THE BERKELEY PROCESS-FLOW LANGUAGE WIP SYSTEM

by

Christopher J. Hegarty, Lawrence A.Rowe, and Christopher B. Williams

Memorandum No. UCB/ERL M90/77

4 September 1990

THE BERKELEY PROCESS-FLOW LANGUAGE WIP SYSTEM

by

Christopher J. Hegarty, Lawrence A. Rowe,
and Christopher B. Williams

Memorandum No. UCB/ERL M90/77

4 September 1990

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

THE BERKELEY PROCESS-FLOW LANGUAGE WIP SYSTEM

by

Christopher J. Hegarty, Lawrence A. Rowe, and Christopher B. Williams

Memorandum No. UCB/ERL M90/77

4 September 1990

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

The Berkeley Process-Flow Language WIP System[†]

Christopher J. Hegarty, Lawrence A. Rowe, and Christopher B. Williams
Department of Electrical Engineering and Computer Science
University of California, Berkeley, CA 94720

To be presented at
SRC TECHCON '90

Abstract

The design of a formal process-flow programming language and a WIP system that executes it is described. The process-flow language, called the *Berkeley Process-Flow Language* (BPFL), can be used to specify a complete representation of the operations to manufacture and test a semiconductor integrated circuit.

Programs in BPFL can be read by other programs that execute or simulate the process. A WIP system executes a BPFL program. A simulation input generator or a process checker simulates the program. Maintaining one consistent process specification that can be used both in process design and fabrication can reduce design and manufacturing errors.

A WIP system is described that executes BPFL programs. Complex heuristics can be coded into a process-flow to automate the actions typically performed by hand because BPFL is a programming language. The advantage of automating these actions is that detailed records can be maintained automatically that describe what was done. And, successful actions can be easily reviewed and made standard policy. In addition, the WIP run management system is designed to allow dynamic changes to active runs (e.g., changing the process and splitting and merging runs). While dynamic changes do not automate the manufacturing process, they are necessary in practice.

1. Introduction

This paper describes the use of a formal language in a computer-integrated manufacturing (CIM) system to specify the process used to manufacture an integrated circuit (IC). The formal specification language is the Berkeley Process-Flow Language (BPFL). BPFL is a programming language that can be used to specify the information needed to manufacture an IC (e.g., masks, materials, equipment, operations, and tests).

The following goals influenced the design of BPFL:

(1) Allow all manufacturing operations to be specified including lot splits and merges, conditional tests, feed-forward and feed-backward control, rework, timing constraints, equipment and operator communication, and exception handling.

(2) Separate the facility specific information from the process specification to make it easier to change equipment in a fab or to move a process to another fab.

(3) Allow a process specification to be used as input to other programs (e.g., process simulators and checkers and factory scheduling systems) to reduce the time required to design a process and manufacture product.

A program written in BPFL, called a *process-flow*, is read by other programs that execute or simulate the process. An example of a program that executes a process-flow is a work-in-progress (WIP) system. Statements in the process-flow cause commands to be sent to equipment connected to the CIM system or to people who operate the equipment (e.g., load and run a particular furnace recipe).

A program that simulates a process-flow can check the program for errors (e.g., putting wafers with resist in a furnace tube) or generate input for other programs (e.g., a process simulator or a scheduling system). Statements in the process-flow change the state of the program that is simulating the process (e.g., keeping track of the material on a wafer) or cause simulator commands to be output. Figure 1 depicts the information flow in a program that generates commands for the PROSE simulator [8].

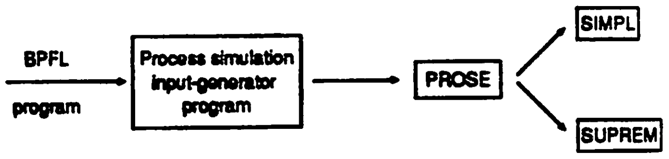


Figure 1: Information flow in a process simulation input generator.

Most WIP systems are either paper-based or run-sheet systems. Paper-based systems use an informal textual language to specify the process. The process-flow contains operator instructions that specify equipment recipes and what should be done for each step. A printed copy of the process-flow is taped to the lot and passed along with it as the lot is moved around the fab.

A run-sheet system uses a formal language to specify the process-flow[1]. The process-flow language in these systems includes commands to communicate with an operator through a form displayed on a terminal and, in some cases, communicate with equipment. However, these languages are not full-function programming languages so data structures (e.g., arrays, records, etc.) and control structures (e.g., conditional statements, looping statements, exception handling, etc.) are not provided[2]. Data and control structures are needed in a process-flow language in order to specify conditional processing (e.g., if it has been a long time since preventative maintenance was done on a piece of equipment, tweak the recipe parameters to compensate for the change in equipment performance) and feed-forward and feed-backward control. More importantly, exception handling mechanisms are needed to allow the CIM system to respond to unanticipated events (e.g., equipment failures).

Several research groups are working on process-flow representations that have the power of a modern programming language [2,3,6,7]. Two different representations are being explored: knowledge-based and programming language. While there are differences in representation, the real differences are the abstractions being developed to represent processing entities (e.g., equipment, materials, profiles, etc.) and operations (e.g., procedures, parameter passing, object-oriented programming, etc.). BPFL uses a programming language representation. The advantage of this representation is that a full complement of control and data structures can be provided which allows sophisticated decision making to be incorporated into the process-flow. Moreover, it emphasizes both the correct and incorrect behavior of the process.

† This research was supported by the National Science Foundation (Grant MIP-8715557) and the Semiconductor Research Corporation, Philips/Signetics Corporation, Harris Corporation, Texas Instruments, National Semiconductor, Intel Corporation, Rockwell International, Motorola, Inc., and Siemens Corporation with a matching grant from the State of California's MICRO program.

1 The most widely used commercial run-sheet systems are WORKSTREAM, formerly called COMETS, from Consilium Inc., Mountain View, California and PROMIS from Promis Systems Inc., Toronto, Canada.

2 WORKSTREAM has a scripting language with some control structures that can be called from a run-sheet command. However, these commands cannot be executed directly in the run-sheet, which severely limits the flexibility of the system.

This paper describes the use of BPFL by a WIP system. The WIP system allows operators and process engineers to manage runs (e.g., start, stop, and suspend runs), to change a run in process (e.g., modify the process-flow, add or remove wafers, or split or merge runs), and to browse a log that tracks processing history. The remainder of the paper is organized as follows. Section 2 presents an overview of the CIM system. Section 3 briefly describes BPFL. Section 4 describes the WIP run management system. And, section 5 describes the status of the prototype system.

2. CIM System Architecture

Figure 2 shows a typical computing system in a fab. Many different sized computers (e.g., micros, minis, and mainframes) are connected to and communicate through a local area network. Equipment is connected to workcell computers, people communicate with the system through workstations and terminals, and data is stored in databases located on different machines. Ideally, programs on any computer can communicate with programs and databases on any other computer.

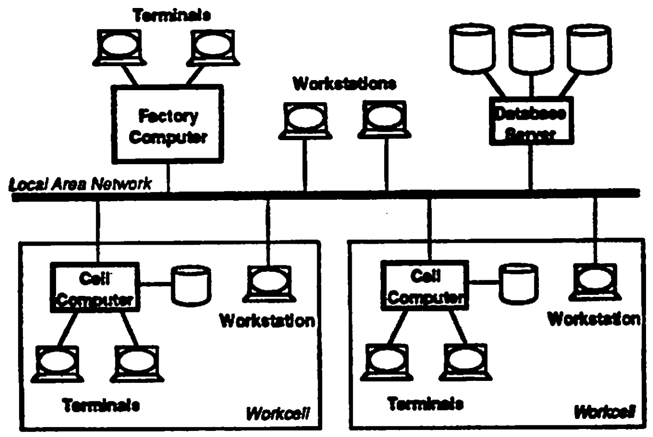


Figure 2: Typical computing system in fab.

The software architecture that runs on this distributed computing system must be a distributed software system. A key element of this software system is a shared CIM database that contains information about the fabrication facility (e.g., work areas and equipment), processes (e.g., masks and process-flows), WIP (e.g., lots, wafers, and processing history), equipment (e.g., status, recipes, maintenance logs, etc.), test data, product inventory, and orders. Applications treat this database as a centralized, single-site database, but it is actually a distributed database (DDBMS). Moreover, it is a heterogeneous DDBMS because data is stored in files and in different DBMS's.

The software architecture of the WIP system is shown in figure 3. The system is composed of many processes that communicate with users, equipment, and the CIM database. The main process is the WIP interpreter that executes BPFL programs. A run corresponds to an execution of a BPFL program. Each run is represented by data structures that contain the run state (e.g., the next statement to execute, the names and values of local variables created by the program, and data retrieved from the database). The WIP interpreter executes many BPFL programs at the same time. In other words, it is a server process.

The user interface process(es) support communication with operators. Operators at different locations in the fab can communicate with any run by connecting to the WIP interpreter through their user interface process. BPFL *user-dialog* commands are sent to the appropriate user interface process[3]. The user interface process is an Application-By-Forms program in the current prototype[4].

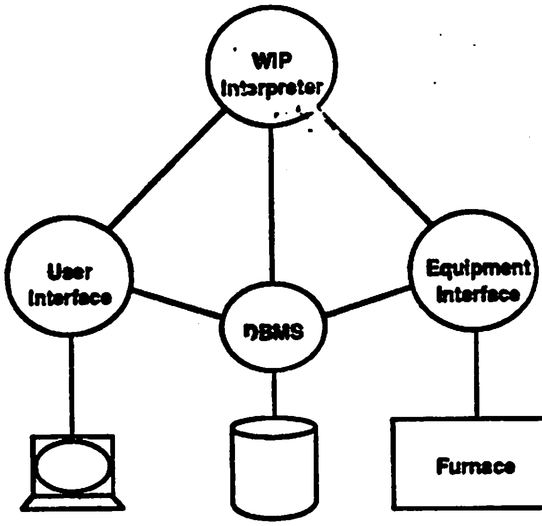


Figure 3: WIP system architecture.

The equipment interface process(es) support communication with equipment. This process is an instance of Wood's SECS server [9]. An object-oriented SECS interface is defined within BPFL. Methods are defined for high level equipment operations (e.g., run recipe, monitor run, fetch equipment status, etc.). These methods are implemented by remote procedure calls on SECS commands handled by the server.

All processes in the WIP system communicate with the CIM database. The WIP interpreter checkpoints the state of runs in the database so that other users and programs can access run information and so that active runs can be recovered if a computer or network fails. The user interface process uses form definitions stored in the database and allows the user to browse the CIM database (e.g., active runs, run logs, equipment status, etc.). The equipment interface process accesses equipment information stored in the database.

3. BPFL

The current version of BPFL is implemented as an extension to Common Lisp [5]. Lisp was chosen because it is well-suited to writing programs that manipulate programs (e.g., the interpreters) and because it is a good environment for experimenting with programming language designs. This representation for BPFL is unsuitable for novice computer users so a graphical user interface is being developed. Similar interfaces are being developed by other research groups [1, 3].

A process-flow is represented by a BPFL procedure. A procedure is a sequence of steps composed of function calls on BPFL procedures or primitives or Lisp functions. Figure 4 shows a fragment of BPFL that allocates some wafers, implants a well, and drives it in[5].

Arguments can be passed to procedures either by position or by name.

3 In a low volume fab such as the Berkeley Microlab, a user moves to a different terminal and reconnects to the run. The WIP system sends the command to the user interface process at the appropriate workcell in a high volume fab with many operators.

4 Application-By-Forms is a product of Ingres Corporation, Alameda, California.

5 A modified notation that is similar to a conventional algebraic language is used instead of Lisp to make the programs easier to read. Keywords are displayed in a bold font. Identifiers and constants are displayed in a roman font. Keywords and identifiers may contain the dash character ('-') to improve their readability.

```
defflow CMOS(masks lot-size)
begin
   step ALLOCATE-WAFERS begin
      allocate-lot(size: lot-size, lot-name: 'product, spec: ...);
      allocate-lot(size: 1, lot-name: 'well, spec: ...);
   end;
   step WELL-FORMATION begin
      with-lot (product, well) do
         wet-oxide(time: {11 min}, temp: {1000 degC}, thickness: {100 nm});
      pattern(get-mask(mask-name: 'NWELL, mask-set: masks));
      with-lot (product, well) do
         implantation(dopant: #m(phosphorus), dose: {4e12 /cm^2}, ...);
      step WELL-DRIVE-IN begin
         with-lot (product, well) do
            oxide-etch()
         ...
      end;
      measure-oxide-thickness(wafer: pick-wafer(product), location: "well");
   end;
   ...
end;
```

Figure 4: Definition of a BPFL procedure.

Arguments passed by name can be passed in any order because the formal argument name precedes the value in the call. For example, the *allocate-lot* procedure in the figure uses named argument passing to pass its three arguments: the number of wafers to allocate (*size*), the lot in which to put the wafers (*lot-name*), and a specification of the properties of the desired wafers (*spec*).

Wafers allocated by a BPFL program are placed in *lots*. Built-in lot names are provided for *product, rework and scrap*. In addition, a process-flow can define new lot names (e.g., *well*) that can be used to hold test wafers or subsets of wafers that will receive special processing. A wafer can be in several lots at the same time. In actuality, all wafers are stored in one carrier. The mapping between wafers and lots is maintained by the interpreter (e.g., the WIP system). During processing, particular wafers are identified by scribes on the back of the wafer. The WIP interpreter specifies operations in terms of the wafer scribes but the process-flow uses lot variables to specify which wafers should be processed. This simplifies the coding of the process.

The step statement specifies a process step. The statement has a name (e.g., *ALLOCATE-WAFERS* or *WELL-FORMATION*) and a body. The body contains the operations in the step. Notice that steps can be nested as illustrated by the *WELL-DRIVE-IN* step in the *WELL-FORMATION* step.

The with-lot statement specifies the wafers that should be processed by the operations in its body. For example, the wafers in the *product* and *well* lots are processed by the *wet-oxide* operation at the beginning of the *WELL-FORMATION* step. This construct gives the process engineer complete control over which wafers are processed.

Expressions in BPFL can include constant values, values with unit designations, and material specifications. Constant values are denoted either by numeric literals (e.g., "32") or by named literals that are quoted (e.g., "'NWELL"). Values with unit designations are denoted by two values enclosed in set brackets. The first value is the magnitude and the second value is the unit designation. For example, "{1000 degC}" represents 1000 degrees Celsius. Material specifications are denoted by an escape sequence ("#m") followed by a material name. For example, "#m(phosphorus)" specifies phosphorus. Materials are represented by instances of objects from a material class hierarchy. Material properties are stored in fields in each object.

BPFL also provides statements to communicate with equipment and operators. The with-equipment statement allocates a piece of equipment, which is represented by an equipment object fetched from the CIM database, executes a sequence of equipment operations, and then deallocates the equipment. The following example shows an oven operation.

```
with-equipment an-oven: 'oven do begin
   set-temp(an-oven, {150 degC});
   wait(an-oven, {30 min});
end
```

The procedures *set-temp* and *wait* are methods defined on the *oven* class. This example uses a generic equipment category (i.e., oven) to specify the desired equipment. The equipment specification can also be a particular instance of equipment (e.g., y2-hard-bake-oven).

Operator communication is specified by the *user-dialog* function. For example, the following statement allocates an instrument that is used to measure oxide thickness and calls a particular frame in the user interface process.

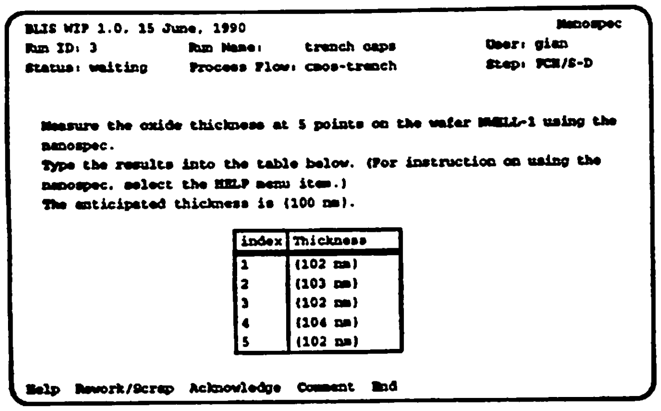
```
with-equipment x: 'nanospec do
   results := user-dialog(frame: 'nanospec,
                          expected: {100 nm});
```

A frame contains a form and a menu of operations. The form contains fields in which data can be entered by or displayed to the user. The menu of operations is listed across the bottom of the frame. The *nanospec* frame called by this code is shown in figure 5. The form in the frame contains instructions that describe the operation to be performed and fields into which the operator can enter the results of the inspection. The operator enters the measurements into the table at the bottom of the form. The system checks that the values use the correct units (e.g., time values would be wrong) and close to the expected value. When the operator finishes inspecting the wafers and entering the results, he or she executes the *Acknowledge* operation which causes the values entered to be returned to the BPFL program and assigned to the variable results.

More details on the design of BPFL and its interpreters is given elsewhere [4].

4. WIP Run Management System

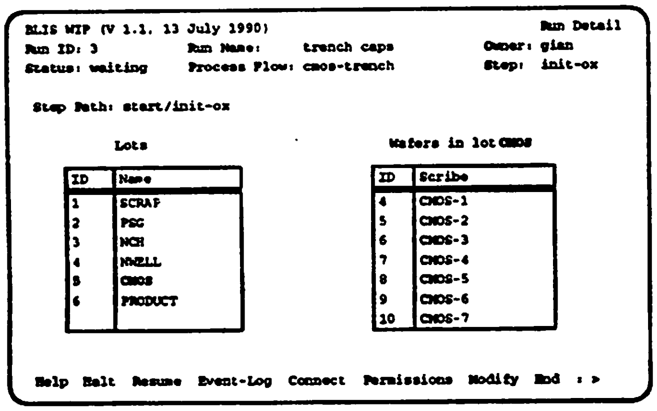
This section describes the WIP run management system. Frames are provided that show summary information about active runs and detailed information about a particular run. Figure 6 shows detailed information about a run. The frame shows the process-flow being executed, the run status, the lots in the run, and the wafers in one lot. Operations are provided that allow the operator to browse the run log, modify the process-flow, and split or merge runs.



```
BLIS WIP 1.0, 15 June, 1990                                       Nanospec
Run ID: 3            Run Name:     trench caps            User: gian
Status: waiting      Process Flow: cmos-trench            Step: PCH/S-D

  Measure the oxide thickness at 5 points on the wafer NWELL-1 using the
  nanospec.
  Type the results into the table below. (For instruction on using the
  nanospec, select the HELP menu item.)
  The anticipated thickness is (100 nm).

                    index Thickness
                    1     (102 nm)
                    2     (103 nm)
                    3     (102 nm)
                    4     (104 nm)
                    5     (102 nm)

Help  Rework/Scrap  Acknowledge  Comment  End
```

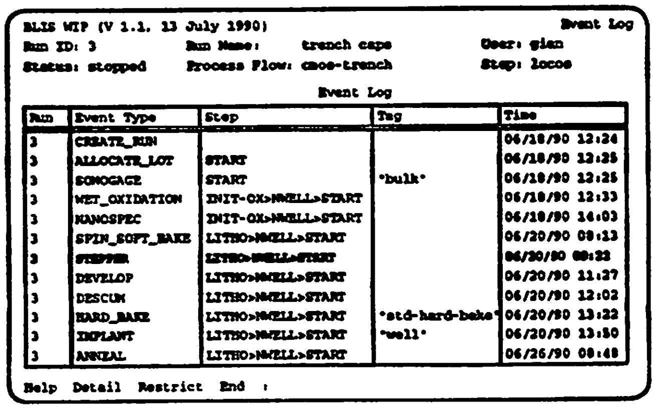
Figure 5: *Nanospec* frame.

Figure 7 shows the frame that displays the event log that is called when the *Event-Log* operation is executed. The table in the middle of the form is a scrollable window on all log entries written by the run. More information about log entries can be examined by executing the *Detail* operation. Other frames are provided that allow users to browse the log entries written by more than one run. A description of the database design for the log and example queries are given in the previously referenced paper [4].



```
BLIS WIP (V 1.1, 13 July 1990)                                  Run Detail
Run ID: 3            Run Name:     trench caps            Owner: gian
Status: waiting      Process Flow: cmos-trench            Step:  init-ox

 Step Path: start/init-ox

        Lots                               Wafers in lot CMOS

   ID   Name                            ID   Scribe
   1    SCRAP                           4    CMOS-1
   2    PSG                             5    CMOS-2
   3    NCH                             6    CMOS-3
   4    NWELL                           7    CMOS-4
   5    CMOS                            8    CMOS-5
   6    PRODUCT                         9    CMOS-6
                                        10   CMOS-7

 Help  Halt  Resume  Event-Log  Connect  Permissions  Modify  End  : >
```

Figure 6: *Run-detail* frame.



```
BLIS WIP (V 1.1, 13 July 1990)                                   Event Log
Run ID: 3            Run Name:     trench caps            User: gian
Status: stopped      Process Flow: cmos-trench            Step: locos
                                  Event Log
Run  Event Type       Step                 Tag               Time
3    CREATE_RUN                                              06/18/90 12:24
3    ALLOCATE_LOT     START                                  06/18/90 12:25
3    SONOGAGE         START                'bulk'            06/18/90 12:25
3    WET_OXIDATION    INIT-OX>NWELL>START                    06/18/90 12:33
3    NANOSPEC         INIT-OX>NWELL>START                    06/18/90 14:03
3    SPIN_SOFT_BAKE   LITHO>NWELL>START                      06/20/90 08:13
3    STEPPER          LITHO>NWELL>START                      06/20/90 08:22
3    DEVELOP          LITHO>NWELL>START                      06/20/90 11:27
3    DESCUM           LITHO>NWELL>START                      06/20/90 12:02
3    HARD_BAKE        LITHO>NWELL>START    'std-hard-bake'   06/20/90 13:22
3    IMPLANT          LITHO>NWELL>START    'well'            06/20/90 13:30
3    ANNEAL           LITHO>NWELL>START                      06/26/90 08:48
Help  Detail  Restrict  End  :
```

Figure 7: *Event-Log* frame.

The *Modify* operation in the *run-detail* frame allows a process engineer or operator to tweak the process (e.g., change equipment settings), modify the process-flow (e.g., adding or deleting an operation), or modify the run. BPFL programs are stored in a version control system so changes can either be made to the process-flow for just this run or to the process-flow used by other runs. A permissions mechanism is provided so that changes can only be made by people who are authorized to do so.

The engineer or operator can also split a run into two or more runs or merge one or more runs into another run. In a run split, wafers can be distributed to lots in the new run(s) and new wafers can be allocated. In a run merge, wafers from the different runs can be merged into one or more lots in the resulting run. A lot split might be used when a batch operation (e.g., a furnace run) cannot hold a full lot and the fab policy is to split a lot so that the operation is full. A lot merge might be used when the number of remaining product wafers falls below some threshold. These operations can be executed by hand or they can be invoked by another program.

5. Status and Conclusions

The WIP system described above is almost complete. The interpreter has been implemented and tested and the user interface is mostly done. The functions that remain to be implemented are the version control system for process-flows and the operations to modify process-flows and runs. The WIP interpreter is approximately 10,000 lines of Lisp. We have coded a CMOS process used in the Berkeley Microlab. The process-flow is approximately 650 lines of code and it uses approximately 75 frames. Many procedures in the process-flow belong to a library that we expect will be used by many other process-flows.

References

1. R. Hartzell, *Personal Communication*, Texas Instruments, Dallas TX, Jan. 1989.
2. M. B. McIlrath and D.S. Boning, "Integrating Process Design and Manufacture," *Proc. 1989 SRC IFM-IC Workshop*, College Station, TX, Nov. 1989.
3. J. Y. Pan, J.M. Tenenbaum and J. Glicksman, "A Framework for Knowledge-Based Computer-Integrated Manufacturing," *IEEE Trans. on Semiconductor Manufacturing* 2, 2 (May 1989).
4. L. A. Rowe, C. B. Williams and C. J. Hegarty, "The Design of the Berkeley Process-Flow Language," Electronics Research Lab. Memo 90/62, U.C. Berkeley, August 1990.
5. G. L. Steele, *Common Lisp - The Language*, Digital Press, 1984.
6. E. M. Voorhees, "A Work-In-Progress Tracking System for Experimental Manufacturing," *Proc. 2nd Intl Conf. on Data and Knowledge Sys. for Manuf. and Eng.*, Gaithesburg MD, Oct. 1989.
7. J. S. Wenstrand, I. Hiroshi and R. W. Dutton, "A Manufacturing-Oriented Environment for Synthesis of Fabrication Processes," *Proc. 1989 ICCAD*, Nov. 1989.
8. A. S. Wong, "An Integrated Graphical Environment for Operating IC Process Simulators," Electronics Research Lab. Memo 89/67, U.C. Berkeley, May 1989.
9. E. J. Wood, H. Schenck and J. Wijaya, "Networking and Object-Oriented Coding for SECS Communications," *Proc. Automated IC Manufacturing Symposium*, Fall Electrochemical Society Meeting, Oct. 1987.