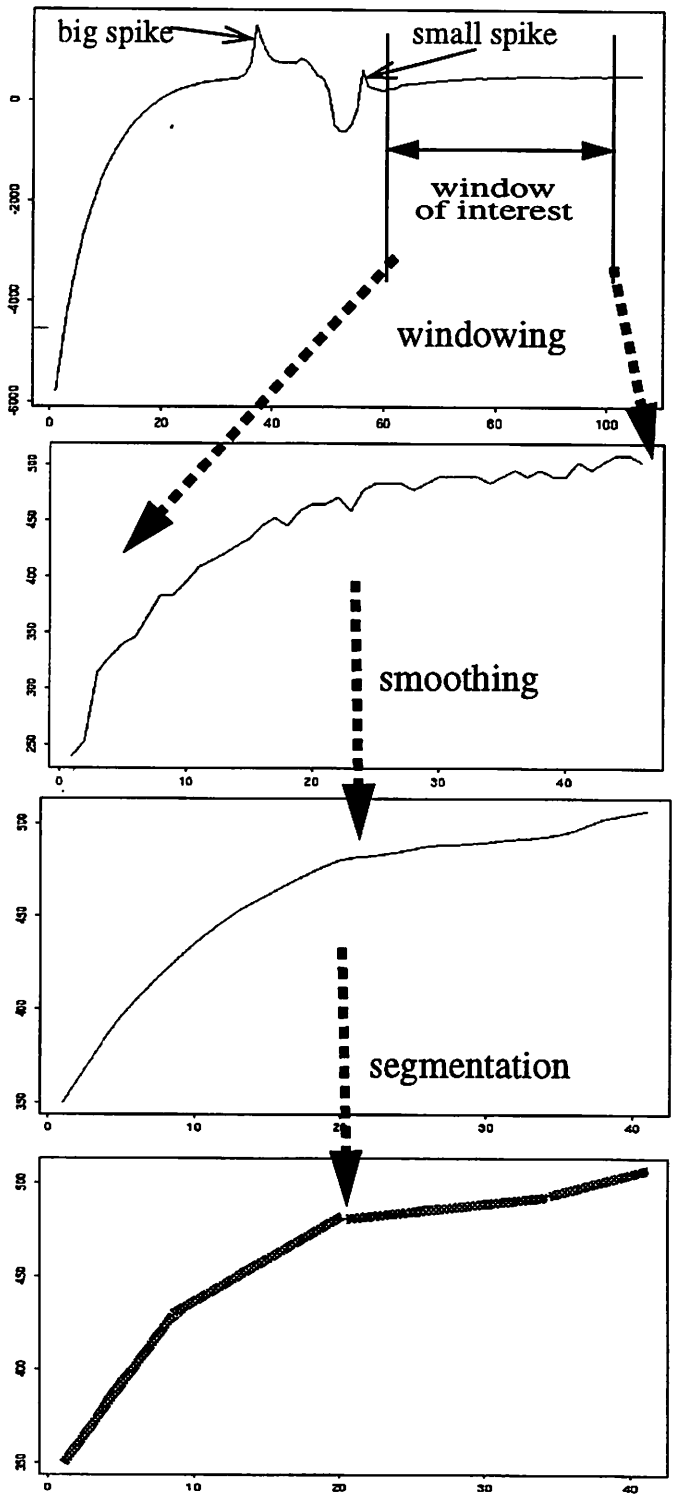


Figure 4.2. The process flow of the preprocessor.
The sample rate of the original signal is 2 samples per sec.

4.2.2. Encoder

Figure 4.3 shows the encoding scheme which converts the sequence of segments into a string of integers. Five integers are used for encoding the slope of the segments: 2 (fast increasing), 1 (slowly increasing), 0 (almost flat), -1 (slowly decreasing), and -2 (fast decreasing). For the windowed waveform of the capacitance manometer, we consider a segment with a slope of magnitude more than 10 units/sec to be fast changing, less than 4 units/sec to be almost flat, and the in-between values to be slowly changing.

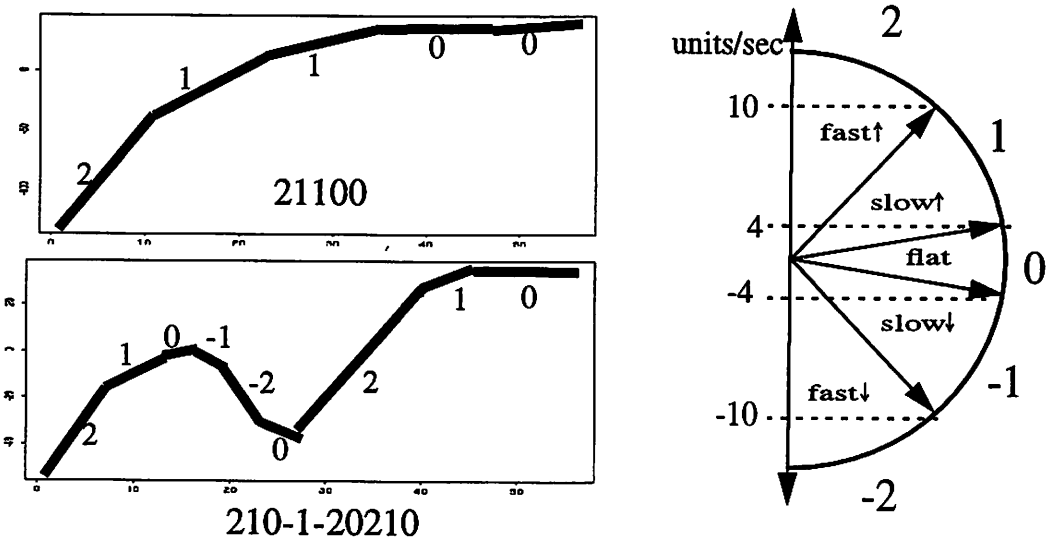


Figure 4.3. The encoding scheme.

4.2.3. Classifier

The classifier's operation is based on regular expression representation. Regular expressions are used to build the classifier. The expressions are used for matching the encoding string from the raw data. Assuming that x is an integer variable, we show some examples of regular expressions:

x*: zero or more x, i.e., <empty>, x, xx, or xxxxxxx.

x+: one or more x, i.e., x, xx, or xxxxxxx.

x?: zero or one x, i.e.,<empty>, x.

x|y: either x or y.

For example, the following strings are all represented by the same regular expression, 2*1+0*.

2*1+0*: 2222110000, 22221, or 100000.

Based on the process engineer's knowledge on the different waveforms, the classifier can be built to describe the shape of waveforms with one regular expression for each. After an incoming etching waveform is encoded into an integer string, the classifier will try to match the string to one of the regular expressions, and thus determine its category. For example, the following regular expressions can be used to describe the waveforms shown in Figure 4.3:

2+1+0+: describes a curve that first increases rapidly, then stabilizes and finally flattens out. (I.e. the first encoding example 21100 from Figure 4.3)

2+ {1*0*(-1)*(-2)*(-1)*0*1*2*} 1+0+: describes curves that are the sum of the 2+1+0+ curve and a possible negative spike. (I.e. the second encoding example 210-1-20210 from Figure 4.3. Notice that the expression within the curly brackets represents the spike).

However, actual real-time signals may evolve quite a bit over time. The normal waveform may be "stretched" in time or amplitude; the spikes of type 1 and 2 can appear at various times, with varying amplitude and duration, relative to the base waveform. Care should be taken when one derives a regular expression, so that the expression is flexible enough to accept variants of the waveform. Let us discuss in some detail the regular expressions for the five different waveforms in our data.

4.2.3.1. Regular expressions for five waveform categories

Normal:

The normal waveform has a shape similar to the first example in Figure 4.3. It first increases rapidly, then stabilizes and flattens out. However, expression 2+1+0+ will not be

appropriate enough to describe this decreasing trend of positive slope. Due to the "rubber stretching" effect, sometimes slope code of 2 or 1 might not appear in the integer sting. The engineer must exercise discretion in deriving the regular expression. Strings without a "2" or "1" should be accepted. An expression that would accommodate this range of signals is **2*1+0+ | 2+1*0+.**

Type 1:

The type 1 waveform is the sum of the normal waveform and a negative noisy peak. Because the amplitudes of the peaks are different, and they are added to an increasing curve, the encoding representation might not contain negative slopes. For instance, see the waveform in Figure 4.4. Also, the peak might appear in any position relative to the normal curve, so it is necessary to consider all scenarios of where the peak appears. The notation of Nxy is used for describing the peak, where N stands for negative peak; x is the slope encoding value before the peak; y is the slope encoding value after the peak. The encoding for the peak is in this format:

(starting segment, left arm, right arm, ending segment)

N_{end} stands for the negative peak occurring at end of the waveform; flat segments do not need to appear after N_{end}. P_{end} is the positive peak defined similiarly. For the peak coding N22, **(2 (-2|-1|0|1)+ (0|1|2)* 2),** i.e., the negative peak occurring within the fast increasing "2" region, line segments with slope code less than 2 will be considered as a valid left arm; also, it is not necessary to have a right arm.

N22=(2 (-2|-1|0|1)+ (0|1|2)* 2)
N21=(2 (-2|-1|0)+ (0|1|2)* 1)
N11=(1 (-2|-1|0)+ (0|1|2)* 1)
N10=(1 (-2|-1)+ (0|1|2)* 0)
N00=(0 (-2|-1)+ (0|1|2)* 0)
N_{end}=(0 (-2|-1)+ (0|1|2)*)
Type1=2*{N22}?2*{N21}?1*{N11}?1*{N10}?0*{N00}?0*{N_{end}}?

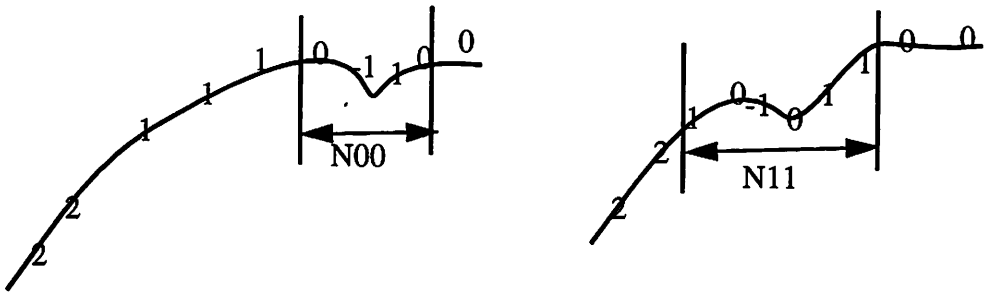


Figure 4.4. Two examples of negative peaks.

Type 2:

The type 2 waveform is the sum of the normal waveform and a positive noisy peak. Similar to the negative-peak type 1 case, it is necessary to consider all scenarios of where the positive peak appears. Notice that there are no P22 and P21. A positive peak has segments with slopes greater than the segments before it. However, as "2" is the largest slope coding value, it is not possible to have a segment with the slope coding value greater than 2. Thus under this coding scheme, it is not possible to have a possible peak within, or right after a region of segments with coding values "2."

P11=(1 2+ (1|0|-1|-2)* 1)
P10=(1 2+ (1|0|-1|-2)* 0)
P00=(0 (1|2)+ (0|-1|-2)* 0)
P_{end}=(0 (1|2)+ (0|-1|-2)*)
Type2=2*1*{P11}?1*{P10}?0*{P00}?0*{P_{end}}?

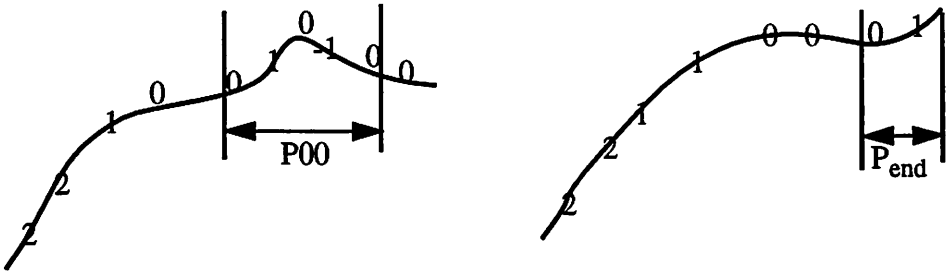


Figure 4.5. Two examples of postive peaks.

Type 3:

The type 3 waveform is more or less the inverted version of the normal waveform. It first decreases rapidly, then stabilizes and flattens out. Again, due to the "rubber stretching" effect, there may not be a "-1" or a "0" in the encoding strings, so the regular expression for type 3 is **(-2)+(-1)*0+ | (-2)+(-1)+0***.

Type 4:

The type 4 waveform has a more complicated valley-like shape. There is a bump at the bottom of the valley. The expression is

(-2)*(-1)+0*(-2)+(-1)*0*1*2*1*0*(-1)*(-2)*0*1*2+1*0*.

4.2.4. First-pass classification result

Table 4-1 summarizes the classification result based on the system described above.

Table 4-1. Waveform category distribution, first-pass result.

Type	Normal	1 & 2	3	4	unknown
Correct	1180	221	9	2	3
Miss	2	0	1	0	--

Let us examine this table. There are 3 "unknown" signals that could not be classified as any of the predetermined types. Many normal waveforms may have small spikes; proper smoothing and quantizing prevents them from showing up in the encoding. The two misclassifications for the normal begin with a "1" followed by "2s" instead of beginning with a "2". The type 3 misclassification has a small negative spike. Lastly, the system basically cannot distinguish if there is a positive or negative spike to the normal template, although it is able to detect a significant slope change in the otherwise monotonically increasing waveform. This ambiguity can be explained by Figure 4.6. Depending on how we interpret the different curve regions, we might come up with a positive or a negative spike for the same curve.

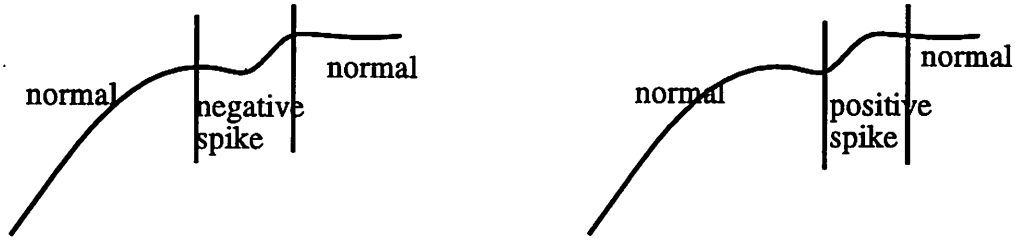


Figure 4.6. Two possible interpretations of the shape of the same curve.

4.2.5. Spike Evaluator

One apparent way to resolve this structural ambiguity is to add quantitative measuring ability to the classifier, in order to find out the sign and magnitude of the spikes. Similar schemes have been implemented for ECG waveform analysis. For example, for more accurate ECG waveform classification, Koski, et al. [3] compute the amplitude and duration of candidate P wave and T wave. Based on these numerical attributes, the wave in question is designated as a noisy waves, a P wave or a T wave. Here, a spike evaluator is proposed to measure the magnitude and sign of spikes (Figure 4.7). We first take the smoothed signal, centered, and standardized by its standard deviation, and we then subtract a reference sig-

nal. On the residual plot, the maximum peak value represents the value of the spike. In our study, we put a threshold of 0.3, which means that if the spike is less than 0.3 times the standard deviation of the signal, we consider the process to be normal. Using this criterion, 60 examples of type 1, and 29 of type 2 are classified as faulty. The improved results are shown in Table 4-2. Notice that a small spike added to a signal is a very common phenomenon. It should not be a surprise that out of the ten type 3 signals, one has a small spike. If we construct a spike evaluator for fault type 3, the one classifying error due to the small negative spike added to the signal would be corrected as well.

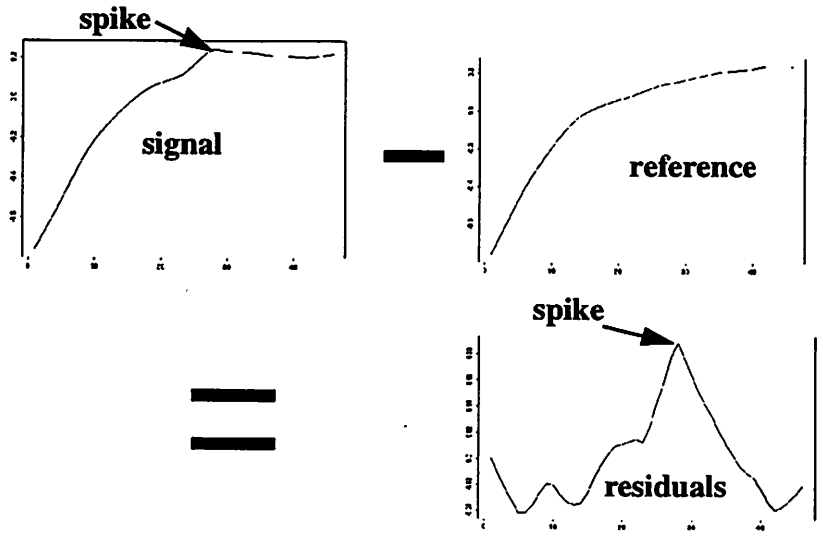


Figure 4.7. The way to measure the spike magnitude in the classifier.

Table 4-2. Improved waveform classification with the spike evaluator.

Type	Normal	1	2	3	4	unknown
Correct	1312	60	29	9	2	3
Miss	2	0	0	1	0	--

Chapter 5 An Advanced Case Study: Analysis of the "High Speed" Data

5.1. Introduction

High speed data is acquired during the plasma ignition stage, before any etching occurs. This stage is the transition between the pre-etch and the main etch. The sample rate is 100 samples per second, instead of one or two samples per second, as was the case when monitoring during the entire etching period. The high sampling rate is needed to capture the detail of the transition waveform, and this is where the term "high speed" comes from.

Our assumption about the "high speed" waveform is that each waveform corresponds to an operating condition. The goal of the analysis is to describe the shape of a waveform and thus determine its operating condition. There are two designed parameters for the operating conditions, namely, "tune" and "load." They can be assigned to different experimental levels, such as "high," "medium-high," "baseline," "medium-low," and "low."

Let us examine some waveforms. Figure 5.2 shows two baseline waveforms and two medium-low tune and load waveforms. For the baseline waveforms, the region between the first and second spikes might be somewhat different, otherwise, the two waveforms will have very similar structures. For the medium low tune and load waveforms, the region after the big positive peak can be quite different. Also, we can infer from inspection that the negative peak can be sometimes narrower (as in the first waveform) and sometimes wider (as in the second waveform). A human brain can effortlessly analyze the waveforms and come up the above observations. We will build our automated analysis system with these observations in mind.

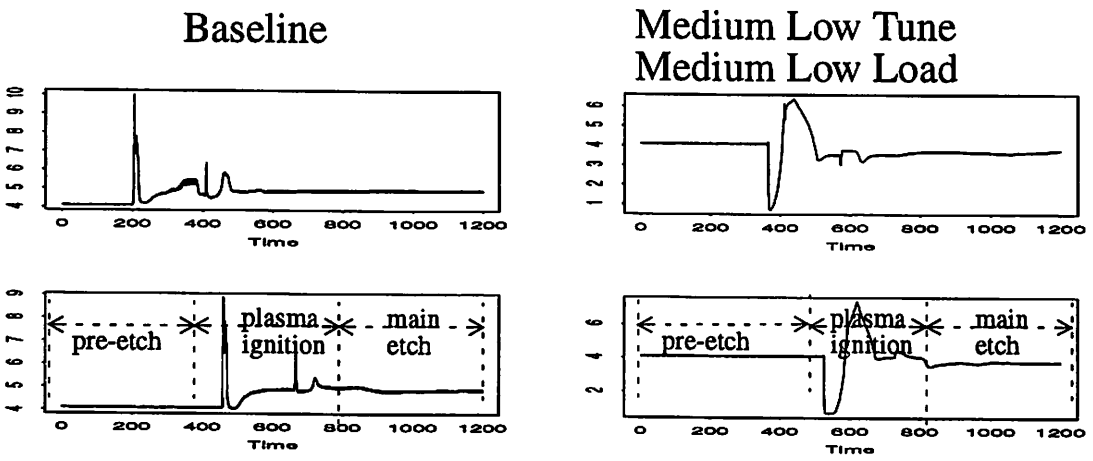


Figure 5.1. Two types of TCP line impedance waveforms for two different operation conditions.

Syntactic analysis is partly science and partly art. For accurate classification, the importance of engineering judgement cannot be overemphasized. This means that the rules encoded in the system are going to be highly specific to the nature of the data. The "high speed" data waveforms are much more complicated than the main-etch waveforms we analyzed previously, so we cannot use the analytic scheme for the main-etch waveforms. Using line segments as the primitive elements would make the classifier extremely complicated. Also, using slope attributes alone would not adequately describe the "high speed" waveforms.

Horowitz proposes a syntactic algorithm for detecting peaks in ECG signals [6]. Belforte uses a peak-coding table look-up method to analyze ECG signals [7]. After taking the first derivative on the raw ECG data, the waveform is parsed into peaks. Based on the amplitude and duration of a peak, a letter code is assigned to it. Trahanias and Skordalakis suggest using peak and segment as two types of primitives, and one can build a hierarchy for a waveform from the primitives in a bottom-up fashion [8][9]. However, the use of a "peak" as a primitive can be troublesome. Notice that if a positive peak is followed by a negative peak, the two peaks will share a common arm in the middle. That is, a lower-level element is being shared by two higher-level elements; this will complicate the syntactic

structure description. Also, it may be difficult to define the duration and amplitude of a peak if the left and right arms of a peak are uneven. Nevertheless, we believe that the recognition of complicated waveforms can be done in a fairly straightforward way, as discussed next.

5.2. The analysis system

A new scheme is proposed for recognizing "high speed" waveforms. Three types of primitives are used: UP (monotonically increasing), FLAT (approximately constant), and DOWN (monotonically decreasing). Each primitive consists of small straight line segments. For our data, the line segments with slope between -0.1 and 0.1 unit per data point are considered FLAT; less than -0.1, DOWN; greater than 0.1, UP. See Figure 5.3 for drawing of the primitives, and Figure 5.4 for the block diagram of the analysis system.

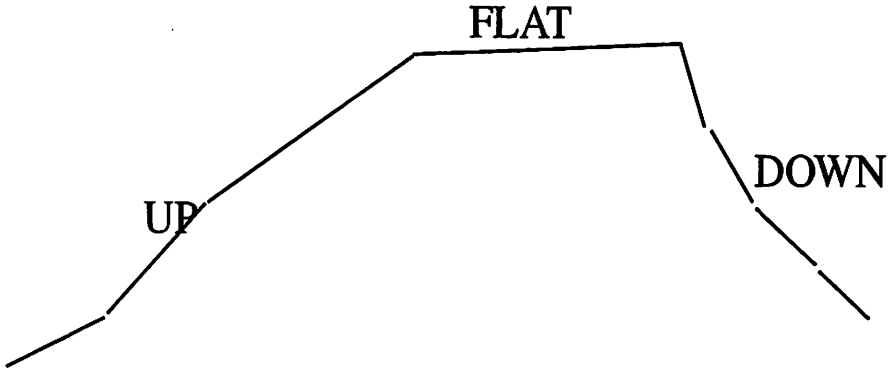


Figure 5.2. Illustration of three types of primitives.

5.2.1. Noise filter and segmentation

One encounters noisy line segments in extracting the primitives. See Table 5-1 (the rule table) for different cases of noisy segments. Also, look at Figure 5.1 where in the second baseline waveform, a lot of noisy segments occur between the first two spikes. Antonacopoulos, Economous [11] and Skordalakis [12] discuss some techniques for detecting and processing noisy peaks. Let us examine a few ways to filter out the noise.

Noise filtering scheme 1: smoothing

Here, the objective is to implement a low pass filter in order to eliminate high frequency noise. The Locally Weighted Scatter Plot Smoothing Algorithm from Chapter 4 works very well for the simple main etch waveform. However, if we apply the algorithm on a baseline waveform, we will lose some detail of the first spike. Also, the ratio between the magnitude of the first spike and that of second spike gets altered. Originally, the ratio is roughly 3:1. In the smoothed waveform, it is less than 2:1. This is an undesirable effect of the smoothing operation.

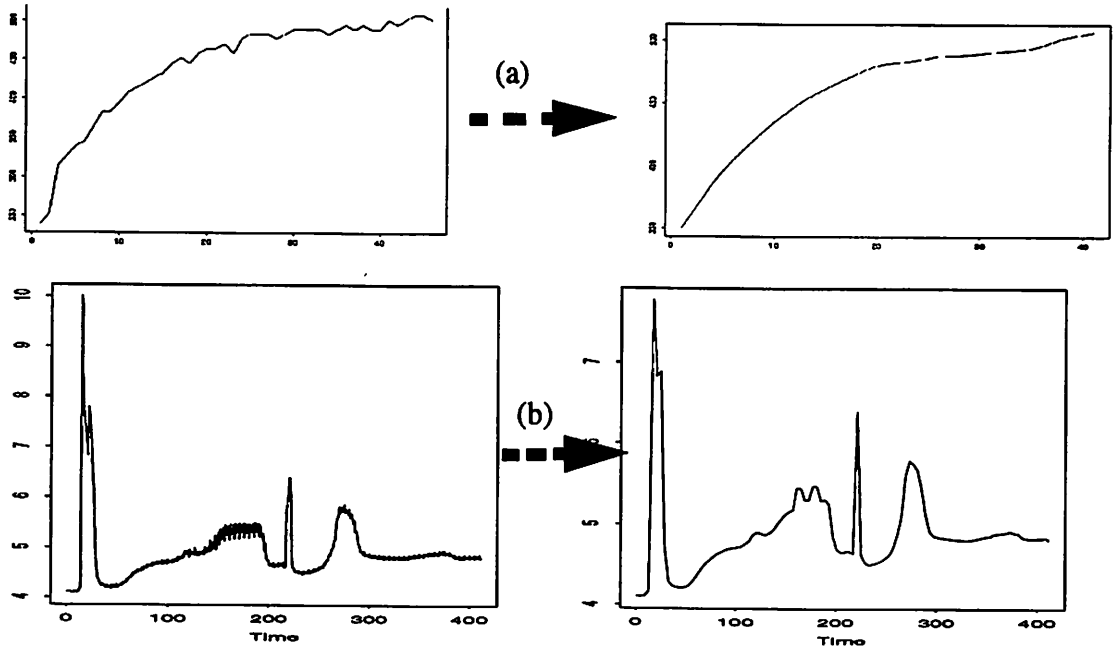


Figure 5.3. Smoothing. (a) on a main-etch waveform. (b) on a "high speed" baseline waveform.

Noise filtering scheme 2: piecewise-linear approximation

One can also attempt to filter out the noisy segments by applying a piecewise-linear approximation to the waveform, with the proper tolerance for the fitting error. The scheme is illustrated in the first part of Figure 5.4. However, for a complicated waveform, picking a good tolerance is hard, because some noise might be signal-dependent. Some region of the signal can be very noisy, while others might be relatively noise-free. If the tolerance is

too big, the approximation will not be adequate; if the tolerance is too small, filtering will not be effective in the noisy region. (See the second part of Figure 5.5)

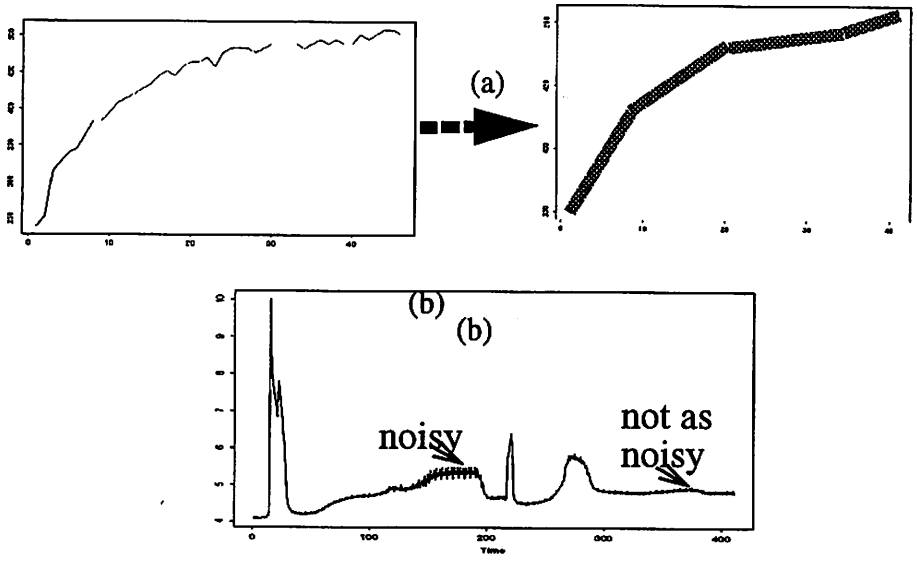
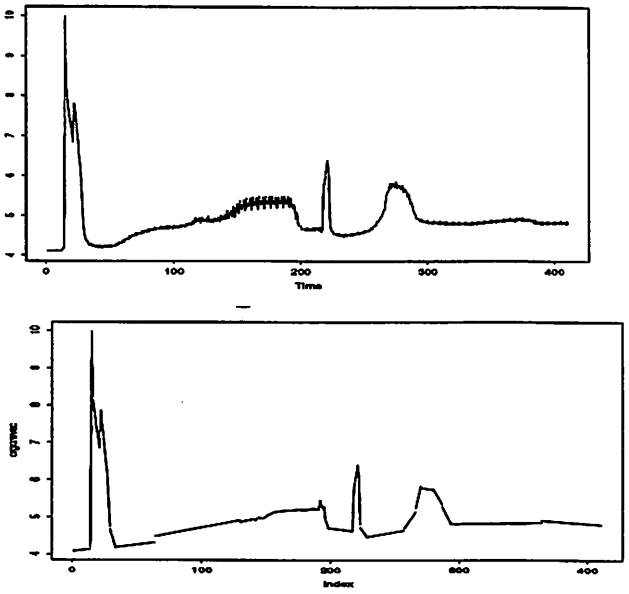


Figure 5.4. . (a) piecewise-linear approximation on a main-etch waveform. (b) Examination of a "high speed" baseline waveform.

Noise filtering scheme 3: processing noisy segments

This scheme first employs a piecewise linear approximation with a small tolerance, selected to be effective for the relatively noise-free regions of the waveform. We subsequently extract the UP, FLAT, and DOWN primitives, based on the rules shown in Table 5-1. In the high speed data, any line segment with duration less than 5 and amplitude less than 0.4 is considered noisy.



Not done yet!
More rules needed
to be applied.

Figure 5.5. Illustration of noisy peak processing with rules d and e (from Table 5-1) applied.

Table 5-1. Rules for processing noisy segments

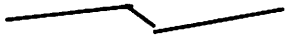
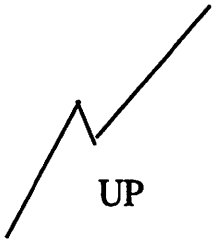
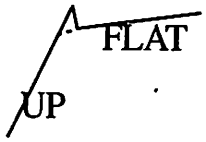
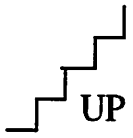
Waveform	Rule Description
FLAT 	(a) A noisy segment between two long FLAT segments. We consider all three segments as a FLAT primitive candidate.
UP 	(b) A noisy segment between two big UP segments. We consider all three segments as an UP primitive candidate.

Table 5-1. Rules for processing noisy segments

	c) A noisy segment a UP segment and a FLAT segment. A horizontal line is drawn through the noisy region and consider the region and the FLAT segment as a FLAT primitive candidate.
	(d) Noisy segments occur consecutively, alternating between UP and FLAT primitives. We consider the entire region as UP.
	(e) Noisy segments occur consecutively, alternating between UP and DOWN primitives. We use lines to connect bottoms of the segments. From the slope of the those lines, we determine the slope attribute for the region.

The variants of the above rules can be derived easily. For instance, the dual of (b); two negative DOWN segments with a positive-sloped noisy segment in between will be considered as a DOWN primitive candidate. By iterating over the above rules a few times, all the noisy segments should be filtered out.

5.2.2. Encoding

Three attributes are used to describe each primitive, in the form of {S, D, A}, where,

S is the slope code, which can be -1 (DOWN), 0 (FLAT), and 1 (UP);

D is the duration code which can be 0, 1, and 2, in order of length. If duration of a primitive is less than 10, D=0; between 10 and 30, D=1; greater than 30, D=2;

A is the amplitude code which can be 0, 1, and 2, in order of magnitude. If amplitude of a primitive is less than 0.4, A=0; between 0.4 and 2, D=1; greater than 2, D=2.

The criteria for quantization can be assigned based on the process engineer's experience with the signal. One should try to make the number of primitives corresponding to each quantized value roughly the same. This will make the task of building the classifier easier. Consider the case where we use a very strict criterion on the FLAT primitive, in which case only line segments with slope very close to zero will be assigned slope code of

0. Then the number of FLAT primitives will be very small, and it simply defeats the purpose of having a FLAT attribute; since the FLAT attribute were to be left largely unused, we might as well just two attributes, UP and DOWN.

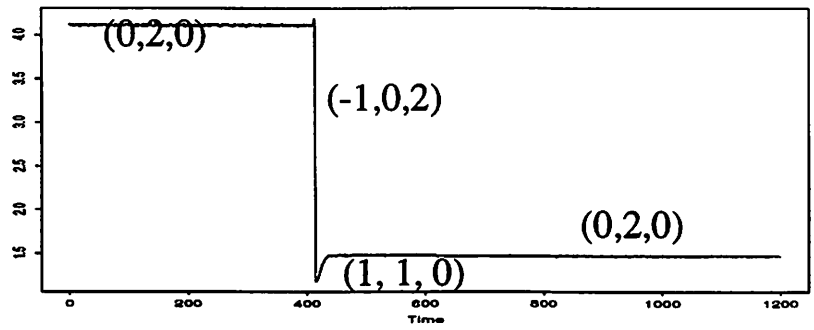


Figure 5.6. An encoding example. This is the low-tune and high-load waveform.

For the above low-tune and high-load waveform, the list of the numerical values for the primitives is

(0, 412, 0.02) (-1, 1, 2.99) (1, 23, 0.35) (0, 763, 0.01), which can be coded as,

(0, 2, 0) (-1, 0, 2) (1, 1, 0) (0, 2, 0).

5.2.3. Building the classifier

The syntactic rules have to be created to take into account the error tolerances used in extracting the primitives. In training the classifier, one should be careful with primitives close to the boundary value. If there is a reason to believe that the corresponding primitive of the subsequent waveform may take on either of the two encoding values which share a common boundary, we should use the logical OR ("|") operator on the two values, so that both values will be accepted.

Consider the third primitive of the above waveform. Its amplitude is 0.35, which is fairly close to the boundary value of 0.4. We should make the classifier accept both 0 and 1 for the amplitude attribute. The classifier for the waveform can be,

(0, 2, 0) (-1, 0, 2) (1, 1, *0|1*) (0, 2, 0).

Indeed, engineering intuition is of great help in building the classifier. Let us get back to the observation on the LHext waveform, which we mentioned in the introduction of this chapter. The basic idea is to write the regular expression based on the common region. Anything attached to the common region will be acceptable.

{Common}{Anything}

Anything = -2 | -1 | 0 | 1 | 2 | , | (|)

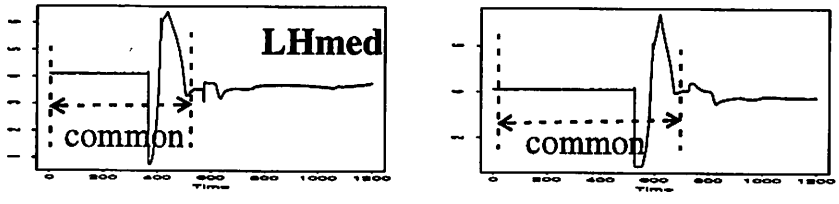


Figure 5.7. Highlight the common region in two waveforms collected at the same operating conditions.

In this case, the common region is a big FLAT segment followed by a negative peak and a positive peak. Notice that the top portion of the negative peak might be relatively flat. Therefore, after segmentation, a small FLAT primitive corresponding to the top of the negative peak might exist. With this in mind, the common region can be coded as follows:

top_flat = (0,0|1,0)

common= (0,2,0)(-1,0,2)*{top_flat}?*(1,2,2)(-1,0,1)(1,0|1,0|1)(-1,2,2)

Please see Appendix C for the flex code of the classifier.

5.3. Experiment setup

Figure 5.8 shows the basic schematic of the LAM 9400 plasma etcher, which is a transformer-coupled plasma (TCP) system. The inductive planar coils at the top of the chamber are wound from near the center to the outer radius of the chamber. Plasma is created by applying RF power to the inductive coil. Another RF power source is applied to the substrate for ion-bombardment of the wafer. There is one matching network for each RF source. The upper one is a capacitive network, consisting of two variable capacitors, the tune vane capacitor and the load capacitor. The lower one is a L-type network; the variable circuit elements are the tune vane capacitor and the load coil (see Figure 5.9). A matching network tries to match the impedance it "sees," as to maximize the power transfer from the RF source to the plasma. During the matching operation, we can acquire a list of signals from each network. Some useful signals for fault detection and diagnosis are listed in Table 5.2. For this work, we analyze TCP line impedance waveforms for classifying machine operating condition. For this classification purpose, it is sufficient to analyze just one signal. Multiple-signal analysis is still under investigation.

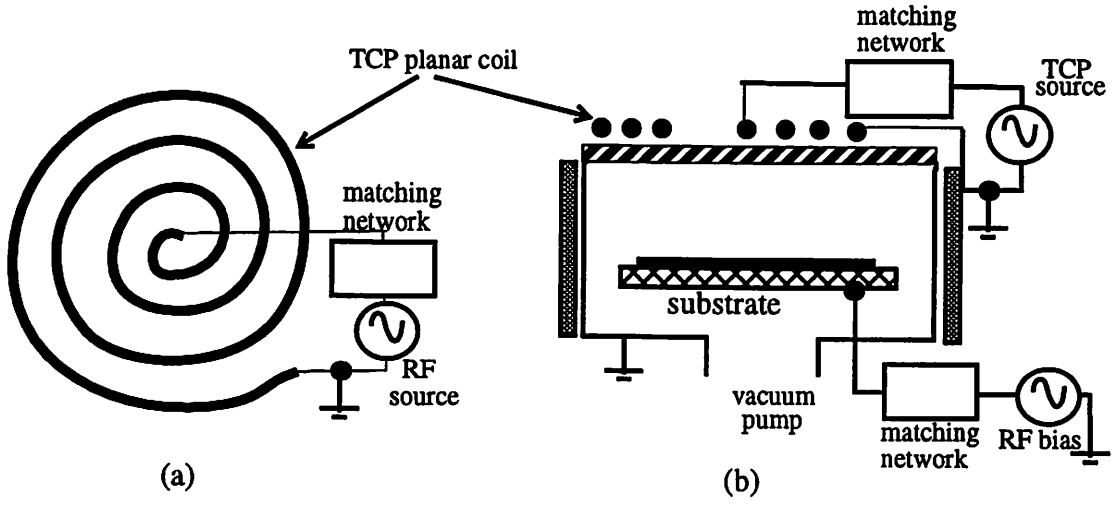


Figure 5.8 a) Top view of the inductive planar coil. b) The side-view illustration of a TCP system.[24]

Table 5.2. Real-Time Signals Collected for the Lam TCP 9400

	Position	Description
Upper Matching Network	TCP Tune Vane Capacitor Command	Value for the tune vane capacitor to match
	TCP Load Capacitor Command	Value for the load capacitor to match
	TCP Phase Control	Control signal of phase error between the current and voltage at the top coil
	TCP Tune Vane Capacitor Position	Position of the tune vane capacitor of the upper matching network for the top coil
	TCP Load Capacitor Position	Position of the load capacitor of the upper matching network for the top coil
	TCP Line Impedance	Apparent input impedance of the upper matching network
Lower Matching Network	RF Tune Vane Capacitor Control	Control signal for the tune vane capacitor of the lower matching network
	RF load coil Control	Control signal for the load coil of the lower matching network
	RF Tune Vane Capacitor Position	Position of the tune vane capacitor of the lower matching network
	RF Load Coil Position	Position of the load coil of the lower matching network
	RF power	Power transferring to the substrate
	RF Line Impedance	Apparent input impedance of the lower-matching network
	RF voltage	Substrate bias with respect to ground

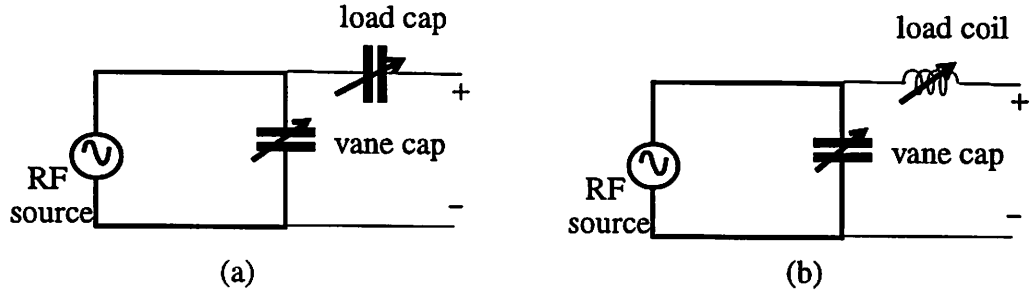


Figure 5.9 a) A capacitive matching network. b) An L-type matching network.

The two designed parameters for the operating conditions, "tune" and "load," are the pre-specified values for the two variable capacitors of the upper matching network to follow. As mentioned in the introduction, they each can have one of the experimental designed levels of "high," "medium high," "baseline," "medium low," and "low."

Each parameter is on a standardized scale, shown in Figure 5.9. "H" and "L" stand for high and low, respectively; "ext" and "med" stand for extreme and medium respectively. "HLext" means that the operating condition of extremely high tune and extremely low load. There are nine operating conditions: Baseline, HHext, LLext, HLext, LHext, HHmed, LLmed, HLmed, LHmed.

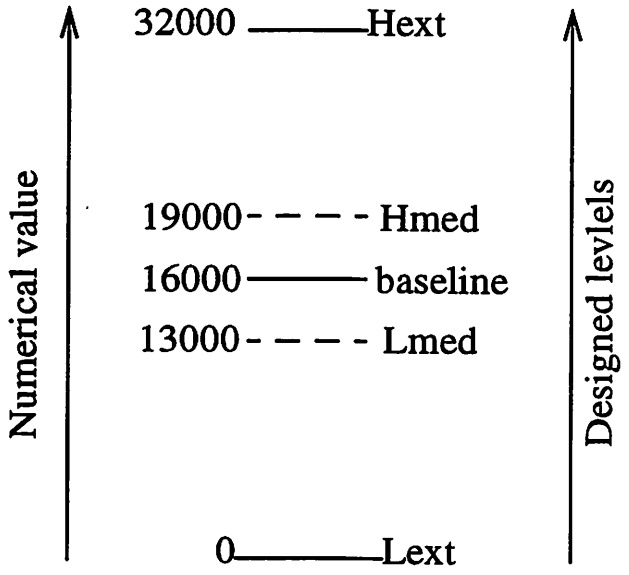


Figure 5.10. The designed-level description of the parameters tune and load.

5.4. Result and discussion

The results are summarized in Table 5-2. The bold italic wafer numbers signify the misclassified cases. The baseline miss has to do with the fact that a routine spike is significantly weaker in the other signals, so that the second and third peaks of the line impedance

signal disappear. the LLmed miss has to do with a high spike occurring in the common region, so that the recognizable pattern is greatly "damaged." Notice that if the high spike occurs far away from the common region (first waveform of Figure 5.9), the common region will not be altered, and thus classifying error will not occur.

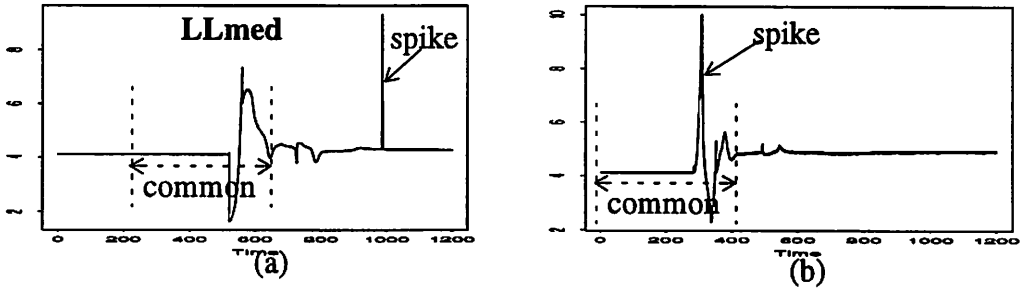


Figure 5.11. The high-spike effect on the waveforms. (a) The spike occurs far away from the common region. (b) The spike occurs right at the common region.

Finally, for the LHmed wafers, #26 and #28 waveforms are similar to LLmed ones (see Figure 5.8 and Figure 5.9). #27 waveform is similar to LHext ones (Figure 5.7). This means that #27 waveform is totally different from those of #26 and #28. As a matter of fact, the similarity which confused the classifying task is so great that even a human expert will not be able to make a distinction. This implies that probably any pattern recognition scheme will not tell those confounded waveforms apart. Therefore, the author will not consider this as a classification error.

Table 5-3. Result summary. The italic wafer numbers signify misclassification.

Type	Wafer number	Comment
Baseline	1,2,3,*16*	missing spike
HHext	4,5,6	
LLext	7,8,9	
HLext	10,11,12	
LHext	13,14,15	
HHmed	17,18,19	
LLmed	*20*,21,22	extra spike

Table 5-3. Result summary. The italic wafer numbers signify misclassification.

HLmed	23,24,25	
LHmed	26,27,28	confused, with LHx, LLm.

Chapter 6 Conclusion and Final Remarks

The syntactic method is shown to be robust and accurate for fault detection and diagnosis in plasma etching. For the successful operation of this system, the expertise of the process engineer plays a key role. The system complements the process engineer's expertise in interpreting the etching signals, therefore, parameters of the system must be trained to suit the engineer's needs. The following are a few observations on the method.

6.1. Similarity to DSP techniques

At a glance, syntactic analysis is quite similar to the encoding and decoding techniques in digital signal processing (DSP). In DSP, the engineer first defines a number of logical values, and assigns a voltage or frequency level for each logical value. The data is presented with a stream of logical values, encoded into physical signal levels (voltage or frequency), and transmitted over noisy channels. The receiver will try to ignore the noise in the received signal, and try to match it to a predefined logical value.

In syntactic analysis, we define a number of fault categories based on our experience. For diagnosing a plasma etching signal, we would ignore effects such as machine aging, preventive maintenance, chamber memory, small spikes, and so on, and try to match the signal to a predefined fault category. The similarities between syntactic analysis and DSP are highlighted in Figure 6.1.

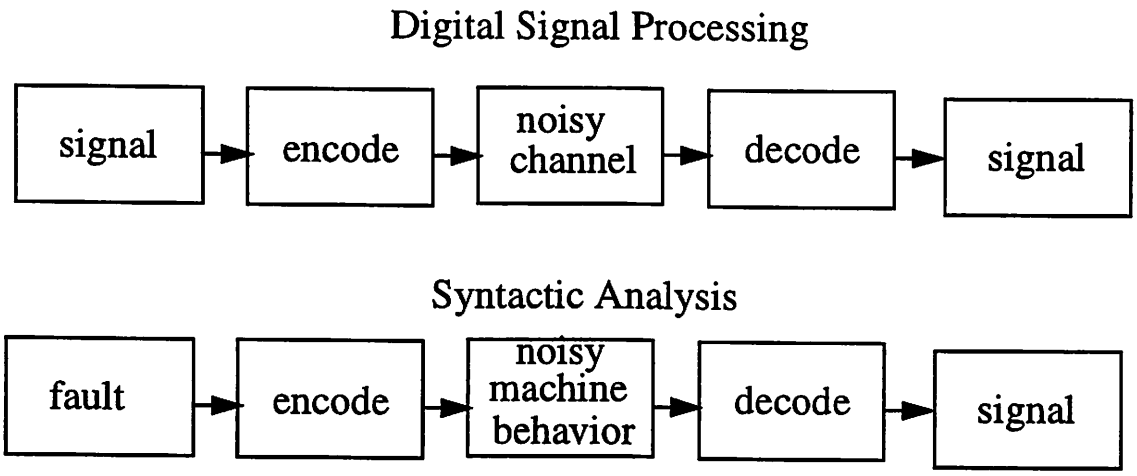


Figure 6.1. Architecture of the overall syntactic system.

6.2. Comments on General Methods

The syntactic techniques for solving the classification problems in this thesis may appear *ad hoc*. The reader might wonder if there is any general syntactic method for all the patterns, such that total automation can be achieved. As a result, one can build a general system to analyze any waveforms without the help of an engineer. In our experience, this is not likely. In the literature, uses of syntactic analysis to recognize objects tend to be pattern dependent. Many people use context-free grammar. For a different pattern, a different set of segmentation primitives must be used; a different grammar must be specified; also, different attribute information, such as segment length, time duration, and amplitude, may need to be considered (this is done usually by using attributed grammar, described briefly in 6.3). Similarly, for the plasma main etch signal pattern, line segments are used as the segmentation primitives; a regular grammar (a subset of context-free grammar) is specified, and the spike magnitude is the attribute considered.

You and Fu [22] propose a general 2-d shape recognition method, in which curve segments are used as primitives. Each curve has four attributes: direction, length, total angle change, and the degree of symmetry. Also, the angle between two adjacent curve segments is considered. While this method can describe a 2-D pattern in detail, it may complicate the task of classifying plasma ignition waveforms. Obviously, in You and Fu's

scheme, if curve segments are used as primitives, the grammar for classification will be extremely complicated. However, if we use monotonic segments (UP, FLAT, DOWN) as the primitives, with the qualitative attribute of amplitude and duration, the classifier's grammar will be very simple.

6.3. Future direction

6.3.1. Attributed grammar for maintenance scheduling

Since the real-time data waveforms of plasma etch drift constantly due to machine aging, the waveform is significantly different between the beginning and the end of a maintenance cycle. Since the real-time etch waveform reflects the actual etching behavior of the machine, it would be very helpful if we can capture the amount of drift of a plasma etch signal, such that preventive maintenance can be scheduled according to how much the shape of the waveform has changed. Attributed grammar can be used to achieve this. There are two parts to attributed grammar: the qualitative part and the quantitative part. The qualitative part focuses on the rough structural description of the waveform. Loosely speaking, it is the grand human impression on the signal, which we mainly use throughout this thesis for classification purpose. The quantitative part is the numerical measurement of the waveform attributes, for instance, the amplitude and duration of a peak, the distance between peaks, etc.

6.3.2. Automate building of a classification system

Building a classification system can be a very tedious task. It includes works of examining samples of waveforms, selecting noise tolerance and segmentation criteria, writing down description strings, etc. Also, the quality of the resulting system is highly dependent on the level of expertise of the trainer. Therefore, it is highly desirable to automate part of the building process. For instance, there are usually more than one plasma etcher in a fabrication plant. After a classification system is built for a plasma etcher, due to machine variation, it cannot be directly used on another etcher. Nevertheless, we do not have to build a new classification system from scratch. We can keep the same description strings for the waveforms, and based on these strings, select the noise tolerance and segmentation criteria. Some handwriting recognition system operates with the similar idea. If a new person wants

to use the system , s/he must provide some writing samples to the training system, so the classification system can "learn" the unique writing habit of this particular person (page 80, [23]). We can borrow the automation idea and implement a smiliar training system for plasma etching.

References

[1] Bruha, I.; Madhavan, G. P. "Use of attributed grammars for pattern recognition of evoked potentials," *IEEE Transactions on Systems, Man., and Cybernetics*. Nov./ Dec. 1988, vol. 18, (no. 6): 1046-9.

[2] Horowitz, S.L. "A syntactic algorithm for peak detection in waveforms with applications to cardiography," *Communications of the ACM*, May 1975, vol.18, (no.5):281-5.

[3] Koski, A.; Juhola, M.; Meriste, M. "Syntactic recognition of ECG signals by attributed finite automata," *Pattern Recognition*, Dec. 1995, vol.28, (no.12):1927-40.

[4] Papakonstantinou, G.; Skordalakis, E.; Gritzali, F. "An attribute grammar for QRS detection," *Pattern Recognition*, 1986, vol.19, (no.4):297-303.

[5] Udupa, J.K.; Murthy, I. S.N. "Syntactic approach to ECG rhythm analysis," *IEEE Transactions on Biomedical Engineering*, July 1980, vol.BME-27, (no.7):370-5.

[6] Horowitz, S.L. "A syntactic algorithm for peak detection in waveforms with applications to cardiography," *Communications of the ACM*, May 1975, vol.18, (no.5):281-5.

[7] Belforte, G.; de Mori, R.; Ferraris, F. "A contribution to the automatic processing of electrocardiograms using syntactic methods," *IEEE Transactions on Biomedical Engineering*, March 1979, vol.BME-26, (no.3):125-36.

[8] Trahanias, P.; Skordalakis, E. "Bottom-up approach to the ECG pattern-recognition problem," *Medical & Biological Engineering & Computing*, May 1989, vol.27, (no.3):221-9.

[9] Trahanias, P.; Skordalakis, E. "Syntactic pattern recognition of the ECG," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, July 1990, vol.12, (no.7):648-57.

[10] Skordalakis, E. "Syntactic ECG processing: a review," *Pattern Recognition*, 1986, vol.19, (no.4):305-13.

[11] Antonacopoulos, A.; Economou, A. "A structural approach for smoothing noisy peak-shaped analytical signals," *Chemometrics and Intelligent Laboratory Systems*, 6 July 1998, vol.41, (no.1):31-42.

[12] Skordalakis, E. "Recognition of noisy peaks in ECG waveforms," *Computers and Biomedical Research*, June 1984, vol.17, (no.3):208-21.

[13] Cleveland, W. S. "Robust locally weighted regression and smoothing scatterplots," *Journal of the American Statistical Association,* Dec. 1979, vol 74, (no. 368): 829-836.

[14] Lee, S.; Spanos, C. "Prediction of wafer state after plasma processing using real-time tool data," *IEEE Trans. on Semiconductor Manufacturing*, Aug 1995, Vol 8, (no 2): 252-61.

[15] Lee, S. "Semiconductor Equipment Analysis and Wafer State Prediction System Using Real-time Data," PhD. dissertation, December 1994.

[16] Huang, Herbert W. "Adaptive and Predictive Modeling for Real-Time Statistical Process Control," M.S. thesis, 1996.

[17] White, D.A.; Boning, D.; Butler, S.W.; Barna, G.G. "Spatial characterization of wafer state using principal component analysis of optical emission spectra in plasma etch," *IEEE Transactions on Semiconductor Manufacturing*, Feb. 1997, vol.10, (no.1):52-61.

[18] Brillinger, D. *Time Series: Data Analysis and Theory*, Expanded Edition, Holden-Day, San Francisco, California, 1981.

[19] Brillinger, D. "The Digital Rainbow: Some History and Applications of Numerical Spectrum Analysis," *The Canadian Journal of Statistics*, 1993, Vol. 21, (no. 1):1-19

[20] Rompelman, O et al. "Heart Rate Variability in Relation to Psychological Factors," Ergonomics, 1980, Vol. 23, (no. 12):1101-1115

[21] Fu, K. S. *Syntactic Pattern Recognition and Applications*, Prentice-Hall, Englewood Cliffs, New Jersey, 1982.

[22] You, K. C. and Fu, K. S., "A Syntactic Approach to Shape Recognition Using Attributed Grammars," *IEEE Trans. Syst. Man Cybern. SMC-9*, June 1979:334-45

[23] Nadler, M.; Smith, E. P. *Pattern Recognition Engineering*, Wiley, New York, 1993.

[24] M. Lieberman and A. Lichtenberg, *Principles of Plasma Discharges and Materials Processing*, John Wiley and Sons, New York, 1994.

[25] Chen, R. "OES-base Sensing for Plasma Processing in IC Manufacturing," PhD. dissertation, December 1997.

[26] SIA Semiconductor Industry Association, *The National Technology Roadmap for Semiconductors, 1997 Ed.*

Appendix A Splus Code for Data Encoding

```
#return the bottom index of the down hill,
#where the signal begin to rise
GetDownHill<-function(startIdx, endIdx, enclist, threshold)
{
        downhillamp <- 0
        newStartIdx <- startIdx
       newNegBegIdx <-0
        while(downhillamp < threshold) {
                for(idx in newStartIdx:endIdx) {
                        if(enclist[[idx]][1] < 0){
                           newNegBegIdx <- idx
                                break
                    }
                    if(idx==endIdx)
                           return(c(-1,-1))
                }
                downhillamp <- 0
             negIdx_newNegBegIdx
                while(enclist[[negIdx]][1] < 0) {
                        downhillamp <- downhillamp + enclist[[negIdx]][3]
                        negIdx <- negIdx + 1
                        if(negIdx > endIdx)
                                break
                }
                {
                        if(downhillamp >= threshold){
                           #print("downhillamp=")
                           #print(c(newNegBegIdx,negIdx, downhillamp))
                                return(c(newNegBegIdx,negIdx))
                    }
                        else{
                           #print("not big enough downhillamp=")
                           #print(c(newNegBegIdx,negIdx, downhillamp))
                            newStartIdx <- negIdx

                    }
                }
        }
}
#return the peak index of the up hill,
#where the signal begin to fall
GetUpHill<-function(startIdx, endIdx, enclist, threshold)
{
        uphillamp <- 0
```

```
        newStartIdx <- startIdx
       newPosBegIdx <-0
        while(uphillamp < threshold) {
                for(idx in newStartIdx:endIdx) {
                        if(enclist[[idx]][1] > 0){
                           newPosBegIdx <- idx
                                break
                    }
                    if(idx==endIdx)
                           return(c(-1,-1))
                }
                uphillamp <- 0
             posIdx_newPosBegIdx
                while(enclist[[posIdx]][1] > 0) {
                        uphillamp <- uphillamp + enclist[[posIdx]][3]
                        posIdx <- posIdx + 1
                        if(posIdx > endIdx)
                                break
                }
                {
                        if(uphillamp >= threshold){
                           #print("uphillamp=")
                           #print(c(newPosBegIdx,posIdx,uphillamp))
                           return(c(newPosBegIdx,posIdx))
                    }
                        else{
                           #print("not big enough uphillamp=")
                           #print(c(newPosBegIdx,posIdx, uphillamp))
                           newStartIdx <- posIdx

                    }
                }
        }
}


GetUpSeg<-function(startIdx, endIdx, enclist)
{
        upsegamp <- 0
        newStartIdx <- startIdx
             newPosBegIdx <-0
             for(idx in newStartIdx:endIdx) {
                     if(enclist[[idx]][1] > 0){
                          newPosBegIdx <- idx
                                break
                    }
                    if(idx==endIdx)
                           return(c(-1,-1,-1))
                }
                upsegamp <- 0
             posIdx_newPosBegIdx
                while(enclist[[posIdx]][1] > 0) {
                        upsegamp <- upsegamp + enclist[[posIdx]][3]
                        posIdx <- posIdx + 1
                        if(posIdx > endIdx)
                                break
                }
             #print("The upSegment found is")
             #print(c(newPosBegIdx,posIdx, upsegamp))
```

```
            return(c(newPosBegIdx,posIdx, upsegamp))

}

GetDownSeg<-function(startIdx, endIdx, enclist)
{
        downsegamp <- 0
        newStartIdx <- startIdx
             newNegBegIdx <-0
             for(idx in newStartIdx:endIdx) {
                     if(enclist[[idx]][1] < 0){
                          newNegBegIdx <- idx
                               break
                     }
                     if(idx==endIdx)
                          return(c(-1,-1,-1))
               }
               downsegamp <- 0
             negIdx_newNegBegIdx
               while(enclist[[negIdx]][1] < 0) {
                       downsegamp <- downsegamp + enclist[[negIdx]][3]
                       negIdx <- negIdx + 1
                       if(negIdx > endIdx)
                               break
               }
             #print("The downSegment found is")
             #print(c(newNegBegIdx,negIdx, downsegamp))
             return(c(newNegBegIdx,negIdx, downsegamp))

}


Segmentation<-function(curve, smooth, smoothFactor, linearFitErrorTolerance){
       (if(smooth){
             rawvecmatrix_lowess(1:length(curve), curve, f=smoothFactor)
             rawvec_rawvecmatrix$y
             rawvec_rawvec-mean(rawvec)
       }
       else
             rawvec_curve-mean(curve)
       }

       seglist_list()
       enclist_list()
       modlist_list()
       e_linearFitErrorTolerance
       k_1; h_1; i_1; j_1
       j_j+k
       while(j<=length(curve)){
             Gh_rawvec[i:j]
             xvar_i:j
             ##print(xvar)
             #print(Gh)
             fitline_lm(Gh~xvar)
             maxerror_max(abs(predict(fitline)-rawvec[i:j]))
             if(j>=length(curve)){ #reaching the end of the curve
                   if(maxerror<e){
                   enclist[h]_list(c(fitline$coefficients[2], j-i, max(Gh)-
min(Gh)))
```

```
                    seglist[h]_list(Gh)
                    modlist[h]_list(fitline)
                    #lines(xvar, predict(fitline))
                    h_h+1; i_j; j_j+k
                    }
             }
             if(maxerror>e){
                    j_j-k
                    Gh_rawvec[i:j]
                    xvar_i:j
                    #print(xvar)
                    #print(Gh)
                    fitline_lm(Gh~xvar)
                    enclist[h]_list(c(fitline$coefficients[2], j-i, max(Gh)-
min(Gh)))
                    seglist[h]_list(Gh)
                    modlist[h]_list(fitline)
                    #lines(xvar, predict(fitline))
                    h_h+1; i_j; j_j+k
             }
             else {
                    j_j+k; next
             }
      }
      #print(enclist)
      return(enclist)
}

SegmentationDrawLines<-function(curve, smooth, smoothFactor, linearFitErrorTol-
erance){
      (if(smooth){
             rawvecmatrix_lowess(1:length(curve), curve, f=smoothFactor)
             rawvec_rawvecmatrix$y
             rawvec_rawvec-mean(rawvec)
      }
      else
             rawvec_curve-mean(curve)
      }

      seglist_list()
      enclist_list()
      modlist_list()
      e_linearFitErrorTolerance
      k_1; h_1; i_1; j_1
      j_j+k
      while(j<=length(curve)){
             Gh_rawvec[i:j]
             xvar_i:j
             ##print(xvar)
             #print(Gh)
             fitline_lm(Gh~xvar)
             maxerror_max(abs(predict(fitline)-rawvec[i:j]))
             if(j>=length(curve)){ #reaching the end of the curve
                    if(maxerror<e){
                    enclist[h]_list(c(fitline$coefficients[2], j-i, max(Gh)-
min(Gh)))
                    seglist[h]_list(Gh)
                    modlist[h]_list(fitline)
                    lines(xvar, predict(fitline))
```

```
                    h_h+1; i_j; j_j+k
                    )
             }
             if(maxerror>e){
                    j_j-k
                    Gh_rawvec[i:j]
                    xvar_i:j
                    #print(xvar)
                    #print(Gh)
                    fitline_lm(Gh~xvar)
                    enclist[h]_list(c(fitline$coefficients[2], j-i, max(Gh)-
min(Gh)))
                    seglist[h]_list(Gh)
                    modlist[h]_list(fitline)
                    lines(xvar, predict(fitline))
                    h_h+1; i_j; j_j+k
             }
             else {
                    j_j+k; next
             }
      }
      #print(enclist)
      return(enclist)
}

encodingFunct<-function(enclist, fastlim, slowlim){
      encoding_list()
      for(slopei in 1:(length(enclist))){
             slopeval_enclist[[slopei]][[1]]
             if (slopeval>fastlim) {
                    encoding[slopei]_2}
             else if (slopeval <= fastlim && slopeval>slowlim) {
                    encoding[slopei]_1}
             else if (slopeval<=slowlim && slopeval>= -slowlim) {
                    encoding[slopei]_0}
             else if (slopeval< -slowlim && slopeval>= -fastlim) {
                    encoding[slopei]_-1}
             else if(slopeval< -fastlim) {
                    encoding[slopei]_-2}
      }
      return(encoding)
}

#just look at the end of the third downhill.
cutwindow<-function(rawvec, postcutdelay, winlen){
      enclist_Segmentation(rawvec, F, 0, 10)
      newIdx_1
      for(idx in 1:3){
             downHillSeg_GetDownHill(newIdx, length(enclist), enclist, 100)
             newIdx_downHillSeg[2]
      }

      newWaveIdx_0
      for(subidx in 1:newIdx)
             newWaveIdx_newWaveIdx+enclist[[subidx]][2]
      #print("newWaveIdx=")
      #print(newWaveIdx)
      s_newWaveIdx+postcutdelay
      return(rawvec[s:(s+winlen)])
```

```
}

GetPosSpike<-function(startIdx, endIdx, enclist, threshold)
{
        findSpike_F
        newStartIdx_startIdx
        while(!findSpike){
                upHillSeg_GetUpHill(newStartIdx, endIdx, enclist, threshold)
                downSeg_GetDownSeg(upHillSeg[2], endIdx, enclist)
                {if(downSeg[3]>threshold) findSpike_T
                else newStartIdx_downSeg[2]}
        }
        return(c(upHillSeg[1],upHillSeg[2], downSeg[1], downSeg[2]))
}

GetNegSpike<-function(startIdx, endIdx, enclist, threshold)
{
        findSpike_F
        newStartIdx_startIdx
        while(!findSpike){
                downHillSeg_GetDownHill(newStartIdx, endIdx, enclist, threshold)
                upSeg_GetUpSeg(downHillSeg[2], endIdx, enclist);
                {if(upSeg[3]>threshold) findSpike_T
                else newStartIdx_upSeg[2]}

        }
        return(c(downHillSeg[1], downHillSeg[2], upSeg[1], upSeg[2]))
}

#scan thru, get one pos spike, then one neg spike, and then
#one pos spike
cutwindownew<-function(rawvec, postcutdelay, winlen){
        enclist_Segmentation(rawvec, F, 0, 10)

        newIdx_1
        firstPosSpike_GetPosSpike(newIdx, length(enclist), enclist, 400)
        #print("firstPosSpike=")
        #print(firstPosSpike)
        fallingPoint_firstPosSpike[3]
        firstNegSpike_GetNegSpike(fallingPoint, length(enclist), enclist, 400)
        #print("firstNegSpike=")
        #print(firstNegSpike)
        risingPoint_firstNegSpike[3]

        secondPosSpike_GetPosSpike(risingPoint, length(enclist), enclist, 100)
        #print("secondPosSpike=")
        #print(secondPosSpike)

        #summing up the duration
        newWaveIdx_0;
        for(subidx in 1:secondPosSpike[4])
                newWaveIdx_newWaveIdx+enclist[[subidx]][2]
        #print("newWaveIdx=")
        #print(newWaveIdx)
        s_newWaveIdx+postcutdelay
        return(rawvec[s:(s+winlen)])
}

#try to filter out two spikes
```

```
cutwindownew2<-function(rawvec, postcutdelay, winlen){
	firstnumpoints_100
	#enclist_Segmentation(rawvec, F, 0, 10)
	enclist_Segmentation(rawvec[1:firstnumpoints], F, 0, 10)
	#print(enclist)
	newIdx_1
	firstPosSpike_GetPosSpike(newIdx, length(enclist), enclist, 400)
	#print("firstPosSpike=")
	#print(firstPosSpike)

	risingPoint_firstPosSpike[4]
	secondPosSpike_GetPosSpike(risingPoint, length(enclist), enclist, 150)
	#print("secondPosSpike=")
	#print(secondPosSpike)

	#summing up the duration
	newWaveIdx_0;
	for(subidx in 1:secondPosSpike[4])
		newWaveIdx_newWaveIdx+enclist[[subidx]][2]
	#print("newWaveIdx=")
	#print(newWaveIdx)
	s_newWaveIdx+postcutdelay
	print("The new window start at:")
	print(s)
	return(rawvec[s:(s+winlen)])
}
newwin_vector()
testm_matrix( scan("file.dat"  , skip=1), ncol=26, byrow=TRUE)
colidx_5 #for selecting capacitance manometer
orgrawvec_testm[,colidx]
orgrawvec_orgrawvec[15:min(200,length(orgrawvec))]
newwin_cutwindownew2(orgrawvec, 5, 45);
}
goodrefvecmatrix_lowess(1:length(newwin), newwin, f=.2)
goodrefvec_goodrefvecmatrix$y
enclist_SegmentationDrawLines(somevec, F, 0.0, 1)
encoding_encodingFunct(enclist, 5, 2)
print(encoding)
```

Appendix B Splus Code of Segmentation for the High Speed Data

```
runt<-function(){
for(i in 6:29){
if(i <10)
   filename_paste("wafer00", i,".dat", sep="")
else
   filename_paste("wafer0", i, ".dat", sep="")

cat(filename, file="runlog", fill=T, append=T);
testm_matrix(scan(filename), ncol=13, byrow=TRUE)
colidx_6 #for selecting TCP line impedance
orgrawvec_testm[1:1200,colidx]
motif()
plot(orgrawvec, type="n")
enclist_SegmentationSpeed(orgrawvec, F, 0.0, 0.05)
alist_UpFlatDownSegmentation(enclist, slopethresh=0.01)
cat("UpFlatDownSegmentation:", length(alist), file="runlog", fill=T, append=T);
alist_ReArrangeList2(alist, 5, 0.4, 0.01 )
cat("ReArrangeList2:", length(alist), file="runlog", fill=T, append=T);
#alist_UpFlatDownSegmentation2(enclist, 5, 0.4, slopethresh=0.01)

#Filter small pos peak three times
alist_FilterSmallPosPeak(alist, 5, 0.4, 0.01)
cat("FilterSmallPosPeak:", length(alist), file="runlog", fill=T, append=T);
alist_FilterSmallPosPeak(alist, 5, 0.4, 0.01)
cat("FilterSmallPosPeak:", length(alist), file="runlog", fill=T, append=T);
alist_FilterSmallPosPeak(alist, 5, 0.4, 0.01)
cat("FilterSmallPosPeak:", length(alist), file="runlog", fill=T, append=T);
alist_ReArrangeList(alist)
alist_ReArrangeList2(alist, 5, 0.4, 0.01 )
cat("ReArrangeList2:", length(alist), file="runlog", fill=T, append=T);

alist_FilterSmallLeftBigRight(alist, 5, 0.4, 0.01)
alist_ReArrangeList(alist)
alist_ReArrangeList2(alist, 5, 0.4, 0.01 )
cat("FilterSmallLeftBigRight:", length(alist), file="runlog", fill=T, append=T);

alist_FilterSmallRightBigLeft(alist, 5, 0.4, 0.01)
alist_ReArrangeList(alist)
alist_ReArrangeList2(alist, 5, 0.4, 0.01 )
cat("FilterSmallRightBigLeft:", length(alist), file="runlog", fill=T, append=T);
```

```
alist_ConcatFlat(alist, 5, 0.4, 0.01)
alist_ReArrangeList(alist)
alist_ReArrangeList2(alist, 5, 0.4, 0.01 )
cat("ConcatFlat:", length(alist), file="runlog", fill=T, append=T);

alist_ConcatDown(alist, 5, 0.4, 0.01)
alist_ReArrangeList(alist)
alist_ReArrangeList2(alist, 5, 0.4, 0.01 )
cat(":ConcatDown", length(alist), file="runlog", fill=T, append=T);

alist_ConcatUp(alist, 5, 0.4, 0.01)
alist_ReArrangeList(alist)
alist_ReArrangeList2(alist, 5, 0.4, 0.01 )
cat("ConcatUp:", length(alist), file="runlog", fill=T, append=T);



#PrintAllDurationAmp(alist)
plot(orgrawvec, type="n")
plotUpFlatDownSegments(alist)
}
}


#notice for UpFlatDownSegments[[i]][[j]][[4]], i is the big seg idx, j is the small
seg idx,
#4 is the
#slope attribute (1, 0, or -1) based on slopethresh,
#The small flat segments would be mark as part of the up and down hill
UpFlatDownSegmentation2<-function(enclist, smalldur, smallamp, slopethresh){
UpFlatDownSegments_list()
UpFlatDownIdx_0
totalseg_length(enclist)
SegmentCount_1
auxseg_enclist[[SegmentCount]]
slopeval_auxseg[[3]]$coefficients[2]
duration_auxseg[[2]]-auxseg[[1]]
ampl_max(predict(auxseg[[3]]))-min(predict(auxseg[[3]]))
while(SegmentCount<=totalseg){
   cursegidx_0
   UpFlatDownIdx_UpFlatDownIdx+1
   if(slopeval>slopethresh){
      while(slopeval>slopethresh ||
        (abs(slopeval)<=slopethresh &&
        duration<smalldur && ampl<smallamp) ){
        cursegidx_cursegidx+1
        UpFlatDownSegments[[UpFlatDownIdx]][cursegidx]_list(c(auxseg,1))
        SegmentCount_SegmentCount+1
        if(SegmentCount<=totalseg){
          auxseg_enclist[[SegmentCount]]
          slopeval_auxseg[[3]]$coefficients[2]
          duration_auxseg[[2]]-auxseg[[1]]
          ampl_max(predict(auxseg[[3]]))-min(predict(auxseg[[3]]))
        }
        else break
      }
   }
   else if(abs(slopeval)<=slopethresh){
      while(abs(slopeval)<=slopethresh){
```

```
      cursegidx_cursegidx+1
      UpFlatDownSegments[[UpFlatDownIdx]][cursegidx]_list(c(auxseg,0))
      SegmentCount_SegmentCount+1
      if(SegmentCount<=totalseg){
       auxseg_enclist[[SegmentCount]]
       slopeval_auxseg[[3]]$coefficients[2]
      }
      else break
     }
   }
   else if(slopeval < (-slopethresh)){
     while(slopeval < (-slopethresh)||
      (abs(slopeval)<=slopethresh &&
      duration<smalldur && ampl<smallamp) ){
      cursegidx_cursegidx+1
      UpFlatDownSegments[[UpFlatDownIdx]][cursegidx]_list(c(auxseg,-1))
      SegmentCount_SegmentCount+1
      if(SegmentCount<=totalseg){
       auxseg_enclist[[SegmentCount]]
       slopeval_auxseg[[3]]$coefficients[2]
       duration_auxseg[[2]]-auxseg[[1]]
       ampl_max(predict(auxseg[[3]]))-min(predict(auxseg[[3]]))
      }
      else break
     }
   }

}
return(UpFlatDownSegments)
}

ReArrangeList2<-function(updownlist, smalldur, smallamp, slopethresh){
newlist_list()
newidx_0
i_1
while(i <= length(updownlist)){
   insertlist_list()
   curelm_updownlist[[i]]
   curflag_curelm[[1]][[4]]
   if(curflag==1){
     while(curflag==1 || SmallFlat(curelm, smalldur, smallamp)){
      if(curflag==1)
       insertlist_append(insertlist, curelm)
      else{
       for(curi in 1:length(curelm))
          curelm[[curi]][4]_1
       insertlist_append(insertlist, curelm)

      }
      i_i+1
      if(i <= length(updownlist)){
       curelm_updownlist[[i]]
       curflag_curelm[[1]][[4]]
      }
      else break
     }

   }
   else if(curflag==0){
```

```
      while(curflag==0){
        insertlist_append(insertlist, curelm)
        i_i+1
        if(i <= length(updownlist)){
          curelm_updownlist[[i]]
          curflag_curelm[[1]][[4]]
        }
        else break
      }
   }
   else if(curflag==-1){
      while(curflag==-1 || SmallFlat(curelm, smalldur, smallamp)){
        if(curflag==-1)
          insertlist_append(insertlist,curelm)
        else{
          for(curi in 1:length(curelm))
             curelm[[curi]][4]_-1
          insertlist_append(insertlist, curelm)

        }
        i_i+1
        if(i <= length(updownlist)){
          curelm_updownlist[[i]]
          curflag_curelm[[1]][[4]]
        }
        else break
      }

   }
   newlist_append(newlist, list(insertlist))
}

return(newlist)
}


SmallArm<-function(monotoniclist, smalldur, smallamp){
dur_DurationSegments(monotoniclist)
amp_AmpSegments(monotoniclist)
slopeflag_monotoniclist[[1]][[4]]
smallarm_ifelse(dur<smalldur && amp<smallamp && slopeflag!=0, T, F)
return(smallarm)
}
BigArm<-function(monotoniclist, smalldur, smallamp){
dur_DurationSegments(monotoniclist)
amp_AmpSegments(monotoniclist)
slopeflag_monotoniclist[[1]][[4]]
bigarm_ifelse(!(dur<smalldur && amp<smallamp) && slopeflag!=0, T, F)
return(bigarm)
}

SmallSeg<-function(monotoniclist, smalldur, smallamp){
dur_DurationSegments(monotoniclist)
amp_AmpSegments(monotoniclist)
slopeflag_monotoniclist[[1]][[4]]
smallseg_ifelse(dur<smalldur && amp<smallamp, T, F)
return(smallseg)
}
```

```
BigSeg<-function(monotoniclist, smalldur, smallamp){
dur_DurationSegments(monotoniclist)
amp_AmpSegments(monotoniclist)
slopeflag_monotoniclist[[1]][[4]]
bigseg_ifelse(!(dur<smalldur && amp<smallamp), T, F)
return(bigseg)
}


#Two big flat, one small in the middle, concat them together
ConcatFlat<-function(updownlist, smalldur, smallamp, slopethresh){
newlist_list()
newidx_0
i_1
while((i+2) <= length(updownlist)){
   if(i==1 && SmallArm(updownlist[[i]], smalldur, smallamp) &&
      BigFlat(updownlist[[i+1]], smalldur, smallamp)){
      newarm_list()
      lp_updownlist[[i]]
      for(ai in 1:length(lp))
       newarm_append(newarm, list(c(lp[[ai]][[1]], lp[[ai]][[2]],
         lp[[ai]][3], 0)) )
      #print("The small idx is:"); print(i);
      newlist_append(newlist, list(newarm))
      newlist_append(newlist, updownlist[i+1])
      i_i+1; next
   }

   if (BigFlat(updownlist[[i]], smalldur, smallamp) &&
      SmallArm(updownlist[[i+1]], smalldur, smallamp)
      &&BigFlat(updownlist[[i+2]], smalldur, smallamp)){
      newarm_list()
      lp_updownlist[[i+1]]
      for(ai in 1:length(lp))
       newarm_append(newarm, list(c(lp[[ai]][[1]], lp[[ai]][[2]],
         lp[[ai]][3], 0)))
      #print("The small idx is:"); print(i+1);
      newlist_append(newlist, updownlist[i])
      newlist_append(newlist, list(newarm))
      i_i+2
   }
   else{
      newlist_append(newlist, updownlist[i])
      i_i+1
   }
}

if((i+1)==length(updownlist) && BigFlat(updownlist[[i]], smalldur, smallamp)
   && SmallArm(updownlist[[i+1]], smalldur, smallamp)){
      newarm_list()
      lp_updownlist[[i+1]]
      for(ai in 1:length(lp))
       newarm_append(newarm, list(c(lp[[ai]][[1]], lp[[ai]][[2]],
         lp[[ai]][3], 0)))
      #print("The small idx is:"); print(i+1);
      newlist_append(newlist, updownlist[i])
      newlist_append(newlist, list(newarm))
      i_i+2
}
```

```
else if(i <= length(updownlist))
   for(j in i:length(updownlist))
      newlist_append(newlist, updownlist[j])
return(newlist)
}

ConcatDown<-function(updownlist, smalldur, smallamp, slopethresh){
newlist_list()
newidx_0
i_1
while((i+2) <= length(updownlist)){
   if(i==1 && SmallSeg(updownlist[[i]], smalldur, smallamp) &&
      BigRightArm(updownlist[[i+1]], smalldur, smallamp)){
      newarm_list()
      lp_updownlist[[i]]
      for(ai in 1:length(lp))
       newarm_append(newarm, list(c(lp[[ai]][[1]], lp[[ai]][[2]],
         lp[[ai]][3], -1)) )
      #print("The small idx is:"); print(i);
      newlist_append(newlist, list(newarm))
      newlist_append(newlist, updownlist[i+1])
      i_i+1; next
   }

   if (BigRightArm(updownlist[[i]], smalldur, smallamp) &&
      SmallSeg(updownlist[[i+1]], smalldur, smallamp)
      &&BigRightArm(updownlist[[i+2]], smalldur, smallamp)){
      newarm_list()
      lp_updownlist[[i+1]]
      for(ai in 1:length(lp))
       newarm_append(newarm, list(c(lp[[ai]][[1]], lp[[ai]][[2]],
         lp[[ai]][3], -1)))
      #print("The small idx is:"); print(i+1);
      newlist_append(newlist, updownlist[i])
      newlist_append(newlist, list(newarm))
      i_i+2
   }
   else{
      newlist_append(newlist, updownlist[i])
      i_i+1
   }
}

if((i+1)==length(updownlist) && BigRightArm(updownlist[[i]], smalldur, smallamp)
   && SmallSeg(updownlist[[i+1]], smalldur, smallamp)){
      newarm_list()
      lp_updownlist[[i+1]]
      for(ai in 1:length(lp))
       newarm_append(newarm, list(c(lp[[ai]][[1]], lp[[ai]][[2]],
         lp[[ai]][3], -1)))
      #print("The small idx is:"); print(i+1);
      newlist_append(newlist, updownlist[i])
      newlist_append(newlist, list(newarm))
      i_i+2
}
else if(i <= length(updownlist))
   for(j in i:length(updownlist))
      newlist_append(newlist, updownlist[j])
return(newlist)
```

```
}

ConcatUp<-function(updownlist, smalldur, smallamp, slopethresh){
newlist_list()
newidx_0
i_1
while((i+2) <= length(updownlist)){
   if(i==1 && SmallSeg(updownlist[[i]], smalldur, smallamp) &&
      BigLeftArm(updownlist[[i+1]], smalldur, smallamp)){
      newarm_list()
      lp_updownlist[[i]]
      for(ai in 1:length(lp))
       newarm_append(newarm, list(c(lp[[ai]][[1]], lp[[ai]][[2]],
         lp[[ai]][3], 1)) )
      #print("The small idx is:"); print(i);
      newlist_append(newlist, list(newarm))
      newlist_append(newlist, updownlist[i+1])
      i_i+1; next
   }

   if (BigLeftArm(updownlist[[i]], smalldur, smallamp) &&
      SmallSeg(updownlist[[i+1]], smalldur, smallamp)
      &&BigLeftArm(updownlist[[i+2]], smalldur, smallamp)){
      newarm_list()
      lp_updownlist[[i+1]]
      for(ai in 1:length(lp))
       newarm_append(newarm, list(c(lp[[ai]][[1]], lp[[ai]][[2]],
         lp[[ai]][3], 1)))
      #print("The small idx is:"); print(i+1);
      newlist_append(newlist, updownlist[i])
      newlist_append(newlist, list(newarm))
      i_i+2
   }
   else{
      newlist_append(newlist, updownlist[i])
      i_i+1
   }
}

if((i+1)==length(updownlist) && BigLeftArm(updownlist[[i]], smalldur, smallamp)
   && SmallSeg(updownlist[[i+1]], smalldur, smallamp)){
      newarm_list()
      lp_updownlist[[i+1]]
      for(ai in 1:length(lp))
       newarm_append(newarm, list(c(lp[[ai]][[1]], lp[[ai]][[2]],
         lp[[ai]][3], 1)))
      #print("The small idx is:"); print(i+1);
      newlist_append(newlist, updownlist[i])
      newlist_append(newlist, list(newarm))
      i_i+2
}
else if(i <= length(updownlist))
   for(j in i:length(updownlist))
      newlist_append(newlist, updownlist[j])
return(newlist)
}
```

```
#this is a wrong function, since the flat segment's duration is
#not considered. don't use.
FilterSmallArm<-function(updownlist, smalldur, smallamp, slopethresh){
newlist_list()
newidx_0
i_1
while((i+2) <= length(updownlist)){
   if (updownlist[[i]][[1]][[4]]==0 &&
      SmallArm(updownlist[[i+1]], smalldur, smallamp)
      &&updownlist[[i+2]][[1]][[4]]==0 &&
      DurationSegments(updownlist[[i+1]])<DurationSegments(updownlist[[i]])&&
      DurationSegments(updownlist[[i+1]])<DurationSegments(updownlist[[i+2]])){
      flatleft_updownlist[[i]]
      segcount_length(flatleft)
      leftseg_flatleft[[segcount]]
      leftbasetime_leftseg[[2]]
      leftbaseval_predict(leftseg[[3]])[length(predict(leftseg[[3]]))]

      flatright_updownlist[[i+2]]
      rightseg_flatright[[1]]
      rightbasetime_rightseg[[1]]
      rightbaseval_predict(rightseg[[3]])[1]
      meanval_(leftbaseval+rightbaseval)/2
      yvar_c(meanval, meanval)
      xvar_c(leftbasetime, rightbasetime)
      filtermod_lm(yvar~xvar)
      slopeflag_slopeflagfunc((filtermod$coefficients)[2], slopethresh)
      insertelm_list(c(leftbasetime, rightbasetime, list(filtermod), slopeflag))
      #print("filter segment"); print(i+1);
      #print("previous flag"); print(updownlist[[i+1]][[1]][[4]]);
      #print("new flag"); print(slopeflag);

      newlist_append(newlist, updownlist[i])
      newlist_append(newlist, list(insertelm))
      i_i+2
   }
   else{
      newlist_append(newlist, updownlist[i])
      i_i+1
   }
}
if(i <= length(updownlist))
   for(j in i:length(updownlist))
   newlist_append(newlist, updownlist[j])
return(newlist)
}


#The following four functions try to exam the monotonic curve's status, for neg
peaks.
NSmallLeftArm<-function(monotoniclist, smalldur, smallamp){
dur_DurationSegments(monotoniclist)
amp_AmpSegments(monotoniclist)
```

```
slopeflag_monotoniclist[[1]][[4]]
smallleft_ifelse(dur<smalldur && amp<smallamp && slopeflag==-1, T, F)
return(smallleft)
}
NSmallRightArm<-function(monotoniclist, smalldur, smallamp){
dur_DurationSegments(monotoniclist)
amp_AmpSegments(monotoniclist)
slopeflag_monotoniclist[[1]][[4]]
smallright_ifelse(dur<smalldur && amp<smallamp && slopeflag==1, T, F)
return(smallright)
}

NBigLeftArm<-function(monotoniclist, smalldur, smallamp){
dur_DurationSegments(monotoniclist)
amp_AmpSegments(monotoniclist)
slopeflag_monotoniclist[[1]][[4]]
bigleft_ifelse( !(dur<smalldur && amp<smallamp) && slopeflag==-1, T, F)
return(bigleft)
}

NBigRightArm<-function(monotoniclist, smalldur, smallamp){
dur_DurationSegments(monotoniclist)
amp_AmpSegments(monotoniclist)
slopeflag_monotoniclist[[1]][[4]]
bigright_ifelse( !(dur<smalldur && amp<smallamp) && slopeflag==1, T, F)
return(bigright)
}



FilterSmallNegPeak<-function(updownlist, smalldur, smallamp, slopethresh){
newlist_list()
newidx_0
i_1
while(i < length(updownlist)){
   #cat("i is", i, fill=T);
   #newidx_newidx+1
   if (NSmallLeftArm(updownlist[[i]], smalldur, smallamp)){
      if (NSmallRightArm(updownlist[[i+1]], smalldur, smallamp)){
       leftmostseg_updownlist[[i]][[1]]
       rightmostseg_updownlist[[i+1]][[length(updownlist[[i+1]])]]
       lefttime_leftmostseg[[1]]
       righttime_rightmostseg[[2]]
       leftbaseval_predict(leftmostseg[[3]])[1]
       rightsegval_predict(rightmostseg[[3]])
       rightbaseval_rightsegval[length(rightsegval)]
       yvar_c(leftbaseval, rightbaseval)
       xvar_c(lefttime, righttime)
       filtermod_lm(yvar~xvar)
       slopeflag_slopeflagfunc((filtermod$coefficients)[2], slopethresh)
       insertelm_list(c(lefttime, righttime, list(filtermod), slopeflag))
       newlist_append(newlist, list(insertelm))
       i_i+2
       }
      else{
       newlist_append(newlist, updownlist[i])
       i_i+1
      }
```

```
   }
   else{
      newlist_append(newlist, updownlist[i])
      i_i+1
   }
}
if(i==length(updownlist))
   newlist_append(newlist, updownlist[i])
return(newlist)
}

FilterNSmallLeftBigRight<-function(updownlist, smalldur, smallamp, slopethresh){
newlist_list()
newidx_0
i_1
while(i < length(updownlist)){
   if (NSmallLeftArm(updownlist[[i]], smalldur, smallamp) &&
      NBigRightArm(updownlist[[i+1]], smalldur, smallamp)){
#     print("The old two arms are")
#     print(updownlist[[i]])
#     print(updownlist[[i+1]])

      #compute the filtering, use leftarm base, draw a horizontal
      #line, intersect rightarm. line(leftarm base, intersect) got
      #inserted into the new list.  Also, anything before
      #RightArm intersected segment got discarded. Intersected segment's
      #beginning index is replaced by the intersected index.
      leftbasetime_updownlist[[i]][[1]][[1]]
      leftbaseval_predict(updownlist[[i]][[1]][[3]])[1]
      newrightarm_list()
      rightArmInter_IntersectMonSeg(updownlist[[i+1]], leftbaseval)
      segidx_rightArmInter[1]
      interTime_rightArmInter[2]
      if(segidx==0){
       newlist_append(newlist, updownlist[i])
       newlist_append(newlist, updownlist[i+1])
       i_i+2
      }
      else{
       interLineSeg_updownlist[[i+1]][[segidx]]
       if(interTime==interLineSeg[[2]]){
        if(segidx==length(updownlist[[i+1]])){
        yvar_c(leftbaseval, leftbaseval)
        xvar_c(leftbasetime, interTime)
        filtermod_lm(yvar~xvar)
        slopeflag_slopeflagfunc((filtermod$coefficients)[2], slopethresh)
        insertelm_list(c(leftbasetime, interTime, list(filtermod), slopeflag))
        newlist_append(newlist, list(insertelm))
#       print("The new single left arms are")
#       print(newlist[[length(newlist)]])
        i_i+2
        }
        else{
        for(ridx in (segidx+1):length(updownlist[[i+1]]))
           newrightarm_append(newrightarm, updownlist[[i+1]][ridx])
#       print("The new right arm"); print(newrightarm)
        yvar_c(leftbaseval, leftbaseval)
        xvar_c(leftbasetime, interTime)
        filtermod_lm(yvar~xvar)
```

```
        slopeflag_slopeflagfunc((filtermod$coefficients)[2], slopethresh)
        insertelm_list(c(leftbasetime, interTime, list(filtermod), slopeflag))
        newlist_append(newlist, list(insertelm))
        newlist_append(newlist, list(newrightarm))
#       print("The new two arms are")
#       print(newlist[[length(newlist)-1]])
#       print(newlist[[length(newlist)]])
        i_i+2
        )
      }
      else{
        #non-special case
#       print("Old inter seg");print(interLineSeg)
        tempSeg_list(c(interTime, interLineSeg[[2]], interLineSeg[3],
             interLineSeg[[4]]))
        interLineSeg_tempSeg
#       print("New inter seg");print(interLineSeg)
        newrightarm_append(newrightarm, interLineSeg)
        if(length(updownlist[[i+1]])>segidx)
          for(ridx in (segidx+1):length(updownlist[[i+1]]))
            newrightarm_append(newrightarm, updownlist[[i+1]][ridx])
#       print("The new right arm"); print(newrightarm)
        yvar_c(leftbaseval, leftbaseval)
        xvar_c(leftbasetime, interTime)
        filtermod_lm(yvar~xvar)
        slopeflag_slopeflagfunc((filtermod$coefficients)[2], slopethresh)
        insertelm_list(c(leftbasetime, interTime, list(filtermod), slopeflag))
        newlist_append(newlist, list(insertelm))
        newlist_append(newlist, list(newrightarm))
#       print("The new two arms are")
#       print(newlist[[length(newlist)-1]])
#       print(newlist[[length(newlist)]])
        i_i+2
      }
    }

  }
  else{
    newlist_append(newlist, updownlist[i])
    i_i+1
  }
}
if(i == length(updownlist))
  newlist_append(newlist, updownlist[i])

return(newlist)
}

FilterNSmallRightBigLeft<-function(updownlist, smalldur, smallamp, slopethresh){
newlist_list()
newidx_0
i_1
while(i < length(updownlist)){
  if (NBigLeftArm(updownlist[[i]], smalldur, smallamp) &&
    NSmallRightArm(updownlist[[i+1]], smalldur, smallamp)){
    #print("The old two arms are")
    #print(updownlist[[i]])
    #print(updownlist[[i+1]])
```

```
#compute the filtering, use rightarm base, draw a horizontal
#line, intersect leftarm. line(intersect, rightarm base) got
#inserted into the new list.  Also, anything after
#LeftArm intersected segment got discarded. Intersected segment's
#ending index is replaced by the intersected index.
smallright_updownlist[[i+1]]
segcount_length(smallright)
rightbasetime_smallright[[segcount]][[2]]
predrightval_predict(smallright[[segcount]][[3]])
rightbaseval_predrightval[length(predrightval)]
newleftarm_list()
leftArmInter_IntersectMonSeg(updownlist[[i]], rightbaseval)
segidx_leftArmInter[1]
interTime_leftArmInter[2]
if(segidx==0){
 newlist_append(newlist, updownlist[i])
 newlist_append(newlist, updownlist[i+1])
 i_i+2
}
else{
 interLineSeg_updownlist[[i]][[segidx]]
 if(interTime==interLineSeg[[1]]){
  if(segidx==1){
  yvar_c(rightbaseval, rightbaseval)
  xvar_c(interTime, rightbasetime)
  filtermod_lm(yvar~xvar)
  slopeflag_slopeflagfunc((filtermod$coefficients)[2], slopethresh)
  insertelm_list(c(interTime, rightbasetime, list(filtermod), slopeflag))
  newlist_append(newlist, list(insertelm))
  #print("The new single left arms are")
  #print(newlist[[length(newlist)]])
  i_i+2
  }
  else{
  for(lidx in 1:(segidx-1))
     newleftarm_append(newleftarm, updownlist[[i]][lidx])
  #print("The new left arm"); #print(newleftarm)
  yvar_c(rightbaseval, rightbaseval)
  xvar_c(interTime, rightbasetime)
  filtermod_lm(yvar~xvar)
  slopeflag_slopeflagfunc((filtermod$coefficients)[2], slopethresh)
  insertelm_list(c(interTime, rightbasetime, list(filtermod), slopeflag))
  newlist_append(newlist, list(newleftarm))
  newlist_append(newlist, list(insertelm))
  #print("The new two arms are")
  #print(newlist[[length(newlist)-1]])
  #print(newlist[[length(newlist)]])
  i_i+2
  }
 }
 else{
  #non-special case

  if(segidx>1)
     for(lidx in 1:(segidx-1))
        newleftarm_append(newleftarm, updownlist[[i]][lidx])


  #print("Old inter seg");print(interLineSeg)
```

```
        tempSeg_list(c(interLineSeg[[1]], interTime, interLineSeg[3],
             interLineSeg[[4]]))
        interLineSeg_tempSeg
        #print("New inter seg");print(interLineSeg)
        newleftarm_append(newleftarm, interLineSeg)

        #print("The new left arm"); print(newleftarm)
        yvar_c(rightbaseval, rightbaseval)
        xvar_c(interTime, rightbasetime)
        filtermod_lm(yvar~xvar)
        slopeflag_slopeflagfunc((filtermod$coefficients)[2], slopethresh)
        insertelm_list(c(interTime, rightbasetime, list(filtermod), slopeflag))
        newlist_append(newlist, list(newleftarm))
        newlist_append(newlist, list(insertelm))
        #print("The new two arms are")
        #print(newlist[[length(newlist)-1]])
        #print(newlist[[length(newlist)]])
        i_i+2
      }
    }
  }
  else{
    newlist_append(newlist, updownlist[i])
    i_i+1
  }
}
if(i == length(updownlist))
  newlist_append(newlist, updownlist[i])
return(newlist)
}
```

Appendix C The Classifer for the High Speed Data

```
/*encoding.c*/
#include <stdio.h>
#include <string.h>
int GetDurCode(int dur){
  if (dur <10)
    return 0;
  else if(dur >= 10 && dur <=30)
    return 1;
  else if(dur > 30)
    return 2;
  printf("Error, wrong duration sign %d\n", dur);
  exit(1);
}

int GetAmpCode(float amp){
  if (amp <0.4)
    return 0;
  else if(amp >= 0.4 && amp <=2)
    return 1;
  else if(amp > 2)
    return 2;
  printf("Error, wrong amp sign %d\n", amp);
  exit(1);
}

main(){
  int idx, dur, slopecode, durcode, ampcode;
  float amp;
  FILE *inf, *outf;
  inf=fopen("forhist.dat.txt", "r");
  outf=fopen("code.list", "w");
  while(fscanf(inf, "%d%d%f%d",&idx, &dur, &amp, &slopecode)!=EOF){
    if(idx==1) fprintf(outf, "\n");
    fprintf(outf, "(%d,%d,%d)",slopecode,
            GetDurCode(dur),GetAmpCode(amp));
  }
  fprintf(outf, "\n");
}


/*classfier.l*/
/* scanner for a toy Pascal-like language */

%{
/* need this for the call to atof() below */
```

```
#include <math.h>
        char *fileName;
%)
SYM     "("|")"|","|0|1|2|"-1"
DIGIT   [0-9]
ID      [a-z][a-z0-9]*
ONEORTWO 1|2
AI "("0,2,0")("1,0,2")("-1,0,2")("1,0,1")("-1,1,2")("0,1,0")"
AII  {SYM}
AIII "("1,0,(1|2)")("-1,(0|1),(1|2)")("0,(1|2),0")"
AIV  "("1,(0|1),(0|1)")("-1,(0|1),(0|1)")("0,2,0")"
BI "("0,2,0")("-1,0,1")("1,2,2")("0,1,0")("-1,1,2")("1,1,2")("-1,1,2")"
BII "("1,0,1")("-1,0,1")("1,1,1")("-1,(0|1),(0|1)")"
BIII1 "("0,0,0")("1,(0|1),0")("0,2,(0|1)")"
BIII2 "("0,1,0")("-1,0,0")("0,2,0")"
BIII {BIII1}|{BIII2}

Cop_pos_spike "("1,0,1")("-1,1,1")"
Cop_flat "("0,(0|1),0")"
Cop_down "("-1,(0|1),(0|1)")"
Ccommon "("-1,2,2")"{Cop_flat}?"("1,2,2")("0,1,0")("-
1,1,2")"{Cop_pos_spike}?"("1,1,2")("-1,(0|1),2")"
D "("0,2,0")("-1,0,2")("1,1,0")("0,2,(0|1)")"
Fcommon "("1,1,2")("-1,0,2")("1,1,2")("-1,0,1")("1,0,2")("-1,1,2")"
Hcommon "("0,2,0")("-1,0,2")"{Cop_flat}?"("1,2,2")("-
1,0,1")("1,(0|1),(0|1)")"{Cop_flat}?"("-1,2,2")"
Icommon "("0,2,0")("1,0,2")("-1,1,2")("0,2,(0|1)")"
%%
{AI}{AII}{AIII}{AIV}$     {
            printf( "Type A matched: %s %s\n", yytext,
                    fileName);
            }

{BI}{BII}{BIII}$     {
            printf( "Type B matched: %s %s\n", yytext,
                    fileName);
            }


{SYM}*{Ccommon}{SYM}*$     {
            printf( "Type C matched: %s %s\n", yytext,
                    fileName);
            }
 SYM}*{Fcommon}{SYM}*$     {
            printf( "Type F matched: %s %s\n", yytext,
                    fileName);
            }


{Hcommon}{SYM}*$     {
            printf( "Type H matched: %s %s\n", yytext,
                    fileName);
            }

{Icommon}{SYM}*$     {
            printf( "Type I matched: %s %s\n", yytext,
                    fileName);
            }
```

```
(D)$     {
             printf( "Type D matched: %s %s\n", yytext,
                     fileName);
             }



%%

main( argc, argv )
int argc;
char **argv;
{
++argv, --argc;  /*("-2")+("-1")*0*$      ( skip over program name */
if ( argc > 0 ){
yyin = fopen( argv[0], "r" );
fileName=argv[0];
}
else{
fileName="Standard input";
yyin = stdin;
}
yylex();
}
```