UNDERGRADUATE REPORT

REU Report: Process Integration with Wafer Yield: An Exercise in Computer-Based Modules

by SunJun Park
Advisor: Gary W. Rubloff

U.G. 99-3

ISR develops, applies and teaches advanced methodologies of design and analysis to solve complex, hierarchical, heterogeneous and dynamic problems of engineering technology and systems for industry and government.

ISR is a permanent institute of the University of Maryland, within the Glenn L. Martin Institute of Technology/A. James Clark School of Engineering. It is a National Science Foundation Engineering Research Center.

Web site http://www.isr.umd.edu

Process Integration and Wafer Yield: A Practice in Computer-based Education Modules

By SunJun Park, North Carolina State University, Raleigh
Dr. Gary Rubloff, Advisor
Research Experience for Undergraduates
Institute for Systems Research
University of Maryland at College Park
National Science Foundation

August 5, 1999

Table of Contents:

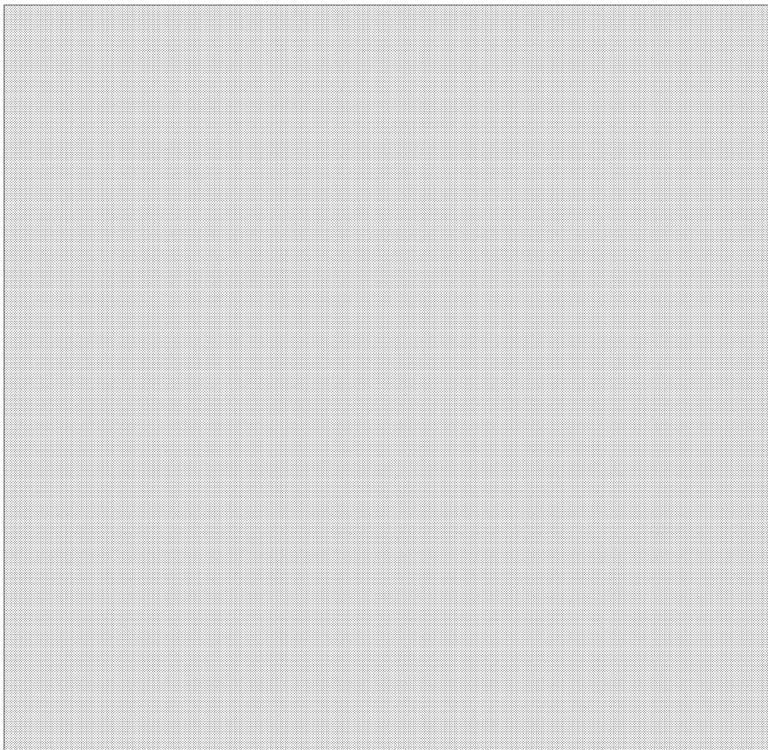
I.	Introduction	3
II.	Problem	3
III.	Tool Selection	4
IV.	The OLE Advantage	4
V.	Excel Algorithm	6
VI.	Graphical User Interface Design	7
VII.	Future Direction of the Project	9
VIII.	Special Thanks	10
IX.	Bibliography	11

I. Introduction

With the growing complexity of modern technology, it is necessary to create new means for teaching the fundamental ideas behind the latest processes. Although current programs do an adequate job, they prove at times to be confusing and difficult to visualize, creating a steep learning curve for basic concepts. Silicon wafer manufacturing is no exception to this problem: with an incredibly wide range of factors affecting yield, it is a difficult, but necessary task to educate workers on the complex processes.

The Institute for Systems Research (ISR) at the University of Maryland (UMD) created an innovative solution to this difficult problem. They proposed taking advantage of the latest technological developments, particularly in the field of information technology, and utilizing it to create powerful, realistic simulations to teach fundamental ideas in a hands-on, interactive environment. The Center for Engineering Learning Systems (CELS) was created by ISR in conjunction with the Human-Computer Interface Lab (HCIL) at UMD to take an engineering approach to the creation of these educational modules.

II. Problem



Improving yield has always been a difficult problem for chip manufacturers. Electronic chips are built on large silicon wafers that are treated and sliced before being pressed into a final product. The problem is that the actual yield per wafer in a modern manufacturing facility is poor, resulting in up to only seventy percent of the

chips actually meeting the necessary standards. The financial costs of this are tremendous: if a wafer can potentially create one hundred thousand dollars in product, but has a mere seventy percent yield, then thirty thousand dollars in product is lost per wafer.

With such large losses in manufacturing, companies have considerable incentive to train their employees in maximizing yield. This module assists in this endeavor by demonstrating how slight variations in several basic factors can make consequential differences in total yield. In addition, it shows users the random variations that naturally occur in the process, giving a realistic portrait of the different factors. This is accomplished by generating random numbers that follows the Gaussian curve. By taking this approach, the module shows users the effects that changes in the standard deviation can make.

III. Tool Selection

Older CELS models were created using Microsoft's Visual Basic development kit. However, the platform was switched to Borland Delphi, a visual kit based on the Object Pascal language, due to several distinct advantages. First, it offered an easy means of creating custom components, a task at which the Microsoft product fell short. Delphi-produced modules also demonstrated faster performance and execution speeds. In addition, they did not require the smorgasbord of additional DLL's to run, making distribution of the software an easier task.

The Windows 95 dependent Delphi was also selected for its efficiency and simplicity. Although other platform-independent languages, such as JAVA, were available, they carried heavy amounts of overhead. Since these modules depended upon Windows-based applications, the platform independence advantage was completely nullified, leaving the other languages with nothing but unnecessary baggage.

Microsoft Excel was selected as the engine to calculate the wafer data. The OLE controls integrated into the software allowed for easy integration into the Delphi-based module. The cell-based nature of silicon wafers was easily represented in the spreadsheet environment. Also, Excel provided all the necessary statistical tools to carry out the advanced random number generation required for the project.

After a prototype of the Delphi-Excel wafer map was created and successfully tested, the entire project was integrated into the SimPLE framework, created by CELS for a concurrently running project. Since it was designed for use with the VisSim software package, several modifications were made to it to combine the two. Further discussion on these changes is documented in section VI.

The question is asked: why use Delphi as a front end to Excel and VisSim? After all, the both software packages come with powerful graphing functions and significant flexibility. The response is simple: while both programs can be used as

training simulations, the representation of data by both is both muddled and confusing. Both also require significant training to properly use them. Most technology workers are competent enough to learn the packages, however, they lack the time, and their employers lack the resources, to become proficient in them. By using a front end Graphical User Interface, such as Delphi, the users can easily train themselves in using the module in a few minutes, then utilize the rest of their time in actual systems training.

IV. The OLE Advantage

The integration of Excel's number-crunching abilities and Delphi's straightforward user interface is made possible by OLE, an attempt by Microsoft to allow data exchange between the numerous components that use the Windows environment.[1] Built upon the Component Object Model (COM), OLE allows data to be transferred between two very different programs, in this case, the Excel spreadsheet and Delphi development environment.

Data transferred into Delphi by OLE is done through a special class of variables located in the COMobj library, known as the **variant**. The variant is the simplest route by which Delphi manipulates COM objects. Through it, OLE objects are created and remotely controlled, in this case, an instance of Excel. The following code fragment shows how a new instance of Excel is created (the lines are numbered for reference purposes only):

```
1: var
2:    excel: variant;
3:    x: string;
4:
5: begin
6:    excel := CreateOleObject('Excel.Application');
7:    excel.workbooks.open('foo.xls');
8:    x := excel.workbooks.sheet[1].cells[1,1];
9: end;
```

Figure 4.1 – Sample code segment demonstrating OLE

After the variant is declared (line 2), the CreateOleObject command is invoked through it (line 6). The specification, 'Excel.Application', indicates that an instance of the Excel application should be created. The declaration associates the object to the variable, allowing the user to manipulate the object through simple commands. In the example, the worksheet foo.xls is opened inside the instance.

It is important to note the organizational hierarchy of the OLE objects: while traditional visual platforms determine their order based on what they are derived from, whereas OLE objects are organized according to what objects can access it and be accessed from it[2]. Thus, in line eight, the Excel Application initiates workbook one, goes to the first sheet, then stores the value of cell [1,1] in x.

Also notice that the data taken from the Excel cell is transferred as a string. When the contents of a cell are read by OLE, it is converted into string format when transferred. If the cell contains an equation, then the result of the equation will be sent not the equation itself. This is convenient for Delphi, since all displayed data is stored in a string format.

There are disadvantages to OLE however. The biggest point speed of the transfer process between Delphi and Excel. It is not necessarily the amount of data, but the number of times OLE must be invoked which causes the slowdowns. While this may be inconsequential when transferring one or two variables, it is a serious problem in this module, where over two hundred variables can be sent at a single time. A partial solution was created for this situation: instead of reading each individual cell, the module would request a Excel range of cells at a single time. This greatly cut down the maximum number of transfers for the wafers, allowing 184 requests to be reduced to a mere two. Unfortunately, twenty-four of the transfers still require individual requests due to their nature. This was reduced even further though, by accessing them only when changes have been made.

V. Excel Algorithm

All the calculations in the module are based on the basic capacitance formula:

$$C = \epsilon A / D$$

Equation 5.1 – Capacitance formula

C is capacitance, A is the capacitance area, D is the oxide thickness, and ϵ is the dielectric constant. The oxide thickness is determined by two factors: the growth time and the growth rate. The latter is determined by this formula as a function of time:

$$B/A\ (T) = D_o e^{-Ea/kT}$$

Equation 5.2 – Oxide Growth Rate function

The D_o coefficient used in this case is for Silicon <100> using dry O_2, which is 3.71×10^6 μm/hr. E_a, the activation energy, is 2.00 eV. T, the temperature of the process, is converted from Celsius to Kelvin while k is set at 8.63×10^{-5}. The actual oxide growth is then calculated by multiplying the growth rate by the time plus offset, which are determined by the user.

The above however was considered the simpler section to code. It merely required the plugging in of the formula and a few minor unit adjustments when entered into Excel. The actual difficulty in programming the spreadsheet was in creating the algorithm for temperature.

The temperature of a chip is determined by seven different factors: center temperature, center location, radial gradient, x and y plane gradients, randomness, and

the standard deviation. In order to maximize efficiency, it was necessary to create an algorithm that integrated all the above factors. The resulting formula is as follows:

```
=RandomSwitch * ((CenterTemp - (RadialGradient * (SQRT(ABS((xLocation
-xCenterShift)^2 + (yLocation - yCenterShift)^2)))) + (xGradient *
(xLocation - xCenterShift)) + (xGradient * (yLocation - yCenterShift)))
- NORMINV(RAND(),CenterTemp - (RadialGradient*(SQRT(ABS((xLocation -
xCenterShift)^2 + (yLocation - yCenterShift)^2)))) + (xGradient *
(xLocation - xCenterShift)) + (xGradient * (yLocation -
yCenterShift)),StandardDeviation)) + CenterTemp - (RadialGradient *
(SQRT(ABS((xLocation - xCenterShift)^2 + (yLocation -
yCenterShift)^2)))) + (xGradient * (xLocation – xCenterShift)) +
(xGradient *( yLocation - yCenterShift))
```

Equation 5.3 – Excel Temperature Algorithm

The formula can be broken down into two basic parts. The foundation is based in the last segment:

```
CenterTemp - (RadialGradient * (SQRT(ABS((xLocation - xCenterShift)^2 +
(yLocation - yCenterShift)^2))))
```

Equation 5.4 – Excel Temperature Algorithm backbone

This part simply determines the temperature based on the location of the central heat source, the center temperature, and the location of the cell on the wafer.

The following is then added to the formula above:

```
(xGradient * (xLocation – xCenterShift)) + (xGradient *( yLocation -
yCenterShift))
```

Equation 5.3 – Gradient shift segment of Excel Temperature algorithm

This adjusts the temperature for the x and y gradients.

The other half of the formula merely takes the total from above and uses it as the mean in generating a random number based on the standard deviation. This number is not a true random number per say; it is actually a number generated on odds determined by a Gaussian curve that uses the determined mean and user specified standard deviation using the NORMINV command. This value is then added to the ideal mean and returned to the user. A toggle is added that switches between one and zero. If the user decided to get the ideal conditions instead of the random ones, the value is multiplied by zero, thus canceling out the generated number to zero.

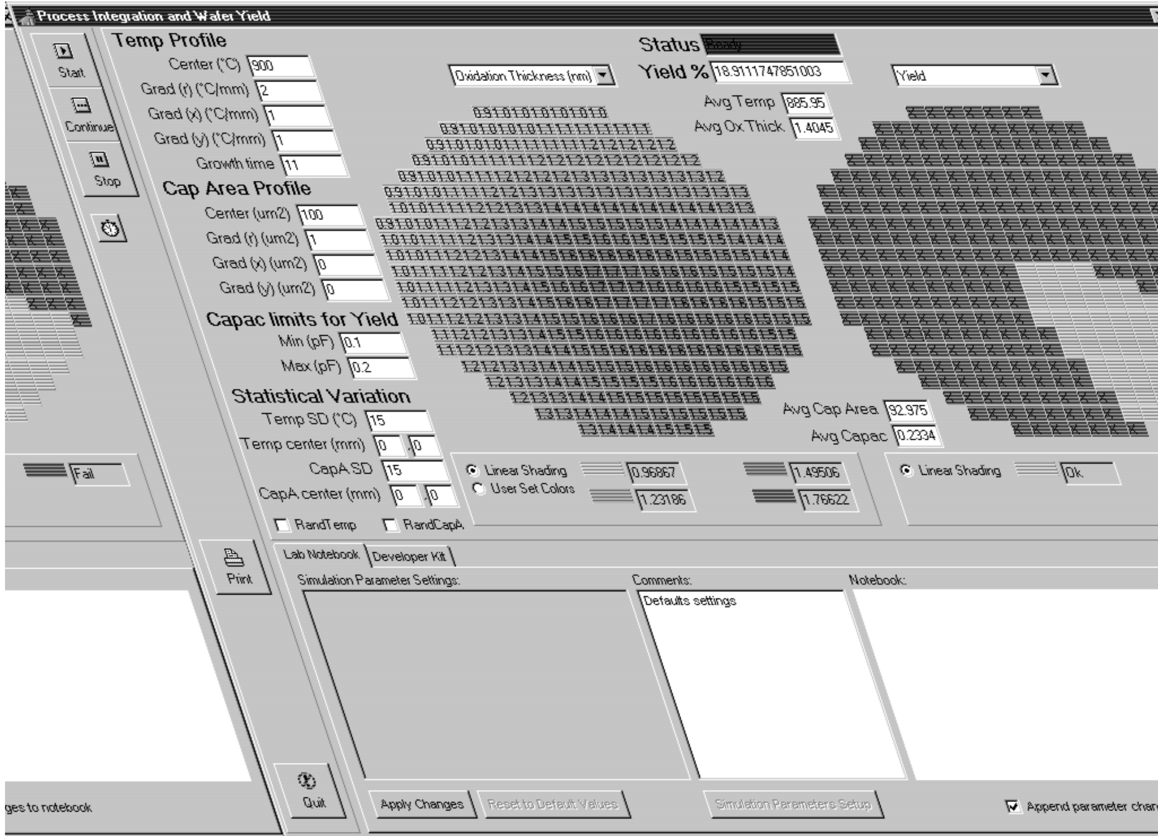
A similar formula is used for the capacitance area. Even though the units and actual numbers differ, the basic formula is exactly the same.

VI. Graphical User Interface Design

In education modules, the presentation of data is as critical, if not more, than the generation process of data. With the large number of factors the user can manipulate and the vast sheets data available to him, any mishap in the flow of data between the user and computer can result in the failure of the entire module.

The five sets of data available to the user show various statistics on the various chips on the wafer. The sets are temperature, oxide thickness, capacitance area, capacitance, and total wafer yield. The total yield percent, average temperature, oxide thickness, area, and capacitance are also available. The user can set numerous properties. The temperature of the hot spot, the location of the hot spot, radial and planar gradients for temperature and capacitance area, the standard deviation for temperature and capacitance area, and the acceptable range of capacitance.

It was decided to build the module on top of the already tested SimPLE framework, created by the HCIL. Although originally designed for VisSim, the various modules available to the framework proved quite useful. In particular, the html-based guidance and instructional system, proven effective and successful in previous HCIL projects, was well worth the effort of adapting the template to Excel.



ig 6.1 – Graphical User Interface (GUI) for Wafer Module

The actual conversion was a rather tricky endeavor. The time-based controls and functions were removed since they were irrelevant in the wafer module. In addition, all traces of the VisSim controls needed to be removed, else set the stage for a potential conflict between VisSim and Excel. Some minor esthetics, such as the red grid indicating a halted simulation, was adjusted as well.

In the primary display panel, two basic wafer maps were created. This allowed the user to spot trends by comparing two separate sets of data simultaneously. The data could be easily changed by selecting the desired data from a pull down menu,

which also served as the wafer title.

There are two cases in regards to the display of the wafer map: the yield map and the numerical data maps. The yield map displays to the user which chips are usable on the map. Usable chips are colored yellow while unusable chips are marked with an 'X' and colored red. On the panel below the wafer map, a legend will remind the user which color is used to indicate each status.

The numerical data maps offer the user two ways to view the data. They can either have the computer generate a linearly based coloring scheme or define their own parameters for up to four colors. The computer-generated scheme takes the minimum and maximum value from the graph and runs it through this formula:

$$(\text{Value} - \text{Minimum}) / (\text{Maximum} - \text{Minimum})$$

Equation 6.2 – RGB color shade formula

This generates a decimal number that gives the approximate location of where the value is between the two boundaries. This number is then multiplied by 255 and used in an RGB value, producing the corresponding shade on a range from red (maximum) to yellow (minimum). The legend below the wafer map then defines the color at the minimum, one-third, two-thirds, and maximum. The boxes are set ReadOnly so that the user won't accidentally change the scale.

The user-defined scheme allows the user to select the maximum value that each color represents. Therefore, the first color represents all values equal and less than the first defined value, the second color defines the range between the first value and the second, the third color defines from the second to the third, and the final color represents all values greater than the third color. This gives the user greater flexibility in viewing the data.

VII. Future Direction of the Project

There are several possible routes for future development of this module. One idea, given by a former Intel yield worker, suggests introducing a model that would take into consideration particle contamination. This could possibly be implemented by introducing a statistical model based upon the rating of the manufacturing center and

clean rooms.

Improvements could also be made to the module, mainly in improving the efficiency of the code and implementation of a more object oriented design. Large segments of code could be broken down into simpler functions and procedures.

Another possibility is to bypass the OLE system all together and directly manipulate the COM objects through Delphi. This would be a more code intensive and difficult procedure, but could greatly improve speed and efficiency.

A more complicated coloring scheme could be introduced, giving the user even more flexibility in viewing data. This could come in the form of an exponential and logarithmic coloring scale, in addition to the current user-defined and linear display.

The base material worked with could also be made variable. In this module, the entire project was set at the statistics for dry oxidation of Silicion <100>. However, the user could be offered the option of changing this, allowing them to view the differences between different materials.

VIII. Special Thanks

Special thanks to the following people: my advisor, Dr. Gary Rubloff, who made this entire summer experience worthwhile, the Delphi guru Anne Rose at HCIL who patiently assisted me with code support throughout the entire project, Brian Conaghan, Intel's factory firefighter, who's luck with the stock market continues to amaze me, Rama, for keeping the lab from getting too quiet... don't get lost in Alaska, Rock, good luck with the baby!, Laurant, for fighting the evil NT machine when it acted up, and the rest of the Simulations Lab for tolerating my presence and my random questions and comments.

[0] Brockschmidt, Kraig. "OLE Integration Technologies: A Technical Overview." http://msdn.microsoft.com/library/techart/msdn_ddjole.htm

[0] Calvert, Charlie. "Delphi and Microsoft Office: Automating Excel and Word."