Demonstration of a Process-Independent Run-to-Run Controller

Roland Telfeyan and James Moyne
The University of Michigan
Ann Arbor, Michigan 48109-2108
telfeyan@umich.edu

Arnon Hurwitz
SEMATECH
Austin, TX 78741-6499
arnon.hurwitz@sematech.org

John Taylor
Compugenesis, Inc.
Austin, TX 78739
jtaylor@compugen.com

Abstract

A process-independent generic cell controller (GCC) has been developed as a part of a supervisory control framework for semiconductor manufacturing processes. It can function in a hybrid, dynamic process environment where facility structure, networks, controllers, equipment, products, and operations are constantly changing. The GCC is designed primarily as a run-to-run (R2R) controller that can be controlled by inter-process factory controllers, can incorporate real-time controllers, and is currently being adapted for transfer to industry. A GCC implementation has performed optimization and control of plasma etching and chemical-mechanical planarization, illustrating process independence.

Introduction

This paper examines a design for a *generic cell controller* (GCC) which has been implemented and tested in the run-to-run (R2R) control of two semiconductor manufacturing processes: RIE and CMP. The GCC is best understood in its context within the model of a fully-automated VLSI fabrication facility. This widely-accepted model denotes a hierarchical control structure shown in Figure 1. At the top of the hierarchy, a factory controller acts as a supervisory production controller and a gateway to upper management. At the bottom, the leaves are equipment involved in VLSI wafer processing.

Each *group* of equipment dedicated to a particular manufacturing process is called a *cell* and is controlled by a *cell controller.* The cell controllers divide the task of wafer processing amongst the various equipment involved in the particular process.

The GCC is an event-driven mechanism, responding to high-level commands from a parent controller and executing these commands by sending a series of lower-level commands to subordinate controllers, be they equipment controllers or other GCCs. The GCC is a run-to-run (R2R) controller. R2R control is a form of discrete process and machine control in which the product recipe with respect to a particular machine and machine process is modified at an *ex-situ* process-to-process level as opposed to an *in-situ* level, i.e.,

the product recipe is modified between machine runs rather than during runs. A major emphasis in R2R control research has been on the development and implementation of algorithms for discrete process control. The ranges of use of each of these methods though generally overlapping, vary greatly in many cases. Thus, in an ideal discrete control system, a number of these methods could be utilized in a complementary fashion.

Thus, the main significance of the GCC is that it integrates and automates the interoperation of not only all the equipment in the cell, but also all the control algorithms used for discrete process control within the cell. The GCC is the glue, or enabler, which binds all these hardware and software components together, automating the cell. The remainder of this paper examines the GCC R2R controller design and demonstrates how this design is able to bind together, automate, and control all of the equipment that make up a manufacturing cell.

Design Requirements, Approach, and Implementation

A practical and reusable process-independent GCC must have the capabilities listed below. For each capability, design requirements are specified, then the approach to the design and implementation of the current version of the GCC (1.0) is summarized.

Dynamic control scheme. The GCC's source of control knowledge should be persistent yet dynamic, capable of being updated in mid-process and capable of being adapted to many different processes by the user. One way to achieve this goal is to have the control knowledge stored in a database rather than in static code. By storing the control knowledge in a database, it makes the knowledge capable of being changed or modified quickly, easily, and at any time. This dynamic control knowledge enables the GCC to respond quickly and easily to changes in the steps required to carry out a process.

Approach. When the GCC receives an event, the control scheme determines the action to be taken in order to service that event. That control action consists of a sequence of messages sent to modules (explained below) to carry out the desired task. In other words, the control scheme is a list of event-action pairs, where an event represents a high-level command to the GCC and the action is a list of module invocations. The control scheme is stored in a dynamic database rather than in static code, so that it is able to be modified even during the execution of a process if necessary. This dynamic control scheme is critical to the GCC's quality of process-independence, and it is the mechanism by which modules are bound to the GCC.

Simply put, for each incoming event, the GCC control knowledge maps the event into a series of commands to modules. Changing or adding to the control knowledge requires no re-coding or re-linking, because the control knowledge is stored in a dynamic database. It is therefore easy for the GCC to have different sets of control knowledge for different processes, and it is easy for the GCC to adapt or add to its knowledge on the fly, that is, even

in the midst of carrying out a run. The ability to adapt while running allows the GCC to accommodate a wide range of conditions within a process.

In the current implementation, the GCC control knowledge is defined in terms of a relational database and is implemented in SQL. The GCC database access layer supports SQL servers from multiple vendors: ORACLE, SYBASE, and QuickBase. There are future plans to create a graphical user interface to the control knowledge, complete with a learning mechanism that will query the user for the desired action when an unknown event is received.[2]

Plug-and-play integration of external software modules. The GCC must be able to inter-operate with software programs such as process model builders, optimizers, control algorithms, and equipment interfaces. Thus, the GCC must define a generic interface to all such types of software programs. These types of programs, called *modules*, must be able dynamically to connect and disconnect from the GCC without any code modification. There should be a module interface which facilitates the passing of arbitrary data, determined at runtime, between the GCC and the module. The module interface must allow users to develop custom modules or third-party developers to produce shrink-wrapped modules. This generic and dynamic interface to software modules is critical to the requirement of process independence (below).

Approach. A suite of software objects have been developed as part of a software interface specification for the GCC, which meets all of the above requirements. For further information, this specification is available upon request.[4] A full discussion is beyond the scope of this paper, but the essential job of these core objects is to facilitate the interfacing software modules to the GCC. Third-party integrators can make their software a module to the GCC in a very short amount of time by using these objects. They hide all of the details and provide a clean and simple interface.

Process-independence. By virtue of the foregoing two design requirements, the GCC has defined a generic interface to the process and the equipment being controlled. Because the control scheme mechanism is dynamic, the GCC can be programmed with control knowledge for any process. Because software modules and equipment interface modules and be integrated to the GCC's well-defined interface specification, the GCC can work with any software or equipment related to a given process.

No matter what tool is being used or what software and hardware is being used to control it, the interface to the GCC is well-defined, allowing for the interchangeability of these software and hardware. Further, the GCC is able to work with a new process without any code change, recompilation, or relinking. New equipment controller modules are able to be integrated into the GCC without any code modifications, and they are able to connect to and disconnect from the GCC dynamically.

The relationship of modular software and dynamic control scheme to process independence is illustrated in Figure 3. To prepare the GCC to handle a new process, one must insure that there is a control scheme for the desired process, and that there are suitable modules that the control scheme will invoke. There is no re-compilation or re-linking required.

Complementary operation of many control and optimization methods. The GCC has the ability to run several controllers and optimizers concurrently, taking the advice given by the most appropriate controller or optimizer to use for the current run. Fuzzy heuristics are used to make the decision as to which controller or optimizer is best for the current run. Figure 4 shows how several optimization and control modules can be used in concert. This fuzzy branch determination mechanism is described elsewhere.[1]

Ability to provide R2R control with or without in-situ control. The GCC should always be able to provide R2R control. Further, it should be able to delegate in-situ, real-time control to subordinate equipment controllers, if they exist.

Approach. The GCC is a major component of a multi-level control system that includes real-time equipment and process control as well as soft-real-time process control components operating in conjunction with the sequential control component. The process control problem is broken down into three control components: in-situ process factory control, in-situ process control, and discrete (R2R) product control. In the ideal case these three control loops operate concurrently in a hierarchical feedback fashion (Figure 2). In the ideal factory control environment, this multi-level process control system described is a component of a higher level factory control system that provides for process integration and inter-process control.

Platform independence. The GCC and all of its components should run on multiple hardware architectures and operating systems.

Approach. The current GCC implementation runs on NEXTSTEP (Mach UNIX) which runs on multiple hardware platforms: Intel 486 and Pentium, HP PA-RISC, and Sun SPARC. In the future, this list will also include PowerPC and DEC Alpha. In the near future, the GCC will also be running on the following operating systems: Solaris (PowerPC, SPARC, Intel), Windows NT (Intel), and DEC UNIX (Alpha). Most of these development efforts are well on their way to completion.

GCC Software Components

The GCC is suite of software applications (Figure 5) which consists of the GCC *kernel* and a set of *modules*. One of the modules is a graphical user interface (GUI) which presents an abstract view of the cell (Figure 6), showing the high-level commands that the

GCC can accept on behalf of the cell, together with the list of modules that will actually carry out the work to execute high-level commands.

The GCC Kernel

The GCC waits for events and reacts with corresponding actions. When the GCC receives a command, an event is posted, and the GCC consults its database to determine a corresponding action, an ordered-list of invocations. An invocation is a message sent to a target. The targets in this case are the various modules connected to the GCC.

The actions corresponding to events (the control scheme) are programmed and stored in the GCC's database by the user. (The database has a low-level API, but currently it has no high-level API tailored to programming this control knowledge, and thus it also has no GUI. These two deficiencies will be addressed in the next major release.)

The modules to be invoked by the control scheme are either purchased as third-party packages or can be tailor-made by the user. Instructions for writing a module are straightforward and are available on request.[4]

The GCC Graphical User Interface

For each manufacturing cell, the graphical user interface (GUI) will present a window with two browsers (Figure 6):

- **Commands:** The high-level commands that the GCC is able to carry out on behalf of this cell
- **Modules:** The modules associated with those commands. The modules are the list of targets that will be invoked by the GCC to carry out the desired command.

A module can be opened by double-clicking on its entry in the **Modules** browser. Its own GUI will display. A command may be sent to the GCC by double-clicking on its entry in the **Commands** browser.

The GCC Modules

Modules identify themselves and the manufacturing cell to which they belong by name, as a character string. The name of the cell and the names of the modules can be edited as character strings within the module applications at runtime, allowing dynamic binding of the character strings listed in the database to the actual software applications to which they correspond, running as part of the distributed GCC application suite.

All modules have certain common attributes and behaviors. If one runs a module with the kernel not running, one will notice that the module automatically tries to find the kernel and bind to it. If the module can't find the kernel, it allows one to either:

- Try to search for the kernel again
- Specify a host name (either a specific name or * to have it dynamically search for the kernel running on any local host).
- Run the module in stand-alone mode. For passive modules, like the **GccGui** which means nothing without the **GccKernel**, stand-alone mode is not significant. However, for a module like **MitGradual**,[3] which enables one to configure a problem model and run simulations off-line, running stand-alone without the **GccKernel** is useful.

The specific modules created for this release (1.0) are not only functional as modules of the GCC but also as stand-alone application programs:

- **MitGradual:** MIT R2R Gradual Mode program
- **GccTool:** A generic tool controller GUI program
- **GccMetrology:** A metrology equipment controller GUI program
- **GccHistory:** A graphic history/plotting program

The MitGradual Module (Figure 7) is a linear-approximation controller. It contains a linear RSM model of the tool, and given metrology, it calculates the next best recipe. With respect to the GCC, it responds to the message **control** by programmatically accepting the metrology and returning the new recipe. The **MitGradual** module supports saving and loading of different problem setups. This enables the same software, when run twice, to be used to control more than one process.

The GccTool Module (Figure 8) is an equipment interface module. Its job is to maintain SECS communications with an actual equipment controller. For example, when the **GccTool** module receives a command from the GCC to download a recipe and start, it in turn issues SECS *recipe download* and *process start* commands to the equipment controller. With respect to the GCC, it responds to the messages **setRecipe** and **start**. It also monitors the process and sends a **done** message to the GCC when the process is done.

The GccMetrology Module (Figure 9) is also an equipment interface module. When it receives a command from the GCC to get the metrology, it prompts the user to type in metrology values which it will then send back to the GCC.

In contrast to the **GccTool** module which is totally automated, this **GccMetrology** module is designed to ask for human intervention. It does not assume a direct SECS channel to a metrology station for two reasons: (1) it allows us to demonstrate that hybrid forms of communication are possible, and (2) we do not yet have an automatic data acquisition method from our metrology equipment.

It could be the task of an integrator to take the **GccMetrology** module and actually make it communicate over the network to the corresponding equipment controller, so the user would not have to type in values (sneaker net). Nevertheless, the interaction and interface of this module to the GCC would not change.

The GccHistory Module (Figure 10) produces dynamic graphic plots and event traces of the control history of the process. Specifically, the **GccHistory** module produces:

- A graphic plot of the inputs
- A graphic plot of the outputs
- A formatted, human-readable trace of the data
- A raw, machine-readable trace of the data. This raw data is suitable for being imported into another off-line data analysis/plotting package.

The "Demonstration"

In preparing the GCC for a particular cell process, we must (1) add control knowledge (i.e., a control scheme) to the database and (2) make sure the required modules are present and initialized. When this preparation is complete, we are ready to send commands to the GCC to achieve R2R control.

Adding a Control Scheme. Suppose that we wish to have the GCC respond to the high-level message **planarize**. The resultant control scheme might look as shown in Figure 11, where incoming commands are in bold, and module invocations are in brackets.

Initializing the Modules. The only module we have to initialize is the **MitGradual** module, since it must be set up with the appropriate RSM model; that is, parameter inputs and outputs, their constraints, model coefficients, current optimal recipe, and current run number.

Sending a Command to the GCC. The GCC design may be introduced by tracing the control/optimization cycle of one run for a specific implementation developed to provide run-to-run control of a process. (Also see Figure 11.) At the beginning of the run, a high-level process command (e.g., the command **planarize(start)**) is sent to the GCC by a process engineer or factory controller. The operator can send such a command to the GCC by pressing a button (see Figure 6).

The GCC accesses its database and retrieves the equipment identification and process recipe. The GCC sends the recipe to the equipment interface module which forwards the recipe to the appropriate equipment controller which in turn downloads the recipe into the actual equipment. After the wafer to be processed is loaded into the equipment, the equipment interface module sends a command to the equipment controller to start the process.

When the module monitoring the process determines that the process has been completed, it informs the GCC (e.g., the command **planarize(done)**). As a result, the GCC invokes the metrology module, which retrieves the measured, post-process data. When the metrology is done, the GCC then invokes the process optimization and control module, which examines this data and attempts to determine a "better" equipment recipe to correspond to the high level process recipe; i.e., it uses one or more optimization and control branches (Figure 4) and modifies the existing equipment recipe so as to better achieve process target values. As the last step in the run cycle, the updated equipment recipe is stored in the GCC database by a history module and linked to the appropriate high level process recipe by the optimization and control module.

Implementation Case Studies

The GCC has performed R2R optimization and control in two case studies (1) the plasma or reactive-ion etching (RIE) process and (2) the chemical-mechanical planarization (CMP) process.

Experiments conducted for the first case study of plasma etching involved the etching of a simple 3-layer PolySilicon/SiO2/Si wafer (Figure 12). The experiment was tailored towards maintaining target parameters of average etch rate and over etch depth while controlling the input equipment parameters of etch time and oxygen flow. Other etch recipe parameters such as power, pressure, and various gas flows are kept constant. In the RIE experimental data, both etch rate target changed, and oxygen flow sensory information perturbed.With open-loop operation there is by definition no capability for compensation to target shifts or process perturbation. However with R2R control, the system effectively compensated for process shift and drift.

Experiments conducted for R2R control of a CMP process involved the planarization of an unpatterned wafer (Figure 14). The planarizer experiment was designed to maintain target parameters of removal rate and uniformity while controlling the input equipment parameters of down pressure, back pressure, carrier velocity, and platen velocity. Other planarizer recipe parameters such as pad age are monitored but obviously not controllable. Experimental results (Figure 15) illustrate that the controller accurately compensated for a drop in removal rate while maintaining an acceptable level of uniformity.

Conclusions and Future Work

A process-independent controller, the Generic Cell Controller (GCC), has been designed and developed for the run-to-run (R2R) control and optimization of semiconductor manufacturing processes. This GCC design utilizes sequential control and optimization branches in a complementary fashion. The design has the qualities of dynamic adaptation to different control processes and dynamic incorporation of third-party software control

modules. Thus it is able to be adapted and customized to any R2R control process. A GCC implementation has effectively controlled two different processes, RIE and CMP, without any change of code or design, illustrating process independence.

Our research is continuing in the area of module development. We currently have a multi-branch decision-maker module which has been designed, implemented, and tested elsewhere.[1] We plan to integrate that multi-branch module together with modules for model building and optimization with our existing GCC setup. We also plan to create a graphical user interface to the control knowledge, making it easier to program and maintain. We are developing a learning mechanism that will use this graphical user interface to query the operator for the desired action when an unknown event is received. The current GCC implementation is presently being adapted for transfer into industry.

Acknowledgments

The authors gratefully acknowledge the assistance of Hossein Etemad and Jeff Fournier. This research is supported by SEMATECH, the Semiconductor Research Corp., and Compugenesis, Inc.

References

1. J. R. Moyne, N. Chaudhry, R. Telfeyan, “Adaptive Extensions to a Multi-Branch Run-to-Run controller for Plasma Etching”, Journal of Vacuum Science and Technology, May/June 1995.

2. J. R. Moyne and L.C. McAfee, Jr. “A generic cell controller for the automated VLSI manufacturing facility,” IEEE Transactions on Semiconductor Manufacturing, May 1992.

3. W. Moyne and E. Sachs, “MIT RbR Control Server,” software system and documentation, MIT, Cambridge, MA, November 1994.

4. R. Telfeyan and J. R. Moyne, “Generic Cell Controller Interface Specification”, University of Michigan, Department of EECS, internal draft May 1995.

5. J. R. Moyne, “Generic Cell Controlling Method and Apparatus for Computer Integrated Manufacturing System,” Patent application filed with the United States Patent and Trademark Office. Filed, August 1991; Formal notification of allowance, May 1995.

FIGURES

Figure 1. Hierarchical model of the automated VLSI manufacturing facility

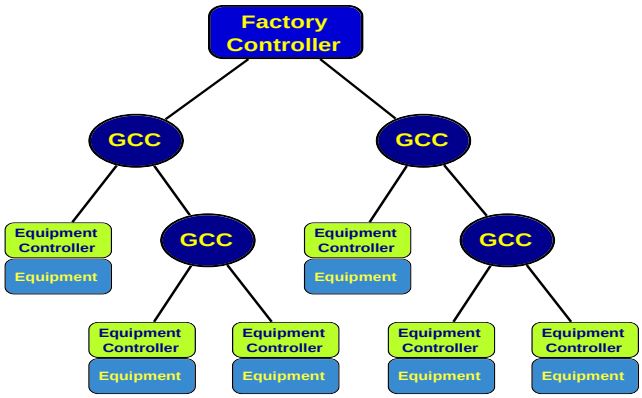

Figure 2. How the GCC delegates in-situ and real-time control to subordinate controllers

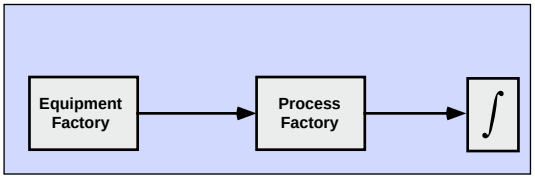

Figure 3. Process Independence of the GCC

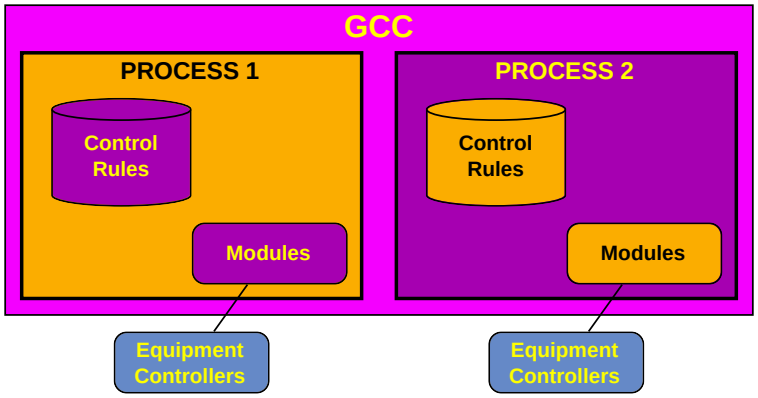

Figure 4. GCC Process Optimization and Control Algorithm

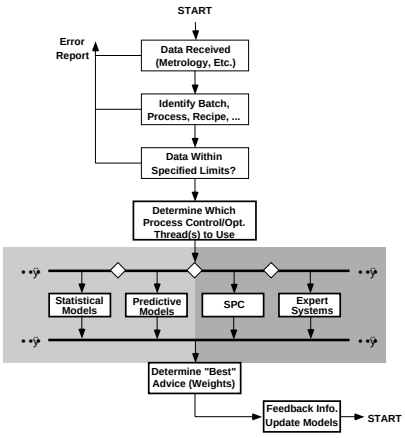

Figure 5. Block Diagram of the GCC

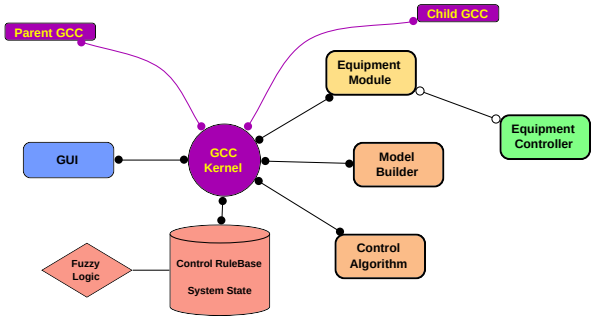

Figure 6. A Cell Represented in the GCC GUI

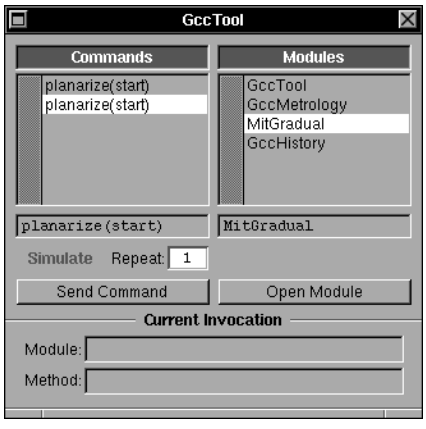

Figure 7. MIT R2R Gradual Mode Module

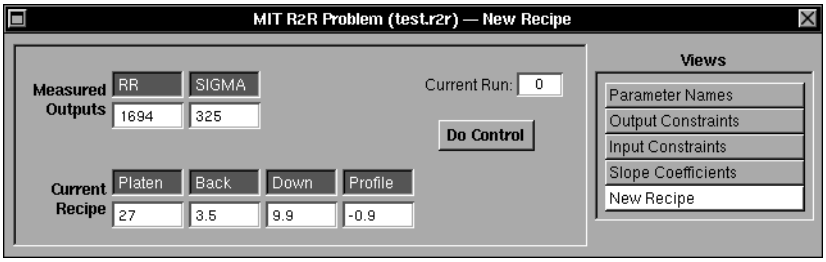

Figure 8. GCC Tool Interface Module

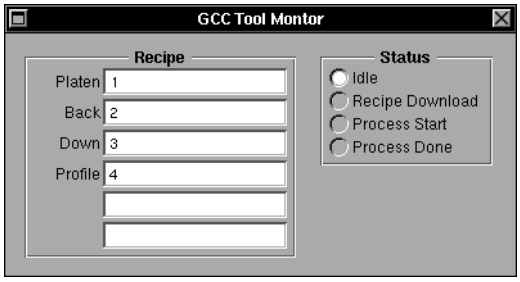

Figure 9. GCC Metrology Interface Module

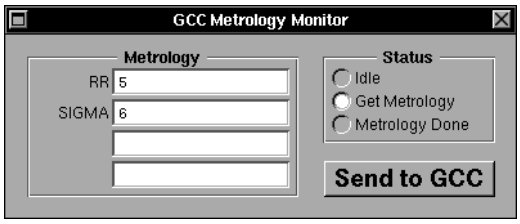

Figure 10. GCC History Output Plot

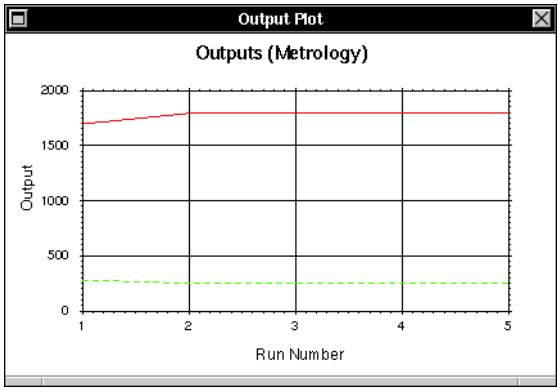

Figure 11. Sample GCC control scheme to service a “planarize” command

```
planarize(start)
    [GccHistory getRecipeAndRunNumber]
    [GccTool downloadRecipe]
    [GccTool start]
planarize(done)
    [GccMetrology getMetrology]
planarize(metrologyDone)
    [MitGradual control]
    [GccHistory storeRunData]
```

Figure 12. RIE Experimental Setup

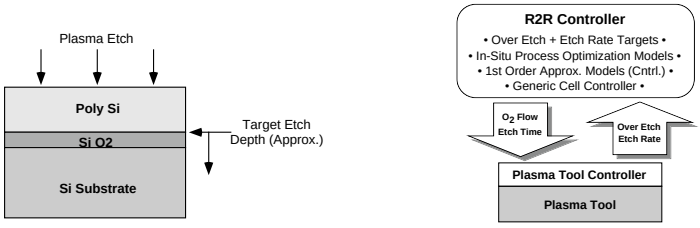

Figure 13. CMP Experimental Setup

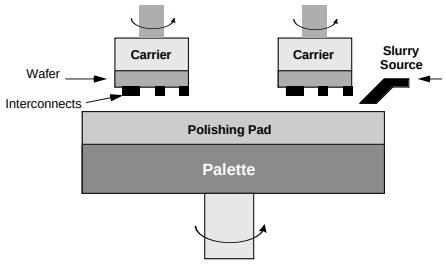

Figure 14. Experimental data: R2R Control of CMP Process