INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

Université d'Ottawa · University of Ottawa

Periodic Analysis of Critical Dimension Measurements in the Photolithography Manufacturing Process

by

Erin L. Chapman

A thesis submitted to the Faculty of Graduate and Post-Doctoral Studies
In partial fulfillment of the requirements for the degree of

Master of Applied Science

in the
Department of Chemical Engineering
University of Ottawa
Ottawa, Canada

ABSTRACT

ModIV is Nortel Networks' semiconductor fabrication facility. It specializes in producing complex designs at low cost in a short period of time. This feature allows ModIV to serve as both a production and research facility.

The semiconductor manufacturing process is a series of precise layering steps carried out on silicon wafers. Each layering step occurs in one of three fabrication areas: device formation, photolithography and interconnect.

One of the biggest difficulties associated with troubleshooting problems in the semiconductor process is due to the lack of measurements performed on production batches. Most of the production measuring steps occur in the photolithography area.

In the photolithography area there are three sequential processing steps: coating, exposing and developing. For capacity and maintenance reasons, there are three pieces of equipment that can be used interchangeably at each of the three processing steps. This means that there are three coaters, three steppers and three developers. Measurements can only be taken after the developer step. The equipment interchangeability creates a problem when trying to detect the root cause of processing problems. Currently each processing level has its own control chart that incorporates all of the processing paths for that particular level. The primary objective of this work was to develop a method of isolating the root cause of processing problems.

A data extraction and sorting scheme was developed to create data files that could be analyzed for periodic behavior. A model building algorithm was created to detect periodic behavior and a series of stochastic simulations were used to determine the quantity of data required to detect the underlying signals with different signal to noise ratios. An experiment was conducted in the photolithography area to intentionally cause a problem. This problem isolation method was used, and the correct root cause of the problem was found.

ABRÉGÉ

ModIV est un laboratoire de manufacture de semiconducteurs pour la compagnie Nortel Networks. Le département se spécialise dans la fabrication de circuits complexes mais surtout et avant tout, dans la fabrication de semiconducteurs dans un délai de temps minime. ModIV est à la fois une manufacture et aussi un centre de recherche qui vise à développer de nouvelles architectures de semiconduteurs.

Le fabrication de semiconducteur comprend la reproduction de plusieurs étapes précises sur des pastilles de silicium. Chaque étape de la fabrication peut être classée en trois groupes: formation de composantes, micro lithographie et finalement, interconnection de composantes.

Un des plus grands défis associé à la diagnostique de problèmes dans l'industrie de semiconducteurs est dû au manque de mesures concrètes sur les lots de production. La grande majorité des mesures sont prises dans la section de micro lithographie. Cette section de la fabrication peut être résumée en trois étapes consécutives: le revêtement, l'exposition et le développement. Pour des raisons de capacité et aussi de maintien, trois outils interchangeables se retrouvent dans chacune des trois étapes. Ceci veut dire qu'il y a trois systèmes de revêtement, d'exposition et aussi de développement. Les mesures de qualités peuvent seulement être prises après le développement. Puisque les lots de production peuvent se déplacer sur n'importe quel système de revêtement, d'exposition ou de développement, le défi est de localiser le système défectueux. Présentement, chaque niveau de reproduction (micro lithographique) appartient à un tableau de données associées à toutes les étapes. L'objectif premier de ce travail était de développer une nouvelle méthode diagnostique afin de localiser les sources de problèmes.

Un système d'extraction de données et de triage a été développé pour créer des classeurs de données qui peuvent par la suite être analysées pour des caracteristiques périodiques. Un modèle algorithmique fut créé afin de localiser des signatures périodiques. Une série de

simulation stochastique a été utilisée pour déterminer la quantité de données nécessaires pour détecter des signaux de bases appartenant au rapport de signal au bruit.

Une expérience fut menée dans la section de micro lithographie; un problème fictif fut créé. Cette méthode algorithmique d'isolation fut utlisée et la source exacte du problème fut découverte.

ACKNOWLEDGEMENTS

I would like to thank Dr. David McLean of the University of Ottawa for his support and guidance throughout my masters degree. There were many times I needed someone to talk with and he was always there to listen and to point me in the right direction.

I would like to thank Jean Proulx and Ken Hardage of Nortel Networks for their encouragement, it helped me to work through the tough problems. I would also like to thank the Mod IV Photolithography group at Nortel Networks for their help in understanding the semiconductor manufacturing process. Your time and patience were greatly appreciated.

I would like to thank Sheila Copp and Patrick Burn for their assistance in writing the PROMIS script file.

I would also like to extend my most sincere words of appreciation to Mike King of Nortel Networks for financially supporting this project.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

a_k	Fourier series coefficients
a_p	Fourier series coefficients
b_p	Fourier series coefficients
$\|D_j\|$	Sum of the squared difference vector where vector j is begin subtracted from the data series
$\|D_o\|$	Sum of the squared original data series
f	frequency
$I(\omega_p)$	height of the histogram
k	integer value from $-\infty$ to ∞.
n	integer value from 0 to ∞
N	Total number of data points in the data series
N_1	position within the period where the block signal becomes zero
p	the number of frequencies that can be observed in a descrete data set
SSD	Sum of Squared Data
R_p	amplitude of the pth harmonic
T	period
x_i	the i^{th} value of the centered data sequence
X	sorted data
Δt	sampling interval
$\delta(x)$	Dirac delta function
$\Im(y)$	Fourier Transform of y
ϕ_p	phase of the pth harmonic
ω	angular frequency
$\nabla(i,j)$	element of the impulse matrix showing the i^{th} value of the j^{th} impulse vector

Photolithography Terminology

CD	critical dimension
CD Key	the area in the scribe channel where the CD is measured
CHECK	the first RS1 sorting program
CPU	Central Processing Unit
Coater	the processing equipment that coats each wafer with a thin layer of photoresist.
DCOP	Data Collection Operation Procedure
Developer	the processing equipment that develops pattern on the wafer
Die	the product microchip
Lot ID	the unique alphanumeric identification given to each lot of wafers
Measurement Site	the location on the wafer where the SEM made a measurement
Period	the number of data points in a sequence before it is repeated
PROMIS	PRoduction Operation and Management Information System
PM	Preventative Maintence
Reticle	the glass plate used in the stepper that is used to create the exposure pattern on the wafers
RS1	statistical software
Scribe Channel	the area between the product dies in the stepper shot
Script (PROMIS)	a series of commands that are stored in a file that can be executed with one command.
SEM	semi-automatic Scanning Electron beam Microscope
Stepper	the processing equipment that exposes the wafer with a UV light pattern
Stepper shot	the size of the exposure field on a wafer
SPECTRAL	the second RS1 sorting program

CHAPTER 1 INTRODUCTION

ModIV is Nortel Networks semiconductor fabrication facility. It specializes in producing complex designs at low cost in a short period of time. Currently, there are over 400 active designs in ModIV of which 40 to 60 may be in production at any given time. This feature allows ModIV to serve as both a production and research facility.

The semiconductor manufacturing process begins with bare silicon wafers and builds on them in a series of precise layering steps. Each processing step occurs in one of three processing areas: device formation, photolithography and interconnect. The photolithography area is the hub of all activity in the semiconductor manufacturing process. It creates the pattern on the wafers that is required for the other two processing areas.

In order to ensure that the pattern is laid down correctly, critical dimensions (CDs) in the wafer pattern are measured. These CD measurements are recorded in the production operation and management information system (PROMIS) in data collection operation procedures (DCOPs) which analyze the CD data for process control problems.

The photolithography process consists of three primary steps – coating, exposing and developing. It works in a very similar way to photography. Initially an extremely thin UV light sensitive coating is spun on to the silicon wafers. This is similar to the photographic film you

buy in a store. Then, the coated wafer is exposed to a UV light pattern. This is similar to taking a picture with a camera. The last step, as in photography, is development of the exposed image. The pattern is then visually inspected and, depending on the sampling plan, critical dimensions within the pattern are measured.

The process described in this thesis pertains specifically to the one used at Mod IV of Nortel Networks, but is typical of the processes used in the semiconductor industry.

Since there are many integrated circuit designs in production at any particular time in Mod IV, it is not possible nor cost effective to have processing equipment dedicated to each active design. Instead, there are several pieces of each type of equipment available for the photolithography processing steps. This kind of flexibility provides ModIV with the ability to produce product quickly, which is essential for its profitability.

One of the problems associated with this flexibility is the difficulty in detecting a process problem associated with a specific series of equipment. Currently, the exact process equipment flow for a batch of wafers is random. This is because there are multiple pieces of equipment at each of the three processing steps, and the assignment of equipment is left up to the discretion of the operator. This ensures that each batch of wafers spends the minimum amount of time waiting for a piece of processing equipment to become available.

There are currently 3 coater machines, 3 stepper machines and 3 developer machines, which comprise a total of 27 different processing paths that a batch could follow. With so many equipment processing combinations, it would be easy for a processing problem, associated with a particular piece of equipment, to appear as a random out-of-control point in a control chart. Although CD data are monitored at key processing levels using Shewart (i.e., R and X-bar) control charts, there is currently no control or diagnostic tool that specifically examines the role of each piece of processing equipment.

Another problem associated with the photolithography process is the lack of information available at each processing step. Critical dimensions can be measured only after a series of three processing steps. This makes it very difficult to determine which processing step is the cause of the problem. Consequently there is a need for a diagnostic tool that could help engineering personnel determine the root cause of processing problems in the photolithography area.

Such a diagnostic tool should be able to extract CD data along with identifying characteristics (e.g., measurement position, wafer #, coater #, stepper #, developer #), sort the CD data into appropriate sequences and analyze the data for trends in a way that engineering personnel could easily understand. This kind of diagnostic tool would provide an easy link between CD measurements and the nature of the processing conditions in the photolithography area. In this way this tool would be able to identify the processing characteristics that were the cause of the underlying problem with the process.

Progler (1995) suggested a sorting technique that ordered the data in a periodic fashion. He then used "spectral analysis" and periodic modeling to detect periodic out-of–control signals which could be directly related to the identifying characteristics of this process. Although his approach seemed to offer significant potential, there were several unanswered questions about his modeling methodology and the robustness of his approach for low signal to noise ratios and limited data situations.

The overall goal of this work was to build and test a diagnostic tool based on Progler's approach which could be applied, but not restricted to the photolithography process.

The specific objectives of this thesis were:

1. To develop a data extraction and sorting technique to create data sequences from which such periodic trends in product quality variables can be identified.
2. To develop a model building algorithm to detect the underlying periodic signals in a data sequence

3. To use stochastic simulations to determine the quantity of data required for different signal to noise ratios to detect underlying signals.
4. To test the methodology using real data into which known signals have been embedded.

This thesis consists of five chapters: Introduction, Background and Review of Literature, Methodology, Results and Discussion, Conclusions and Recommendations. The second chapter, entitled Background and Review of Literature, gives the reader a broad understanding of the photolithography area by describing the photolithography process in detail. The ideas behind periodic analysis are then introduced as the work by Progler is described and critiqued. Chapter Three describes in detail all of the extracting, sorting and analysis techniques used to diagnose periodic error in CD data. The methodology for tests on the photolithography process is also described in this chapter. Chapter Four shows typical results from the stochastic simulations and discusses some of the current limitations for this diagnostic tool. The results from the experiment are show as well as results from the process where no known problem exist. This thesis ends with Conclusions and Recommendations in Chapter Five. Copies of all the software are provided in Appendices A,B,C,D and F. Secondary data are given in Appendix E while sample calculations are outlined in Appendix G.

CHAPTER 2 BACKGROUND AND REVIEW OF LITERATURE

This chapter describes the photolithography process and the techniques uses to sustain the process control in the photolithography area. It describes and critiques the data sorting and analysis techniques used by Progler (1995) and other literature sources that describe important techniques in periodic modeling.

2.1 Process Management

The silicon fabrication process is very complex. To ensure that the right processing steps are carried out in the correct order, a computerized system is used to control the process flow of all batches through the manufacturing process. This system is called the PRoduction Operation Management Information System or PROMIS. All aspects of production planning, fabrication, testing and process monitoring are controlled through this system.

A lot is started when a unique lot ID is assigned to a batch of silicon wafers. A label with a unique bar code and lot ID is placed on the outside of the box containing the wafers. This is used to identify and track the lot throughout the process. This same lot ID is laser-scribed into

each wafer along with a wafer number. (The terms 'batch' and 'lot' are synonymous for a group of wafers that are processed and measured together.)

Each batch contains up to 25 wafers and each wafer within the batch is uniquely identified. For example, the seventh wafer of a lot would have the lot ID laser-scribed into the wafer with a suffix of '.07'. After the lot ID is created, it is assigned a particular process flow in PROMIS. A process flow is the sequence of processing conditions and recipes that are used to process a batch of wafers throughout the manufacturing process. There are many similarities between the process flows of different products.

The processing of each lot is controlled by tracking it into and out of each piece of equipment in the process. PROMIS does not allow a lot to be tracked into the wrong piece of equipment, nor does it allow the process flow to continue when data are missing or the entered data constitute an out-of-control signal in the data base. PROMIS stores all processing information: the time the lot was tracked in, what operator tracked it in, what piece of equipment the lot was tracked into, etc.

2.2 Photolithography

The photolithography process described in this work is the one used in Mod IV at Nortel Networks fabrication facility in Nepean, Ontario. It is however, a typical example of the manufacturing process used in the semiconductor industry.

The semiconductor manufacturing process builds a microchip through a series of precise layering steps. The design of the microchip requires that each layer be built directly on top of the previous layer. The alignment of these layers is controlled in the patterning process which is carried out in the photolithography area.

The photolithography process consists of three sequential processing steps: coating, exposing and developing. The coating step is carried out on a coater track, the exposing step is carried out on a stepper and the developing step is carried out on a developer track. These processing steps are shown in Figure 1.

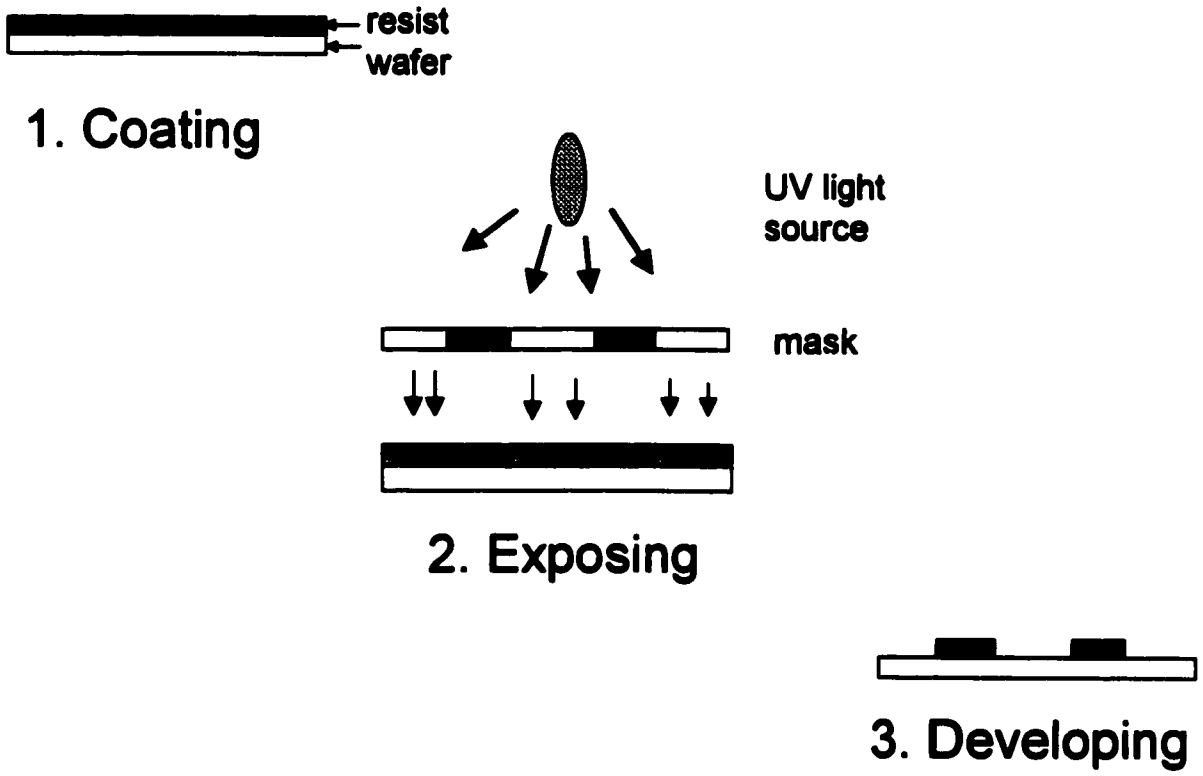


Figure 1: The Photolithography Process

2.2.1 Coating

When a lot enters the photolithography area it is ready to be tracked into the coating step. To do this an operator looks at the list of coater tracks available for processing in PROMIS and tracks the lot into an appropriate coater track. PROMIS then downloads the recipe number to the selected coater track and tells the operator onto which of the four loading positions to put the lot. The actual recipes reside on the coater tracks themselves and contain all of the operating conditions for how each batch is processed on the coater track.

After the lot is loaded on the coater track, a robot checks each slot in the cassette carrier for wafers. The top wafer is then taken to a hot plate where it is coated with adhesive. The robot then transfers the wafer to a cooling plate. After the wafer is cooled it is moved to the coating chamber. The coating chamber is used to spin on an ultra thin film of photoresist. After the entire wafer is coated with resist, a solvent is used to remove the resist on the edge of the wafer. The wafer is then moved to another hot plate where it is soft baked. Once the soft bake is finished the wafer travels back to the carrier and is deposited into the same slot that it was taken from initially.

There are two types of photoresist: positive and negative. Positive resist contains cross-linked polymers that are very difficult to remove from the surface of the wafer. These cross-links break down in the presence of UV light. Thus, when using a positive resist, the places in the pattern where light flows through the reticle are where the resist will be washed away in the developer step. The opposite is true for negative resist; instead of breaking down the cross-linking process the UV light initiates cross-linking. Different chemicals are used in the developer tracks to develop positive and negative photoresists.

Each wafer travels independently through the coater track. Only one wafer enters each of the four processing chambers at a time. Out of the three coater tracks considered in this work, two of these tracks have a dual coater bowl system. This means that on the same coater track there are two coating chambers where two wafers can be coated simultaneously.
Once the lot has been processed, the operator removes the lot from the coater track and tracks the lot out of the coating step in PROMIS.

2.2.2 Exposing

After the lot is tracked out of the coater, it is tracked into one of the steppers. The stepper contains a UV-light source that exposes a pattern into the coated wafers. The exposure pattern that is used on the wafers is stored on a glass plate called a reticle. Since the pattern

on the reticle is product and process layer specific, it is tracked into PROMIS at the same time that the lot is tracked into the stepper equipment.

The wafer is exposed in an array of shots across the surface of the wafer. Each shot contains the exact pattern in the reticle, which, depending on the size of the die, may contain more than one die. A typical stepper shot layout can be seen in Figure 2.

Surrounding each die on the wafer is a narrow path called a scribe channel. Figure 2 shows the details of a typical shot including the scribe channel. The primary reason for the scribe channel is to provide a path for the saw to use when it cuts up the wafer into individual dies. Since scribe channels exist all across the wafer and are processed exactly the same as the product die, the scribe channel serves as an excellent test location. On the reticle, test structures with the same critical dimensions as the product die are built into the scribe channel. These are the critical dimensions that are measured and stored in PROMIS as a means to monitor the control of each processing step in photolithography.

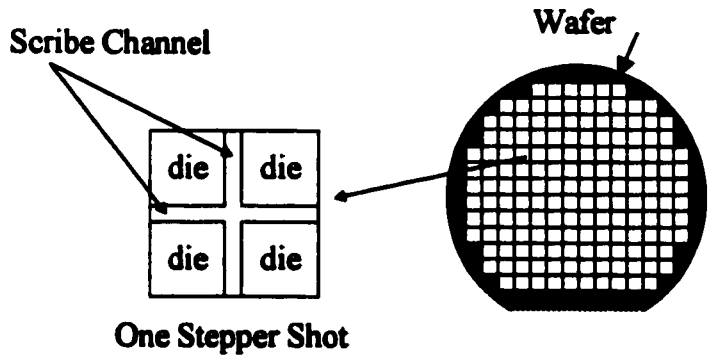


Figure 2 Layout of Stepper Shots on a Wafer

Each reticle also contains special alignment marks that are printed into the wafers. It is these alignment marks that the stepper looks for to ensure the current pattern aligns with previous

layers on the wafer. This alignment is critical because the performance of the device depends on each of the processing levels lining up.

2.2.3 Developer

The developer step removes the unwanted photoresist one wafer at a time in a developer bowl. Once the wafer is in the developer bowl, the wafer is spun and developer chemical is poured over the surface of the wafer. Development time is controlled to ensure that all the unwanted resist is washed from the surface of the wafer. Each chemical in the developer track is specially formulated to work with a specific type of photoresist in the coater track.

2.2.4 Inspection

After leaving the developer, the wafers are inspected at a macro level. This step requires an operator to visually inspect the wafers, with the aid of a manual microscope, to ensure that the pattern does not have any visible defects. At this level, the operator is mainly looking for scratches covering more than 5 die and any discoloration due to non-uniform processing that affects more than thirty percent of the wafer.

After passing macro inspection, the lot, depending on the sampling plan and level, is passed to micro inspection. It is here where the semi-automatic Scanning Electron beam Microscope (SEM) is used to examine the critical features of the pattern. The term 'critical feature' refers to the size of the smallest dimension in the resist pattern on that process layer. In order to reduce the impact of measuring the actual microchip, critical dimensions (CDs) are measured from the CD Key. This is located in the scribe channel. The type of measurement depends on the nature of the critical feature. On some processing levels the critical feature is the width of a line while on other levels, the width of a gap between a set of lines is of concern. A trained SEM operator measures the critical dimension (CD) on the CD key for a predetermined number of locations across the wafer and records these measurements in PROMIS. This series

of measurements is carried out in the same locations and in the same order on each tested wafer. The information concerning the location and order of the measurements is stored in the SEM as a job file. Each processing level that is measured on the SEM has a different job file. This job file is programmed by engineering personnel. After the wafer's critical dimensions (CDs) are measured, the CDs are entered into PROMIS as part of the micro inspection step and are immediately part of the control charting process. If the CDs result in any control chart violations, based on a subset of the Nelson Rules (Nelson, 1984), the lot is immediately put on hold and the sustaining engineer is notified.

2.3 Process Monitoring and Control

Each critical layer in the photolithography area has an associated sampling plan where CD data are collected. The extent of the sampling plan depends on the processing level and its impact on the final product. In some cases this means every lot is measured and in other cases it means that as few as one in five lots is measured.

Very few measurements in the semiconductor manufacturing process are actually performed on production wafers. Consequently, it is very important to have other ways of monitoring the process. PROMIS schedules and keeps track of a series of process equipment checks. The nature and frequency of these checks depend on the impact of process equipment malfunctions. This impact is assessed by engineering personnel who also determine the necessary frequency of the process checks.

The checks in the photolithography area ensure that comparable sets of equipment are in control and can be used interchangeably. PROMIS checks to ensure that the data entered from the tasks meets the required control limits. If the data entered do not meet the control limits, the equipment becomes unavailable to production.

Each piece of processing equipment is also thoroughly cleaned as a preventative maintenance (PM) initiative. These PMs are carried out by specially trained personnel and all results and times are recorded in PROMIS.

2.3.1 Coater Track Checks

In order to maintain the required thickness of photoresist, thickness checks are carried out on dummy wafers every 12 hours. If the measured thickness on these dummy wafers is not within specification, the spin speed is adjusted and a new set of dummy wafers are coated and measured. This process is carried out until the dummy wafer measurements are within the specification limits. The spin speed is usually adjusted to compensate for changes in resist thickness caused by atmospheric pressure changes. The specification limits for the mean photoresist thickness and thickness range are not derived from a set of design criteria; instead they are based on the resist thickness suggested by the supplier. The resist thickness checks are carried out on all of the coater tracks. This ensures that all coater tracks are operating within the same set of specification limits, and hence can be used interchangeably.

In order to test that each coater track is working the same, three wafers are coated, one on each coater track. These three wafers are then exposed with a column pattern of UV light energy increment shots. Each of the three wafers are exposed and developed on the same equipment set. The first increment where the photoresist is fully removed is compared for the three wafers. If the increment that was first cleared of photoresist is the same across all wafers, the coater matching task is complete; if not, adjustments to the coater tracks are made and the process is repeated.

2.3.2 Stepper Checks

In order to test that each of the steppers is working the same, a special exposure test is carried out. A wafer is coated with photoresist and then exposed as a column with blocks of

increasing UV light energy. Each stepper exposes the same wafer with one column of blocks with the same UV light energy increments. The wafer is then developed. The UV light energy required to remove all of the photoresist should be at the same energy level across all three columns if the steppers are well matched. If the steppers are not matched, adjustments to the UV light energy are made and the test is repeated until all three steppers match.

Each block in the column represents a certain amount of UV light energy. The energy required to clear a block of resist is recorded in PROMIS. This comes from the column of blocks of increasing UV light energy. Each block corresponds to a certain energy level. It is left up to the eye of the operator to best judge which block in the column is the first to be completely clear of photoresist.

2.3.3 Developer Checks

In order to determine that each developer is working the same, three wafers are coated and exposed using the same coater and stepper. The wafers are then developed on the three different developer tracks. If the developers are all working the same way, all wafers should have the same exposure pattern on each of the three wafers. Developer matching relies mainly on the developer chemical since the recipe for developing wafers is the same on each developer track. This test mainly ensures that no chemical degradation has occurred. If the developers do not match, the chemistry of the developer is examined. The developer process is currently the most robust process in photolithography and unmatched developer tracks are very rare.

2.3.4 Diagnostic Methods

With the photolithography process, it is very difficult to determine why a CD value has gone out of control. The current response to such a situation is to rework CDs that are out of control. This means that the resist pattern is stripped off and the coater, stepper, developer

process is repeated. The rework process adds a significant cost to the manufacturing process both in time and materials.

Recall that since the specific processing equipment is random, a slight problem with one piece of equipment will appear as a random point in the control charts. The engineering diagnostics are further complicated by the number of processing steps that occur between measurements. It is both difficult and time consuming with the current engineering tools to evaluate each of the different possible causes of an out-of-control signal.

There are very few published articles dealing with the specifics of problem solving and root cause identification in photolithography or more generally for multistep processes. Most new manufacturing techniques are kept proprietary by the respective companies. This is primarily due to the fiercely competitive nature of the microelectronics industry, where even slight process improvements can mean hundreds of thousands of dollars in savings. However, a paper by C. J. Progler (1995) entitled "Systematic Analysis of CD Data" spawned the idea for the diagnostic procedure developed in this work.

2.4 Periodic Analysis

Using a photolithography example, Progler showed how appropriate sorting of CD data could be coupled with a periodic analysis to identify the existence of a processing problem and also provide valuable pointers to the root cause of the problem. In this section, Progler's approach will be outlined and evaluated.

The first step in Progler's approach is to sort the CD data so that the unique processing attributes are spaced evenly apart. For example, if CD data exist from two steppers (S1 and S2), and for each batch that was processed through these steppers two wafers (W1 and W2) were sampled from each batch, and for each wafer that was measured, five sites

(M1:M5) were measured in exactly the same locations on the wafer, the CD data sequence would look like

CD Data[1:20]=S1[W1(M1:M5)]S1[W2(M1:M5)]S2[W1(M1:M5)]S2[W2(M1:M5)]

The data sequence shown in here has multiple unique processing attributes. For example, every fifth measurement comes from the same location on the wafer surface, every tenth measurement comes from the same wafer number. When this data sequence is repeated, every twentieth measurement comes from the same stepper, wafer and measurement number. It is these periods (5, 10 and 20) that Progler creates from his sorting scheme.
Progler identifies three types of periodic signals that could exist in the photolithography process. They all stem from the wafer sampling plan and the sequencing of the CD data. CD measurements are performed on a semi-automatic scanning electron-beam microscope (SEM). Due to the semiautomatic nature of the SEM, the location of the measurement on the wafers surface is always the same; also the order in which the CDs are measured is always the same. This sequential measurement scheme is critical to being able to identify within wafer periodic effects.

If a wafer is measured on a SEM in five different places, and the CD data have been sorted so that measurements on a wafer appear in the same repeating sequence (site 1, site 2, site 3, site 4, site 5), then every fifth measurement will always be from the same location on a wafer. If for some reason one of the five measurements was always higher than the other four measurements, a periodic signal in the data would be observed. This type of signal is called an impulse signal, because the same data point stands out from the other data points in each period of the sequence. The period in this case, would be five, the number of data points that are measured on each wafer. More generally a period is the number of data points in a sequence before the sequence is repeated. Figure 3 shows an impulse signal with a period of ten data measurements, where the first measurement in the period is consistently larger than the other wafer measurements. The magnitude of the data in this figure and throughout this thesis can be considered to be the deviation of the measured value from its target. The impulse

signal shown in Figure 3 depicts a perfect impulse signal (i.e. without noise) of magnitude one.

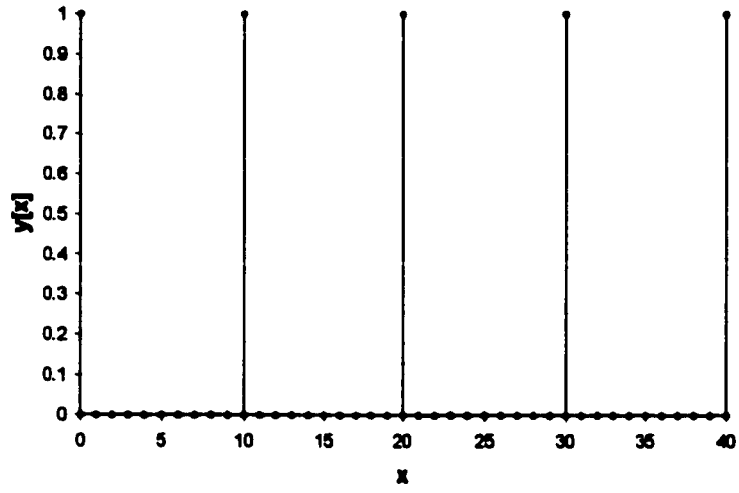


Figure 3 Impulse Signal with a Period of 10

The second type of periodic signal that Progler identifies is a two-impulse signal. This type of signal may occur, depending on the sampling plan, if two measurements are in the same affected area of a processing problem. This may occur if only part of the wafer is affected by a processing problem. For example, if the stepper was out of focus in one area of the wafer, the CDs that were in focus would be different than the CDs not in focus. Figure 4 shows a two-impulse signal where the first and third measurements in a period of 20 are affected by some type of processing problem. The first measurement of the impulse signal is referenced by x=0, similarly the third measurement is referenced by x=2. This convention is shown in Figure 4.

The third type of signal that Progler identifies is a block effect. This type of signal may occur if an entire wafer is affected by a processing problem. For example a block effect may be caused by an incorrect exposure setting being entered into the stepper. This way, all measurements on the wafer would be affected. A typical block effect with a period of 10 can be seen in Figure 5.

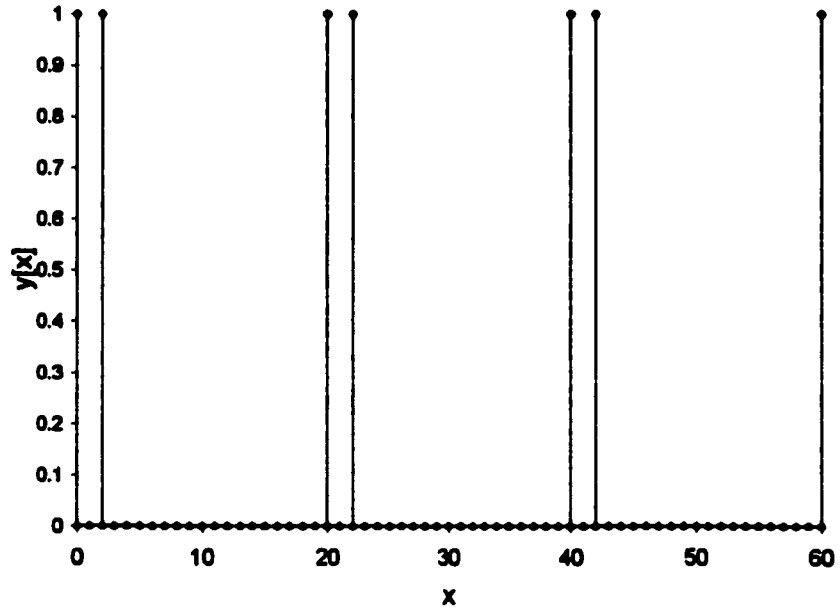


Figure 4 Two-Impulse Signal with a Period of 20

In Progler's paper he gives an example of each type of signal and presents the associated discrete and frequency domain models. The models for the impulse, two impulse and block effect are given in equations1-10, respectively.

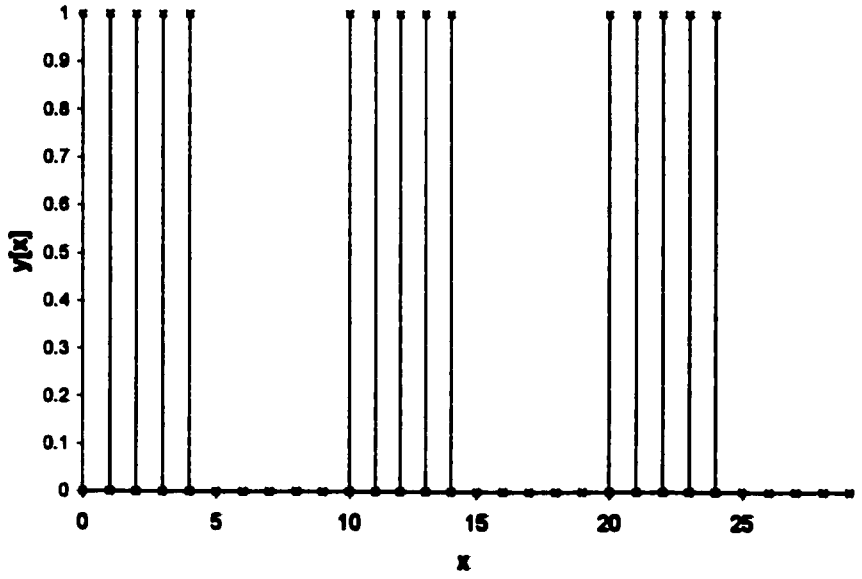


Figure 5 Block Signal of Length 5 with a Period of 10

A discrete impulse signal with a period of 10 is described by

$$y[x]=\sum_{n=0}^{\infty}\delta(x-n10) \qquad (1)$$

where x is only defined for integers and represents the position of the measurement in a sequence, n is a counter for the number of repeating periods, δ(x) is the Dirac delta function defined as

$$\delta(x)=1 \text{ for } x=0$$
$$\delta(x)=0 \text{ for } x=0 \qquad (2)$$

Taking the Fourier transform of equation (1) gives the frequency domain representation for this impulse signal

$$\Im(y)=\sum_{n=0}^{\infty}\delta\left(f-\frac{n}{10}\right) \qquad (3)$$

where f is the frequency.

A discrete two-impulse signal with impulses at locations 0 and 2 in a period of 20 is described by

$$y[x]=\sum_{n=0}^{\infty}[\delta(x-n20)+\delta(x-20n-2)] \qquad (4)$$

The Fourier transform of this signal is described by

$$\Im(y)=\cos(2\pi f)\sum_{n=0}^{\infty}\delta\left(f-\frac{n}{20}\right) \qquad (5)$$

Progler's expression for the block signal appears to be incorrect in his paper. The correct expression for the block signal model is shown in the following equations.

For the block signal

$$y[x]=1, \text{ for } x\le N_1$$
$$y[x]=0, \text{ for } x>N_1 \qquad (6)$$

where

$$N_1 \le N \quad \text{and} \quad y[x+N] = y[x] \tag{7}$$

The Fourier transform of the block signal is

$$\Im(y) = 2\pi \sum_{k=-\infty}^{\infty} a_k \delta\left(\omega - \frac{2\pi k}{N}\right) \tag{8}$$

where ω is angular frequency defined as

$$\omega = 2\pi f \tag{9}$$

and

$$a_k = \frac{\sin[(2\pi k / N)(N_1 + 1/2)]}{N \sin[2\pi k / 2N]} \quad \text{for k} \neq 0, \pm\text{N}, \pm 2\text{N},... \tag{10}$$

$$a_k = \frac{2N_1 + 1}{N} \quad \text{for k} = 0, \pm\text{N}, \pm 2\text{N},...$$

A major limitation with Progler's models is that the location of the signal, within the period, (i.e. the phase information) is lost. As a result, one can't tell the difference between a signal that begins at the first observation in the period from a signal that begins at another location in the period. This information becomes even more important when more than one signal exists in a set of data. Without the phase information, signals can no longer be uniquely identified with the models Progler presents in his paper.

The best way to illustrate this point is through the use of an example. Consider a 5 point block signal with a period of 10 data points (see Figure 5). If this block signal was added to an impulse signal with a period of 10 data points, the combined signal could be one of various results. As stated above no phase information is given, so all we know is that within those 10 data points there is a block effect and an impulse signal. The way these two signals exist within the period changes the resultant signal drastically. Consider the case where the block signal of five begins at the first point of the period and the impulse occurs at the third point in the period. This resultant signal can be seen in Figure 6. Consider another case where the

block signal begins at the 6th data point and the impulse signal remains at the third point of the period. This signal is shown in Figure 7. Both of these signals would yield the same model form using Progler's signal models and yet their respective signals are very different.

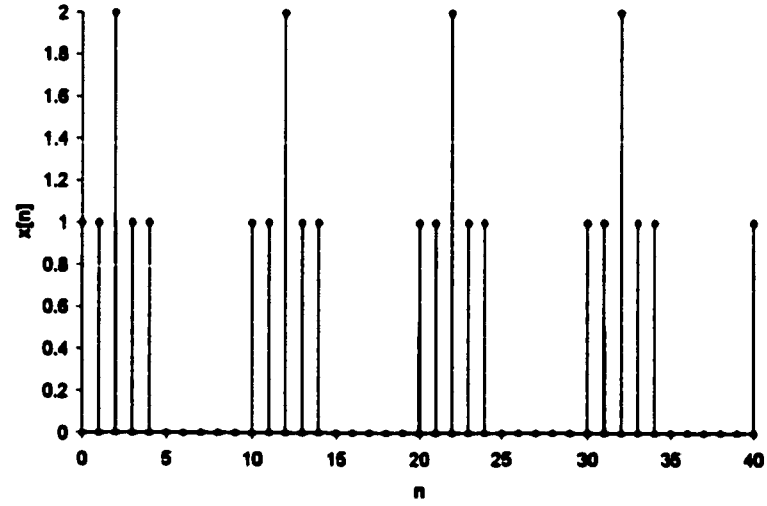


Figure 6 Block Signal (0 to 4) + Impulse (@2)

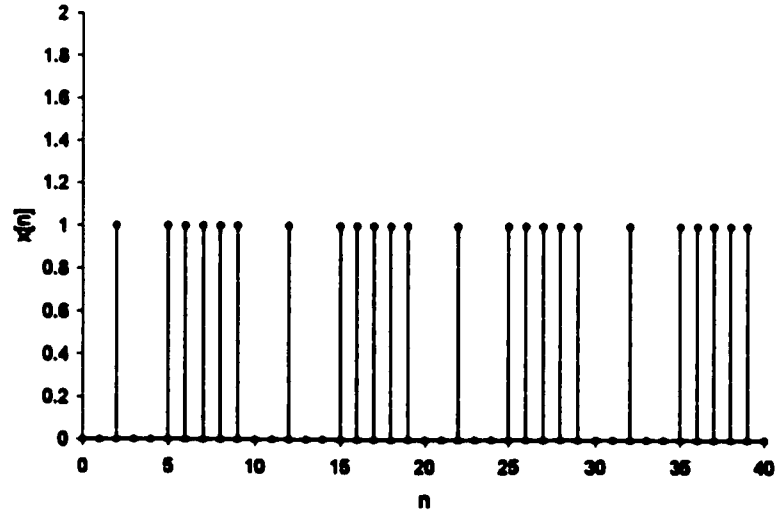


Figure 7 Block Signal (5 to 9) + Impulse (@2)

Another problem with Progler's models is that they contain no information about the magnitude of the signals. This is very important because the relative size of the signal shows how much it is disturbing the process and could possibly aid in diagnosing the root cause of the problem.

It is also unfortunate that Progler does not explain the details of how he uses the three signal models to determine if the sorted data contain periodic errors. Progler uses a periodogram to graphically analyze the sorted data for periodic errors. However, the task of correctly modeling a periodic error from a periodogram is very difficult if more than one underlying signal exists in the data set. If only one signal exists within the data it is fairly easy to recognize because it would look like one of the three types of signals that Progler describes. It should be noted that Progler's two-impulse signal is only accurate for signals that differ in phase by one observation. There are many other possibilities depending on the length of the period.

In Progler's example, potential differences in CDs obtained from the use of two reticles was of interest. After ordering the CD data so that measurements on wafers processed with one reticle were followed by CD data from wafers processed with the second reticle, a difference was detected indicating there was a problem with one of the reticles. According to experts at Nortel Networks in Ottawa Ontario, this type of shift in data would be extremely obvious and is a very weak demonstration of the use of this methodology. In fact, simply plotting the data sequences reveals this effect.

Progler did not examine the effects of signal to noise ratios, nor did he examine the amount of available data and its effect on the sensitivity and robustness of his approach. This is essential information for proper use and application of this technique. Although this approach seemed to have potential, the many unanswered questions provided much of the motivation for this thesis.

The periodogram is a way of quantifying the variance contained in a data set (Chatfield 1997, Beneka et al.,1988). The variance of data at different frequencies is plotted. When the variance at one frequency is larger than others, a periodic effect is occurring at that particular frequency. One of the first steps that is usually taken when performing any type of variance analysis is centering of the data. In the photolithography example, this means that the CD target value for the processing level is subtracted from the data. The resulting data then contain only deviations from this target value. This centered data set is analyzed for variance on the periodogram because any variation that the periodogram detects is a direct result of a data point being off target.

The potential for periodogram analysis was discussed from a process control point of view by Beneka et al.(1988). It was suggested that periodic data trends may exist within the realm of the statistical process control limits. It is well known in the semiconductor industry that weather patterns affect the fabrication process. This is thought to occur because the photolithography process is extreme sensitive to atmospheric pressure and humidity. This kind of periodic effect may be visible with periodic analysis if sufficient data are used.

Beneka et al.(1988) used periodogram analysis for the average wafer values so as to insure the independence of all measurements. This, however, is not necessarily the case when a piece of processing equipment has just been cleaned. There is a period of time where the equipment is reaching a steady state. Anything processed through a machine reaching steady state would be correlated with other batches that were processed during the time in which the equipment was reaching steady state. Performing periodogram analysis on wafer averages also results in losing all information with respect to specific wafer sites and any effects that may exist across the surface of the wafer.

While trying to gain a better understanding of Progler's work, a key observation was made. The magnitude of the observed signal in the periodogram is directly a function of the number of periods in the data string. This means that the more periods one has in a data string, the stronger the signal and hence the periodic effect is easier to detect. This is a very important

observation because the stronger the signal is, the easier it is to detect. This effect was demonstrated in this thesis though the use of stochastic simulations.

Some problems associated with detecting a weak signal have been outlined by Yeung et al. (1996). A weak signal is characterized by a small signal to noise ratio. This is the case for the photolithography application because we are looking for periodic signals in data that would not necessarily be picked up with standard control charting techniques. One of the problems associated with the detection of weak signals, is the quantity of data required to detect the signal (Yeung et al., 1996, Beneka et al., 1988). The problem is generally associated with the quantity of time it takes to process large amounts of data. This is typical of electrical engineering applications dealing with the transmission and reception of signals on a real time basis. Such limitations are not so critical in our study because calculations are done off line.

The modeling algorithm used in this work is supported by a key point made by Auslander et al. (1990) in his work regarding advanced signal-processing algorithms. The time it takes to find a signal can be greatly decreased when the span of the search is narrowed. This philosophy was used in the modeling algorithm. Periods in increments of 5 were examined beginning with an initial period of 5 up to and including 45 data points. This 5 measurement increment incorporates all of the possible periodic behaviors that could be observed in the process.

A second key point that Auslander makes is to use all the process information you have when trying to detect the signal. For example, if the signal to be detected is known, then a filter should be used that exactly matches the period of the signal to be detected. Since not all information is known about the type of periodic signal an exact matching filter cannot be used. Instead a series of filters should be used that model the possible periodic processing effects in the photolithography case.

A third concept that Auslander explains is the windowing function. The windowing function is a concept that spans a period of data in which a signal is expected to be observed. This works

along with the idea of using all available information for the type of signal you expect. This is the very idea that was used in the modeling algorithm developed in this work.

CHAPTER 3 METHODOLOGY

The purpose of this work was to develop a method that could detect a periodic signal in a given data set. The methodology consists of three steps: data extraction, data sorting and data analysis. This chapter explains each of the three parts and their role in periodic signal detection. Later in the chapter, tests of the effectiveness of this method using both simulated and process data are described.

Three software programs involving two different computer platforms were used in creating this methodology. The need to work with two platforms resulted from all data files being stored on PROMIS, which is operated on a VAX platform. Data extraction was performed using PROMIS. The extracted data file was then transferred to the Unix environment where it was sorted by a program written in RPL. RPL is the programming language for RS1, the statistical software package used at Nortel Networks. The output of the sorting routine was analyzed by model building and periodogram programs, both written in Matlab.

3.1 Data Extraction

The data extraction routine is a script file written in PROMIS. The user is prompted for a Data Collection Operational Procedure (DCOP) and a time period for the data extraction. In order to obtain all of the necessary information (CD values, lot ID, wafer number, test

location, processing equipment ID), the script must extract two sets of data and merge the information into one file based on the lot ID.

The database in PROMIS is very difficult to work with due largely to poor documentation. There are many field sorting parameters that do not come with adequate explanations or comments. In fact, some of the sorting fields have no explanation at all. This lack of documentation made writing this script extremely difficult and time consuming. Initially this script was to be written by one of the PROMIS script experts. After this attempt failed, another expert was asked to write the script. It was not until the author became directly involved on a daily basis that the script was successful. This effort took the author of this work approximately six weeks to complete after all experts had failed.

Table 1 shows a sample output table from the PROMIS script that was developed to extract CD data. The first column is labeled LOT_ID. It contains the lot identification used to track the lot throughout the manufacturing process. The second and third columns are labeled STAGE and RECNO, these are both internal PROMIS labels used in extracting the data. The next thing to note is the fourth column labeled COMPID. COMPID is short for Component ID. The COMPID contains the Lot ID as the root part of the number but the extension (.1 in this case) is the order in which the wafers, in the same batch, were measured on the SEM. For example, in Table 1, the first five rows contain the same number and extension. This is because there are five sites being measured per wafer. The COMPID for the second wafer being measured has the extension '.2'. The next thing to notice about this table, is the fifth column, SITE. This column contains the numeric sequence of SEM measurements performed on each wafer. The maximum value in this column is the total number of sites (or locations) measured on each wafer. The sixth column CD_DATA contains the actual measurement values obtained from the SEM. The seventh column EQUIP is the first of two EQUIP columns, and refers to the equipment that the batch was measured on. The eighth and nineth columns entitled ET_TIME refer to the date and time that the batch was measured with the measurement equipment.The tenth column, RECNO is short for Record Number and is used internally in the PROMIS data base. The eleventh column, EQUIP, refers to the processing

equipment that was used on the batch, while the twelveth and thirteenth column, entitled TRACKINTIME, refer to the date and time that the batch was processed with said equipment. The eleventh, twelveth and thirteenth columns in Table 1 are separated with a vertical line and are to be read as a group of three columns. The fourth row of the eleventh column is"403WED". This term serves only as a placeholder in the data.

From looking at the SITE and COMPID columns, it is easy to determine the sampling plan for this DCOP. In this example two wafers are measured per lot and five sites are measured per wafer. In all there are ten measurements per lot. The specific equipment that this batch was processed with, at this particular processing level, is shown in order of process flow in the second column entitled EQUIP.

Table 1 PROMIS Data Extraction Output

LOT_ID	STAGE	RECNO	COMPID	SITE	CD_DATA	EQUIP	BT_TIME		RECNO	EQUIP	TRACKINTIME	
W9905471.1	4P1_PE	74	W9905471.1	1	0.882	403SEM	1-Mar-99	4:24:32	5791	407COAT	1-Mar-99	1:48:20
W9905471.1	4P1_PE	74	W9905471.1	2	0.879	403SEM	1-Mar-99	4:24:32	5792	407STEP	1-Mar-99	2:17:48
W9905471.1	4P1_PE	74	W9905471.1	3	0.873	403SEM	1-Mar-99	4:24:32	5793	409DEV	1-Mar-99	2:52:59
W9905471.1	4P1_PE	74	W9905471.1	4	0.869	403SEM	1-Mar-99	4:24:32	5794	403WED	1-Mar-99	4:17:27
W9905471.1	4P1_PE	74	W9905471.1	5	0.88	403SEM	1-Mar-99	4:24:32	5795	403WED	1-Mar-99	4:17:27
W9905471.1	4P1_PE	74	W9905471.2	1	0.875	403SEM	1-Mar-99	4:24:32	5795	403WED	1-Mar-99	4:17:27
W9905471.1	4P1_PE	74	W9905471.2	2	0.86	403SEM	1-Mar-99	4:24:32	5795	403WED	1-Mar-99	4:17:27
W9905471.1	4P1_PE	74	W9905471.2	3	0.863	403SEM	1-Mar-99	4:24:32	5795	403WED	1-Mar-99	4:17:27
W9905471.1	4P1_PE	74	W9905471.2	4	0.861	403SEM	1-Mar-99	4:24:32	5795	403WED	1-Mar-99	4:17:27
W9905471.1	4P1_PE	74	W9905471.2	5	0.864	403SEM	1-Mar-99	4:24:32	5795	403WED	1-Mar-99	4:17:27

The output from the PROMIS script should be called ERIN4.PRT;1. PROMIS keeps old versions of files such that new data extraction have the same name with an incremental ending (i.e., ERIN4.PRT;2). The RS1 program cannot read the incremental ending so it is important to make sure that no other version of ERIN4.PRT exists before running the data extraction script.

The ERIN4.PRT;1 file has to be transferred to the RS1 platform for data sorting. There are two RS1 computer platforms: PC and Unix. When using the PC platform, the file has to be put into the work folder of the RS1 program in order for RS1 to recognize it. When using the

Unix platform the file has to be put in the home directory. The location of this file is important because the exact path of the file is required for it to be called from the program written in RPL.

The data extraction software is provided in Appendix A.

3.2 Data Sorting

The biggest problem in writing the PROMIS script file was finding a way to link the data from each lot to the equipment that each lot was processed with. A compromise was reached, such that the processing equipment for each lot was listed in a column. This is shown in Table 1.

Sometimes an error occurs while a batch is being processed. When this happens the operator has to reprocess the batch in what is called a rework loop. This means that the lot has to be stripped of the current processing and reprocessed. The data extraction script captures all of the processing steps that were carried out on each batch for a particular processing level. This processing sequence information is used to screen out batches that have been reworked. This should not significantly impact the size of the data set because the number of reworked batches is low in photolithography.

Once the data extraction file from PROMIS is copied to where the RS1 program can read it, the program CHECK is run. The purpose of CHECK is to screen out batches that have been reworked and to presort the data extraction file. CHECK does not prompt the user for the file name of the PROMIS script output file. The exact format of the file name is critical to accessing the file and since it is a nonstandard filename, it was decided to put the filename directly into the CHECK program so that users would not be frustrated trying to type the name of the file. The CHECK program instead prompts the user for a filename to store the presorted output. It then prompts the user for the processing flow of the batch. To do this, the user is prompted with a series of two questions that apply to the photolithography area. The

first question asks what type of processing equipment is used (coater, stepper, or developer). The second question then asks if there are more processing steps. If the user answers yes, the two questions are repeated. This dialogue continues until the user answers "no" to the question "are there more processing steps?". This dialogue determines the correct process flow for batches that have not been reworked. A comparison is carried out and any batches that do not have the exact process flow, as determined by the user, are screened out of the data extract. The names of the processing equipment in the photolithography area are coded into the CHECK program. If any equipment names are added or changed this program needs to be updated.

In order to make sorting easier, CHECK makes a column out of each processing step in photolithography. More specifically, CHECK transposes the processing equipment information, on a lot by lot basis, to a row vector. The result of transposing the process equipment information from column 11 in Table 1 is shown in Table 2. In the example shown in Table 2, column 4 always pertains to the first piece of processing equipment in the photolithography process (eg. coater track), column 5 will always pertain to the second processing step, etc.

Table 2 Expanded Process Equipment Format

EQUIP	TRACKINTIME										
407COAT	1-Mar-99	1:48:20	407COAT	407STEP	409DEV	403WED	403WED	403WED	403WED	403WED	403WED
407STEP	1-Mar-99	2:17:48	407COAT	407STEP	409DEV	403WED	403WED	403WED	403WED	403WED	403WED
409DEV	1-Mar-99	2:52:59	407COAT	407STEP	409DEV	403WED	403WED	403WED	403WED	403WED	403WED
403WED	1-Mar-99	4:17:27	407COAT	407STEP	409DEV	403WED	403WED	403WED	403WED	403WED	403WED
403WED	1-Mar-99	4:17:27	407COAT	407STEP	409DEV	403WED	403WED	403WED	403WED	403WED	403WED
403WED	1-Mar-99	4:17:27	407COAT	407STEP	409DEV	403WED	403WED	403WED	403WED	403WED	403WED
403WED	1-Mar-99	4:17:27	407COAT	407STEP	409DEV	403WED	403WED	403WED	403WED	403WED	403WED
403WED	1-Mar-99	4:17:27	407COAT	407STEP	409DEV	403WED	403WED	403WED	403WED	403WED	403WED
403WED	1-Mar-99	4:17:27	407COAT	407STEP	409DEV	403WED	403WED	403WED	403WED	403WED	403WED

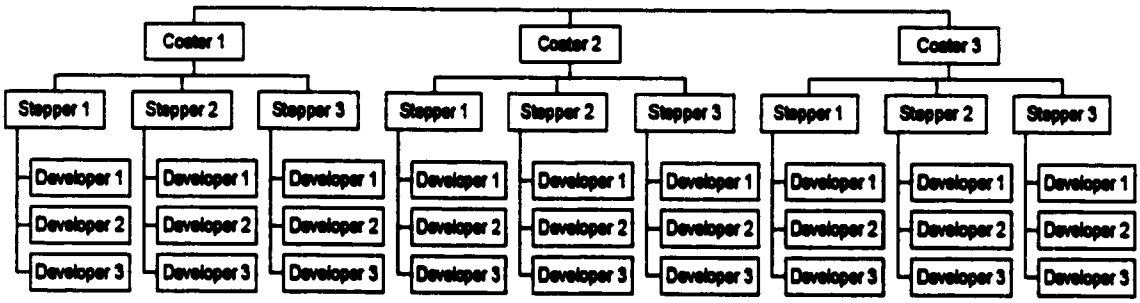
Since the processing equipment for each processing step is now contained in its own column, further sorting algorithms become manageable. Table 1 and Table 2 show the format of the CD data after the CHECK program has been run. This presorted table of CD data, that

CHECK created, is stored as the filename that the user entered. The CHECK program can be found in Appendix B.

Now that the data are in a format that can be sorted easily, they need to be sorted. The SPECTRAL program is the second RS1 program that is used in sorting the data into periodic data series. SPECTRAL uses the presorted output from the CHECK program as its input.

SPECTRAL sorts the CD data into unique processing paths. For the photolithography example with three coaters (C1,C2,C3), three steppers (S1,S2,S3) and three developers (D1,D2,D3), there are 27 different processing paths (eg., C1S1D1). Figure 8 shows the 27 different processing paths in the photolithography area.

Figure 8 The 27 Processing Paths in Photolithography



The SPECTRAL program then counts the number of batches that were processed with each processing path (eg. there might be eight batches that were processed through the processing path C1S1D1 in the time frame of the data extraction).

The SPECTRAL program then creates periodic data series using the sorted processing paths. Each data series is comprised of a number of repeating data sequences that relate to a specific type of processing equipment. For example, in the photolithography area there are three coaters; thus a data sequence for three coaters would be C1C2C3.

In order to accurately examine the effects of one piece of processing equipment, the processing paths in the data sequence can only differ by one (i.e., the processing equipment that one is examining). A data sequence in which the coater is being examined is shown in the following data sequence: C1S1D1,C2S1D1,C3S1D1.

The number of repeating periods in the data series is dependent on the amount of data in the data extraction. The size of the data extraction file is limited by the time period for the extraction. The time period for the data extraction is controlled by the user. As the time period for the data extraction increases, the number of batches, in the data extract, that were processed with each processing path increases. This is to say that, since you are trying to obtain enough data to create repeating data sequences, the data extraction should be sufficiently large such that there are multiple batches for each processing path. The time frame for such a data extraction depends entirely on the processing level being examined and how the production planning group has scheduled batches through the photolithography area. For example, if the production planning group started batches in such a way that, for a certain period of time, there were no batches at a particular processing level in photolithography. Thus, if an extraction was performed for this time period, for that particular processing level, the data extraction file would be empty. Consequently it is important to look at the size of the data extraction file to ensure sufficient data has been extracted before proceeding with further analysis.

In order to ensure that there are complete data sequences in the data series, the number of batches that were processed with each processing path are counted. The processing path with the least number of batches becomes the limiting data set when the processing paths are combined to make data series. This is why the minimum number of batches for the processing paths that are being compared, is the number of repeating data sequences that exists in the data series. For example, if there were eight batches that were processed with the data sequence C1S1D1, ten batches that were processed with C2S1D1 and four batches that were processed with C3S1D1, the largest data series that could be made from these processing paths would consist of four repeated data sequences.

Since we are trying to identify the existence of a processing problem as well as the cause of one, we need to examine all combinations of processing paths for periodic signals. In the photolithography example, this means analyzing 27 different processing paths. In order to simplify this task and reduce the amount of data required to analyze these paths , CD data is examined on the basis of processing equipment. This way, all of the possible periodic processing problems that could exist in the photolithography area are examined. This means that for each piece of equipment, there are nine processing paths to be examined and hence there are nine data series to analyze for periodic errors. For example, if we were to analyze the coater equipment set for periodic errors we would sort the CD data into nine data series where each data series contained one of nine periodic sequences shown below.

Sequence 1: C1S1D1,C2S1D1,C3S1D1
Sequence 2: C1S1D2,C2S1D2,C3S1D2
Sequence 3: C1S1D3,C2S1D3,C3S1D3
Sequence 4: C1S2D1,C2S2D1,C3S2D1
Sequence 5: C1S2D2,C2S2D2,C3S2D2
Sequence 6: C1S2D3,C2S2D3,C3S2D3
Sequence 7: C1S3D1,C2S3D1,C3S3D1
Sequence 8: C1S3D2,C2S3D2,C3S3D2
Sequence 9: C1S3D3,C2S3D3,C3S3D3

Similarly, there are nine periodic series to be analyzed from the stepper and nine periodic series to be analyzed from the developer. These data series are created individually in the SPECTRAL program and are the final sorted CD data that are required for periodic analysis.

The sampling plan directly effects the way that data are collected. It determines the number of sites that are measured per wafer, the number of wafers measured per batch and the number of batches that are measured in the production process. The sampling plan is not the same at all processing levels. This is mainly due to the difference in minimum feature sizes at each processing level. The smaller the minimum feature, the more difficult it is to make. This results

in more sampling on the levels that have smaller feature sizes. In the photolithography area, sampling plans range from one batch in five being measured to every batch being measured. Within that range, the number of wafers being measured varies from one to three wafers and five sites are measured per wafer.

With different sampling plans comes different periodic data sequences. For example when five sites are measured per wafer per batch, the sequence that compares 3 coater tracks would be repeated every 15 data points. The repeating sequence would look like this.

C1s1,C1s2,C1s3,C1s4,C1s5,C2s1,C2s2,C2s3,C2s4,C2s5,C3s1,C3s2,C3s3,C3s4,C3s5

Where C1s1 represents the CD datum that was measured at site 1 on a wafer that was coated with coater 1. C1s2 represents the CD datum that was measured at site 2 on a wafer that was coated with coater 1. The sequence if repeated would be periodic in two ways: every five data points the measurement location on a wafer is repeated, and every fifteen data points the processing equipment is repeated.

When the sampling plan involves more that one wafer, the repeating sequence is longer. For example, a sampling plan of 2 wafers per batch having five sites measured per wafer would have a repeating sequence of 30 data points. The sampling plan directly effects the periodicity of the sequence files and the possible periodic nature that could be observed in the modeling program.

The RPL sorting program creates a data sequence for each processing path comparison. This, depending on the data extraction, can create up to 27 different data sequences to analyze.

The sorting software CHECK and SPECTRAL is provided in Appendix B.

3.3 The Modeling Program Algorithm

The sorting technique used in this work was essentially the same technique Progler (1995) used. The modeling technique in this work differs from Progler's work because the phase information in the model is maintained. This simplifies the number of signals that need to be considered for the periodic model because any periodic signal can be modeled using a combination of impulse signals with various periods. All discrete signals can be built from combinations of the impulse signal (Oppenheim, 1997). Progler's two-impulse signal is only the resultant sum of two single impulses occurring within the same period. The same is true for a block signal. Since this approach maintains the phase information of the model's signal, more information as to the root cause of the process problem is available to engineering personnel. This information is very valuable because it gives the engineer more information about the root cause of the process upset. If for example, the model detected a signal that had a period of 5 and had an impulse at the first measurement location, the engineer would know where to look in the process for problems that could occur at that specific location. However if the model could only tell the engineer that there was a periodic upset occurring every 5^{th} point, they would need to do some more investigation as to what part of the wafer was being affected.

The modeling program takes one of the data sequences from the sorting routine and counts the number of CD measurements in the sequence. In the photolithography area, CD measurements are carried out a groups of 5. Since the CD measurements are recorded in the same order in which they are measured, the data sequences are all periodic in the measurement location. This means that when we are looking for periodic trends, we should always look for trends in data with a period of 5. This is the smallest periodic sequence that can be found in the data that relates to the process and explains why the model algorithm is set up to look for periods in increments of 5.

In order to create a generic modeling program for all of the photolithography processing levels, the modeling program considers all of the periodic sequences that could exist in a data

sequence. Since the minimum periodic sequence has a period of 5 and periodic sequences can only exist in multiples of 5, all of the multiples of 5 from 5 to 45 are tested in order. A period of 45 exists when three wafers are measured per batch and three pieces of equipment are compared. This is the largest sampling period used in photolithography.

The modeling algorithm selects the first of nine periods (period = 5, 10, 15, 20, 25, 30, 35, 40, 45) in which to evaluate the model. In order to determine if the data sequence should be examined for a particular period, the total number of points in the data sequence is divided by the suggested period. If the remainder is zero, the number of repeating periods is an exact multiple of the period and the selected period is used in determining a periodic model for the data sequence. If the selected period does not divide evenly into the total number of data points in the data sequence it is not considered in modeling the data sequence.

An example of a period that would be considered is a period of 10 and the total number of data points being 120. In the photolithography area a period of 10 could exist when two wafers are measured per batch at five sites per wafer. The 120 data points could be from10 measurements per equipment, across three pieces of equipment (period of 30) repeated four times.

An example of when the period would not be considered for the model would be if the selected period was 20 and the data sequence had 150 data points. In this case a period of 20 would not be considered because it does not divide evenly into 150.

This methodology only analyzes complete periods of data. The result of this is that not all of the data from the extraction are being used. Alternatives for filling in missing data points were considered, but the algorithm for determining these missing points could amplify non-periodic data to look periodic. In the case where only the real data are used, the resulting periodic signal model holds more merit.

After the modeling algorithm has selected an appropriate period with which to model the data sequence with and has calculated the number of repeating periods in the data sequence, the modeling program creates a series of impulse signals. For example, if the period was 5 and the number of repeating periods was 30, ten impulses would be created each of length 30. The first 5 impulse signals would have magnitudes of 0.1, while the next 5 impulses would have magnitudes of –0.1. Each impulse signal has a different phase in the period. The resulting impulse signals are shown in Table 3 . It is important to realize that these signals would be repeated 30 times for the example described above with 30 repeating periods.

Table 3 Impulse Signal with a Period of Five for Use in the Modeling Algorithm

1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th
0.1	0	0	0	0	-0.1	0	0	0	0
0	0.1	0	0	0	0	-0.1	0	0	0
0	0	0.1	0	0	0	0	-0.1	0	0
0	0	0	0.1	0	0	0	0	-0.1	0
0	0	0	0	0.1	0	0	0	0	-0.1

The magnitude of the impulse signal (0.1 and –0.1) is significant because it allows the model to detect both positive and negative periodic deviations from the CD target value.

The CD data are centered about the target value for the processing level. The target CD value is entered by the user at the beginning of the modeling program. This target value is subtracted from all of the data points. If for example all of the CD data points were exactly on target, the centered data sequence would be a series of zeros.

The magnitude of the data sequence, $|D_o|$, is calculated by squaring each of the values of the centered data sequence and adding them together. This original sum is stored for later comparison.

Each impulse signal has the same length as the signal to be modeled. The objective of the impulse signals is to determine if they are part of the original signal. This is done by subtracting each impulse independently from the signal to be modeled. If the sum of the

squared values of the difference signal, $|D_j|$, see equation (11), is less than the original sum, then the impulse is considered to be part of the original signal, see equation (11).

$$|D_j| = \sum_{i=1}^{N} (x_i - \nabla(i,j))^2 \qquad (11)$$

where x_i is the i^{th} value of the centered data sequence

$\nabla(i,j)$ is the i^{th} value of the j^{th} impulse vector

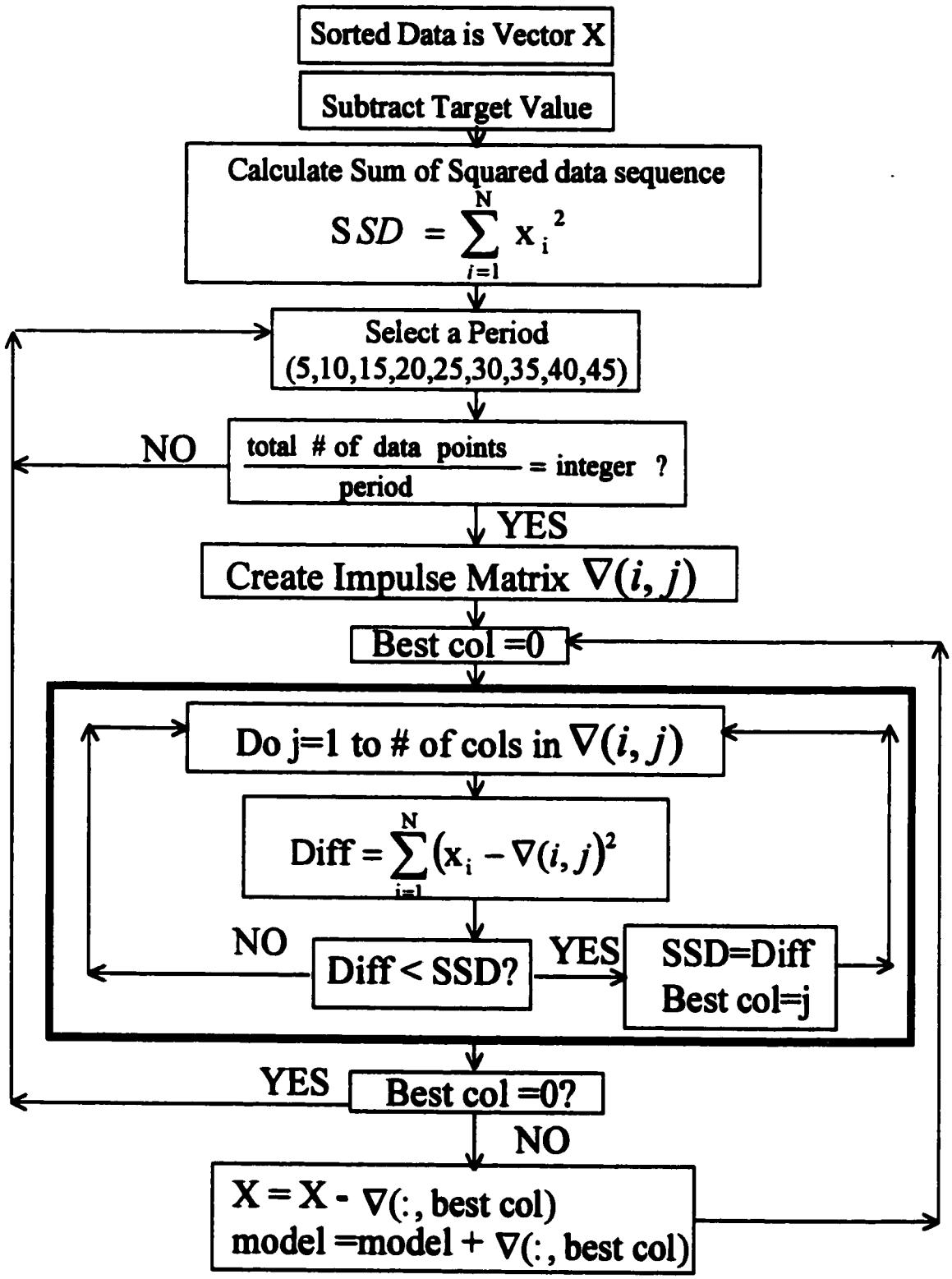
The magnitude of the difference signal is calculated for each impulse signal. The impulse signal that most reduces the magnitude of the impulse signal is added to the model for the data sequence and subtracted from the centered data sequence. This new data sequence is used for further modeling. This process is repeated until none of the impulse signals reduce the magnitude of the difference signal. When this happens a new period is chosen. The modeling algorithm is shown in Figure 9 as a flow sheet.

The magnitude of the impulse signal is important because it directly impacts the ability of a signal to be detected. If a periodic signal exists in the centered data sequence that is less than half of the magnitude of the impulse signal, it will not be detected. One of the issues surrounding the magnitude of the impulse signal is the degree to which random noise could be mistaken for a periodic signal. If the magnitude of the impulse signal is too small it is very easy to get false periodic signals, however if the magnitude of the impulse signal is too large, the model will not be sensitive enough to detect smaller periodic signals.

The modeling program can be found in Appendix C.

3.4 Simulations

The purpose of the simulation part of this work was to determine the sensitivity and repeatability of the modeling algorithm with different signal to noise ratios. A simulation



N = total number of data points

Figure 9 Modeling Algorithm Flow Chart

program (see Appendix D) was written in MATLAB that used the same modeling algorithm as the modeling program. The modeling algorithm was applied multiple times at different noise levels to determine how many times out of the total number of simulations that the correct signal was detected.

The simulation program first creates a periodic signal. This is the signal that the modeling algorithm tries to detect. The actual data signal that the modeling algorithm sees contains the original signal plus a specific level of random noise. The number of times that the original signal is correctly detected is recorded as a fraction of the total number of simulations that were carried out. This testing process was repeated for each period (i.e. 5, 10, 15, 20, 25, 30, 35, 40, 45) that the modeling program considers. The length of the data series, namely the number of repeating periods, was also changed to determine its effect on the algorithms ability to determine the correct signal.

In order to determine how effective the modeling program was at detecting periodic signals, it was tested without noise with various different signals.(i.e. signals with different magnitudes, different types of signals, signals with different periods and signals with different numbers of repeating periods). The modeling algorithm performed perfectly in all cases. Since the type, size and magnitude of the original signal did not seem to affect the modeling algorithms' ability to detect the correct signal, only one type of signal was chosen to be used for the simulations. This signal was an impulse signal, such that the impulse was located at the first observation in each period. An example of such an impulse signal can be seen in Figure 3.

To studying the effect of signal to noise ratio, the magnitude of original signal that the algorithm was trying to detect, was kept constant. Only the magnitude of the noise was changed.

The random number generator 'randn' in the MATLAB program was used to create normally distributed random numbers with a mean of zero and a standard deviation of one. These randomly generated values were scaled by the desired standard deviation for each particular

simulation. These noise values were then added to the signal under study to produce simulated CD measurements.

3.5 Spectral Analysis

Spectral analysis is a methodology for detecting periodic trends in time series data. It is based on Fourier analysis but with a modification so that a stationary stochastic time series can be studied. Fourier analysis breaks down time series into frequency components of variance. The variance contributions are best viewed in a plot called a periodogram.

Before initiating a discussion about spectral analysis, it is a good idea to review a few basics. The relationship between frequency and period (or wavelength) is fundamental to the comprehension of the following discussion. Studying data for periodic trends is the same as examining the frequency at which the data is periodic. This inverse relationship is shown in equation (12).

$$f = \frac{1}{T} \qquad (12)$$

Sine and cosine are probably the most familiar periodic functions. The Fourier theorem states that any "well behaved" periodic function x(t) with period p, can be expressed as a sum of cosine and sine functions. This is known as a Fourier series. The Dirichlet conditions are the restrictions on the "well behaved" functions and they require the function, x(t) to have a finite number of maxima and minima, a finite number of discontinuities in the range over which it is specified in and be absolutely integratable. Equation (13) shows a Fourier series representation of a periodic function x(t),

$$x(t) = \sum_{p=0}^{N} \left[a_p \cos(\omega_p t) + b_p \sin(\omega_p t) \right] \qquad (13)$$

The a_p and b_p terms in equation (13) are called Fourier series coefficients. The angular frequency of the periodic variation, ω_p, has the units of radians per time unit. In order to have

a better physical meaning of frequency, ω_p is divided by 2π to give frequency the units of cycles per unit time. This relationship is shown in equation (14).

$$f = \frac{\omega_p}{2\pi} \qquad (14)$$

In most of the theoretical development of spectral analysis angular frequency will be used, but it is important for the reader to understand and know the difference between angular frequency and frequency.

Since a summation of cosine and sine terms can represent any periodic function, they are used as the building blocks in Fourier analysis. The objective is to find the coefficients and frequencies that best model the periodic data series x(t).

The data we are considering are discrete. Thus we need to use a discrete Fourier series so that x(t) is only defined on the integers 1, 2, ... N. N is the total number of observations in the data series. An important aspect in modeling a discrete periodic time series comes from the fact that two exact models, with different frequencies can both accurately model the same data set. Consider the models cos(0.5ω) and cos(0.25ω) and a sampling interval of π shown in Figure 10. Three data points from a periodic signal x(ω) result in 2π. The data points are $(0,1),(\pi,1)$ and $(2\pi,1)$. Both models describe the data set exactly and are said to be aliases with the current sampling interval. However, if the sampling interval was increased to $\pi/2$, the data set would contain two additional data points ($\pi/2$, 1) and ($3\pi/2$,1). This additional information would reveal cos(0.25w) to be the correct model. This shows the benefits of frequent sampling.

Aliases have common periods and thus can be indistinguishable from one another at a particular sampling interval. The best way to picture this common period is with a graph of the cosine function. Figure 10. shows how changing the frequency of the cosine function affects the period of the cosine function. The cos (ω) function has a frequency of 1 and a period of

2π. The function cos(0.5w) has a frequency of 0.5 and a period of 4π. The cos(0.25ω) has a frequency of 0.25 and a period of 8π.

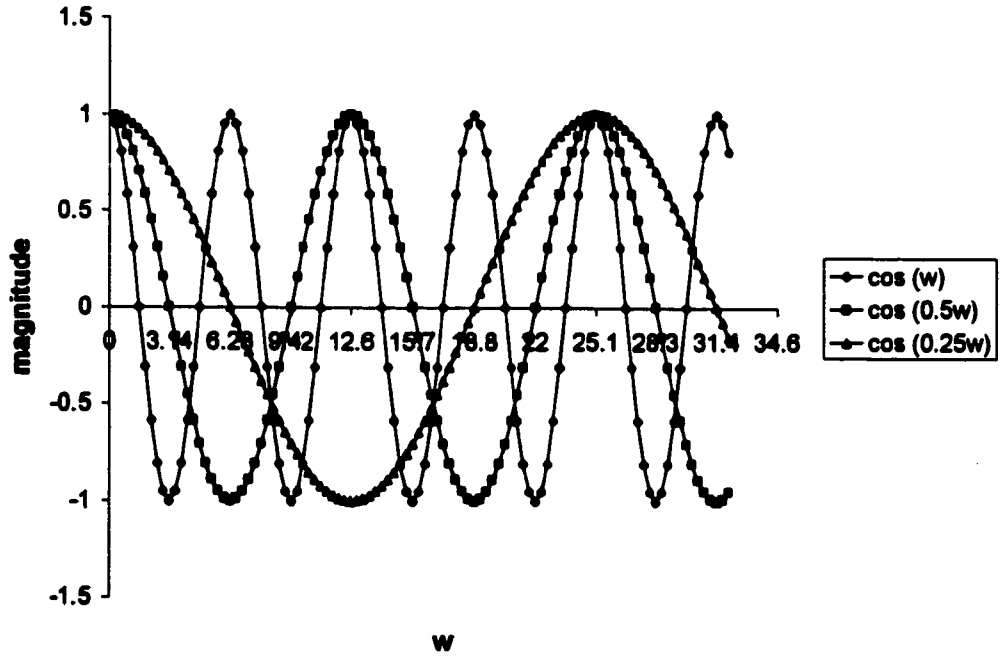


Figure 10 Cosine Plot showing Signal Aliasing

For a discrete process, there is a finite number of frequencies that exist within a data set. This is best explained through the use of an example. If you were to measure the air temperature at noon, once a day, every day for a month, you would have approximately 30 measurements (depending on the month). Since the temperature is only measured once a day, there is no way to tell the temperature at any other time of day, nor is it possible to detect any temperature cycle that occurs within a day. The fastest temperature cycle that could be observed requires two days of data. In other words, the fastest cycle that could be measured is a two day cycle. This requires 3 data points. Usually the starting point is not numbered, so the second and third data points are called the first and second observations. The maximum frequency, or shortest cyclic period, that could be detected in a discrete data set is 1 cycle per 2 observations or ½. This maximum frequency, in angular frequency, is called the Nyquist frequency.

$$f_N = \frac{1}{2} \longrightarrow \omega_N = \frac{2\pi}{2} = \pi \qquad (15)$$

The formal definition of Nyquist frequency is shown in equation (16), where Δt is the sampling interval.

$$f_N = \frac{\pi}{\Delta t} \qquad (16)$$

Consider the slowest frequency that could be modeled from a data set. Again consider the month of air temperature data, the slowest frequency we could observe would be one cycle in the entire data set of 30 observations or 1/30. We would not be able to obtain monthly trends because we only have one month of data. The number of data points in the data series is directly related to the maximum cycle length that can be studied. The slowest frequency that can be observed is thus 1/N or 2π/N in angular frequency.

If there is a finite number of observations in a discrete data set, there is a finite number of frequencies that can be observed in that data set. Since integer values are required for a discrete time series, the angular frequency can be thought of as.

$$\omega_p = \frac{2\pi p}{N} \qquad (17)$$

Recognizing the limits on frequency to be 1/N and π,the integer values that p can take on are p=1,2,..., N/2. Using this information and rewriting equation (13) yields equation (18).

$$x(t) = a_0 + \sum_{p=1}^{N/2-1}\left[a_p \cos\left(\frac{2\pi p t}{N}\right) + b_p \sin\left(\frac{2\pi p t}{N}\right)\right] + a_{N/2}\cos(\pi t) \qquad (18)$$

When the function x(t) has a period of p, the first term in the series is a_0.The second term in the series (r =1) has the cosine and sine terms with the fundamental period p. The third term in the series (r =2) has the cosine and sine terms with a period of p/2. This frequency is called the second harmonic. The fourth term in the series (r =3) represents cosine and sine terms with a

period of p/3 and is called the third harmonic and so on. The important thing to note is that all cosine and sine terms whose period is an integer fraction of p will also have a period of p.

There is no loss of generality in restricting the frequency domain to the range (0, π). This is because all other integer frequencies have aliases in the frequency range (0,π). This can be seen in equation (19)

$$\begin{aligned} \cos(\omega t + k\pi t) &= \cos(\omega t) \qquad k, t \text{ integers with } k \text{ even} \\ \cos(\omega t + k\pi t) &= \cos(\pi - \omega)t \quad k, t \text{ integers with } k \text{ odd} \end{aligned} \tag{19}$$

The Fourier series coefficients are given in equation (20)

$$\begin{aligned} a_0 &= \bar{x} \\ a_p &= \frac{2}{N}\left[\sum x_t \cos\left(\frac{2\pi pt}{N}\right)\right] \quad \text{for p} = 1,\ldots,\text{N/2} - 1 \\ b_p &= \frac{2}{N}\left[\sum x_t \sin\left(\frac{2\pi pt}{N}\right)\right] \quad \text{for p} = 1,\ldots,\text{N/2} - 1 \\ a_{N/2} &= \frac{\sum(-1)^t x_t}{N} \end{aligned} \tag{20}$$

3.5.1 Periodogram

The objective of the periodogram is to break down the variance of the data set into frequency components. This way we can analyze how much variation is being caused by the N/2 frequencies.

We are looking for periodic trends in our data set. We are expecting no trends, and thus the sum of square errors represents the total corrected sum of squared errors,

$$|D_o| = \sum_{t=1}^{N}\left(x_t - \bar{x}\right)^2 \tag{21}$$

The periodic model for the frequency w_p is given by

$$x(t) = \sum_{t=1}^{N} \left(\hat{a}_p \cos \omega_p t + \hat{b}_p \sin \omega_p t \right) \qquad (22)$$

Since the expected value of the data series is zero, after the data has been adjusted by the target value, the frequency model component of the total corrected sum of squared errors is given by equation (23)

$$|D| = \sum_{t=1}^{N} \left(\hat{a}_p \cos \omega_p t + \hat{b}_p \sin \omega_p t \right)^2 \qquad (23)$$

Equation (23) can be simplified with the four following trigonometric identities.

$$\begin{aligned}
&\sum_{t=1}^{N} \cos \omega_p t \cos \omega_p t = N && \text{for p = N/2} \\
&\sum_{t=1}^{N} \cos \omega_p t \cos \omega_p t = N/2 && \text{for p} \neq \text{N/2} \\
&\sum_{t=1}^{N} \sin \omega_p t \sin \omega_p t = 0 && \text{for p = N/2} \\
&\sum_{t=1}^{N} \sin \omega_p t \sin \omega_p t = N/2 && \text{for p} \neq \text{N/2}
\end{aligned} \qquad (24)$$

If you expand equation (23), you have

$$|D| = a_p{}^2 \sum_{t=1}^{N} \cos \omega_p t \cos \omega_p t + 2 a_p b_p \sum_{t=1}^{N} \cos \omega_p t \sin \omega_p t + b_p{}^2 \sum_{t=1}^{N} \sin \omega_p t \sin \omega_p t \qquad (25)$$

which by using equation (24) gives

$$\begin{aligned}
|D| &= \left(a_p{}^2 + b_p{}^2 \right) N/2 && \text{for p} \neq \text{N/2} \\
|D| &= a_p{}^2 N && \text{for p = N/2}
\end{aligned} \qquad (26)$$

Equation (22) can also be represented by

$$a_p \cos(\omega_p t) + b_p \sin(\omega_p t) = R_p \cos(\omega_p t + \phi_p) \qquad (27)$$

where R_p is the amplitude of the p^{th} harmonic and can be calculated from

$$R_p = \sqrt{\left(a_p^2 + b_p^2\right)} \qquad (28)$$

The phase of the p^{th} harmonic ϕ_p is calculated from

$$\phi_p = \tan^{-1}\left(-b_p/a_p\right) \qquad (29)$$

The contribution of the p^{th} harmonic (where $p \neq N/2$) to the total sum of squared error is

$$|D_p| = \frac{N}{2}\left(a_p^2 + b_p^2\right) = \frac{NR_p^2}{2} \qquad (30)$$

The periodic component of the total sum of squares for one frequency (ω_p) is shown in equation (23). The contribution of all N/2 frequencies to the sum of squared error is

$$|D| = \sum_{t=1}^{N}(x_t - \bar{x})^2 = \sum_{p=1}^{N/2}\sum_{t=1}^{N}\left(\hat{a}_p \cos\omega_p t + \hat{b}_p \sin\omega_p t\right)^2 \qquad (31)$$

In this way equation (31) can be represented, using equations (26) and (28) as

$$|D| = \sum_{t=1}^{N}(x_t - \bar{x})^2 = N\sum_{p=1}^{N/2-1}\frac{R_p^2}{2} + Na_{N/2}^2 \qquad (32)$$

The left hand side of equation (32) looks very much like the equation for variance, with the exception of a denominator of (N-1).When N is very large, N and N-1 are very close, thus dividing both sides of equation (32) by N, shows the relationship between the p^{th} harmonic and its contribution to the variance of the data series.

$$VAR(x) = \frac{\sum_{t=1}^{N}(x_t - \bar{x})^2}{N} = \sum_{p=1}^{N/2-1}\frac{R_p^2}{2} + a_{N/2}^2 \qquad (33)$$

where $R_p^2/2$ is the actual contribution of the pth harmonic to the total variance, for p=1 to N/2-1, and $a_{N/2}^2$ is the contribution of the harmonic where p=N/2.

Since we are looking for ways to examine the variance contribution of each discrete frequency we can plot $R_p^2/2$ versus $\omega_p = 2\pi p/N$. This gives a line spectrum. Since most time series have

continuous spectra it becomes inappropriate to plot a line spectrum, instead we use a histogram-like plot where we consider $R_p^2/2$ as the contribution to the total variance in the range $\omega_p \pm \pi/N$. The height of each segment of the histogram is given by

$$\therefore \text{Height of Histogram } I(\omega_p) = \frac{\text{area of pth element}}{\text{width of pth element}} = \frac{\frac{R_p^2}{2}}{2\pi/N} = \frac{NR_p^2}{4\pi} \qquad (34)$$

As usual, equation (34) only applies for p=1 to N/2-1. When $\omega_p=\pi$, $a_{N/2}^2$ is the contribution of the $N/2^{th}$ harmonic. This plot of $I(\omega_p)$ versus ω is called a periodogram even though $I(\omega_p)$ is a function of frequency rather than period. Other authors define the periodogram plot slightly differently, but the advantage of the definition described in equation(34) is that the total area under the periodogram is equal to the total variance of the time series.

Equation (34) can be calculated directly from the time series using

$$I(\omega_p) = \frac{\left(\sum_{t=1}^{N/2} x_t \cos\left(\frac{2\pi pt}{N}\right)\right)^2 + \left(\sum_{t=1}^{N/2} x_t \sin\left(\frac{2\pi pt}{N}\right)\right)^2}{N\pi} \qquad (35)$$

It is important to note that equation (35) is valid for p=1 to N/2.

3.5.2 Applying the Periodogram

A primary requirement for the approach studied in this work was the ability to determine if a given data sequence contained a periodic signal. Two programs were written in MATLAB to perform this analysis. The first program uses a minimization technique to determine a periodic model for the data and the second program creates a periodogram to show how the variance of the data is distributed over the frequency range. Information provided by these programs is critical for the success of the proposed tool in diagnosing periodic trends in data.

The periodogram describes the variance distribution in the frequency domain. If the data were sorted such that 5 data points from each of three coaters (Coater A, Coater B and Coater C) were being analyzed over a finite period of time, the data points would be ordered in a periodic way such that the sequencing order would repeat every 15 data points.

A1,A2,A3,A4,A5,B1,B2,B3,B4,B5,C1,C2,C3,C4,C5,A1...

If there were a significant deviation from a target associated with the first measurement on wafers processed with Coater A (i.e. A1) there would be a spike in the periodogram at a frequency of 1/15 reflecting the periodic occurrence of the deviation every 15 measurements. (i.e., period =15).Thus when examining the periodogram, if there were a frequency peak every 1/15 we would know that the data sequence was periodic with a period of 15.

In order to determine where the periodic effect is coming from, the data is modeled in the time domain using a first program. From the examination of the periodogram, some information is already known about the periodic nature of the data. This information can be used to model the signal. Once a model has been created in the time domain, it can be subtracted from the original data sequence and another periodogram can be made. If the new data sequence has a periodogram that closely resembles white noise, the model is considered to adequately describe the data. Unfortunately, this same pattern on the periodogram could arise from deviations occurring repetitively at any one the three coaters. For example, a deviation occurring for the fifth measurement from a wafer processed on coater B (i.e. B5) would produce the same periodogram as discussed above. This inability to determine the exact location of the upset results from the lack of any phase information in the periodogram. Only information about the magnitude of the Fourier transform is presented in the periodogram. If the periodogram still looks as though there is periodic behavior in the data the model can be modified and the procedure repeated until the periodogram resembles that for white noise.

3.6 Testing the Methods on the Real Process

An experiment was carried out to intentionally inflate the critical dimensions of a batch of wafers. Since only two wafers are normally measured per batch, the batch was divided into sublots and entered into the database as separate batches.

Twelve wafers were intentionally over-exposed so as to increase the critical dimensions measured on the twelve wafers. The idea was to cause a shift in CD data that would not be caught by the control charts. The 12 wafers were divided into 6 batches of 2 wafers in PROMIS. The CD measurements were entered into PROMIS as part of the data collection operation procedure after they were measured on the SEM. Unfortunately, the exposure that was recommended for use was from an older exposure energy correlation chart and the exposure value shifted the critical dimension values such that they were flagged in the control charts.

Since these 12 processed wafers represented a significant amount of measuring time on the SEM, the experiment was continued in order to determine the effectiveness of the modeling algorithm and the periodogram with real process data.

Another experiment was carried out to examine real process data. Regular CD data extracted a period of 60 days using the PROMIS script file SEM_ERIN. This data extract was sorted and then analyzed using the model building algorithm and the periodogram.

CHAPTER 4 RESULTS AND DISCUSSION

The purpose of this section is to present and discuss the results of this work. The layout of this chapter closely follows that of Chapter 3 in that the data extraction results are discussed in the first section, the data sorting results are discussed in the second section and the simulation results are shown and discussed in the third section.

In the final section of this chapter the results of investigations using real data from the photolithography process at Mod IV are presented. Two types of real data were analyzed. The first set of real data comes from the experiment described in section 3.6. The second set of real data comes from a data extraction of regular CD data from the photolithography process. Both data sets are analyzed with the modeling algorithm and the periodogram.

4.1 Data Extraction

The PROMIS data base is very difficult to work with. There are many field sorting parameters that do not come with adequate explanations or comments. In fact some of the sorting fields have no explanation at all. This lack of documentation makes it very difficult to use PROMIS to its full capability. In order to avoid working from the manual menus in PROMIS, many engineering personnel have developed scripts to perform routine data extracts and

calculations. These scripts are highly valued in the PROMIS community because of the time they save the PROMIS user.

The script that was written for the data extraction part of this thesis, namely SEM_ERIN.SCR, is very useful because it links the lot with the processing equipment that was used to process the lot. Previously when a user wanted this information, they would have to enter each line in the SEM_ERIN.SCR script on a command line. Not only is this time consuming but the user would also have to know exactly how to link and sort the files.The SEM_ERIN. SCR script is very useful because now engineering personnel can find the processing equipment for each processing level for multiple batches without knowing the exact sequence of commands.

The data extraction script can be found in Appendix A.

There may be some problems with the script output file in the future if the number of processing steps increases or the sampling plan decreases. This is due to the way the processing equipment is stored in the output file. The number of elements in the column vector associated with each lot is set by the number of CD measured on that lot. If the number of measurements per lot is less than the number of processing steps at that level, the output file is uncertain. Currently, as long as the number of equipment processing steps (three) is less than the number of sites measured per lot (five) the script file should work.

4.2 Data Sorting

The CHECK program uses the output from the SEM_ERIN.SCR script to presort information about the equipment tracks. The user is queried about the general equipment process path for the extracted data set (eg. What is the first processing equipment (coater, stepper, developer)?). The CHECK program screens and deletes the lots containing

nonstandard processing paths. All of the equipment names for the coater, stepper and developer equipment sets are part of the CHECK program. If any equipment names change, or if new equipment is added the CHECK program needs to be updated. Despite efforts to keep the sorting software simple and compact the CHECK and SPECTRAL programs could not be combined due to the size limitation of the text editor for the RS1 software. Efforts were also made to keep maintenance of the software to a minimum. A copy of both the CHECK program and the SPECTRAL program can be found in Appendix B.

The SPECTRAL program was written so that up to seven processing steps could be sorted. The current photolithography application requires only three processing steps (coater, stepper, developer) but this built-in flexibility means that this sorting algorithm could be used in other areas of Mod IV or for an expanded process flow in photolithography. This flexibility is a very important feature to NORTEL and it should increase the likelihood of this methodology being used. If the program is used in other areas of Mod IV, the PROMIS script file and the CHECK program will have to be modified. As long as the processing equipment flow is in separate columns, the SPECTRAL program will work. This type of flexibility was built into the SPECTRAL program so that other process flows, besides that of the photolithography area, could be analyzed with a minimum amount of work.

The spectral program was tested for its sorting abilities throughout its development, and found to be problem free.

4.3 Simulation Results

The purpose of the simulations was to determine the modeling algorithm's ability to detect a periodic signal in the presence of normally distributed random noise.

In order to gain a better understanding of the number of simulations that would be required to have repeatable simulations results a mini-experiment was carried out. This mini-experiment

used a signal to noise ratio (S/N) of 100:1 and varied the number of runs from 25 to 100 simulations for four repeating periods of simulated data.

Table 4 shows the percentage of the simulations for which the modeling algorithm detected the correct underlying signal.

Table 4 Percent of Correct Signals Detected for S/N=100:1 and Four Repeating Periodic Sequences

Period	25 runs	30 runs	35 runs	50 runs	50 runs	100 runs	100 runs	100 runs
5	100%	100%	100%	100%	100%	100%	100%	100%
10	100%	100%	100%	100%	100%	100%	100%	100%
15	100%	100%	100%	100%	100%	100%	100%	100%
20	100%	100%	100%	100%	100%	100%	100%	100%
25	100%	100%	100%	100%	100%	100%	100%	100%
30	100%	100%	100%	100%	100%	100%	100%	100%
35	92%	86.7%	77.1%	84%	80%	87%	87%	87%
40	100%	100%	100%	100%	100%	100%	100%	100%
45	100%	100%	100%	100%	100%	100%	100%	100%

The top row of Table 4 indicates the number of simulations that were used to calculate the average percentage of correct signals detected. For example, the first element indicates that a signal with a period of 5, repeated 4 times was simulated 25 times. For those 25 simulations, the correct signal was detected 100% of the time. Except for the signal having a period of 35, the correct signal was detected 100% of the time regardless of period or number of simulations. The true value for the percentage of correct signals detected by the simulation algorithm should be the same regardless of the number of times the simulation is run. This means that the values observed for the period of 35 should be the same. As the number of simulations was increased, the true value was thought to be obtained. Since each of the 100 run cases all resulted in the same percentage of correct signals detected, namely 87%, it was thought that 100 simulations should be used for future reliability simulations.

After some of the simulations had already been run, we wanted to be able to calculate confidence limits. In order to do this, the 100 simulations were carried out in 5 sets of 20 simulations. The average of the 20 simulations was recorded 5 times. This simulation reporting scheme provides an estimate of variability in the simulation. This, however, reduced the accuracy for the average measurement because instead of averaging 100 simulations, we were now averaging 20 simulations. The result is that average values were now recorded in increments of 5%.

In order to determine if a correct signal was detected the estimated signal model was subtracted from the original (known) signal. The original signal in this case refers to the original signal that does not contain any noise. The elements of this difference vector were squared and added together. If this sum was less than 10^{-5} the signal model was deemed to be correct. The value 10^{-5} was chosen as the selection criterion because of the magnitude of the impulse in the impulse vector was 0.1 (see Table 3) and the original signal is 10 times this value. In this way, an exact model could be found by the algorithm. If the signal model differed from the original known signal by one increment in just one of the elements of the difference vector the resulting difference vector would contain a value of 0.1. Since the algorithm squares the difference vector and sums the values in the vector, a single difference of 0.1 would result in a sum of 0.01. When this minimum difference value was being considered, it was thought that an incremental impulse signal with a magnitude of 0.01 might be used. In this case, the sum of the squared difference vector, would be 10^{-4} for a single element difference. Since this difference could occur with an impulse of 0.01, the difference criteria was set to be 10^{-5}. This criteria is very conservative for the impulse of 0.1, but it effectively identifies signals that are perfectly matched.

A problem regarding the amount of UNIX server time became apparent when other computer users began complaining about not being able to access the UNIX CPU server because it was being occupied 100% with simulations. These simulations, depending on the number of periods used, usually took 2 to 3 days to complete. Since this was significantly impacting other UNIX users on the same server, it was necessary to reduce the number of simulations.

The simulations were reduced to 3 runs of 20 simulations. In order to ensure that there was not a significant amount of information lost in this reduction of simulation runs a 95% confidence interval based on 3 runs of 20 simulations and 6 runs of 20 simulations was calculated. These confidence intervals, along with the mean values from the simulations are shown in Figure 11.

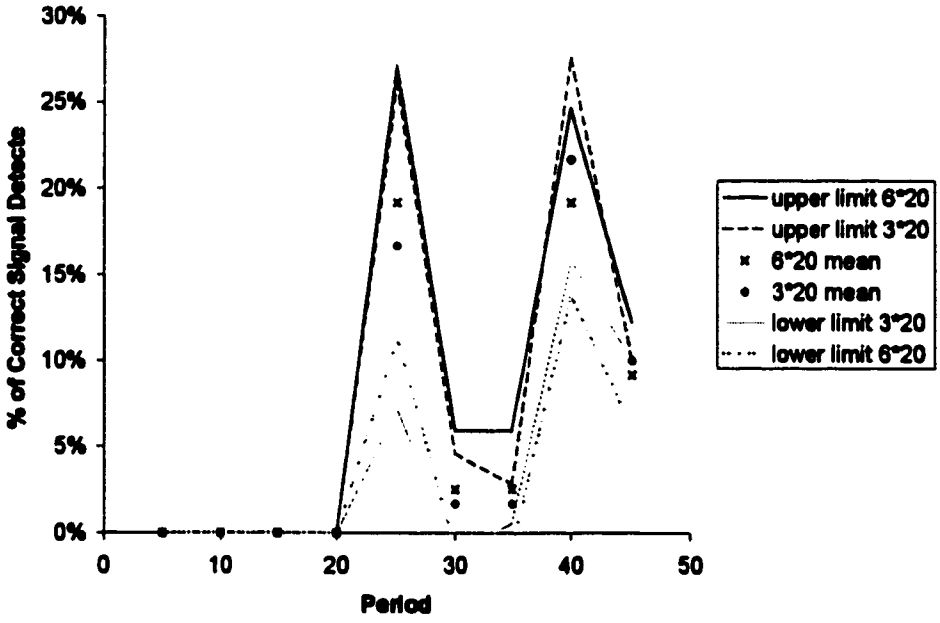


Figure 11 Signal:Noise Ratio 20:1 with 95% Confidence Limits for 4 Repeating Periods

One apparent anomalous point occurred in Figure 11. The confidence limits and mean value for the three runs of 20 simulations at a period of 45 coincided since the three runs produced identical percentages resulting in a zero value for the standard deviation. This was largely due to the limited resolution (±5%) when calculating an integer fraction of twenty. The trend of this chart is typical of other signal-to-noise simulation results, which are shown in Appendix E.

Figure 12 and Figure 13 depict typical simulation results for the relationship between the number of repeating periods and the percentage of correct signals detected when the signal to noise ratio is held constant. As the number of repeating periods increases, the percentage of correct signals detected also increases. These results are based on the average for 20

simulations repeated three times and agree with the literature description of weak signals and the difficulty in detecting them. As the number of repeating periods is increased, the signal is stronger and the detection capabilities are improved.

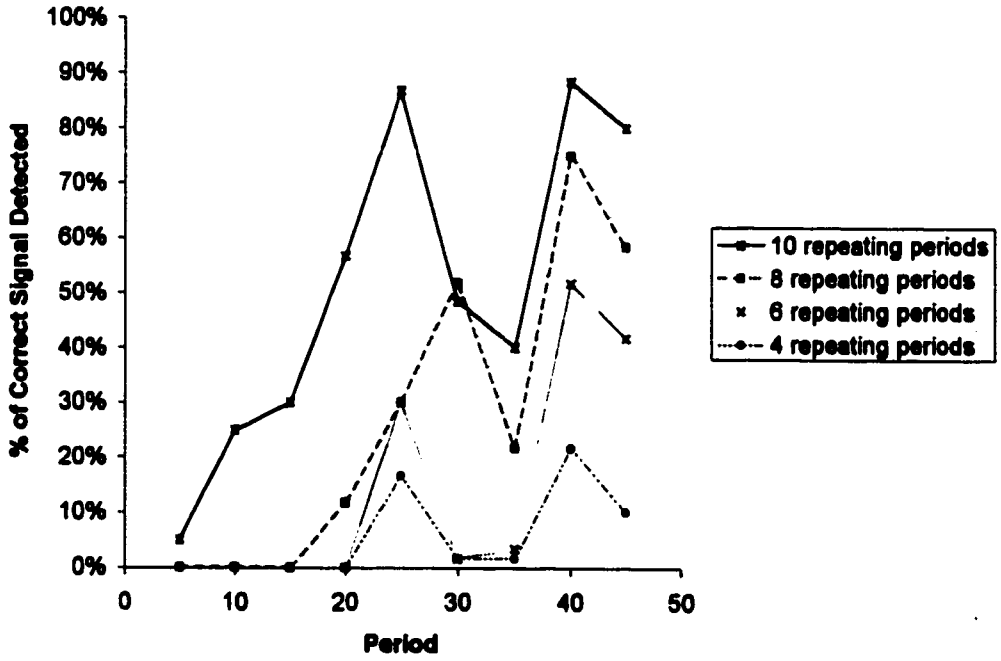


Figure 12 Signal Detection Capabilities based on 20 simulations repeated 3 times with S/N=20 for 4 to 10 Repeating Periods

The other important influence on the modeling algorithm's ability to detect the correct signal is the size of the signal to noise ratio (S/N). Figure 14 depicts typical simulation results for the relationship between the size of the S/N and the percentage of correct signals detected when the number of repeating periods is held constant. These results are based on the average for 20 simulations repeated three times and agree with the literature description of weak signals and the difficulty in detecting them. As the S/N is increased, the signal is stronger and the detection capabilities are improved. The other simulation results can be found in Appendix E.

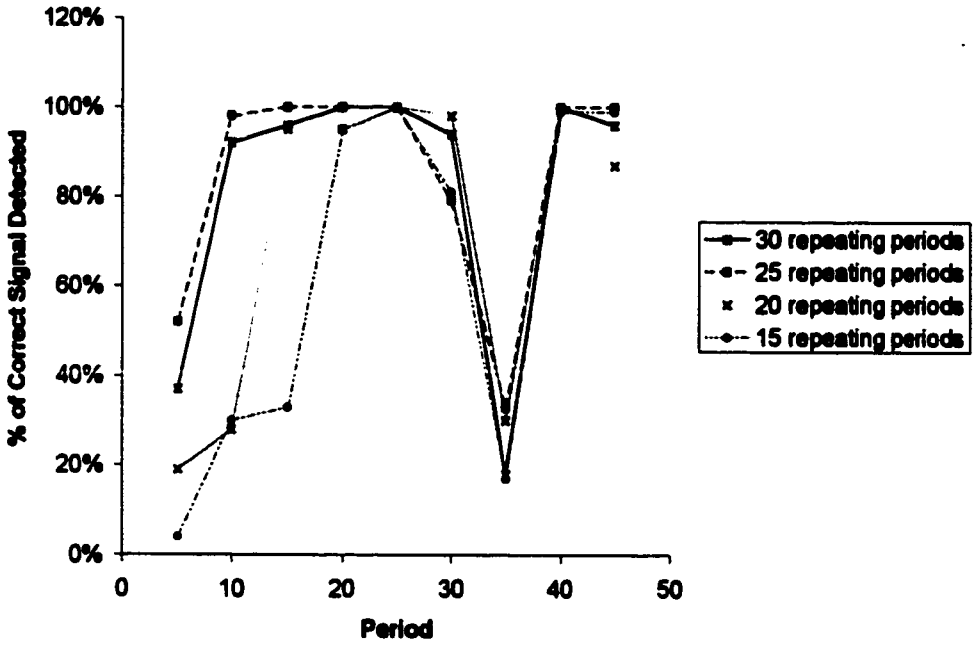


Figure 13 Signal Detection Capabilities based on 20 simulations repeated 3 times with S/N=20 for 15 to 30 Repeating Periods

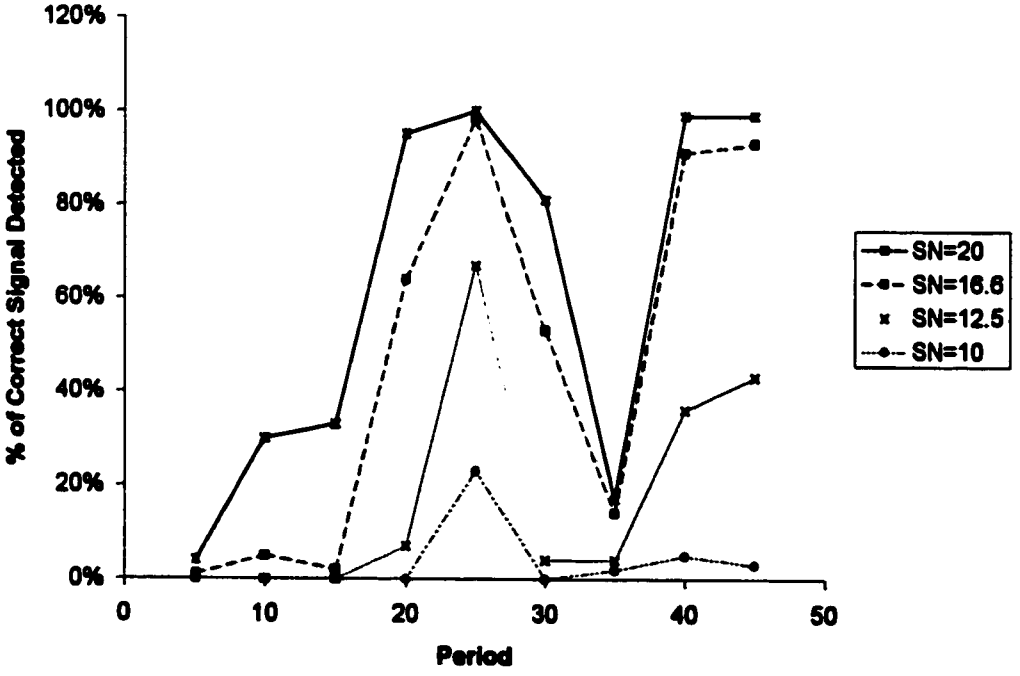


Figure 14 The Signal to Noise Ratio Effect on Data with a Period of 15

The ability of the algorithm to detect a correct signal seemed to be influenced by the period of the underlying signal. This effect was not expected and a series of tests were carried out to find an explanation for this behavior.

Initially the random number generator was suspect. The way in which the random number generator was used was changed when the number of iterations was changed from 5*20 simulations to 3*20 simulations. Initially the MATLAB function 'randn' was used. The seed value for the 'randn' random number generator is set when you begin a session of MATLAB. It is very unlikely that the random number sequence identically matches the periodic simulations that were run, but to eliminate this possibility, the random number generator was changed to choose a different seed value at the beginning of each simulation. This seed value references the computer clock to ensure a random seed. After this change was made, the size of the period still seemed to influence the ability of the algorithm to detect a periodic signal.

The next idea was to examine the number of times the modeling algorithm tried to model a data series. The only way a data series is examined for a periodic signal is if the period to be examined is an integer factor of the total number of data points in the data series. For example, a data series with a period of 35 repeated four times has 140 data points. There are four periods in the modeling algorithm that divide evenly into 140. (i.e. 5, 10,20 and 35). ying signal.

Table 5 shows the number of periods that the modeling program would try to use to model the data series with. Consider the case for four repeating periods, if the number of factors (i.e., the number of wrong periodic signals that are tested in the modeling algorithm +1) influenced the ability for the correct signal to be detected, the best percentage of correct signals detected should be the data series with the least number of factors. This is not the case.

To better examine the number of factors and their influence on the modeling algorithm, a series of alternative simulations were run. These simulations were carried out to determine if the wrong periodic signals were responsible for reducing the percentage of correct signals

detected. In order to do this, the simulation scheme was altered so that only the underlying period of the known (original) signal was used in the modeling algorithm. The results were surprising. As the size of the period increased, the modeling algorithms ability to detect the underlying signal was reduced. Initially it was expected that the size of the period would not affect the ability for the modeling algorithm to detect the correct underlying signal.

Table 5 The number of Factors for Repeating Periods

	# of repeating periods			
size of period	4	6	8	10
5	3	4	4	3
10	4	5	4	4
15	5	5	6	5
20	4	6	4	5
25	4	5	5	3
30	6	6	6	6
35	4	5	5	4
40	4	6	4	5
45	6	5	7	6

Figure 15 and Figure 16 show the results for these alternative simulations. Figure 15 shows the result of increasing the number of repeating periods while keeping the S/N constant. Figure 16 shows the results for reducing the S/N while keeping the number of repeating periods constant. The underlying reason for this behavior seems to be the average strength of the signal. For example an impulse of magnitude one in a period of five has an average strength of 0.2, while an impulse with the same magnitude but with a period of 20, has an average strength of 0.05. This effect is overcome as the number of repeating periods is increased and the strength of the signal comes a certain threshold signal strength. This threshold is the number of repeating periods where the size of the period no longer affects the modeling algorithms ability to detect the correct signal. The threshold value may be defined in two ways. The first threshold value may be defined as the minimum number of repeating periods where the upper control limit contains a value of 100% for the percentage of correct signals detected for each size of period. The second type of threshold value may be more strictly defined as the minimum number of repeating periods where the average percentage of correctly detected signals is 100%. Table 6 shows both definitions of threshold number of

repeating periods for various signal to noise ratios. The reader is reminded that these simulation results are based from the alternative simulation scheme, where only the real underlying period of the signal is used in the algorithm to detect the original signal and thus are not readily observable in the results of the regular simulation algorithm.

In order to test the threshold concept, the results of regular simulations were examined again. More specifically, two data series with the same S/N were compared. The first data series contained more than the threshold number of repeating periods while the other data series contained less than the threshold number of repeating periods. Confidence limits for these two data series were calculated and can be seen in Figure 17. Using the confidence limits for the 10 repeating periods and the confidence limits for the 4 repeating periods, it can be seen that there is a significant difference between these detection percentages for periods 5, 10, and 15. This confirms the idea of a threshold signal strength because there is a significant difference in the two modeling algorithms' abilities to detect these periodic signals.

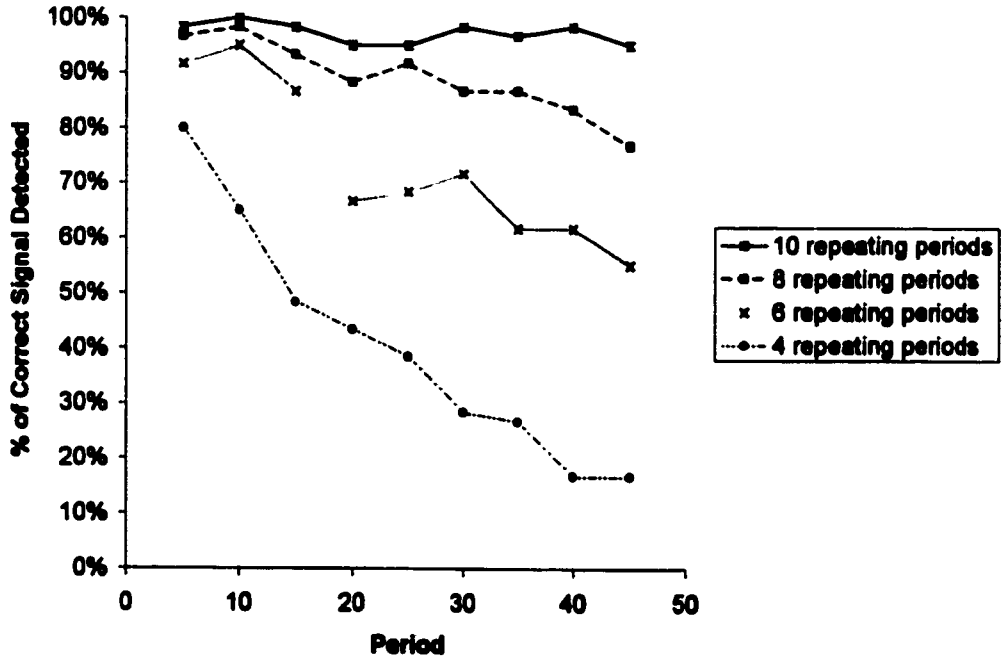


Figure 15 The Influence of the Period on the Algorithm for Signal to Noise Ratio of 20:1

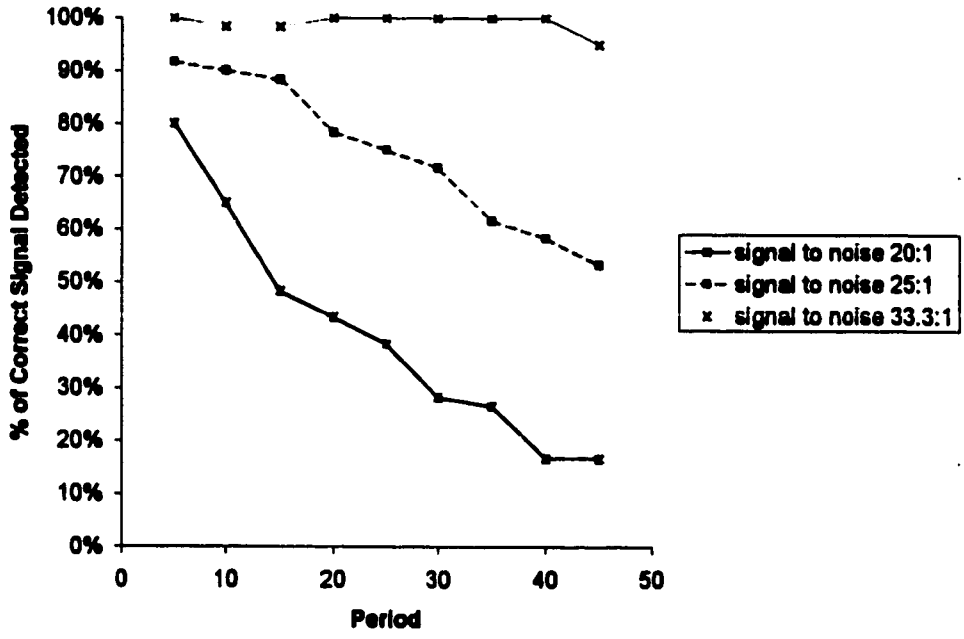


Figure 16 The Effect of the Signal to Noise Ratio on the Algorithm for 4 Repeating periods

Table 6 Threshold Number of Periods

	# of repeating periods	
Signal to Noise Ratio	Upper Confidence Limit	Mean
33.3:1	4	6-8
25:1	8	12-14
20:1	10	20-25

All of the above discussion does not explain why periodic signals with periods of 5,10 and 15 are hardly ever detected when the number of repeating periods is low. The reason that periodic signals with periods of 5, 10 and 15 are rarely detected with a low number of repeating periods (eg. 4) is because the total number of data points in these cases is a period that is examined in the model. For example, when a period of five is repeated four times, there are 20 data points in the series. When the modeling algorithm checks to see if this data series can be modeled with a period of 20 the algorithm says "yes". This allows each individual data point in series to be modeled individually. All that is required for the modeling algorithm not

to detect the correct signal is if the random error in the simulation is sufficiently close to the magnitude of the impulse such that by subtracting in impulse (0.1 or –0.1) the sum of squared difference vector is reduced. In order to fix this problem a minimum number of repeating periods should be added to the algorithm such that, if a period is going to be used for modeling purposes, there will be a minimum number of repeating periods to model.

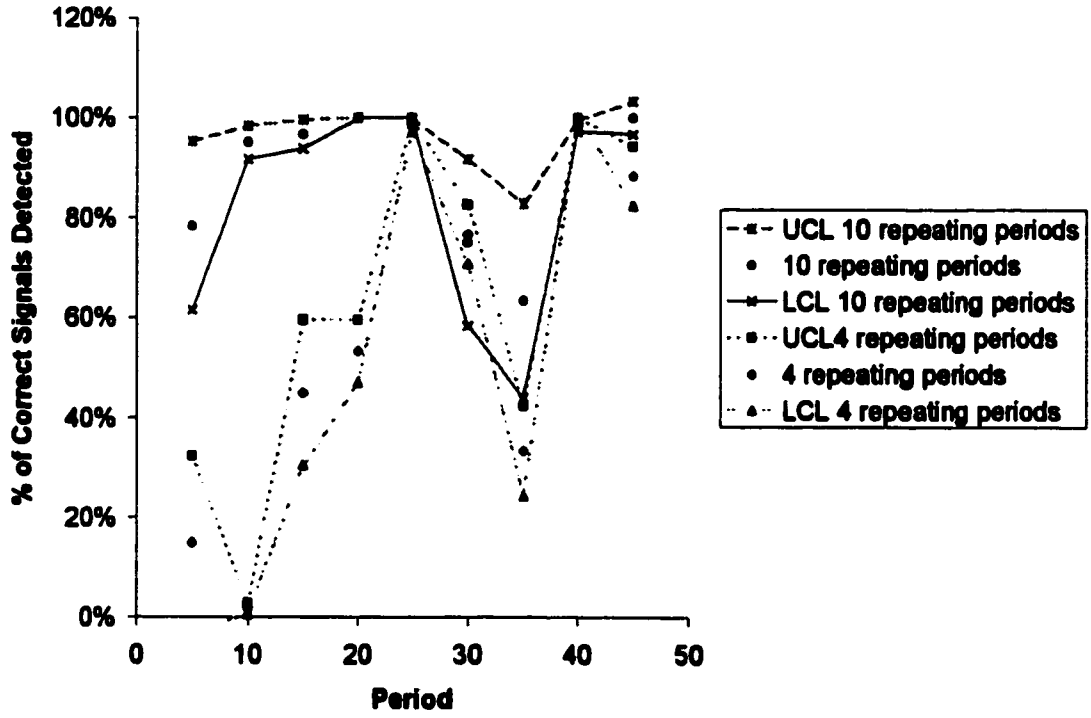


Figure 17 The Effect of the Threshold Number of Repeating Periods

When the idea of the modeling algorithm was conceived, a minimum number of repeating periods was a planned part of the modeling algorithm. It wasn't until the modeling peculiarities were fully examined that it was discovered that the minimum number of repeating periods was not a part of the algorithm.

The modeling algorithm may not be optimal but it does detect periodic signals in the presence of noise. These simulations can provide guidelines for the amount of data required to detect certain levels of signals, which was the purpose of this of this work. It should also be noted

that the main problem with post threshold trends shows that the modeling algorithm has difficulty in detection periods of 35. In the photolithography process, with the current sampling plan, it is not possible for the process to create a data sequence that is periodic with a period of 35.

4.4 Real Process Tests

The purpose of the real process tests was to ensure that the modeling algorithm works for real process data in the same way it worked for simulated data.

4.4.1 Testing with a Known Signal

An experiment was carried out such that twelve wafers were purposefully misprocessed in the photolithography area. The CD values of these wafers were entered into PROMIS as six lots of production wafers. The sampling plan for the processing level was two wafers per batch and five measurements per wafer. The purpose of this experiment was to induce a periodic block signal in real process data.

The data extraction and sorting methods were used to extract and sort the data. The data were then modeled with only six repeating periods. Note that the experimental data came from only one of the three processing paths compared in the data sequence; the other CD data were regular CD data from production.

Table 7 shows the modeling results for the experiment. The column entitled 'induced signal' shows the signal that was intentionally created by the experiment. The column entitled signal model shows the model predictions calculated by the modeling algorithm for the real data and the column entitled 'average data' is the average value of the six real data points for each phase in the period (i.e. with a period of 30 and 6 repeating periods the average phase value was calculated by averaging every 30^{th} data point.) For example, the first phase averages came from averaging the 1^{st}, 31^{st}, 61^{st}, 91^{st}, 121^{st}, and 151^{st} data points.

These results show that the model did identify the induced block signal. It is interesting that other signals were detected in this experiment. The 2nd , 3rd and 8th points had additional observations of 0.1. These observations represent a periodic deviation occurring on the

Table 7 Real Data Model

		3 path model	
	induced signal	signal model	actual signal
1	0.1	0.1	0.128
2	0.1	0.2	0.165
3	0.1	0.2	0.162
4	0.1	0.1	0.144
5	0.1	0.1	0.134
6	0.1	0.1	0.134
7	0.1	0.1	0.147
8	0.1	0.2	0.160
9	0.1	0.1	0.142
10	0.1	0.1	0.140
11	0	0	0.001
12	0	0	0.005
13	0	0	0.007
14	0	0	0.013
15	0	0	-0.021
16	0	0	-0.010
17	0	0	0.007
18	0	0	-0.002
19	0	0	-0.002
20	0	0	0.000
21	0	0	-0.014
22	0	0	0.000
23	0	0	0.000
24	0	0	-0.003
25	0	0	-0.013
26	0	0	-0.008
27	0	0	-0.004
28	0	0	-0.011
29	0	0	-0.020
30	0	0	-0.011

second and third measurement on the first wafer of the first processing path (2nd ,3rd point) and the third measurement on the second wafer for the first processing path (8th point). The only periodic error that would be reasonable to expect would be that of the 3rd and 8th

measurement points. This would represent a consistent deviation from target associated with the 3rd measurement on each wafer that was processed with the first processing path.

Since this diagnostic tool was designed to aid an engineer in determining the root cause of CD data variability, it would be very difficult to explain how some periodic models in the data are more likely to be real than others. The sampling scheme in the photolithography area for how to chose two wafers from a batch is random. This is the reason that there should be no first wafer or second wafer effects. It is distinctly possible that the selection of two wafers is not random and the operators are always choosing the wafers in the same order. If there was a wafer to wafer effect, the modeling program would detect it as a first or second wafer effect. But if we are to believe that the wafers are chosen randomly across the batch there should be no reason for a wafer effect to exist. In any case the modeling algorithm coupled with process knowledge has pointed to possible problems in the process.

An important point to recognize is that the knowledgeable engineer, understands the sampling plan of the process and can identify the periodic nature of the process and the corresponding data. That type of training and knowledge cannot be incorporated into a general modeling program. However, it is possible for a specific modeling program to be set up and specialized to look at specific process and sampling conditions. This would mean that only certain periods would be considered thereby focusing the search on likely causes and those periodic effects that have a reasonable chance of being detected when limited amounts of data were available.

It is interesting that with only 6 repeating periods that the algorithm was able to detect the correct periodic signal for a signal to noise ratio of approximately 5:1. According to the simulation results, with only 6 repeating periods this signal would not be detectable. This just demonstrates that the modeling algorithm works well at detecting periodic data trends and although the signal that is detected may include more than the underlying signal, it points the engineer in the right direction.

The periodogram results for this experiment are shown in Figure 18 ,Figure 19 and Figure 20. The real data from the experiment was used to create the periodogram shown in Figure 18. Figure 19 shows the periodogram for the data after the induced model (see Table 7) has been subtracted from the observed data. Figure 20 shows the experimental data set after the fitted model has been subtracted. After subtracting the fitted model from the measured data a reduction in the variance of the data was observed. This confirms the model from the modeling algorithm. This periodogram program helps the engineer in two ways: it provides a visual representation of the variance distribution of the data and a way to experiment with different signal models to determine their effect on the variance distribution of the in the data set.

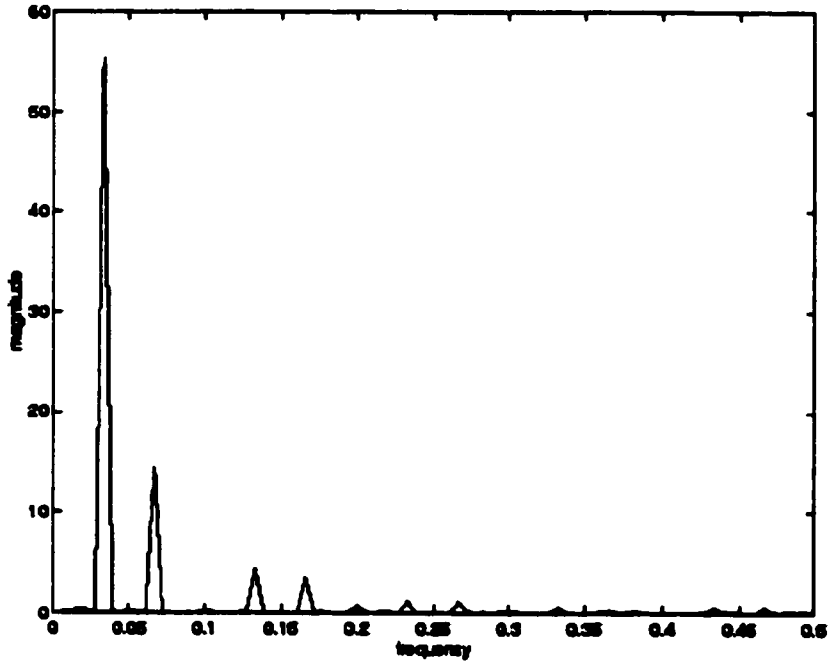


Figure 18 Periodogram for Real Data Experiment

The shape of Figure 18 closely resembles the Fourier transform of the block signal shown in equation (6). This is easy to detect only because there is one predominant signal in the data. The other observation to be made about Figure 18 is the peak around the value 0.0333. This is due to the signal having a period of 30.

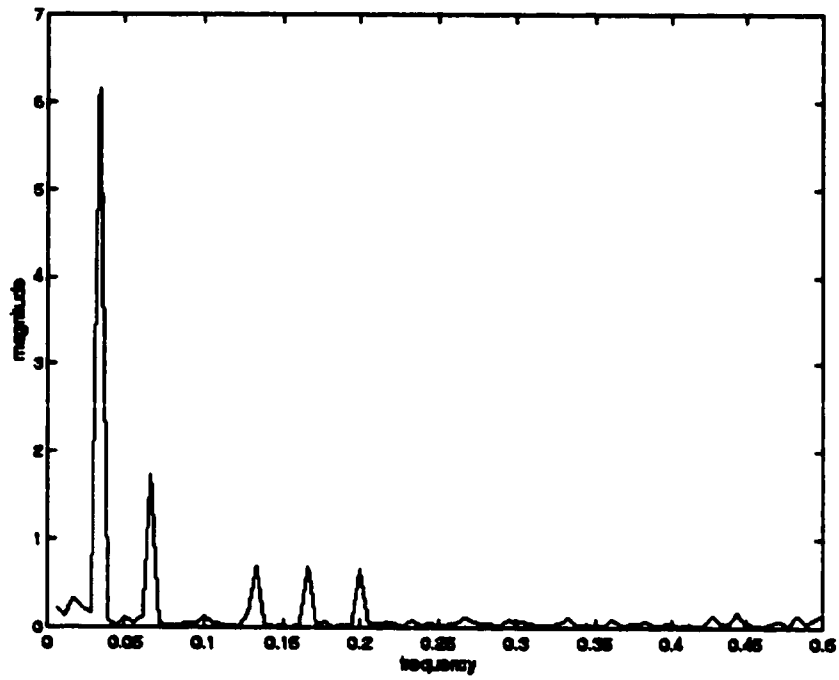


Figure 19 Periodogram for Real Data with the Induced Model Subtracted

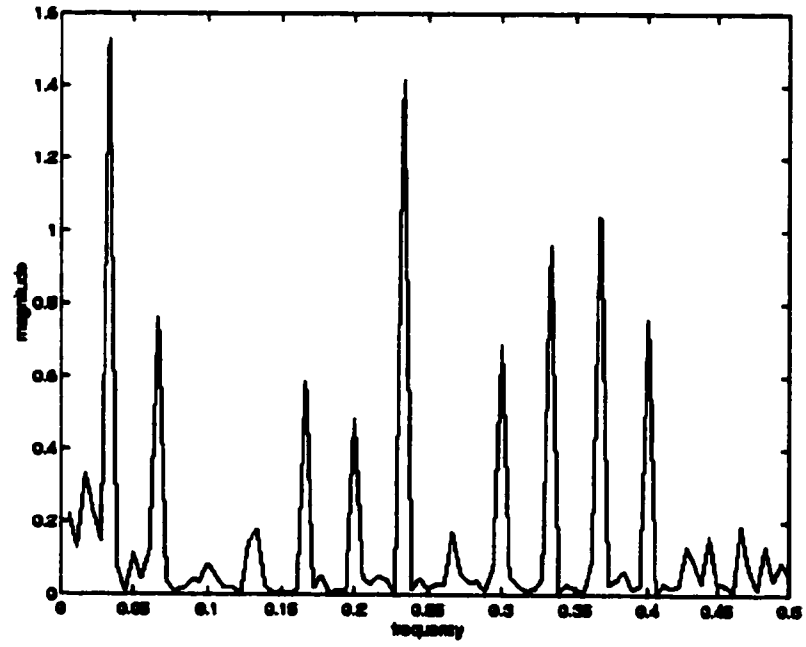


Figure 20 Periodogram for Real Data with the Predicted Model Subtracted

Figure 19 shows a great reduction in variance from Figure 18. The reader is reminded to look at the scale of the periodogram when comparing Figure 18 to Figure 19.There is still a

periodic signal present in Figure 19. That signal is the difference between the induced model and the signal model presented in Table 7. The sum of the squared data series was initially 1.33 (i.e. where only the target is subtracted from the data). The sum of the squared data series after the induced model was subtracted was 0.185. The sum of the squared data series was further reduced to 0.140 when the modeled signal was subtracted from the data series in Figure 18. Figure 20 shows a periodogram where only noise is present, this is characterized by low magnitude values and the spiky nature of the data series as plotted in the frequency domain.

4.4.2 Testing with Historical Process Data

In order to determine if any periodic behavior is occurring during the normal operation conditions in the photolithography area, a data extract was carried out for a period of 60 days. This is the same period of time that a CD measurement remains on a control chart. The modeling algorithm did not detect any periodic behavior, but only two periods of CD data were available from a 60 day data extraction. This result shows the biggest limitation of this modeling tool, that is, the amount of data required to have a sufficient number of repeating periods. The periodogram for this extract is shown in Figure 21. The magnitude of the variance in this data set is even less than that of the previous data set, confirming the findings of the modeling algorithm that no periodic signal is present in this data set.

If there was a periodic error within the photolithography data set, it would have a signal to noise ratio of greater than 3:1. This is because the standard deviation for the control chart is 0.019 and the difference between the upper control limit and the target is 0.06. If the signal to noise ratio was any less, the control chart would detect an out of control signal. The number of repeating periods required to detect this signal is greater than 60 because at a signal to noise ratio of 3:1 the simulation algorithm could not detect any signals with only 60 repeating periods. It is not reasonable to extract 60 periods of CD data from the photolithography area, because it represents about a year or more of data.

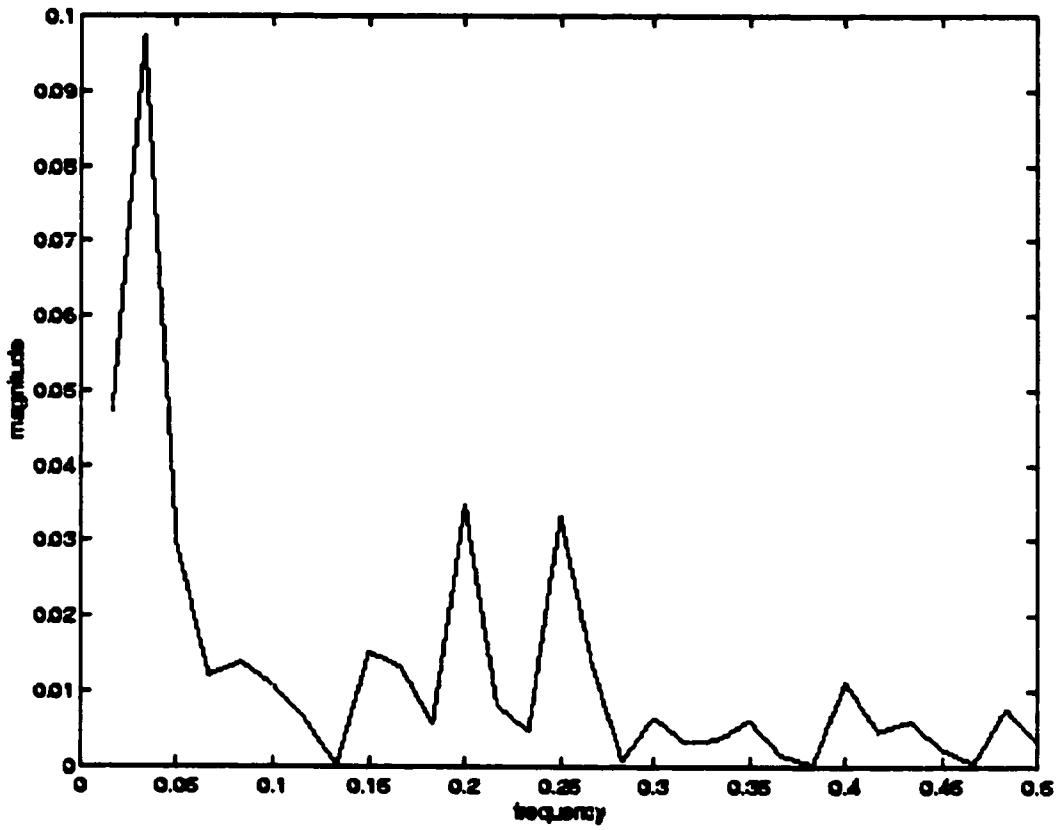


Figure 21 Periodogram for Regular Process CD Data

CHAPTER 5 CONCLUSIONS & RECOMMENDATIONS

5.1 Conclusions and Contributions

1. A data extraction and sorting technique was constructed to create data files for periodic analysis. The data extraction file will be useful to other PROMIS users.

2. A model building algorithm was created that successfully detects the underlying periodic problem. This algorithm was found to work for both simulated and real data sets.

3. A periodogram program was created to provide engineering personnel a way to visualize the variance distribution in a given set of sorted data. The spectral analysis tool is an effective way of determining the root cause of periodic variation within the photolithography process.

4. Stochastic simulations demonstrated the effectiveness of the modeling algorithm to detect underlying signals with different signal to noise ratios. These simulations can be used as a reference in the future.

5. The greater the number of repeating periods in the periodic series, the greater the likelihood of detecting an existing periodic signal. A large amount of data is required to

detect a signal with a small signal to noise ratio.(e.g., For a signal to noise ratio of 25:1 there should be at least 8 repeating periods)

6. The modeling algorithm has greatest difficulty detecting a period of 35. Fortunately a period of 35 can not occur within the current sampling process in the photolithography data.

5.2 Recommendations

The modeling algorithm should be changed such that a minimum number of repeating periods is required before the data series can be modeled.

The threshold number of repeating periods should be studied further in order to confirm their real effect on the modeling algorithms ability to detect the correct signal.

Another photolithography experiment should be carried out at a smaller signal to noise ratio such that the induced signal falls within the control limits of the processing level.

Further investigations should be carried out to determine why a periodic signal with a period of 35 is more difficult to detect with this modeling algorithm.

A less strict sorting algorithm should be investigated, one in which the data series is only sorted on one piece of processing equipment and the sequence of CD data would not necessarily come from the same type of processing path. This might alleviate the limiting data problem but it will also create an increased level of the noise in the sorted data.

CHAPTER 6 REFERENCES

Auslander, L., Buffalano, C., Orr, R., and Tolimieri, R.,. "A Comparison of the Gabor and Short-Time Fourier Transforms for Signal Detection and Feature Extraction in Noisy Environments", SPIE Proceedings of the Advanced Signal-Processing Algorithms, Architectures, and Implementations, 1348, 230-247 (1990).

Beneke, M., Leemis, L.M., Schlegel, R.E., and Foote, B.L., "Spectral Analysis in Quality Control: A Control Chart Based on the Periodogram", Techometrics, 30,63-70 (1988).

Chatfield, C., "The Analysis of Time Series: An Introduction", Chapman & Hall, Great Britain (1996).

Jenkins, G.M., and Watts, D.G. "Spectral Analysis and Its Applications, Holden-Day, San Francisco (1968).

Nelson, L.S. "The Shewart Control Chart – Tests for Special Causes", Journal of Quality Technology, 17,114-116 (1984).

Oppenheim, A.V., Willsky, A.S., and Nawab, S.H., "Signals & Systems", Prentice Hall, New Jersey, (1997).

Progler, C.J. "Systematic Analysis of CD Data" Interface '95 Proceedings for the OCG Microlithography Seminar, October 29-31 1995 San Diego, CA, 57-73 (1995).

Ryan, T.P., "Statistical Methods for Quality Improvement", Wiley, New York (1989).

Yeung,G.K., and Gardner, W.A., "Search-Efficient Methods of Detection of Cyclostationary Signals", IEEE Transactions on Signal Processing, 44, 1214-1223 (1996).

Appendix A

The PROMIS Data Extraction Script

Written with the help of Sandra Copp and Patrick Burns

```
!This data extract was written by Erin Chapman
!
define DCOP
! Enter the DCOP (ex. 4AS07.11)
@
!
!
define STARTTIME
! Enter STARTTIME (ex. -7, 19-AUG-1999)
@
!
!
define ENDTIME
! Enter ENDTIME (ex. --, now, 21-MAY-1998)
@
!
!
set nooutput
!
! EXTRACT engineering data
!
quit
data
datalink
engineering
%DCOP
y
%STARTTIME
y
%ENDTIME
all
n
y
SYS$LOGIN:ERIN1
tres
EQPID
enttime 4
end
hist
stage
```

```
end
end
B
end
B
YES
NO
!
! removing duplicate Lotids - create ERIN2
!
quit
data
datalink
remove
SYS$LOGIN:ERIN1
trailing
LOT_ID
end
SYS$LOGIN:ERIN2
!
!extraction of equipment id - create ERIN3
!
quit
data
datalink
general
hist
SYS$LOGIN:ERIN3
lotid
stage
eqpid
trackintime 4
end
B
lotid
fr
SYS$LOGIN:ERIN2
LOT_ID
LOCATION = 4PHOTO
TRACKINTIME GE %STARTTIME
```

```
end
B
YES
NO
!
!sorting ERIN1
!
quit
data
datalink
sort
SYS$LOGIN:ERIN1
LOT_ID
A
stage
A
end
no
SYS$LOGIN:ERIN1
!
!sorting ERIN2
!
so
SYS$LOGIN:ERIN3
lotid
a
stage
a
end
no
SYS$LOGIN:ERIN3
!
!Joining ERIN1 and ERIN3
!
join
SYS$LOGIN:ERIN1
SYS$LOGIN:ERIN3
LOT_ID
stage
end
```

```
yes
none
lotid
stage
n
none
yes
SYS$LOGIN:ERIN4
!
! create print file to send to RS1
!
quit
data
datalink
print
SYS$LOGIN:ERIN4
no
SYS$LOGIN:ERIN4
```

Appendix B

The RPL Data Sorting Programs

Written by Erin Chapman

Available by request from

Dr.David McLean

University of Ottawa

Department of Chemical Engineering

CHECK

```
Procedure;
/* WARNING THIS PROGRAM SHOULD ONLY BE CALLED ONCE */

/* this procedure reads the erin4.prt file */

tablename=GETTEXT("Please enter a FILENAME (that DOES NOT exist in your
user directory) ");

$EZ_READFILE("sixtydays1.txt", tablename);

del row 0 of table(tablename);
del row 1 of table(tablename);

del col 3 of table(tablename);
del col 9 of table(tablename);


/* coater names */

coat[1]="406coat";
coat[2]="407coat";
coat[3]="409coat";
numcoat=3;

/* stepper names */

stepp[1]="407step";
stepp[2]="408step";
stepp[3]="409step";
numstep=3;

/* developer names */

dev[1]="406dev";
dev[2]="407dev";
dev[3]="409dev";
numdev=3;

lotid=1;
stage=2;
compid=3;
sitecol=4;
cddata=5;
equipmeasure=6;
dateofmeasure=7;
timeofmeasure=8;
processequip=9;
trackindate=10;
trackintime=11;

lastcol=lastcol(tablename);
lastrow=lastrow(tablename);

/* calculates the number of sites being measured */

sites = 1;
do i=1 to lastrow(tablename);
```

```
  if (col(sitecol) row (i) of table(tablename) LT col(sitecol) row (i+1))
    sites=sites+1;
  else
    doexit;
end;

/*  calculates the number of wafers in a batch */

wafers = 1;
do i=1 to lastrow(tablename) by sites while(col(lotid) row(i) of
table(tablename) eq col(lotid) row(i+sites) of table(tablename));
  wafers=wafers+1;
end;

/* finding out the processing flow (coat - coat - step - dev) */

equip[1]=GETRESPONSE("What is the first process step (coater, stepper, or
developer?", TRUE, "coater", "stepper", "developer");

processingsteps=1;
do I=2 to sites*wafers;
  if yesanswer ("Is there another processing step?", EMPTY) then
    begin;
      equip[I]=GETRESPONSE("next type of processing( coater, stepper or
developer)?", TRUE, "coater", "stepper", "developer");
      processingsteps=processingsteps+1;
    end;
  else
    doexit;
end;

 /* determines the line number of reworks */

m=1;
do I=1 to lastrow(tablename) by (sites*wafers);
  do j=1 to processingsteps;
    match="no";
    /* the processing equipment is thought to be coater */
    if (equip[j]="coater")
      begin;
        do k=1 to numcoat;
          equipmatch=row(I+j-1) col (processequip) of table(tablename);
          cmatch=$comparestr(coat[k],equipmatch);
          if (cmatch=4)
            begin;
              match="yes";
              doexit;
            end;
        end;
        /* the process equipment should be coater but it wasn't */
        if (match="no")
          begin;
            rework[m]=I;
            m=m+1;
            doexit;
          end;
      end;
    else
      begin;
        if (equip[j]="stepper")
```

```
          begin;
            do k=1 to numstep;
              equipmatch=row(I+j-1) col (processequip) of table(tablename);
              cmatch=$comparestr(stepp[k],equipmatch);
              if (cmatch=4)
                begin;
                  match="yes";
                  doexit;
                end;
            end;
            if (match="no") then
              begin;
                rework[m]=I;
                m=m+1;
                doexit;
              end;
          end;
        else
          begin;
            if (equip[j]="developer")
              begin;
                do k=1 to numdev;
                  equipmatch=row(I+j-1) col (processequip) of
table(tablename);
                  cmatch=$comparestr(dev[k],equipmatch);
                  if (cmatch=4)
                    begin;
                      match="yes";
                      doexit;
                    end;
                end;
                if (match="no")
                  begin;
                    rework[m]=I;
                    m=m+1;
                    doexit;
                  end;
              end;
          end;
      end;
  end;
end;
numreworks=m-1;

/* deleting reworks */

if (numreworks > 1)
  begin;
    del rows (rework[1]) to (rework[1]+sites*wafers-1) of table(tablename);
    do m=2 to numreworks;
      rework[m]=rework[m]-(m-1)*sites*wafers;
      del rows(rework[m]) to (rework[m]+sites*wafers-1) of
table(tablename);
    end;
  end;

/* putting process equipment into separate columns */

lastcol=lastcol(tablename);
lastrow=lastrow(tablename);
```

```
firstequipcol=lastcol+1;

do r=1 to lastrow by (sites*wafers);
  p=lastcol+1;
  do I=1 to (sites*wafers);
    k=r-1+I;
    col (p) row (r) of table(tablename) = col (processequip) row (k) of
table(tablename);
    p=p+1;
  end;
end;

do r=1 to lastrow by sites*wafers;
  do k=1 to sites*wafers-1;
    do p=firstequipcol to lastcol(tablename);
      col (p) row (r+k) of table(tablename)=col (p) row (r) of
table(tablename);
    end;
  end;
end;


/* end of program */
end;
```

SPECTRAL

```
procedure;

tablename = gettable("table to use: ");
lotid   = getnumber("column containing lotid data: ", FALSE,1);
compid  = getnumber("column containing component id data: ", FALSE,3);
sitecol = getnumber("column containing site numbers: ", FALSE,4);
cddata  = getnumber("column containing cd data: ", FALSE, 5);

/* This routine sorts the coater, stepper, and developer equipment ids.
It uses the SEM_DATA script output file. */

coatercol=getnumber("column containing coater data: ", FALSE,12);
steppercol=getnumber("column containing stepper data: ",FALSE,14);
developercol=getnumber("column containing developer data: ",FALSE,15);

if yesanswer("Is there aquatar processing in this level? ", FALSE)
  then begin;
    aquatarcol=getnumber("column containing aquatar data: ",FALSE,13);
  end;
else
  aquatars=1;
  if yesanswer ("Is there more data you want to include in this analysis?",
FALSE) then
    begin;
      xonecol= getnumber("column containing extra1 data: ",True);
      if yesanswer("Is there still more data you want to use? ", FALSE) then
        ztwocol= getnumber("column containing extra2 data: ",true);
      else
        ztwos=1;
    end;
  else
    begin;
      xones=1;
      ztwos=1;
    end;

/* calculates the number of sites */

sites = 1;
lastrow = lastrow(tablename);
do i=1 to lastrow(tablename);
  if (col(sitecol) row (i) of table(tablename) LT col(sitecol) row (i+1)) then
    sites=sites+1;
  else
    doexit;
end;

/*  calculates the number of wafers in a batch */

wafers = 1;
do i=1 to lastrow(tablename) by sites while(col(lotid) row(i) of table(tablename)
eq col(lotid) row(i+sites) of table(tablename));
  wafers=wafers+1;
end;

/* count the number of coaters */

sort table(tablename) by col coatercol;
coaters=1;
coat[coaters]=1;
k=1;
do i=1 to (lastrow-1);
  if (col(coatercol) row (i) of table(tablename) = col(coatercol) row (i+1) of
table(tablename))
    begin;
```

```
      k=k+1;
      coat[coaters]=k;
    end;
  else
    begin;
      coaters=coaters+1;
      k=1;
    end;
end;
coat[coaters]=k;
totalcoaters=coaters;

/* count the number of steppers */

sort table(tablename) by col steppercol;
steppers=1;
do i=1 to (lastrow-1);
  if (col(steppercol) row (i) of table(tablename) = col(steppercol) row (i+1) of
table(tablename))
    donext;
  else
    begin;
      steppers=steppers+1;
    end;
end;
totalsteppers=steppers;


/* count the number of developers */

sort table(tablename) by col developercol;
developers=1;
do i=1 to (lastrow-1);
  if (col(developercol) row (i) of table(tablename) = col (developercol) row
(i+1) of table(tablename))
    donext;
  else
    begin;
    developers=developers+1;
    end;
end;
totaldevelopers=developers;

/* count the number of aquatars */

if (aquatars=1) then
  totalaquatars=aquatars ;
else
  begin;
    sort table(tablename) by col aquatarcol;
    aquatars=1;
    do i=1 to (lastrow-1);
      if (col(aquatarcol) row (i) of table(tablename) = col(aquatarcol) row (i+1)
of table(tablename))
        donext;
      else
        aquatars=aquatars+1;
    end;
    totalaquatars=aquatars;
  end;

/* count the number of xones */

if (xones = 1) then
  totalextral=xones;
else
  begin;
```

```
    sort table(tablename) by col xonecol;
    xones=1;
    do i=1 to (lastrow-1);
      if (col(xonecol) row (i) of table(tablename) = col(xonecol) row (i+1) of
table(tablename))
        donext;
      else
        xones=xones+1;
    end;
    totalextra1=xones;
  end;

/* count the number of ztwos */

if (ztwos = 1) then
  totalextra2=ztwos;
else
  begin;
    sort table(tablename) by col ztwocol;
    ztwos=1;
    do i=1 to (lastrow-1);
      if (col(ztwocol) row (i) of table(tablename) = col(ztwocol) row (i+1) of
table(tablename))
        donext;
      else
        ztwos=ztwos+1;
    end;
    totalextra2=ztwos;
  end;

/* making names for each coater table */

sort table(tablename) by col(coatercol);

/* i is the number of coaters */
/* r is the row number where the next coater starts */

r=1;
do i=1 to coaters;
  nameofcoater=col(coatercol) row(r) of table(tablename);
  coater_name[i]=CAT("coat",nameofcoater);
  MAKETABLE(coater_name[i]);
  r=r+coat[i];
end;

/* separating tracks into coater tables */

r=1;
do i=1 to coaters;
  ADD ROWS TO TABLE(coater_name[i]) FROM rows r TO (r+coat[i]-1) of
TABLE(tablename);
  r=r+coat[i];
end;

/* This is where the stepper loop starts */

n=1;
do i=1 to coaters;
  k=1;
  steppers=1;
  stepp[steppers]=1;
  sort table(coater_name[i]) by col(steppercol);
  do j=2 to LASTROW(coater_name[i]);
    if (col(steppercol) row (j) of table(coater_name[i]) = col(steppercol) row
(j-1) of table(coater_name[i]))
      begin;
        k=k+1;
```

```
        stepp[steppers] = k;
      end;
    else
      begin;
        steppers=steppers+1;
        k=1;
      end;
  end;
  stepp[steppers]=k;

  /* making individual stepper tables */
  /* s is the number of steppers */
  /* r is the row number where the next stepper starts */

  r=1;
  do s=1 to steppers;
    nameofstepper=col(steppercol) row(r) of table(coater_name[i]);
    nameofcoater=col(coatercol) row(1) of table(coater_name[i]);
    stepper_name[s]=CAT("coat",nameofcoater,"step",nameofstepper);
    MAKETABLE(stepper_name[s]);
    r=r+stepp[s];
  end;

  /* moving stepper tracks into new tables */

  r=1;
  do s=1 to steppers;
    ADD ROWS TO TABLE(stepper_name[s]) FROM rows r TO (r+stepp[s]-1) of
TABLE(coater_name[i]);
    r=r+stepp[s];
  end;

/* start of developer loop */
  do s=1 to steppers;
    k=1;
    developers=1;
    dev[developers]=1;
    sort table(stepper_name[s]) by col(developercol);
    do j=2 to LASTROW(stepper_name[s]);
      if (col(developercol) row (j) of table(stepper_name[s]) = col(developercol)
row (j-1) of table(stepper_name[s]))
        begin;
          k=k+1;
          dev[developers] = k;
        end;
      else
        begin;
          developers=developers+1;
          k=1;
        end;
    end;
    dev[developers]=k;

    /* making individual developer tables */
    /* d is the number of developers */
    /* r is the row number where the next developer starts */

    r=1;
    do d=1 to developers;
      nameofdeveloper=col(developercol) row(r) of table(stepper_name[s]);
      nameofcoater=col(coatercol) row(1) of table(stepper_name[s]);
      nameofstepper=col(steppercol) row (1) of table(stepper_name[s]);

  developer_name[d]=CAT("coat",nameofcoater,"step",nameofstepper,"dev",nameofdeve
loper);
      MAKETABLE(developer_name[d]);
      r=r+dev[d];
```

```
    end;

    /* moving developer tracks into new tables */

    r=1;
    do d=1 to developers;
      ADD ROWS TO TABLE(developer_name[d]) FROM rows r TO (r+dev[d]-1) of
TABLE(stepper_name[s]);
      r=r+dev[d];
    end;

    /* START OF Yes/No AQUATAR LOOP */
    if (totalaquatars=1)
      /* no aquatar loop */
      begin;
        if (totalextra1=1)
          begin;
            if (totalextra2=1)
              /* no aquatar, no extra1, no extra2 */
              begin;
                do d=1 to developers;
                  tablenum[n]=developer_name[d];
                  n=n+1;
                end;
              end;
            else
              /* no aquatar, no extra1, yes extra2 */
              begin;
                do d=1 to developers;
                  k=1;
                  ztwo[ztwos]=1;
                  sort table(developer_name[d]) by col(ztwocol);
                  do j=2 to LASTROW(developer_name[d]);
                    if (col(ztwocol) row (j) of table(developer_name[d]) =
col(ztwocol) row (j-1) of table(developer_name[d]))
                      begin;
                        k=k+1;
                        ztwo[ztwos] = k;
                      end;
                    else
                      begin;
                        ztwos=ztwos+1;
                        k=1;
                      end;
                  end;
                  ztwo[ztwos]=k;

                  /* making individual ztwo tables */
                  /* z is the number of ztwos */
                  /* r is the row number where the next aquatar starts */

                  r=1;
                  do z=1 to ztwos;
                    nameofztwo=col(ztwocol) row(r) of table(developer_name[d]);
                    nameofdeveloper=col(developercol) row (1) of
table(developer_name[d]);
                    nameofcoater=col(coatercol) row(1) of
table(developer_name[d]);
                    nameofstepper=col(steppercol) row (1) of
table(developer_name[d]);
                    ztwo_name[z] = CAT("coat", nameofcoater, "step",
nameofstepper, "dev" ,nameofdeveloper,"ztwo", nameofztwo);
                    MAKETABLE(ztwo_name[z]);
                    r=r+ztwo[z];
                  end;

                  /* moving ztwo tracks into new tables */
```

```
                r=1;
                do z=1 to ztwos;
                  ADD ROWS TO TABLE(ztwo_name[z]) FROM rows r TO (r+ztwo[z]-1)
of TABLE(developer_name[d]);
                  r=r+ztwo[z];
                end;

                do z=1 to ztwos;
                  tablenum[n]=ztwo_name[z];
                  n=n+1;
                end;
              end;
            end;

        end;
      else
        /* no aquatar, yes extra1 */
        begin;
          if (totalextra2=1)
            /* no aquatar, yes extra1, no extra2 */
            begin;
              do d=1 to developers;
                k=1;
                xones=1;
                xone[xones]=1;
                sort table(developer_name[d]) by col(xonecol);
                do j=2 to LASTROW(developer_name[d]);
                  if (col(xonecol) row (j) of table(developer_name[d]) =
col(xonecol) row (j-1) of table(developer_name[d]))
                    begin;
                      k=k+1;
                      xone[xones] = k;
                    end;
                  else
                    begin;
                      xones=xones+1;
                      k=1;
                    end;
                end;
                xone[xones]=k;

              /* making individual xone tables */
              /* x is the number of xones */
              /* r is the row number where the next xone starts */

              r=1;
              do x=1 to xones;
                nameofxone=col(xonecol) row(r) of table(developer_name[d]);
                nameofdeveloper=col(developercol) row (1) of
table(developer_name[d]);
                nameofcoater=col(coatercol) row(1) of table(developer_name[d]);
                nameofstepper=col(steppercol) row (1) of
table(developer_name[d]);

  xone_name[x]=CAT("coat",nameofcoater,"step",nameofstepper,"dev",nameofdeveloper
,"xone", nameofxone);
                MAKETABLE(xone_name[x]);
                r=r+xone[x];
              end;

                /* moving xones tracks into new tables */

                r=1;
                do x=1 to xones;
                  ADD ROWS TO TABLE(xone_name[x]) FROM rows r TO (r+xone[x]-1)
of TABLE(developer_name[d]);
```

```
                    r=r+xone[x];
                  end;

                  do x=1 to xones;
                    tablenum[n]=xone_name[x];
                    n=n+1;
                  end;
                end;
              end;
            else
              /* no aquatar, yes extra1, yes extra2 */
              begin;
                do d=1 to developers;
                  k=1;
                  xones=1;
                  xone[xones]=1;
                  sort table(developer_name[d]) by col(xonecol);
                  do j=2 to LASTROW(developer_name[d]);
                    if (col(xonecol) row (j) of table(developer_name[d]) =
col(xonecol) row (j-1) of table(developer_name[d]))
                      begin;
                        k=k+1;
                        xone[xones] = k;
                      end;
                    else
                      begin;
                        xones = xones +1;
                        k=1;
                      end;
                  end;
                  xone[xones]=k;

                  /* making individual xone tables */
                  /* x is the number of xone */
                  /* r is the row number where the next xones starts */

                  r=1;
                  do x=1 to xones;
                    nameofxone=col(xonecol) row(r) of table(developer_name[d]);
                    nameofdeveloper=col(developercol) row (1) of
table(developer_name[d]);
                    nameofcoater=col(coatercol) row(1) of
table(developer_name[d]);
                    nameofstepper=col(steppercol) row (1) of
table(developer_name[d]);

  xone_name[x]=CAT("coat",nameofcoater,"step",nameofstepper,"dev",nameofdeveloper
, "xone",nameofxone);
                    MAKETABLE(xone_name[x]);
                    r=r+xone[x];
                  end;

                  /* moving xone tracks into new tables */

                  r=1;
                  do x=1 to xones;
                    ADD ROWS TO TABLE(xone_name[x]) FROM rows r TO (r+xone[x]-1)
of TABLE(developer_name[d]);
                    r=r+xone[x];
                  end;

                  /* extra2 loop of (no aquatar, yes extra1, yes extra2) */
                  do x=1 to xones;
                    k=1;
                    ztwos=1;
                    ztwo[ztwos]=1;
                    sort table(xone_name[x]) by col(ztwocol);
```

```
                    do j=2 to LASTROW(xone_name[x]);
                      if (col(ztwocol) row (j) of table(xone_name[x]) =
col(ztwocol) row (j-1) of table(xone_name[x]))
                        begin;
                          k=k+1;
                          ztwo[ztwos] = k;
                        end;
                      else
                        begin;
                          ztwos = ztwos +1;
                          k=1;
                        end;
                    end;
                    ztwo[ztwos]=k;

                    /* making individual ztwos tables */
                    /* z is the number of ztwos */
                    /* r is the row number where the next ztwos starts */

                    r=1;
                    do z=1 to ztwos;
                      nameofztwo=col(ztwocol) row( r) of table(xone_name[x]);
                      nameofxone=col(xonecol) row(1) of table(xone_name[x]);
                      nameofdeveloper=col(developercol) row (1) of
table(xone_name[x]);
                      nameofcoater=col(coatercol) row(1) of table(xone_name[x]);
                      nameofstepper=col(steppercol) row (1) of
table(xone_name[x]);

  ztwo_name[z]=CAT("coat",nameofcoater,"step",nameofstepper,"dev",nameofdeveloper
,"xone", nameofxone,"ztwo",nameofztwo);
                      MAKETABLE(ztwo_name[z]);
                      r=r+ztwo[z];
                    end;

                    /* moving ztwos tracks into new tables */

                    r=1;
                    do z=1 to ztwos;
                      ADD ROWS TO TABLE(ztwo_name[z]) FROM rows r TO (r+ztwo[z]-1)
of TABLE(xone_name[x]);
                      r=r+ztwo[z];
                    end;
                    do z=1 to ztwos;
                      tablenum[n]=ztwo_name[z];
                      n=n+1;
                    end;
                  end;
                end;
              end;

          end;
    end;
  else
    /* aquatar loop */
    begin;
      if (totalextra1=1)
        /* yes aquatar, no extra1 */
        begin;
          if (totalextra2=1)
            /* yes aquatar, no extra1, no extra2 */
            begin;
              do d=1 to developers;
                k=1;
                aquatars=1;
                aqua[aquatars]=1;
                sort table(developer_name[d]) by col(aquatarcol);
```

```
                do j=2 to LASTROW(developer_name[d]);
                  if (col(aquatarcol) row (j) of table(developer_name[d]) =
col(aquatarcol) row (j-1) of table(developer_name[d]))
                    begin;
                      k=k+1;
                      aqua[aquatars] = k;
                    end;
                  else
                    begin;
                      aquatars=aquatars+1;
                      k=1;
                    end;
                end;
                aqua[aquatars]=k;

                /* making individual aquatar tables */
                /* a is the number of aquatars */
                /* r is the row number where the next aquatar starts */

                r=1;
                do a=1 to aquatars;
                  nameofaquatar=col(aquatarcol) row(r) of
table(developer_name[d]);
                  nameofdeveloper=col(developercol) row (1) of
table(developer_name[d]);
                  nameofcoater=col(coatercol) row(1) of table(developer_name[d]);
                  nameofstepper=col(steppercol) row (1) of
table(developer_name[d]);

  aquatar_name[a]=CAT("coat",nameofcoater,"step",nameofstepper,"dev",nameofdevelo
per,"aqua", nameofaquatar);
                  MAKETABLE(aquatar_name[a]);
                  r=r+aqua[a];
                end;

                /* moving aquatar tracks into new tables */

                r=1;
                do a=1 to aquatars;
                  ADD ROWS TO TABLE(aquatar_name[a]) FROM rows r TO (r+aqua[a]-1)
of TABLE(developer_name[d]);
                  r=r+aqua[a];
                end;

                do a=1 to aquatars;
                  tablenum[n]=aquatar_name[a];
                  n=n+1;
                end;
        end;
            end;
          else
            /* yes aquatar, no extra1, yes extra2 */
            begin;
              do d=1 to developers;
                k=1;
                aquatars=1;
                aqua[aquatars]=1;
                sort table(developer_name[d]) by col(aquatarcol);
                do j=2 to LASTROW(developer_name[d]);
                  if (col(aquatarcol) row (j) of table(developer_name[d]) =
col(aquatarcol) row (j-1) of table(developer_name[d]))
                    begin;
                      k=k+1;
                      aqua[aquatars] = k;
                    end;
                  else
                    begin;
```

```
                    aquatars=aquatars+1;
                    k=1;
                  end;
              end;
              aqua[aquatars]=k;

              /* making individual aquatar tables */
              /* a is the number of aquatars */
              /* r is the row number where the next aquatar starts */

              r=1;
              do a=1 to aquatars;
                nameofaquatar=col(aquatarcol) row(r) of
table(developer_name[d]);
                nameofdeveloper=col(developercol) row (1) of
table(developer_name[d]);
                nameofcoater=col(coatercol) row(1) of table(developer_name[d]);
                nameofstepper=col(steppercol) row (1) of
table(developer_name[d]);

  aquatar_name[a]=CAT("coat",nameofcoater,"step",nameofstepper,"dev",nameofdevelo
per,"aqua", nameofaquatar);
                MAKETABLE(aquatar_name[a]);
                r=r+aqua[a];
              end;

              /* moving aquatar tracks into new tables */

              r=1;
              do a=1 to aquatars;
                ADD ROWS TO TABLE(aquatar_name[a]) FROM rows r TO (r+aqua[a]-1)
of TABLE(developer_name[d]);
                r=r+aqua[a];
              end;

              /* extra2 loop of (yes aquatar, no extra1, yes extra2) */
              do a=1 to aquatars;
                k=1;
                ztwos=1;
                ztwo[ztwos]=1;
                sort table(aquatar_name[a]) by col(ztwocol);
                do j=2 to LASTROW(aquatar_name[a]);
                  if (col(ztwocol) row (j) of table(aquatar_name[a]) =
col(ztwocol) row (j-1) of table(aquatar_name[a]))
                    begin;
                      k=k+1;
                      ztwo[ztwos] = k;
                    end;
                  else
                    begin;
                      ztwos = ztwos +1;
                      k=1;
                    end;
                end;
                ztwo[ztwos]=k;

                /* making individual ztwo tables */
                /* z is the number of ztwos */
                /* r is the row number where the next ztwos starts */

                r=1;
                do z=1 to ztwos;
nameofztwo=col(ztwocol) row(r) of table(aquatar_name[a]);
nameofaquatar=col(aquatarcol) row(1) of table(aquatar_name[a]);
                  nameofdeveloper=col(developercol) row (1) of
table(aquatar_name[a]);
                  nameofcoater=col(coatercol) row(1) of table(aquatar_name[a]);
```

```
                    nameofstepper=col(steppercol) row (1) of
table(aquatar_name[a]);

  aquatar_name[a]=CAT("coat",nameofcoater,"step",nameofstepper,"dev",nameofdevelo
per,"aqua", nameofaquatar, "ztwo",nameofztwo);
                    MAKETABLE(ztwo_name[z]);
                    r=r+ztwo[z];
                  end;

                  /* moving ztwo tracks into new tables */

                  r=1;
                  do z=1 to ztwos;
                    ADD ROWS TO TABLE(ztwo_name[z]) FROM rows r TO (r+ztwo[z]-1)
of TABLE(aquatar_name[a]);
                    r=r+ztwo[z];
                  end;
                  do z=1 to ztwos;
                    tablenum[n]=ztwo_name[z];
                    n=n+1;
                  end;
                end;
              end;
            end;
        end;
      else
        /* yes aquatar, yes extra1 */
        begin;
            if (totalextra2=1)
            /* yes aquatar, yes extra1, no extra2 */
            begin;
              do d=1 to developers;
                k=1;
                aquatars=1;
                aqua[aquatars]=1;
                sort table(developer_name[d]) by col(aquatarcol);
                do j=2 to LASTROW(developer_name[d]);
                  if (col(aquatarcol) row (j) of table(developer_name[d]) =
col(aquatarcol) row (j-1) of table(developer_name[d]))
                    begin;
                      k=k+1;
                      aqua[aquatars] = k;
                    end;
                  else
                    begin;
                      aquatars=aquatars+1;
                      k=1;
                    end;
                end;
                aqua[aquatars]=k;

                /* making individual aquatar tables */
                /* a is the number of aquatars */
                /* r is the row number where the next aquatar starts */

                r=1;
                do a=1 to aquatars;
                  nameofaquatar=col(aquatarcol) row (r) of
table(developer_name[d]);
                  nameofdeveloper=col(developercol) row (1) of
table(developer_name[d]);
                  nameofcoater=col(coatercol) row(1) of table(developer_name[d]);
                  nameofstepper=col(steppercol) row (1) of
table(developer_name[d]);

  aquatar_name[a]=CAT("coat",nameofcoater,"step",nameofstepper,"dev",nameofdevelo
per,"aqua", nameofaquatar);
```

```
                MAKETABLE(aquatar_name[a]);
                r=r+aqua[a];
              end;

              /* moving aquatar tracks into new tables */

              r=1;
              do a=1 to aquatars;
                ADD ROWS TO TABLE(aquatar_name[a]) FROM rows r TO (r+aqua[a]-1)
of TABLE(developer_name[d]);
                r=r+aqua[a];
              end;

              /* extra1 loop of (yes aquatar, yes extra1, no extra2) */
              do a=1 to aquatars;
                k=1;
                xones=1;
                xone[xones]=1;
                sort table(aquatar_name[a]) by col(xonecol);
                do j=2 to LASTROW(aquatar_name[a]);
                  if (col(xonecol) row (j) of table(aquatar_name[a]) =
col(xonecol) row (j-1) of table(aquatar_name[a]))
                    begin;
                      k=k+1;
                      xone[xones] = k;
                    end;
                  else
                    begin;
                      xones = xones +1;
                      k=1;
                    end;
                end;
                xone[xones]=k;

                /* making individual xone tables */
                /* a is the number of xone */
                /* r is the row number where the next xones starts */

                r=1;
                do x=1 to xones;
                  nameofxone=col(xonecol) row(r) of table(aquatar_name[a]);
                  nameofaquatar=col(aquatarcol) row(1) of
table(aquatar_name[a]);
                  nameofdeveloper=col(developercol) row (1) of
table(aquatar_name[a]);
                  nameofcoater=col(coatercol) row(1) of table(aquatar_name[a]);
                  nameofstepper=col(steppercol) row (1) of
table(aquatar_name[a]);

  xone_name[x]=CAT("coat",nameofcoater,"step",nameofstepper,"dev",nameofdeveloper
,"aqua", nameofaquatar, "xone",nameofxone);
                  MAKETABLE(xone_name[x]);
                  r=r+xone[x];
                end;

                /* moving xone tracks into new tables */

                r=1;
                do x=1 to xones;
                  ADD ROWS TO TABLE(xone_name[x]) FROM rows r TO (r+xone[x]-1)
of TABLE(aquatar_name[a]);
                  r=r+xone[x];
                end;
                do x=1 to xones;
                  tablenum[n]=xone_name[x];
                  n=n+1;
                end;
```

```
                end;
              end;
            end;
          else
            /* yes aquatar, yes extra1, yes extra2 */
            begin;
              do d=1 to developers;
                k=1;
                aquatars=1;
                aqua[aquatars]=1;
                sort table(developer_name[d]) by col(aquatarcol);
                do j=2 to LASTROW(developer_name[d]);
                  if (col(aquatarcol) row (j) of table(developer_name[d]) =
col(aquatarcol) row (j-1) of table(developer_name[d]))
                    begin;
                      k=k+1;
                      aqua[aquatars] = k;
                    end;
                  else
                    begin;
                      aquatars=aquatars+1;
                      k=1;
                    end;
                end;
                aqua[aquatars]=k;

                /* making individual aquatar tables */
                /* a is the number of aquatars */
                /* r is the row number where the next aquatar starts */

                r=1;
                do a=1 to aquatars;
                  nameofaquatar=col(aquatarcol) row(r) of
table(developer_name[d]);
                  nameofdeveloper=col(developercol) row (1) of
table(developer_name[d]);
                  nameofcoater=col(coatercol) row(1) of table(developer_name[d]);
                  nameofstepper=col(steppercol) row (1) of
table(developer_name[d]);

  aquatar_name[a]=CAT("coat",nameofcoater,"step",nameofstepper,"dev",nameofdevelo
per,"aqua", nameofaquatar);
                  MAKETABLE(aquatar_name[a]);
                  r=r+aqua[a];
                end;

                /* moving aquatar tracks into new tables */

                r=1;
                do a=1 to aquatars;
                  ADD ROWS TO TABLE(aquatar_name[a]) FROM rows r TO (r+aqua[a]-1)
of TABLE(developer_name[d]);
                  r=r+aqua[a];
                end;

                /* extra1 loop of (yes aquatar, yes extra1, yes extra2) */
                  do a=1 to aquatars;
                  k=1;
                  xones=1;
                  xone[xones]=1;
                  sort table(aquatar_name[a]) by col(xonecol);
                  do j=2 to LASTROW(aquatar_name[a]);
                    if (col(xonecol) row (j) of table(aquatar_name[a]) =
col(xonecol) row (j-1) of table(aquatar_name[a]))
                      begin;
                        k=k+1;
                        xone[xones] = k;
```

```
                    end;
                  else
                    begin;
                      xones = xones +1;
                      k=1;
                    end;
                end;
                xone[xones]=k;

                /* making individual xone tables */
                /* x is the number of xone */
                /* r is the row number where the next xones starts */

                r=1;
                do x=1 to xones;
                  nameofxone=col(xonecol) row(r) of table(aquatar_name[a]);
                  nameofaquatar=col(aquatarcol) row(1) of
table(aquatar_name[a]);
                  nameofdeveloper=col(developercol) row (1) of
table(aquatar_name[a]);
                  nameofcoater=col(coatercol) row(1) of table(aquatar_name[a]);
                  nameofstepper=col(steppercol) row (1) of
table(aquatar_name[a]);

  xone_name[x]=CAT("coat",nameofcoater,"step",nameofstepper,"dev",nameofdeveloper
,"aqua", nameofaquatar, "xone",nameofxone);
                  MAKETABLE(xone_name[x]);
                  r=r+xone[x];
                end;

                /* moving xone tracks into new tables */

                r=1;
                do x=1 to xones;
                  ADD ROWS TO TABLE(xone_name[x]) FROM rows r TO (r+xone[x]-1)
of TABLE(aquatar_name[a]);
                  r=r+xone[x];
                end;

                /* extra2 loop of (yes aquatar, yes extra1, yes extra2) */
                do x=1 to xones;
                  k=1;
                  ztwos=1;
                  ztwo[ztwos]=1;
                  sort table(xone_name[x]) by col(ztwocol);
                  do j=2 to LASTROW(xone_name[x]);
                    if (col(ztwocol) row (j) of table(xone_name[x]) =
col(ztwocol) row (j-1) of table(xone_name[x]))
                      begin;
                        k=k+1;
                        ztwo[ztwos] = k;
                      end;
                    else
                      begin;
                        ztwos = ztwos +1;
                        k=1;
                      end;
                  end;
                  ztwo[ztwos]=k;

                  /* making individual ztwo tables */
                  /* z is the number of ztwo */
                  /*  r is the number where the next ztwo starts  */
                  r=1;
                  do z=1 to ztwos;
                    nameofztwo=col(ztwocol) row (r) of table(xone_name[x]);
                    nameofxone=col(xonecol) row (1) of table(xone_name[x]);
```

```
                    nameofaquatar=col(aquatarcol) row(1) of table(xone_name[x]);
                    nameofdeveloper=col(developercol) row (1) of
table(xone_name[x]);
                    nameofcoater=col(coatercol) row(1) of table(xone_name[x]);
                    nameofstepper=col(steppercol) row (1) of
table(xone_name[x]);
                    ztwo_name[z]=CAT("coat",nameofcoater,"step" ,nameofstepper
,"dev",nameofdeveloper,"aqua", nameofaquatar,
"xone",nameofxone,"ztwo",nameofztwo);
                    MAKETABLE(ztwo_name[z]);
                    r=r+ztwo[z];
                  end;

                  /* moving ztwo tracks into new tables */

                  r=1;
                  do z=1 to ztwos;
                    ADD ROWS TO TABLE(ztwo_name[z]) FROM rows r TO (r+ztwo[z]-1)
of TABLE(xone_name[x]);
                    r=r+ztwo[z];
                  end;
                  do z=1 to ztwos;
                    tablenum[n]=ztwo_name[z];
                    n=n+1;
                  end;
                end;
              end;
            end;
          end;
      end;
    /* END OF DEVELOPER LOOP */
    end;
  /* END OF STEPPER LOOP */
  end;
/* END OF COATER LOOP */
end;
lasttable=n-1;
/* DETERMINING THE NUMBER OF COLUMNS FOR EACH PROCESSING STEP */

firstcoatercol=1;
if (totalcoaters=1)
  lastcoatercol=firstcoatercol;
else
  lastcoatercol=totalcoaters;

firststeppercol=lastcoatercol+1;
if (totalsteppers=1)
  laststeppercol=firststeppercol;
else
  laststeppercol=firststeppercol+totalsteppers-1;

firstdevcol=laststeppercol+1;
if (totaldevelopers=1)
  lastdevcol=firstdevcol;
else
  lastdevcol=firstdevcol+totaldevelopers-1;

if (totalaquatars=1)
  begin;
    if (totalextra1=1)
      begin;
        if (totalextra2=1)
          begin;
            lastdatacol=lastdevcol;
          end;
        else
        /* no aquatar, no extra1 yes extra2 */
```

```
          begin;
            firstextra2col=lastdevcol+1;
            lastextra2col=firstextra2col+totalextra2-1;
            lastdatacol=lastextra2col;
          end;
      end;
    else
    /* no aquatar yes extra1 */
      begin;
        if (totalextra2=1)
        /* no aquatar yes extra1 no extra2 */
          begin;
            firstextra1col=lastdevcol+1;
            lastextra1col=firstextra1col+totalextra1-1;
            lastdatacol=lastextra1col;
          end;
        else
        /* no aquatar yes extra1 yes extra2 */
          begin;
            firstextra1col=lastdevcol+1;
            lastextra1col=firstextra1col+totalextra1-1;
            firstextra2col=lastextra1col+1;
            lastextra2col=firstextra2col+totalextra2-1;
            lastdatacol=lastextra2col;
          end;
      end;
  end;
else
/* yes aquatar */
  begin;
    if (totalextra1=1)
      begin;
        if (totalextra2=1)
          begin;
            firstaquacol=lastdevcol+1;
            lastaquacol=firstaquacol+totalaquatars-1;
            lastdatacol=lastaquacol;
          end;
        else
        /* yes aquatar no extra1 yes extra2 */
          begin;
            firstaquacol=lastdevcol+1;
            lastaquacol=firstaquacol+totalaquatars-1;
firstextra2col=lastaquacol+1;
            lastextra2col=firstextra2col+totalextra2-1;
            lastdatacol=lastextra2col;
          end;
      end;
    else
    /* yes aquatar yes extra1 */
      begin;
        if (totalextra2 = 1)
        /* yes aquatar yes extra1 no extra2 */
          begin;
            firstaquacol=lastdevcol+1;
            lastaquacol=firstaquacol+totalaquatars-1;
            firstextra1col=lastaquacol+1;
            lastextra1col=firstextra1col+totalextra1-1;
            lastdatacol=lastextra1col;
          end;
        else
        /* yes aquatar yes extra1 yes extra2 */
          begin;
firstaquacol=lastdevcol+1;
            lastaquacol=firstaquacol+totalaquatars-1;
            firstextra1col=lastaquacol+1;
            lastextra1col=firstextra1col+totalextra1-1;
```

```
            firstextra2col=lastextra1col+1;
            lastextra2col=firstextra2col+totalextra2-1;
            lastdatacol=lastextra2col;
          end;
      end;
  end;

/* SORTING TABLES FOR COATERS */

/* "tracknum[p,j]" contains all of the information about what each final table
contains in terms of processing equipment */

MAKETABLE(tracknum);
MAKETABLE(takencoaters);

do i=1 to lasttable;
  do j=firstcoatercol to lastdatacol;
    tracknum[i,j]= "no";
  end;
  takencoaters[i]="no";
end;

do j=firstcoatercol to lastcoatercol;
  I=1;
  if (takencoaters[i]="yes")
    begin;
      do I=1 to lasttable;
        if (takencoaters[i]="yes")
          donext;
        else
          doexit;
      end;
      takencoaters[i]="yes";
      tracknum[i,j]="yes";
      do p=(i+1) to lasttable;
        if (col(coatercol) row(1) of table(tablenum[i]) = col(coatercol) row(1)
of table(tablenum[p]))
          begin;
            takencoaters[p]="yes";
            tracknum[p,j]="yes";
          end;
        else
          donext;
      end;
    end;
  else
  /* takencoaters[1] = "no" */
    begin;
      takencoaters[i]="yes";
      tracknum[i,j]="yes";
        do p=(i+1) to lasttable;
          if (col(coatercol) row(1) of table(tablenum[i]) = col(coatercol) row(1)
of table(tablenum[p]))
            begin;
              takencoaters[p]="yes";
              tracknum[p,j]="yes";
            end;
          else
            donext;
        end;
    end;
end;

/* sorting tables for steppers */

MAKETABLE(takensteppers);
```

```
do i=1 to lasttable;
  takensteppers[i]="no";
end;

do j=firststeppercol to laststeppercol;
  I=1;
  if (takensteppers[i]="yes")
    begin;
      do I=1 to lasttable;
        if (takensteppers[i]="yes")
          donext;
        else
          doexit;
      end;
      takensteppers[i]="yes";
      tracknum[i,j]="yes";
      do p=(i+1) to lasttable;
        if (col(steppercol) row(1) of table(tablenum[i]) = col(steppercol) row(1)
of table(tablenum[p]))
          begin;
            takensteppers[p]="yes";
            tracknum[p,j]="yes";
          end;
        else
          donext;
      end;
    end;
  else
  /* takensteppers[i]="no" */
    begin;
      takensteppers[i]="yes";
      tracknum[i,j]="yes";
        do p=(i+1) to lasttable;
          if (col(steppercol) row(1) of table(tablenum[i]) = col(steppercol)
row(1) of table(tablenum[p]))
            begin;
              takensteppers[p]="yes";
              tracknum[p,j]="yes";
            end;
          else
            donext;
        end;
    end;
end;

/* sorting tables for developers */

MAKETABLE(takendevelopers);
do i=1 to lasttable;
  takendevelopers[i]="no";
end;

do j=firstdevcol to lastdevcol;
  I=1;
  if (takendevelopers[i]="yes")
    begin;
      do I=1 to lasttable;
        if (takendevelopers[i]="yes")
          donext;
        else
          doexit;
      end;
      takendevelopers[i]="yes";
      tracknum[i,j]="yes";
      do p=(i+1) to lasttable;
        if (col(developercol) row(1) of table(tablenum[i]) = col(developercol)
row(1) of table(tablenum[p]))
```

```
          begin;
            takendevelopers[p]="yes";
            tracknum[p,j]="yes";
          end;
        else
          donext;
      end;
    end;
  else
  /* takendevelopers[I]="no" */
    begin;
      takendevelopers[i]="yes";
      tracknum[i,j]="yes";
        do p=(i+1) to lasttable;
          if (col(developercol) row(1) of table(tablenum[i]) = col(developercol)
row(1) of table(tablenum[p]))
            begin;
              takendevelopers[p]="yes";
              tracknum[p,j]="yes";
            end;
          else
            donext;
        end;
    end;
end;

/* sorting tables for aquatars */

if (totalaquatars GT 1)
  begin;
    MAKETABLE(takenaquatars);
    do I=1 to lasttable;
      takenaquatars[i]="no";
    end;

    do j=firstaquacol to lastaquacol;
      i=1;
        if (takenaquatars[i]="yes")
          begin;
            do I=1 to lasttable;
              if (takenaquatars[i]="yes")
                donext;
              else
                doexit;
            end;
            takenaquatars[i]="yes";
            tracknum[i,j]="yes";
            do p=(i+1) to lasttable;
              if (col(aquatarcol) row(1) of table(tablenum[i]) = col(aquatarcol)
row(1) of table(tablenum[p]))
                begin;
                  takenaquatars[p]="yes";
                  tracknum[p,j]="yes";
                end;
              else
                donext;
            end;
          end;
        else
          begin;
            takenaquatars[i]="yes";
            tracknum[i,j]="yes";
            do p=(i+1) to lasttable;
              if (col(aquatarcol) row(1) of table(tablenum[i]) = col(aquatarcol)
row(1) of table(tablenum[p]))
                begin;
                  takenaquatars[p]="yes";
```

```
                tracknum[p,j]="yes";
              end;
            else
              donext;
          end;
        end;
    end;
  end;
else
/* no aquatars */
continue;

/* sorting tables for extra1 */
if (totalextra1 GT 1)
  begin;
    MAKETABLE(takenextra1);
    do i=1 to lasttable;
      takenextra1[i]="no";
    end;

    do j=firstextra1col to lastextra1col;
      I=1;
      if (takenextra1[i]="yes")
        begin;
          do I=1 to lasttable;
            if (takenextra1[i]="yes")
              donext;
            else
              doexit;
          end;
          takenextra1[i]="yes";
          tracknum[i,j]="yes";
          do p=(i+1) to lasttable;
            if (col(xonecol) row(1) of table(tablenum[i]) = col(xonecol) row(1)
of table(tablenum[p]))
              begin;
                takenextra1[p]="yes";
                tracknum[p,j]="yes";
              end;
            else
              donext;
          end;
        end;
      else
        begin;
          takenextra1[i]="yes";
          tracknum[i,j]="yes";
          do p=(i+1) to lasttable;
            if (col(xonecol) row(1) of table(tablenum[i]) = col(xonecol) row(1)
of table(tablenum[p]))
              begin;
                takenextra1[p]="yes";
                tracknum[p,j]="yes";
              end;
            else
              donext;
          end;
        end;
    end;
  end;
else
/* no extra1 */
  continue;

/* sorting tables for extra2 */
if (totalextra2 GT 1)
  begin;
```

```
    MAKETABLE(takenextra2);
    do i=1 to lasttable;
      takenextra2[i]="no";
    end;

    do j=firstextra2col to lastextra2col;
      i=1;
      if (takenextra2[i]="yes")
        begin;
          do I=1 to lasttable;
            if (takenextra2[i]="yes")
              donext;
            else
              doexit;
          end;
          takenextra2[i]="yes";
          tracknum[i,j]="yes";
          do p=(i+1) to lasttable;
            if (col(ztwocol) row(1) of table(tablenum[i]) = col(ztwocol) row(1)
of table(tablenum[p]))
              begin;
                takenextra2[p]="yes";
                tracknum[p,j]="yes";
              end;
            else
              donext;
          end;
        end;
      else
        begin;
          takenextra2[i]="yes";
          tracknum[i,j]="yes";
          do p=(i+1) to lasttable;
            if (col(ztwocol) row(1) of table(tablenum[i]) = col(ztwocol) row(1)
of table(tablenum[p]))
              begin;
                takenextra2[p]="yes";
                tracknum[p,j]="yes";
              end;
            else
              donext;
          end;
        end;
    end;
  end;
else
/* no extra1 */
  continue;

/* this routine sorts for tables with the exact processing equipment without the
COATER track */

r=1;
n=1;
do s=firststeppercol to laststeppercol;
  do d=firstdevcol to lastdevcol;
    if (totalaquatars=1)
      begin;
        if (totalextra1=1)
          begin;
            if (totalextra2=1)
              begin;
                do p=1 to lasttable;
                  if (tracknum[p,s]="yes" AND tracknum[p,d]="yes")
                    begin;
                      coatorder[r,n]=p;
                      r=r+1;
```

```
                    end;
                  else
                    donext;
                end;
                totalcoatorderrow[n]=(r-1);
                n=n+1;
                r=1;
              end;
            /* no aquatar yes extra2 */
            else
              begin;
                do z=firstextra2col to lastextra2col;
                  do p=1 to lasttable;
                    if (tracknum[p,s]= "yes" AND tracknum[p,d]= "yes" and
tracknum[p,z]= "yes")
                      begin;
                        coatorder[r,n]=p;
                        r=r+1;
                      end;
                    else
                      donext;
                  end;
                  n=n+1;
                  totalcoatorderrow[n-1]=(r-1);
                  r=1;
                end;
              end;
          end;
        /* no aquatar, yes extra1*/
        else
          begin;
            do x=firstextra1col to lastextra1col;
              if (totalextra2=1)
                begin;
                  do p=1 to lasttable;
                    if (tracknum[p,s]= "yes" and tracknum[p,d]= "yes" and
tracknum[p,x]= "yes")
                      begin;
                        coatorder[r,n]=p;
                        r=r+1;
                      end;
                    else
                      donext;
                  end;
                  n=n+1;
                  totalcoatorderrow[n-1]=(r-1);
                  r=1;
                end;
              /* no aquatar, yes extra1, yes extra2 */
              else
                begin;
                  do z=firstextra2col to lastextra2col;
                    do p=1 to lasttable;
                      if (tracknum[p,s]= "yes" and tracknum[p,d]= "yes" and
tracknum[p,x]= "yes" and tracknum[p,z]= "yes")
                        begin;
                          coatorder[r,n]=p;
                          r=r+1;
                        end;
                      else
                        donext;
                    end;
                    n=n+1;
                    totalcoatorderrow[n-1]=(r-1);
                    r=1;
                  /* end of extra2 loop */
                  end;
```

```
              end;
            /* end of extra1 loop */
            end;
          end;
      end;
    /* yes aquatar */
    else
      begin;
        do a=firstaquacol to lastaquacol;
          if (totalextra1=1)
            begin;
              if (totalextra2 =1)
                begin;
                  do p=1 to lasttable;
                    if (tracknum[p,a]= "yes" and tracknum[p,s]= "yes" and
tracknum[p,d]= "yes")
                      begin;
                        coatorder[r,n]=p;
                        r=r+1;
                      end;
                    else
                      donext;
                  end;
                  n=n+1;
                  totalcoatorderrow[n-1]=(r-1);
                  r=1;
                end;
              /* yes aquatar no extra1, yes extra2 */
              else
                begin;
                  do z=firstextra2col to lastextra2col;
                    do p=1 to lasttable;
                      if (tracknum[p,a]= "yes" and tracknum[p,s]= "yes" and
tracknum[p,d]= "yes" and tracknum[p,z]= "yes")
                        begin;
                          coatorder[r,n]=p;
                          r=r+1;
                        end;
                      else
                        donext;
                    end;
                    n=n+1;
                    totalcoatorderrow[n-1]=(r-1);
                    r=1;
                  end;
                end;
            end;
          /* yes aquatar, yes extra1*/
          else
            begin;
              do x=firstextra1col to lastextra1col;
                if (totalextra2 =1)
                  begin;
                    do p=1 to lasttable;
                      if (tracknum[p,a]="yes" and tracknum[p,s]="yes" and
tracknum[p,d]="yes" and tracknum[p,x]= "yes")
                        begin;
                          coatorder[r,n]=p;
                          r=r+1;
                        end;
                      else
                        donext;
                    end;
                    n=n+1;
                    totalcoatorderrow[n-1]=(r-1);
                    r=1;
                  end;
```

```
                /* yes aquatar, yes extra1, yes extra2 */
                else
                  begin;
                    do z=firstextra2col to lastextra2col;
                      do p=1 to lasttable;
                        if (tracknum[p,a]= "yes" and tracknum[p,s]= "yes" and
tracknum[p,d]= "yes" and tracknum[p,x]= "yes" and tracknum[p,z]= "yes")
                          begin;
                            coatorder[r,n]=p;
                            r=r+1;
                          end;
                        else
                          donext;
                      end;
                      n=n+1;
                      totalcoatorderrow[n-1]=(r-1);
                      r=1;
                    /* end of  extra2 loop */
                    end;
                  end;
              /* end of extra1 loop */
              end;
            end;
        /* end of aquatar loop */
        end;
      end;
  /* end of developer loop */
  end;
/* end of stepper loop */
end;
lastcolcoatorder=n-1;


/* this routine sorts for tables with the exact processing equipment without the
STEPPER track */

r=1;
n=1;
do c=firstcoatercol to lastcoatercol;
  do d=firstdevcol to lastdevcol;
    if (totalaquatars=1)
      begin;
        if (totalextra1=1)
          begin;
            if (totalextra2 =1)
              begin;
                do p=1 to lasttable;
                  if (tracknum[p,c]="yes" AND tracknum[p,d]="yes")
                    begin;
                      steporder[r,n]=p;
                      r=r+1;
                    end;
                  else
                    donext;
                end;
                n=n+1;
                totalsteporderrow[n-1]=(r-1);
                r=1;
              end;
            /* no aquatar yes extra2 */
            else
              begin;
                do z=firstextra2col to lastextra2col;
                  do p=1 to lasttable;
                    if (tracknum[p,c]= "yes" AND tracknum[p,d]= "yes" and
tracknum[p,z]= "yes")
                      begin;
```

```
                        steporder[r,n]=p;
                        r=r+1;
                      end;
                    else
                      donext;
                  end;
                  n=n+1;
                  totalsteporderrow[n-1]=(r-1);
                  r=1;
                end;
              end;
          end;
        /* no aquatar, yes extra1*/
        else
          begin;
            do x=firstextra1col to lastextra1col;
              if (totalextra2=1)
                begin;
                  do p=1 to lasttable;
                    if (tracknum[p,c]= "yes" and tracknum[p,d]= "yes" and
tracknum[p,x]= "yes")
                      begin;
                        steporder[r,n]=p;
                        r=r+1;
                      end;
                    else
                      donext;
                  end;
                  n=n+1;
                  totalsteporderrow[n-1]=(r-1);
                  r=1;
                end;
              /* no aquatar, yes extra1, yes extra2 */
              else
                begin;
                  do z=firstextra2col to lastextra2col;
                    do p=1 to lasttable;
                      if (tracknum[p,c]= "yes" and tracknum[p,d]= "yes" and
tracknum[p,x]= "yes" and tracknum[p,z]= "yes")
                        begin;
                          steporder[r,n]=p;
                          r=r+1;
                        end;
                      else
                        donext;
                    end;
                    n=n+1;
                    totalsteporderrow[n-1]=(r-1);
                    r=1;
                  /* end of extra2 loop */
                  end;
                end;
            /* end of extra1 loop */
            end;
          end;
      end;
    /* yes aquatar */
    else
      begin;
        do a=firstaquacol to lastaquacol;
          if (totalextra1=1)
            begin;
              if (totalextra2 =1)
                begin;
                  do p=1 to lasttable;
                    if (tracknum[p,a]= "yes" and tracknum[p,c]= "yes" and
tracknum[p,d]= "yes")
```

```
                    begin;
                      steporder[r,n]=p;
                      r=r+1;
                    end;
                  else
                    donext;
                end;
                n=n+1;
                totalsteporderrow[n-1]=(r-1);
                r=1;
              end;
            /* yes aquatar no extra1, yes extra2 */
            else
              begin;
                do z=firstextra2col to lastextra2col;
                  do p=1 to lasttable;
                    if (tracknum[p,a]= "yes" and tracknum[p,c]= "yes" and
tracknum[p,d]= "yes" and tracknum[p,z]= "yes")
                      begin;
                        steporder[r,n]=p;
                        r=r+1;
                      end;
                    else
                      donext;
                  end;
                  n=n+1;
                  totalsteporderrow[n-1]=(r-1);
                  r=1;
                end;
              end;
          end;
        /* yes aquatar, yes extra1*/
        else
          begin;
            do x=firstextra1col to lastextra1col;
              if (totalextra2 =1)
                begin;
                  do p=1 to lasttable;
                    if (tracknum[p,a]= "yes" and tracknum[p,c]= "yes" and
tracknum[p,d]= "yes" and tracknum[p,x]= "yes")
                      begin;
                        steporder[r,n]=p;
                        r=r+1;
                      end;
                    else
                      donext;
                  end;
                  n=n+1;
                  totalsteporderrow[n-1]=(r-1);
                  r=1;
                end;
              /* yes aquatar, yes extra1, yes extra2 */
              else
                begin;
                  do z=firstextra2col to lastextra2col;
                    do p=1 to lasttable;
                      if (tracknum[p,a]= "yes" and tracknum[p,c]= "yes" and
tracknum[p,d]= "yes" and tracknum[p,x]= "yes" and tracknum[p,z]= "yes")
                        begin;
                          steporder[r,n]=p;
                          r=r+1;
                        end;
                      else
                        donext;
                    end;
                    n=n+1;
                    totalsteporderrow[n-1]=(r-1);
```

```
                    r=1;
                  /* end of  extra2 loop */
                  end;
                end;
              /* end of extra1 loop */
              end;
            end;
        /* end of aquatar loop */
        end;
      end;
  /* end of developer loop */
  end;
/* end of coater loop */
end;
lastcolsteporder=n-1;

/* this routine sorts for tables with the exact processing equipment without the
DEVELOPER track */

r=1;
n=1;
do c=firstcoatercol to lastcoatercol;
  do s=firststeppercol to laststeppercol;
    if (totalaquatars=1)
      begin;
        if (totalextra1=1)
          begin;
            if (totalextra2=1)
              begin;
                do p=1 to lasttable;
                  if (tracknum[p,s]="yes" AND tracknum[p,c]="yes")
                    begin;
                      devorder[r,n]=p;
                      r=r+1;
                    end;
                  else
                    donext;
                end;
                n=n+1;
                totaldevorderrow[n-1]=(r-1);
                r=1;
              end;
            /* no aquatar yes extra2 */
            else
              begin;
                do z=firstextra2col to lastextra2col;
                  do p=1 to lasttable;
                    if (tracknum[p,s]= "yes" AND tracknum[p,c]= "yes" and
tracknum[p,z]= "yes")
                      begin;
                        devorder[r,n]=p;
                        r=r+1;
                      end;
                    else
                      donext;
                end;
                n=n+1;
                totaldevorderrow[n-1]=(r-1);
                r=1;
              end;
            end;
          end;
        /* no aquatar, yes extra1*/
        else
          begin;
            do x=firstextra1col to lastextra1col;
              if (totalextra2=1)
```

```
                begin;
                  do p=1 to lasttable;
                    if (tracknum[p,s]= "yes" and tracknum[p,c]= "yes" and
tracknum[p,x]= "yes")
                      begin;
                        devorder[r,n]=p;
                        r=r+1;
                      end;
                    else
                      donext;
                  end;
                  n=n+1;
                  totaldevorderrow[n-1]=(r-1);
                  r=1;
                end;
              /* no aquatar, yes extra1, yes extra2 */
              else
                begin;
                  do z=firstextra2col to lastextra2col;
                    do p=1 to lasttable;
                      if (tracknum[p,s]= "yes" and tracknum[p,c]= "yes" and
tracknum[p,x]= "yes" and tracknum[p,z]= "yes")
                        begin;
                          devorder[r,n]=p;
                          r=r+1;
                        end;
                      else
                        donext;
                    end;
                    n=n+1;
                    totaldevorderrow[n-1]=(r-1);
                    r=1;
                  /* end of extra2 loop */
                  end;
                end;
            /* end of extra1 loop */
            end;
          end;
      end;
    /* yes aquatar */
    else
      begin;
        do a=firstaquacol to lastaquacol;
          if (totalextra1=1)
            begin;
              if (totalextra2 =1)
                begin;
                  do p=1 to lasttable;
                    if (tracknum[p,a]= "yes" and tracknum[p,s]= "yes" and
tracknum[p,c]= "yes")
                      begin;
                        devorder[r,n]=p;
                        r=r+1;
                      end;
                    else
                      donext;
                  end;
                  n=n+1;
                  totaldevorderrow[n-1]=(r-1);
                  r=1;
                end;
              /* yes aquatar no extra1, yes extra2 */
              else
                begin;
                  do z=firstextra2col to lastextra2col;
                    do p=1 to lasttable;
```

```
                    if (tracknum[p,a]= "yes" and tracknum[p,s]= "yes" and
tracknum[p,c]= "yes" and tracknum[p,z]= "yes")
                      begin;
                        devorder[r,n]=p;
                        r=r+1;
                      end;
                    else
                      donext;
                  end;
                  n=n+1;
                  totaldevorderrow[n-1]=(r-1);
                  r=1;
                end;
              end;
            end;
          /* yes aquatar, yes extra1*/
          else
            begin;
              do x=firstextra1col to lastextra1col;
                if (totalextra2 =1)
                  begin;
                    do p=1 to lasttable;
                      if (tracknum[p,a]= "yes" and tracknum[p,s]= "yes" and
tracknum[p,c]= "yes" and tracknum[p,x]= "yes")
                        begin;
                          devorder[r,n]=p;
                          r=r+1;
                        end;
                      else
                        donext;
                    end;
                    n=n+1;
                    totaldevorderrow[n-1]=(r-1);
                    r=1;
                  end;
                /* yes aquatar, yes extra1, yes extra2 */
                else
                  begin;
                    do z=firstextra2col to lastextra2col;
                      do p=1 to lasttable;
                        if (tracknum[p,a]= "yes" and tracknum[p,s]= "yes" and
tracknum[p,c]= "yes" and tracknum[p,x]= "yes" and tracknum[p,z]= "yes")
                          begin;
                            devorder[r,n]=p;
                            r=r+1;
                          end;
                        else
                          donext;
                      end;
                      n=n+1;
                      totaldevorderrow[n-1]=(r-1);
                      r=1;
                    /* end of  extra2 loop */
                    end;
                  end;
              /* end of extra1 loop */
              end;
            end;
        /* end of aquatar loop */
        end;
      end;
  /* end of stepper loop */
  end;
/* end of coater loop */
end;
lastcoldevorder=n-1;
```

```
/* this routine sorts for tables with the exact processing equipment without the
AQUATAR track */

r=1;
n=1;

if (totalaquatars GT 1)
  begin;
    do c=firstcoatercol to lastcoatercol;
      do s=firststeppercol to laststeppercol;
        do d=firstdevcol to lastdevcol;
          if (totalextra1=1)
            begin;
              if (totalextra2=1)
                begin;
                  do p=1 to lasttable;
                    if (tracknum[p,s]="yes" AND tracknum[p,c]="yes" AND
tracknum[p,d]="yes")
                      begin;
                        aquaorder[r,n]=p;
                        r=r+1;
                      end;
                    else
                      donext;
                  end;
                  n=n+1;
                  totalaquaorderrow[n-1]=(r-1);
                  r=1;
                end;
              /* no extra1 yes extra2 */
              else
                begin;
                  do z=firstextra2col to lastextra2col;
                    do p=1 to lasttable;
                      if (tracknum[p,s]= "yes" AND tracknum[p,c]= "yes" AND
tracknum[p,z]= "yes" AND tracknum[p,d]="yes")
                        begin;
                          aquaorder[r,n]=p;
                          r=r+1;
                        end;
                      else
                        donext;
                    end;
                    n=n+1;
                    totalaquaorderrow[n-1]=(r-1);
                    r=1;
                  end;
                end;
            end;
          /* yes extra1*/
          else
            begin;
              do x=firstextra1col to lastextra1col;
                if (totalextra2=1)
                  begin;
                    do p=1 to lasttable;
                      if (tracknum[p,s]= "yes" and tracknum[p,c]= "yes" and
tracknum[p,x]= "yes" and tracknum[p,d]="yes")
                        begin;
                          aquaorder[r,n]=p;
                          r=r+1;
                        end;
                      else
                        donext;
                    end;
                    n=n+1;
                    totalaquaorderrow[n-1]=(r-1);
```

```
                    r=1;
                  end;
                /* yes extra1, yes extra2 */
                else
                  begin;
                    do z=firstextra2col to lastextra2col;
                      do p=1 to lasttable;
                        if (tracknum[p,s]= "yes" and tracknum[p,c]= "yes" and
tracknum[p,x]= "yes" and tracknum[p,z]= "yes" and tracknum[p,d]="yes")
                          begin;
                            aquaorder[r,n]=p;
                            r=r+1;
                          end;
                        else
                          donext;
                      end;
                      n=n+1;
                      totalaquaorderrow[n-1]=(r-1);
                      r=1;
                    /* end of extra2 loop */
                    end;
                  end;
              /* end of extra1 loop */
              end;
            end;
        /* end of developer loop */
        end;
      /* end of stepper loop */
      end;
    /* end of coater loop */
    end;
    lastcolaquaorder=n-1;
  end;
else
  continue;


/* this routine sorts for tables with the exact processing equipment without the
EXTRA1 track */

r=1;
n=1;
if (totalextra1 GT 1)
  begin;
    do c=firstcoatercol to lastcoatercol;
      do s=firststeppercol to laststeppercol;
        do d=firstdevcol to lastdevcol;
          if (totalaquatars=1)
            begin;
              if (totalextra2 =1)
                begin;
                  do p=1 to lasttable;
                    if (tracknum[p,s]="yes" AND tracknum[p,c]="yes" AND
tracknum[p,d]="yes")
                      begin;
                        extralorder[r,n]=p;
                        r=r+1;
                      end;
                    else
                      donext;
                  end;
                  n=n+1;
                  totalextralorderrow[n-1]=(r-1);
                  r=1;
                end;
              /* no aquatar yes extra2 */
```

```
              else
                begin;
                  do z=firstextra2col to lastextra2col;
                    do p=1 to lasttable;
                      if (tracknum[p,s]= "yes" AND tracknum[p,c]= "yes" AND
tracknum[p,z]= "yes" AND tracknum[p,d]="yes")
                        begin;
                          extra1order[r,n]=p;
                          r=r+1;
                        end;
                      else
                        donext;
                    end;
                    n=n+1;
                    totalextra1orderrow[n-1]=(r-1);
                    r=1;
                  end;
                end;
            end;
          /* yes aquatar*/
          else
            begin;
              do a=firstaquacol to lastaquacol;
                if (totalextra2=1)
                  begin;
                    do p=1 to lasttable;
                      if (tracknum[p,s]= "yes" and tracknum[p,c]= "yes" and
tracknum[p,a]= "yes" and tracknum[p,d]="yes")
                        begin;
                          extra1order[r,n]=p;
                          r=r+1;
                        end;
                      else
                        donext;
                    end;
                    n=n+1;
                    totalextra1orderrow[n-1]=(r-1);
                    r=1;
                  end;
                /* yes aquatar, yes extra2 */
                else
                  begin;
                    do z=firstextra2col to lastextra2col;
                      do p=1 to lasttable;
                        if (tracknum[p,s]= "yes" and tracknum[p,c]= "yes" and
tracknum[p,a]= "yes" and tracknum[p,z]= "yes" and tracknum[p,d]="yes")
                          begin;
                            extra1order[r,n]=p;
                            r=r+1;
                          end;
                        else
                          donext;
                      end;
                      n=n+1;
                      totalextra1orderrow[n-1]=(r-1);
                      r=1;
                    /* end of extra2 loop */
                    end;
                  end;
              /* end of aquatar loop */
              end;
            end;
        /* end of developer loop */
        end;
      /* end of stepper loop */
      end;
    /* end of coater loop */
```

```
    end;
    lastcolextra1order=n-1;;
  end;
else
  continue;

/* this routine sorts for tables with the exact processing equipment without the
EXTRA2 track */

r=1;
n=1;
if (totalextra2 GT 1)
  begin;
    do c=firstcoatercol to lastcoatercol;
      do s=firststeppercol to laststeppercol;
        do d=firstdevcol to lastdevcol;
          if (totalaquatars=1)
            begin;
              if (totalextra1 =1)
                begin;
                  do p=1 to lasttable;
                    if (tracknum[p,s]="yes" AND tracknum[p,c]="yes" AND
tracknum[p,d]="yes")
                      begin;
                        extra2order[r,n]=p;
                        r=r+1;
                      end;
                    else
                      donext;
                  end;
                  n=n+1;
                  totalextra2orderrow[n-1]=(r-1);
                  r=1;
                end;
              /* no aquatar yes extra1 */
              else
                begin;
                  do x=firstextra1col to lastextra1col;
                    do p=1 to lasttable;
                      if (tracknum[p,s]= "yes" AND tracknum[p,c]= "yes" AND
tracknum[p,x]= "yes" AND tracknum[p,d]="yes")
                        begin;
                          extra2order[r,n]=p;
                          r=r+1;
                        end;
                      else
                        donext;
                    end;
                    n=n+1;
                    totalextra2orderrow[n-1]=(r-1);
                    r=1;
                  end;
                end;
            end;
          /* yes aquatar*/
          else
            begin;
              do a=firstaquacol to lastaquacol;
                if (totalextra1=1)
                  begin;
                    do p=1 to lasttable;
                      if (tracknum[p,s]= "yes" and tracknum[p,c]= "yes" and
tracknum[p,a]= "yes" and tracknum[p,d]="yes")
                        begin;
                          extra2order[r,n]=p;
                          r=r+1;
                        end;
```

```
                    else
                      donext;
                  end;
                  n=n+1;
                  totalextra2orderrow[n-1]=(r-1);
                  r=1;
                end;
              /* yes aquatar, yes extra2 */
              else
                begin;
                  do x=firstextra1col to lastextra1col;
                    do p=1 to lasttable;
                      if (tracknum[p,s]= "yes" and tracknum[p,c]= "yes" and
tracknum[p,a]= "yes" and tracknum[p,x]= "yes" and tracknum[p,d]="yes")
                        begin;
                          extra2order[r,n]=p;
                          r=r+1;
                        end;
                      else
                        donext;
                  end;
                  n=n+1;
                  totalextra2orderrow[n-1]=(r-1);
                  r=1;
                /* end of extra1 loop */
                end;
              end;
            /* end of aquatar loop */
            end;
          end;
        /* end of developer loop */
        end;
      /* end of stepper loop */
      end;
    /* end of coater loop */
    end;
    lastcolextra2order=n-1;
  end;
else
  continue;

/* This section creates the tables that are passed to MATLAB for analysis */

do n=1 to lasttable;
  lastrowtable[n]=LASTROW(tablenum[n]);
end;

/* coater ordering from table numbers located in coatorder[ , ]. Each element in
coatorder[ , ] is a table number, each row contains similarly processed */

do c=1 to lastcolcoatorder;
  if (totalcoatorderrow[c]<2)
    begin;
      donext;
    end;
  else
/* this finds the minimum number of rows in the tables with the same equipment
ids (except for coater) */
    begin;
      minrows[c]=LASTROW(tablenum[coatorder[1,c]]);
      do r=2 to totalcoatorderrow[c];
        if (LASTROW(tablenum[coatorder[r,c]]) LT minrows[c])
          begin;
            minrows[c]=LASTROW(tablenum[coatorder[r,c]]);
          end;
      end;
    end;
```

```
  coatmatlab[c]=CAT("coatmatlabin",c);
  MAKETABLE(coatmatlab[c]);
  I=1;
  Do While (I LT minrows[c]);
    Do r=1 to totalcoatorderrow[c];
      ADD ROWS TO TABLE(coatmatlab[c]) FROM rows i TO (i+sites*wafers-1)of
    TABLE(tablenum[coatorder[r,c]]);
    end;
    I=I+sites*wafers;
  end;
  coatcds[c]=CAT("coat_cds",c);
  MAKETABLE(coatcds[c]);
  ADD COLS TO TABLE(coatcds[c]) FROM col cddata of TABLE(coatmatlab[c]);
  Call
$EZ_WRITEFILE(coatcds[c],cat("freq/",coatcds[c]),1,LASTROW(COATCDS[C]),",",FALSE,
empty, empty, empty, true);
end;

/* stepper ordering from table numbers located in steporder[ , ]. Each column in
steporder[ , ] is a table number */

do c=1 to lastcolsteporder;
  if (totalsteporderrow[c]<2)
    begin;
      donext;
    end;
  else
/* this finds the minimum number of rows in the tables with the same equipment
ids (except for stepper) */
    begin;
      minrows[c]=LASTROW(tablenum[steporder[1,c]]);
      do r=2 to totalsteporderrow[c];
        if (LASTROW(tablenum[steporder[r,c]]) LT minrows[c])
          begin;
            minrows[c]=LASTROW(tablenum[steporder[r,c]]);
          end;
      end;
    end;
  stepmatlab[c]=CAT("stepmatlabin",c);
  MAKETABLE(stepmatlab[c]);
  I=1;
  Do While (I LT minrows[c]);
    Do r=1 to totalsteporderrow[c];
      ADD ROWS TO TABLE(stepmatlab[c]) FROM rows i TO (i+sites*wafers-1)of
TABLE(tablenum[steporder[r,c]]);
    end;
    I=I+sites*wafers;
  end;
  stepcds[c]=CAT("step_cds",c);
  MAKETABLE(stepcds[c]);
  ADD COLS TO TABLE(stepcds[c]) FROM col cddata of TABLE(stepmatlab[c]);
  Call $EZ_WRITEFILE(stepcds[c],
cat("freq/",stepcds[c]),1,LASTROW(stepCDS[C]),",",FALSE, empty, empty, empty,
true);

end;

/* developer ordering from table numbers located in devorder[ , ]. Each column in
devorder[ , ] is a table number */

do c=1 to lastcoldevorder;
  if (totaldevorderrow[c]<2)
    begin;
      donext;
    end;
  else
```

```
/* this finds the minimum number of rows in the tables with the same equipment
ids (except for developer) */
    begin;
      minrows[c]=LASTROW(tablenum[devorder[1,c]]);
      do r=2 to totaldevorderrow[c];
        if (LASTROW(tablenum[devorder[r,c]]) LT minrows[c])
          begin;
            minrows[c]=LASTROW(tablenum[devorder[r,c]]);
          end;
      end;
    end;
  devmatlab[c]=CAT("devmatlabin",c);
  MAKETABLE(devmatlab[c]);
  I=1;
  Do While (I LT minrows[c]);
    Do r=1 to totaldevorderrow[c];
      ADD ROWS TO TABLE(devmatlab[c]) FROM rows i TO (i+sites*wafers-1)of
TABLE(tablenum[devorder[r,c]]);
    end;
    I=I+sites*wafers;
  end;
  devcds[c]=CAT("dev_cds",c);
  MAKETABLE(devcds[c]);
  ADD COLS TO TABLE(devcds[c]) FROM col cddata of TABLE(devmatlab[c]);
  Call $EZ_WRITEFILE(devcds[c],
cat("freq/",devcds[c]),1,LASTROW(devCDS[C]),",",FALSE, empty, empty, empty,
true);

end;

/* aquatar ordering from table numbers located in aquaorder[ , ]. Each column in
aquaorder[ , ] is a table number */

if (totalaquatars GT 1)
  begin;
    do c=1 to lastcolaquaorder;
      if (totalaquaorderrow[c]<2)
        begin;
          donext;
        end;
      else
      /* this finds the minimum number of rows in the tables with the same
    equipment ids (except for aquatar) */
        begin;
          minrows[c]=LASTROW(tablenum[aquaorder[1,c]]);
          do r=2 to totalaquaorderrow[c];
            if (LASTROW(tablenum[aquaorder[r,c]]) LT minrows[c])
              begin;
                minrows[c]=LASTROW(tablenum[aquaorder[r,c]]);
              end;
          end;
        end;
      aquamatlab[c]=CAT("aquamatlabin",c);
      MAKETABLE(aquamatlab[c]);
      I=1;
      Do While (I LT minrows[c]);
        Do r=1 to totalaquaorderrow[c];
          ADD ROWS TO TABLE(aquamatlab[c]) FROM rows i TO (i+sites*wafers-1)of
TABLE(tablenum[aquaorder[r,c]]);
        end;
        I=I+sites*wafers;
      end;
    aquacds[c]=CAT("aqua_cds",c);
    MAKETABLE(aquacds[c]);
    ADD COLS TO TABLE(aquacds[c]) FROM col cddata of TABLE(aquamatlab[c]);
```

```
    Call $EZ_WRITEFILE(aquacds[c],
cat("freq/",aquacds[c]),1,LASTROW(aquaCDS[C]),",",FALSE, empty, empty, empty,
true);
    end;
  end;

/* extra1 ordering from table numbers located in extra1order[ , ]. Each column in
extra1order[ , ] is a table number */

if (totalextra1 GT 1)
  begin;
    do c=1 to lastcolextra1order;
      if (totalextra1orderrow[c]<2)
        begin;
          donext;
        end;
      else
      /* this finds the minimum number of rows in the tables with the same
    equipment ids (except for extra1) */
        begin;
          minrows[c]=LASTROW(tablenum[extra1order[1,c]]);
          do r=2 to totalextra1orderrow[c];
            if (LASTROW(tablenum[extra1order[r,c]]) LT minrows[c])
              begin;
                minrows[c]=LASTROW(tablenum[extra1order[r,c]]);
              end;
          end;
        end;
      extra1matlab[c]=CAT("extra1matlabin",c);
      MAKETABLE(extra1matlab[c]);
      I=1;
      Do While (I LT minrows[c]);
        Do r=1 to totalextra1orderrow[c];
          ADD ROWS TO TABLE(extra1matlab[c]) FROM rows i TO (i+sites*wafers-1)of
TABLE(tablenum[extra1order[r,c]]);
        end;
        I=I+sites*wafers;
      end;
    extra1cds[c]=CAT("extra1_cds",c);
    MAKETABLE(extra1cds[c]);
    ADD COLS TO TABLE(extra1cds[c]) FROM col cddata of TABLE(extra1matlab[c]);
    Call $EZ_WRITEFILE(extra1cds[c],
cat("freq/",extra1cds[c]),1,LASTROW(extra1CDS[C]),",",FALSE, empty, empty, empty,
true);
    end;
  end;

/* extra2 ordering from table numbers located in extra2order[ , ]. Each column in
extra2order[ , ] is a table number */

if (totalextra2 GT 1)
  begin;
    do c=1 to lastcolextra2order;
      if (totalextra2orderrow[c]<2)
        begin;
          donext;
        end;
      else
      /* this finds the minimum number of rows in the tables with the same
    equipment ids (except for extra2) */
        begin;
          minrows[c]=LASTROW(tablenum[extra2order[1,c]]);
          do r=2 to totalextra2orderrow[c];
            if (LASTROW(tablenum[extra2order[r,c]]) LT minrows[c])
              begin;
                minrows[c]=LASTROW(tablenum[extra2order[r,c]]);
              end;
```

```
          end;
        end;
      extra2matlab[c]=CAT("extra2matlabin",c);
      MAKETABLE(extra2matlab[c]);
      I=1;
      Do While (I LT minrows[c]);
        Do r=1 to totalextra2orderrow[c];
          ADD ROWS TO TABLE(extra2matlab[c]) FROM rows i TO (i+sites*wafers-1)of
TABLE(tablenum[extra2order[r,c]]);
        end;
        I=I+sites*wafers;
      end;
    extra2cds[c]=CAT("extra2_cds",c);
    MAKETABLE(extra2cds[c]);
    ADD COLS TO TABLE(extra2cds[c]) FROM col cddata of TABLE(extra2matlab[c]);
    Call $E2_WRITEFILE(extra2cds[c],
cat("freq/",extra2cds[c]),1,LASTROW(extra2CDS[C]),",",FALSE, empty, empty, empty,
true);
    end;
  end;

/* end of program */
end;
```

Appendix C

The MATLAB Modeling Program

Written by Erin Chapman

Available by request from

Dr.David McLean

University of Ottawa

Department of Chemical Engineering

```
% this program opens the file and runs the periodogram analysis on it.
clear;

filename=input('Please enter the file name for frequency analysis  ','s');

sites=input('Please enter the number of sites measured per wafer   ');
wafers=input('Please enter the number of wafers measured per batch ');
ofile=fopen(filename,'r');
OS=fscanf(ofile,'%10f');
original=OS;
target=input('Please enter the cd target value  ');
OS=OS-target;
% block;
% OS=OS+blocksignal';
pts=length(OS);
signalmodel(:,1)=zeros(pts,1);
min1=0;
for (i=1:1:pts)
   min1=min1+(OS(i,1))^2;
end
% this determines appropriate periods to use in the analysis
if (pts/(6*wafers*sites) > 1)
   for (T=sites:sites:(pts/4))
      if (rem(pts,T)==0)
         period1=T;
         clear impulse;
         % this is where the impulse signal is created
         for (j=1:1:period1)
            m=1;
            % k is the number of repeating periods
            for (k=1:1:(pts/period1))
               for (i=1:1:period1)
                  if(i==j)
                     a=0.1;
                  else
                     a=0;
                  end
                  impulse(m,j)=a;
                  m=m+1;
               end
```

```
            end
         end
         for (j=(period1+1):1:period1*2)
            m=1;
            % k is the number of repeating period
            for (k=1:1:(pts/period1))
               for (i=1:1:period1)
                  if(i==(j-period1))
                     a=-0.1;
                  else
                     a=0;
                  end
                  impulse(m,j)=a;
                  m=m+1;
               end
            end
         end
         model=1;
         % this is where the signal is modeled
         while(model > 0)
            model=0;
            for (j=1:1:period1*2)
               ss=0;
               signal1=OS(:,1)-impulse(:,j);
               for (e=1:1:pts)
                  ss=ss+(signal1(e,1))^2;
               end
               if (ss < min1)
                  model=j
                  min1=ss
                  signalmodel=signalmodel+impulse(:,model);
                  OS=OS-impulse(:,model);
               end
            end
         end
      end
   end
else
   disp('not enough data');
end
```

Appendix D

The MATLAB Simulation Program

Written by Erin Chapman

Available by request from

Dr.David McLean

University of Ottawa

Department of Chemical Engineering

```
clear;

stdev=0.1;

for (repeats=1:1:3)

   numruns=20;

   impulsestrength=0.1;

   %this program tests signal:noise ratios

   p=0;

   % creating the signal for detection
   for (period=5:5:45);
      p=p+1
      n=0;
      for (numberofperiods=60:20:120);
         n=n+1
         exactsum(p,n)=0;
         pts=period*(numberofperiods);
         m=1;
         clear block;
         for j=1:1:numberofperiods;
            for i=1:1:period;
               if (i < 2)
                  a=1;
               else
                  a=0;
               end
               block(m)=a;
               m=m+1;
            end
         end
         original=block';
         % the "runs" loop tests how repeatable signal detection is.
         sucess=0;
         for runs=1:1:numruns;
            randn('state',sum(100*clock));
```

```
            OS=original+stdev*randn(pts,1);
            clear signalmodel;
            signalmodel(:,1)=zeros(pts,1);
            min1=0;
            % min1 is the variable used to determine if a better
signalmodel has been found.
            % at the perfect signalmodel (which with randome noise will
never happen)
            % the min1 variable will be zero.
            for var1=1:1:pts
               min1=min1+OS(var1,1)^2;
            end
            % The simulation must be the same as the actual program. The
actual program will not know what
            % periodic behaviour to look for exclusively. This way all
possible periods are covered as per
            % the current sampling scheme of 5 sites per wafer and up to 3
wafers per batch and 3 paths of
            % processing.
            for (T=5:5:45);
               if(rem(pts,T)==0)
                     period1=T;
                     clear impulse;
                     % the impulses are created in columns (j)
                     for(j=1:1:period1)
                        m=1;
                        % k is the number of repeating periods
                        for (k=1:1:(pts/period1))
                           for (i=1:1:period1)
                              if(i==j)
                                 a=impulsestrength;
                              else
                                 a=0;
                              end
                              impulse(m,j)=a;
                              m=m+1;
                           end
                        end
                     end
                     % this creates impulses for the negative signals
```

```
                    for (j=(period1+1):1:period1*2)
                       m=1;
                       % k is the number of repeating periods
                       for (k=1:1:(pts/period1))
                          for (i=1:1:period1)
                             if (i==(j-period1))
                                a=-impulsestrength;
                             else
                                a=0;
                             end
                             impulse(m,j)=a;
                             m=m+1;
                          end
                       end
                    end
                    model=1;
                    % this is where the signal gets modeled
                    % the modeling program continues as long as the
previous iteration changed the signalmodel
                    % the variable model is used as the switch to see if
the signalmodel was changed.
                    while (model>0);
                       model=0;
                       for (j=1:1:period1*2)
                          signal1=OS(:,1)-impulse(:,j);
                          ss=0;
                          for var1=1:1:pts
                             ss=ss+signal1(var1,1)^2;
                          end
                          if(ss < min1)
                             model=j;
                             min1=ss;
                          end
                       end
                       if (model>0)
                          signalmodel=signalmodel+impulse(:,model);
                          OS=OS-impulse(:,model);
                       end
                    end
```

```
                    % This is where we check to see if the signalmodel =
original
                    % for some reason the exact model comes up with a very
small difference when it is correct
                    diff=(original-signalmodel);
                    if (sse(diff)<1*10^-5)
                       % found correct model
                       exact(p,n,runs)=1;
                    else
                       exact(p,n,runs)=0;
                    end
                 end
                 % the if loop to make sure the number of periods is an
integer
              end
              % the period loop that the model is using
              exactsum(p,n)=exact(p,n,runs)+exactsum(p,n);
           end
           % the numruns counter loop
           exactavg(p,n,repeats)=exactsum(p,n)/numruns;
        end
        % the number of periods counter loop
     end
     % the period counter loop
  end
  % the repeats counter loop
```

Appendix E

MATLAB Simulation Results

Written by Erin Chapman

Available by request from

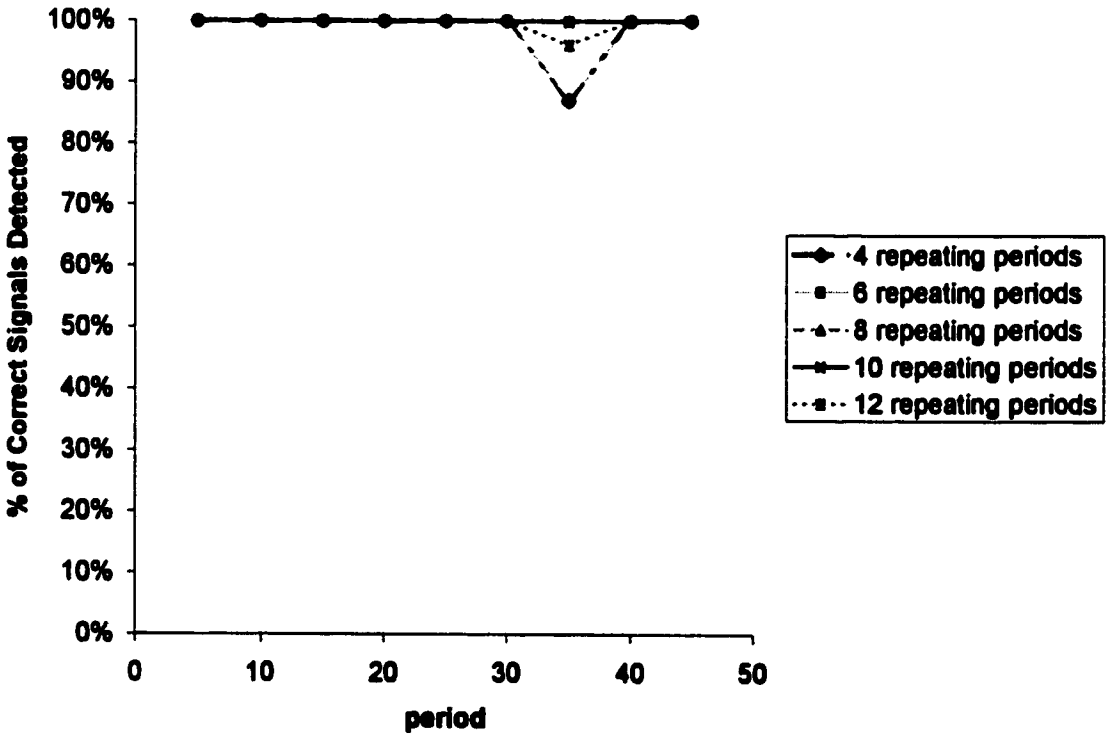
Dr.David McLean

University of Ottawa

Department of Chemical Engineering

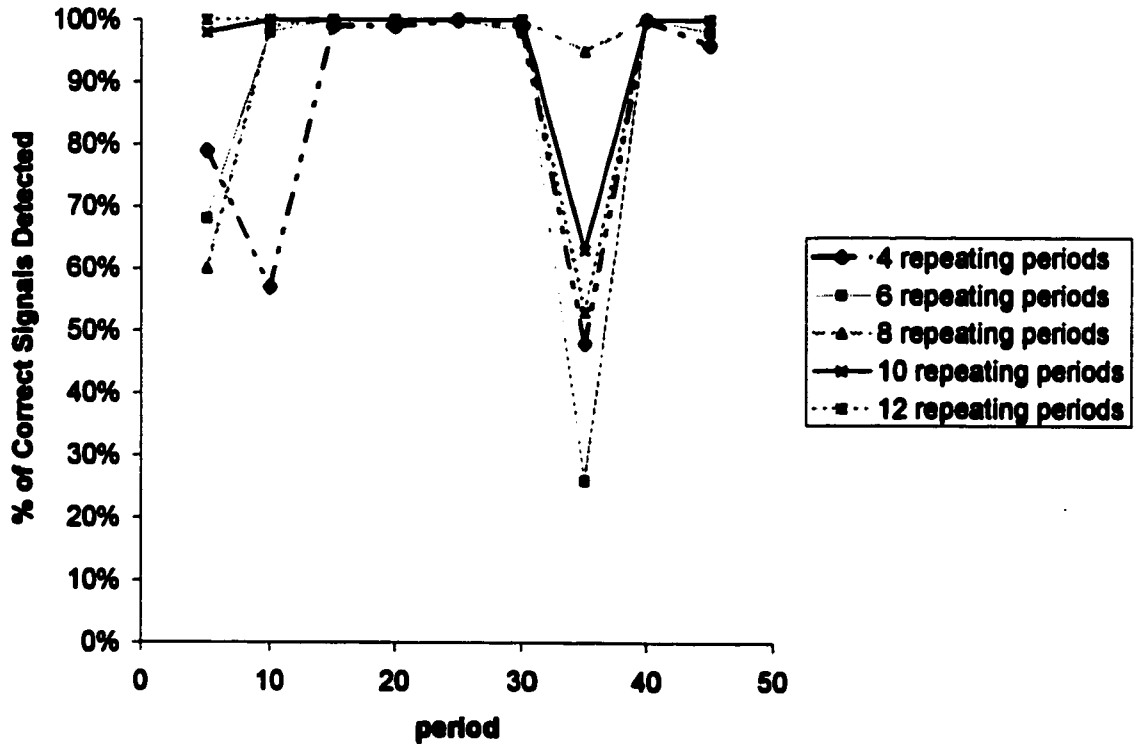
Simulation Results for Signal to Noise Ratio 100:1

100 runs	# of Repeating Periods				
Period	4	6	8	10	12
5	100%	100%	100%	100%	100%
10	100%	100%	100%	100%	100%
15	100%	100%	100%	100%	100%
20	100%	100%	100%	100%	100%
25	100%	100%	100%	100%	100%
30	100%	100%	100%	100%	100%
35	87%	87%	100%	100%	96%
40	100%	100%	100%	100%	100%
45	100%	100%	100%	100%	100%



Simulation Results for Signal to Noise Ratio 50:1

100 runs	# of Repeating Periods				
Period	4	6	8	10	12
5	79%	68%	60%	98%	100%
10	57%	98%	99%	100%	100%
15	99%	100%	100%	100%	100%
20	99%	100%	100%	100%	100%
25	100%	100%	100%	100%	100%
30	99%	98%	100%	100%	100%
35	48%	26%	95%	63%	53%
40	100%	100%	100%	100%	100%
45	96%	98%	100%	100%	100%



Simulation Results for Signal to Noise Ratio 33.3:1

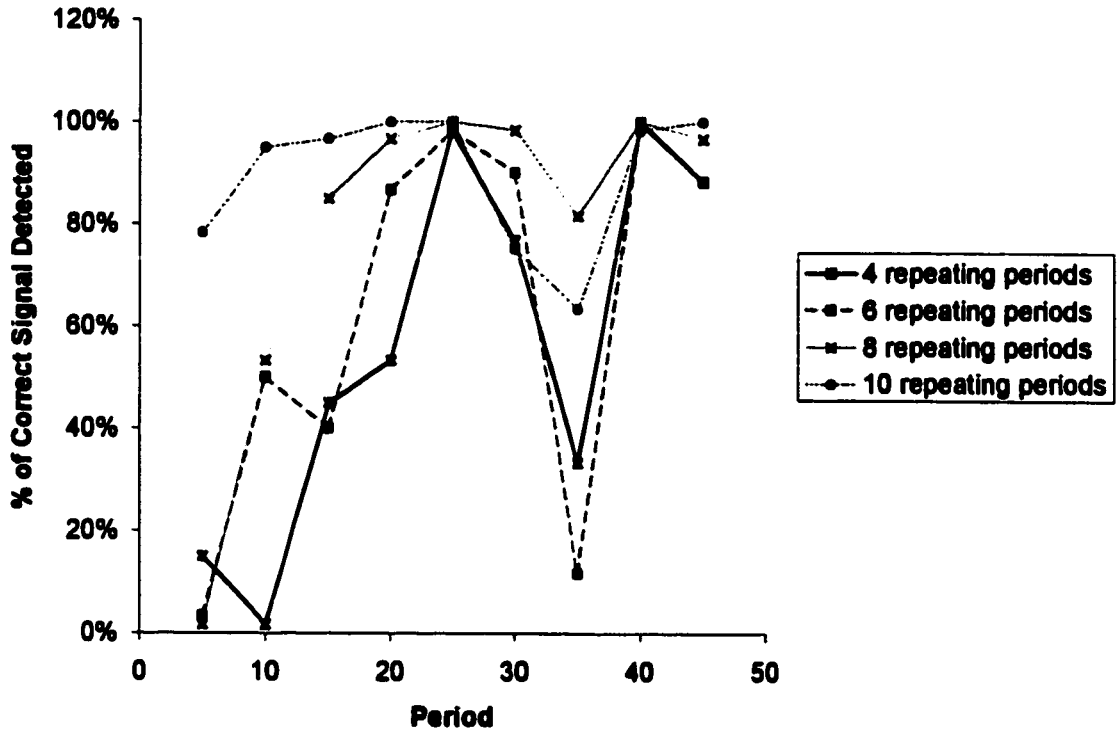
1 20 repeats	# of repeating periods			
period	4	6	8	10
5	0%	10%	0%	70%
10	0%	45%	60%	90%
15	55%	25%	85%	95%
20	55%	80%	100%	100%
25	100%	100%	100%	100%
30	80%	85%	95%	100%
35	30%	10%	75%	50%
40	100%	100%	100%	100%
45	95%	95%	100%	100%

2 20 repeats	# of repeating periods			
period	4	6	8	10
5	30%	0%	0%	50%
10	0%	50%	35%	95%
15	30%	40%	85%	100%
20	45%	90%	90%	100%
25	100%	95%	100%	100%
30	70%	90%	100%	100%
35	45%	5%	80%	30%
40	100%	100%	100%	100%
45	85%	80%	100%	95%

3 20 repeats	# of repeating periods			
period	4	6	8	10
5	15%	0%	5%	70%
10	5%	55%	65%	95%
15	50%	55%	85%	95%
20	60%	90%	100%	100%
25	95%	100%	100%	100%
30	80%	95%	100%	95%
35	25%	20%	90%	40%
40	100%	100%	100%	100%
45	85%	90%	90%	100%

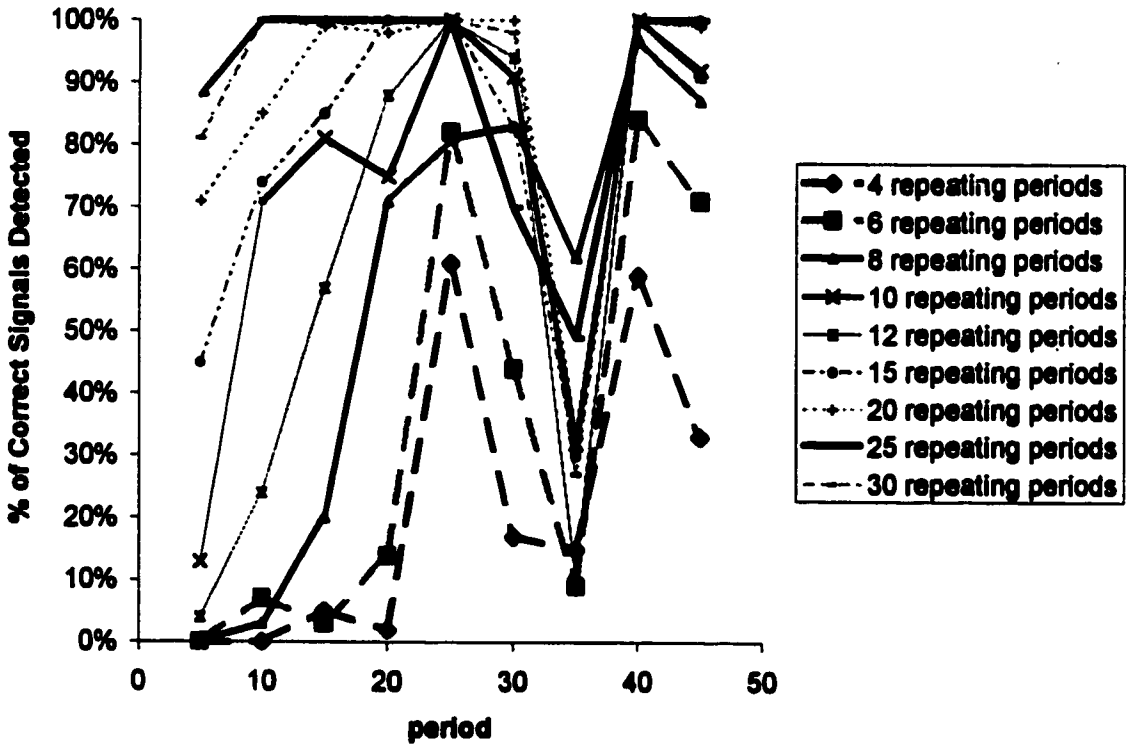
average 20 repeats	# of repeating periods			
period	4	6	8	10
5	15%	3%	2%	78%
10	2%	50%	53%	95%
15	45%	40%	85%	97%
20	53%	87%	97%	100%
25	98%	98%	100%	100%
30	77%	90%	98%	75%
35	33%	12%	82%	63%
40	100%	100%	100%	98%
45	88%	88%	97%	100%

Simulation Results for Signal to Noise Ratio 33.3:1



Simulation Results for Signal to Noise Ratio 25:1

100 runs	# of repeating periods								
Period	4	6	8	10	12	15	20	25	30
5	0%	0%	0%	13%	4%	45%	71%	88%	81%
10	0%	7%	3%	71%	24%	74%	85%	100%	100%
15	5%	3%	20%	81%	57%	85%	99%	100%	99%
20	2%	14%	71%	75%	88%	100%	98%	100%	100%
25	61%	82%	81%	100%	100%	100%	100%	100%	100%
30	17%	44%	83%	91%	94%	83%	100%	70%	98%
35	15%	9%	62%	32%	11%	30%	34%	49%	27%
40	59%	84%	97%	100%	100%	100%	100%	100%	100%
45	33%	71%	87%	92%	91%	100%	99%	100%	99%



Simulation Results for Signal to Noise Ratio 20:1

1 20 repeats

period	# of repeating periods 4	6	8	10	15	20	25	30
5	0%	0%	0%	0%	15%	5%	40%	40%
10	0%	0%	0%	30%	55%	35%	95%	95%
15	0%	0%	5%	30%	40%	85%	100%	85%
20	0%	0%	15%	60%	100%	100%	100%	100%
25	20%	45%	35%	95%	100%	100%	100%	100%
30	10%	5%	25%	60%	70%	100%	85%	95%
35	0%	5%	25%	15%	5%	10%	25%	20%
40	10%	45%	75%	95%	100%	100%	100%	100%
45	15%	25%	40%	85%	100%	100%	100%	95%

2 20 repeats

period	# of repeating periods 4	6	8	10	15	20	25	30
5	0%	0%	0%	0%	10%	15%	55%	40%
10	0%	0%	0%	30%	55%	35%	100%	95%
15	0%	0%	0%	25%	40%	95%	100%	95%
20	0%	10%	15%	40%	95%	95%	100%	100%
25	15%	15%	35%	90%	100%	100%	100%	100%
30	0%	10%	50%	75%	95%	90%	90%	95%
35	10%	0%	25%	25%	25%	0%	55%	25%
40	15%	45%	75%	95%	100%	100%	100%	100%
45	5%	45%	15%	55%	100%	75%	100%	100%

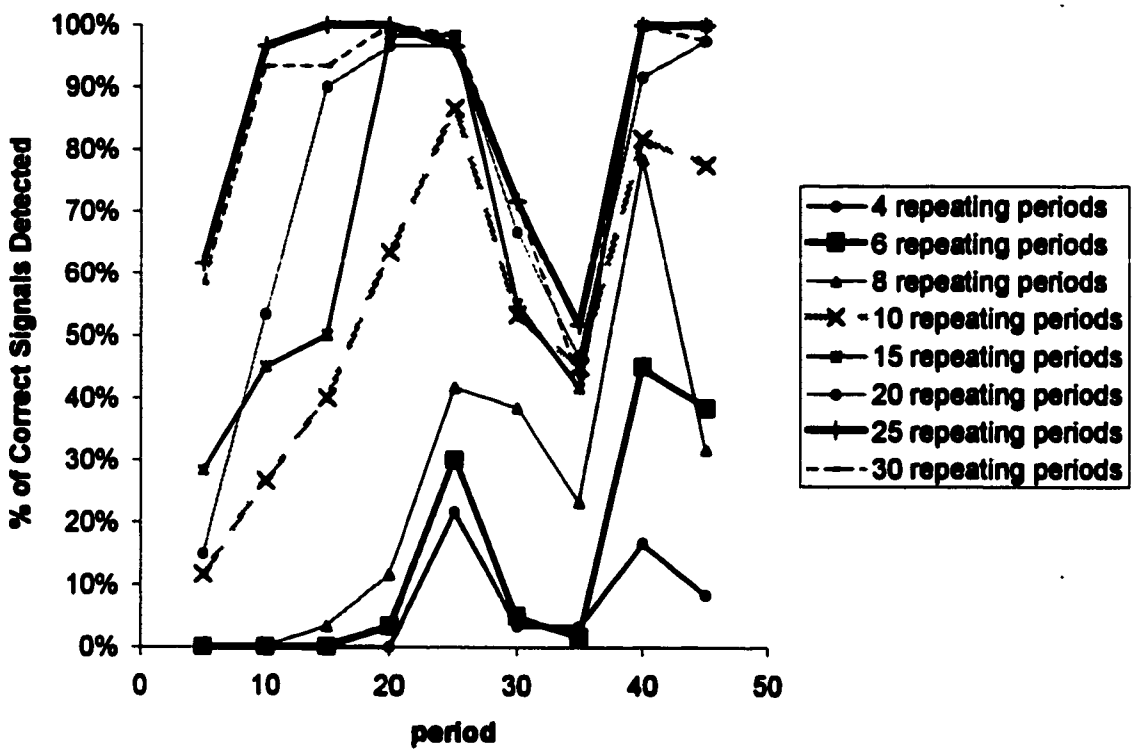
3 20 repeats

period	# of repeating periods 4	6	8	10	15	20	25	30
5	0%	0%	0%	5%	15%	5%	45%	40%
10	0%	0%	0%	25%	40%	30%	95%	90%
15	0%	0%	5%	50%	15%	90%	100%	95%
20	0%	0%	5%	40%	95%	90%	100%	100%
25	30%	30%	55%	90%	100%	100%	100%	100%
30	0%	0%	40%	75%	70%	100%	75%	95%
35	0%	0%	20%	25%	20%	30%	30%	10%
40	25%	45%	85%	95%	100%	100%	100%	100%
45	5%	45%	40%	70%	100%	95%	100%	100%

average 20 repeats

period	# of repeating periods 4	6	8	10	15	20	25	30
5	0%	0%	0%	12%	28%	15%	62%	58%
10	0%	0%	0%	27%	45%	53%	97%	93%
15	0%	0%	3%	40%	50%	90%	100%	93%
20	0%	3%	12%	63%	98%	97%	100%	100%
25	22%	30%	42%	87%	98%	97%	97%	98%
30	3%	5%	38%	53%	55%	67%	72%	72%
35	3%	2%	23%	45%	42%	47%	52%	43%
40	17%	45%	78%	82%	100%	92%	100%	100%
45	8%	38%	32%	78%	100%	98%	100%	98%

Simulation Results for Signal to Noise Ratio 20:1



Simulation Results for Signal to Noise Ratio 16.7:1

1 20 runs

	# of repeating periods				
period	20	40	60	80	100
5	0%	15%	60%	70%	100%
10	0%	70%	95%	100%	100%
15	70%	90%	100%	100%	100%
20	55%	100%	100%	100%	100%
25	100%	100%	100%	100%	100%
30	95%	100%	100%	100%	100%
35	10%	70%	25%	85%	45%
40	100%	100%	100%	100%	100%
45	95%	100%	100%	100%	100%

2 20 runs

	# of repeating periods				
period	20	40	60	80	100
5	5%	5%	70%	75%	100%
10	5%	70%	95%	100%	100%
15	75%	95%	90%	100%	100%
20	35%	100%	100%	100%	100%
25	100%	100%	100%	100%	100%
30	85%	95%	100%	100%	100%
35	5%	65%	15%	95%	65%
40	100%	100%	100%	100%	100%
45	85%	95%	100%	100%	100%

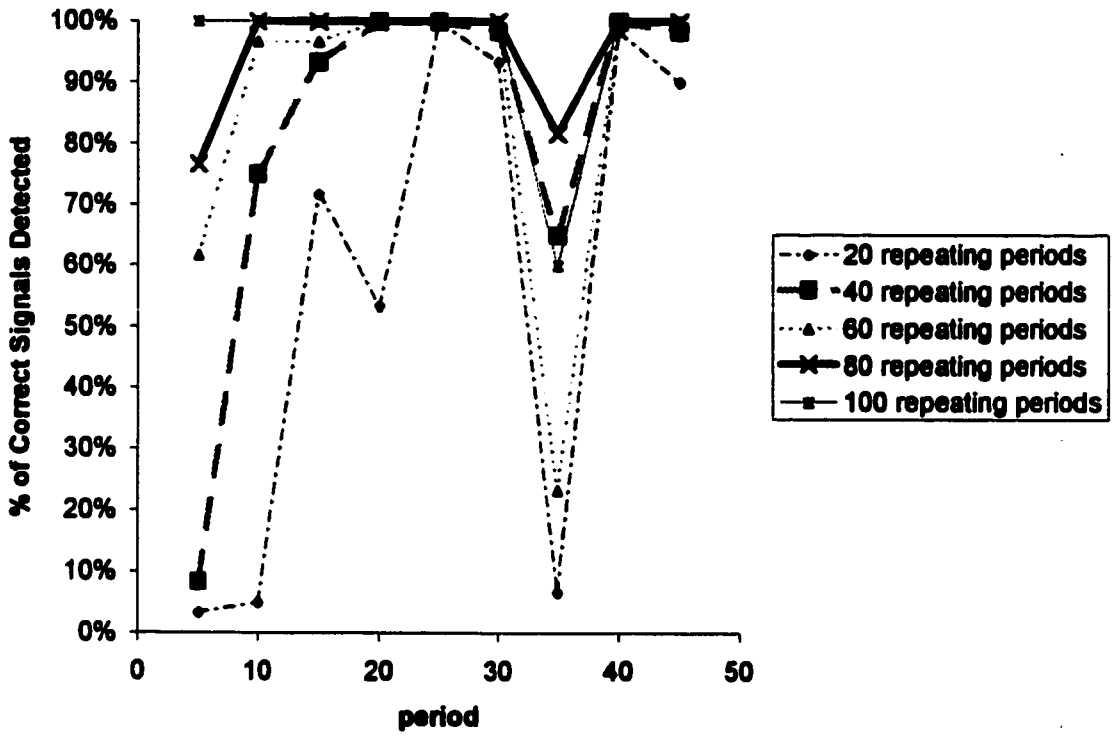
3 20 runs

	# of repeating periods				
period	20	40	60	80	100
5	5%	5%	55%	85%	100%
10	10%	85%	100%	100%	100%
15	70%	95%	100%	100%	100%
20	70%	100%	100%	100%	100%
25	100%	100%	100%	100%	100%
30	100%	100%	100%	100%	100%
35	5%	60%	30%	65%	70%
40	95%	100%	100%	100%	100%
45	90%	100%	100%	100%	100%

average 20 runs

	# of repeating periods				
period	20	40	60	80	100
5	3%	8%	62%	77%	100%
10	5%	75%	97%	100%	100%
15	72%	93%	97%	100%	100%
20	53%	100%	100%	100%	100%
25	100%	100%	100%	100%	100%
30	93%	98%	100%	100%	100%
35	7%	65%	23%	82%	60%
40	98%	100%	100%	100%	100%
45	90%	98%	100%	100%	100%

Simulation Results for Signal to Noise Ratio 16.7:1



Simulation Results for Signal to Noise Ratio 12.5:1

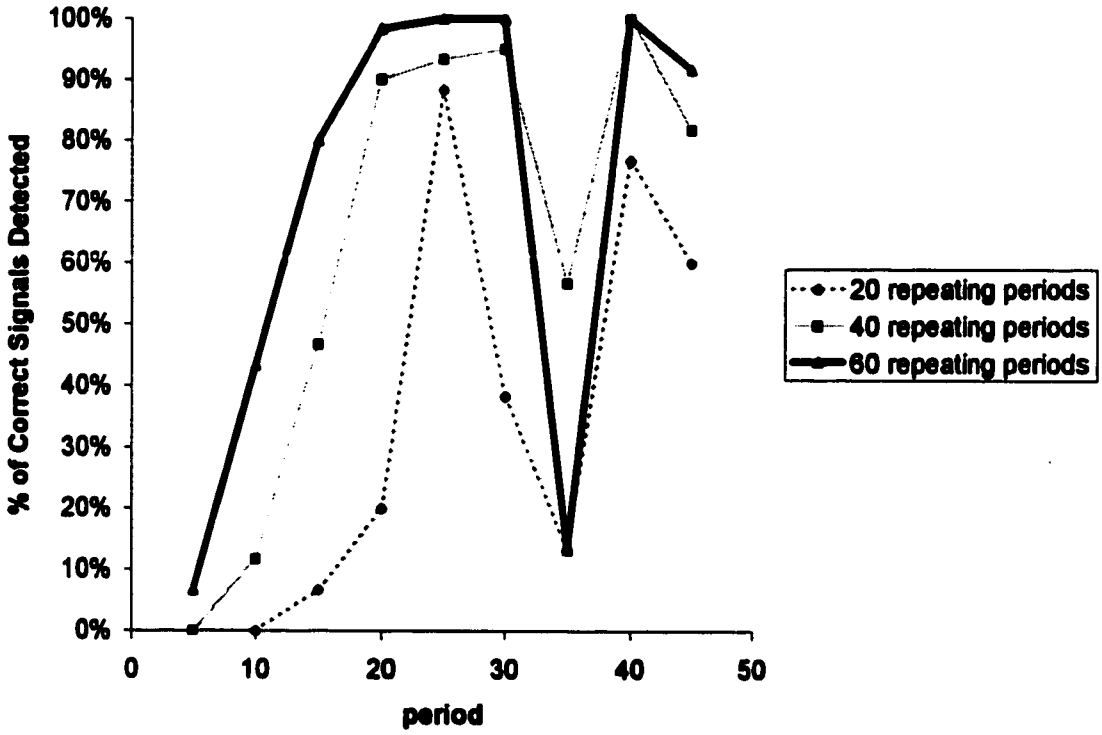
1 20 repeats	# of repeating periods		
period	20	40	60
5	0%	0%	10%
10	0%	15%	45%
15	10%	50%	95%
20	5%	100%	95%
25	85%	85%	100%
30	50%	100%	100%
35	10%	55%	10%
40	90%	100%	100%
45	60%	80%	85%

2 20 repeats	# of repeating periods		
period	20	40	60
5	0%	0%	5%
10	0%	0%	45%
15	5%	35%	70%
20	25%	80%	100%
25	95%	100%	100%
30	25%	100%	100%
35	20%	35%	15%
40	75%	100%	100%
45	55%	80%	100%

3 20 repeats	# of repeating periods		
period	20	40	60
5	0%	0%	5%
10	0%	20%	40%
15	5%	55%	75%
20	30%	90%	100%
25	85%	95%	100%
30	40%	85%	100%
35	10%	80%	15%
40	65%	100%	100%
45	65%	85%	90%

averag 20 repeats	# of repeating periods		
period	20	40	60
5	0%	0%	7%
10	0%	12%	43%
15	7%	47%	80%
20	20%	90%	98%
25	88%	93%	100%
30	38%	95%	100%
35	13%	57%	13%
40	77%	100%	100%
45	60%	82%	92%

Simulation Results for Signal to Noise Ratio 12.5:1



Simulation Results for Signal to Noise Ratio 10:1

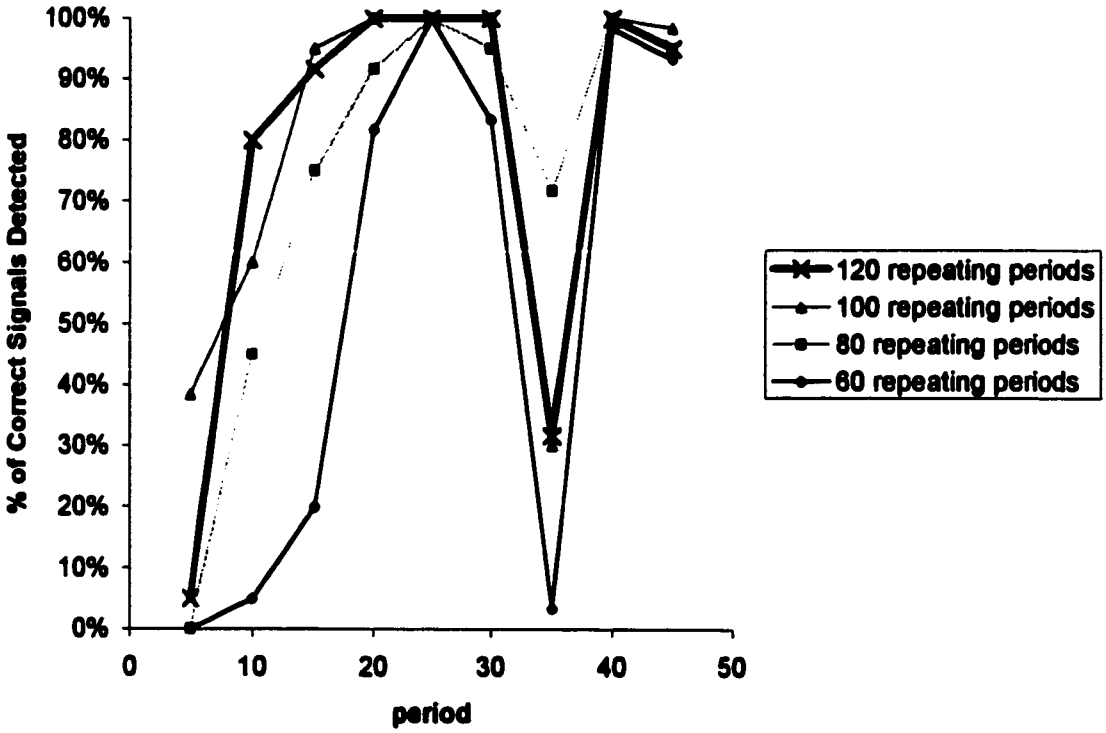
1 20 repeats	# of repeating periods							
period/#of perio	4	6	8	10	60	80	100	120
5	0%	0%	0%	0%	0%	0%	35%	5%
10	0%	0%	0%	0%	10%	45%	55%	75%
15	0%	0%	0%	0%	25%	95%	90%	100%
20	0%	0%	0%	0%	70%	90%	100%	100%
25	0%	0%	0%	5%	100%	100%	100%	100%
30	0%	0%	0%	0%	75%	100%	100%	100%
35	0%	0%	0%	0%	0%	75%	25%	30%
40	0%	0%	0%	0%	100%	100%	100%	100%
45	0%	0%	0%	0%	90%	90%	95%	90%

2 20 repeats	# of repeating periods							
period/#of perio	4	6	8	10	60	80	100	120
5	0%	0%	0%	0%	0%	0%	30%	5%
10	0%	0%	0%	0%	5%	50%	65%	75%
15	0%	0%	0%	0%	5%	70%	95%	90%
20	0%	0%	0%	0%	85%	90%	100%	100%
25	0%	0%	0%	0%	100%	100%	100%	100%
30	0%	0%	0%	0%	85%	95%	100%	100%
35	0%	0%	0%	0%	5%	80%	30%	20%
40	0%	0%	0%	0%	100%	100%	100%	100%
45	0%	0%	0%	0%	90%	100%	100%	95%

3 20 repeats	# of repeating periods							
period/#of perio	4	6	8	10	60	80	100	120
5	0%	0%	0%	0%	0%	0%	50%	5%
10	0%	0%	0%	0%	0%	40%	60%	90%
15	0%	0%	0%	0%	30%	60%	100%	85%
20	0%	0%	0%	0%	90%	95%	100%	100%
25	0%	0%	0%	10%	100%	100%	100%	100%
30	0%	0%	0%	0%	90%	90%	100%	100%
35	0%	0%	0%	0%	5%	60%	35%	45%
40	0%	0%	0%	0%	95%	100%	100%	100%
45	0%	0%	0%	0%	100%	95%	100%	100%

averag 20 repeats	# of repeating periods							
period/#of perio	4	6	8	10	60	80	100	120
5	0%	0%	0%	0%	0%	0%	38%	5%
10	0%	0%	0%	0%	5%	45%	60%	80%
15	0%	0%	0%	0%	20%	75%	95%	92%
20	0%	0%	0%	0%	82%	92%	100%	100%
25	0%	0%	0%	5%	100%	100%	100%	100%
30	0%	0%	0%	0%	83%	95%	100%	100%
35	0%	0%	0%	0%	3%	72%	30%	32%
40	0%	0%	0%	0%	98%	100%	100%	100%
45	0%	0%	0%	0%	93%	95%	98%	95%

Simulation Results for Signal to Noise Ratio 10:1



Appendix F

The MATLAB Periodogram Program

Written by Erin Chapman

Available by request from

Dr.David McLean

University of Ottawa

Department of Chemical Engineering

```
% this program opens the file and runs the periodogram analysis on it.
clear;
filename=input('Please enter the file name for frequency analysis  ','s');
sites=input('Please enter the number of sites measured per wafer   ');
wafers=input('Please enter the number of wafers measured per batch ');
ofile=fopen(filename,'r');
OS=fscanf(ofile,'%10f');
original=OS;
target=input('Please enter the cd target value  ');
OS=OS-target;
% block;
% OS=OS+blocksignal';
pts=length(OS);
signalmodel(:,1)=zeros(pts,1);
min1=0;
for (i=1:1:pts)
   min1=min1+(OS(i,1))^2;
end
% this determines appropriate periods to use in the analysis
if (pts/(6*wafers*sites) > 1)
   for (T=sites:sites:(pts/4))
      if (rem(pts,T)==0)
         period1=T;
         clear impulse;
         % this is where the impulse signal is created
         for (j=1:1:period1)
            m=1;
            % k is the number of repeating periods
            for (k=1:1:(pts/period1))
               for (i=1:1:period1)
                  if(i==j)
                     a=0.1;
                  else
                     a=0;
                  end
                  impulse(m,j)=a;
                  m=m+1;
               end
            end
         end
```

```
            for (j=(period1+1):1:period1*2)
               m=1;
               % k is the number of repeating period
               for (k=1:1:(pts/period1))
                  for (i=1:1:period1)
                     if(i==(j-period1))
                        a=-0.1;
                     else
                        a=0;
                     end
                     impulse(m,j)=a;
                     m=m+1;
                  end
               end
            end
            model=1;
            % this is where the signal is modeled
            while(model > 0)
               model=0;
               for (j=1:1:period1*2)
                  ss=0;
                  signal1=OS(:,1)-impulse(:,j);
                  for (e=1:1:pts)
                     ss=ss+(signal1(e,1))^2;
                  end
                  if (ss < min1)
                     model=j
                     min1=ss
                     signalmodel=signalmodel+impulse(:,model);
                     OS=OS-impulse(:,model);
                  end
               end
            end
         end
      end
   else
      disp('not enough data');
   end
```

Appendix G

Sample Calculations

SAMPLE CALCULATIONS

The 95% confidence limits can be calculated from

$$LCL = \bar{x} - \frac{2s}{\sqrt{n}}$$

$$UCL = \bar{x} + \frac{2s}{\sqrt{n}}$$

where n is the number of observations

x is the sample mean

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}$$

where s is the sample standard deviation

$$s = \sqrt{\frac{\sum_{i=1}^{n} \left(x_i - \bar{x} \right)}{n-1}}$$

In our case, the 3 data points are known from simulation values. Lets use the example of a period of 5 repeating 15 times. The three average values for a signal to noise ratio of 1:20 are: 15%,10% and 15%.

Thus

x =13.33

s =1.712

So a 95 % confidence interval is the mean value of 13.33±1.976