Axiomatic Development of a Machine Control System

by

Kwangduk Douglas Lee

B.S., Mechanical Engineering
Pohang University of Science and Technology, 1998

Submitted to the Department of Mechanical Engineering
In Partial Fullfillment of the Requirements for the Degree of

Master of Science in Mechanical Engineering

at the

Massachusetts Institute of Technology

August 2000
[September 2000]

Signature of Author _____
Department of Mechanical Engineering
August 18, 2000

Certified by _____
Nam P. Suh
Ralph E. Cross Professor of Mechanical Engineering
Thesis Supervisor

Accepted by _____
Ain A. Sonin
Chairman, Department Committee on Graduate Students

Axiomatic Development of a Machine Control System

by

Kwangduk Douglas Lee

Submitted to the Department of Mechanical Engineering
on August 18, 2000 in Partial Fullfillment of the
Requirements for the Degree of Master of Science in
Mechanical Engineering

Abstract

Axiomatic Design is presented as a scientific methodology in designing a complex machine control system. As an example, the CMP α machine control system is developed using the Axiomatic Design framework. The machine is a type of semiconductor processing equipment, which requires numerous actuators and sensors and the intelligent control of them to planarize thin layers of wafers. Signal processing modules, control algorithms, sequential process steps, graphical user interface, process recipe editor and the overall control system structure are all designed by the Axiomatic decomposition. Axiomatic Design is proved to be a very effective tool in control system development. It took less than six months to develop the system and the control system is fully functional without any major error or mistake. The resulting system is clear to understand, easy to maintain and upgrade, and flexible for further development and integration. Although the development has been specific to the CMP α machine, the control system structure and the design methodologies presented in this thesis are universally applicable to the development of any type of machine control system.

Thesis Supervisor: Nam P. Suh
Title: Ralph E. Cross Professor of Mechanical Engineering

Acknowledgements

I would like to acknowledge and thank many people who have supported and helped me in the preparation of this thesis. Professor Suh, my thesis advisor, has been vital in the completion of the thesis and the success of the project. He always has trusted me and endowed me with the strength to finish this work. I am also indebted to other faculty members in the group. Professor Chun has helped me thorough the numerous suggestions and encouragements. Dr. Saka has guided me through his academic advices and lengthy discussions with me.

Many thanks go to my group members, too. Jiun-Yu has helped me with his academic arguments; Jason with his brilliant ideas; Jamie with his hearty collaborations; Amir with his witty cheers.

Many people have helped me during the control system development. I owe a lot to Dr. Oh at United Technologies Corporation, who has been with me during the initial system development. Mr. Cord Ohlenbusch at Ultra Clean International Corp. has helped me with wiring. In fact, the machine power distribution and safety system is his work. I would like to give a special thank to Professor David Trumper for his teachings in the mechatronics course and the suggestions for the controller design. I also wish to acknowledge the help of Mr. Beau Tateyama, who, as a UROP student, helped me with his Visual Basic programming skill in the early stage of the system development.

Numerous people and innumerable instances, which I can not enumerate in a single page, have helped me to accomplish this project, once seemed to be almost unattainable. My family and friends have always been patient and supportive to me, understanding and allowing for me to be 'workholic.' I would like to appreciate all the circumstances around me, which have been so generous to this humble being.

To all the sensible beings in the world...

Table of Contents

List of Figures

List of Tables

Chapter 1. Introduction

1. Machine Control System Design

The scope of a machine control system design is quite extensive. It includes the selection and configuration of major system components, such as computers, controllers, motors, amplifiers, etc., the wiring of those components to establish signal interfaces, the design of open-loop and closed-loop controllers, organizing and programming individual controllers to form a whole scale machine control system, providing an easy-to-use interface to machine operators, and so on. People often identify designing servo controllers with designing a whole machine control system; a user interface shown on the host computer screen with its control system. But they are just parts of a control system. The design and development of a control system involves more than simply designing its subcomponents (although quite important).

A machine control system can be conceived as a set of constituents or subsystems (both hardware and software) with its overhead structure, intended to drive individual components of the machine in a well coordinated manner to achieve the desired changes on target objects with a reduced or minimum human intervention. At a low level, a control system may be used to place a work piece precisely, instead of a human operator performing positioning task. At a higher level, the same control system may cut a complicated three dimensional shape out of the work piece based on a CAD drawing, automatically without a human intervention during the whole process of cutting.

The development (design and subsequent implementation) of a machine control system requires a sound knowledge of physical hardware, control technology, software engineering and system integration, due to the scope involved. In addition, a systematic design tool is required to structure the whole system and to guide the design of its components. Although there are some methodologies in specific disciplines, such as object-oriented programming in software engineering, no comprehensive tool seems to exist tobe able to encompass all the realms involved. Naturally, the development of various control systems has relied on human experience and trial-and-error, without any universal framework. As a result, one control system developed for a certain type of machine can not be used for a different type of machine. At each occasion, the time and cost of a machine control system development is substantially high due to the lack of interchangeability and reusability, the product of an ad hoc development. Also a system developed without any discipline often contains fatal errors undetected during its design phase, resulting in the endless upgrade and maintenance cycle, sometimes harming the very profitability of the system.

The control system design based on Axiomatic Design (AD) is proposed in this thesis. AD provides a concrete scientific methodology for the system development, by identifying Customer Attributes (CAs) first, setting up independent Functional Requirements (FRs), conceiving appropriate Design Parameters (DPs), and also selecting Process Variables (PVs). Instead of relying on speculations and trial-and-errors, AD ensures a coherent new system development by clearly stating FRs and choosing right DPs, and by showing their relationships in terms of design equations. The Axiomatic Design approach eliminates unnecessary couplings and promotes innovative ideas throughout the entire design process. For a more detailed description about Axiomatic Design, please refer to Suh[3, 4].

As an example, the control system for the CMP α machine will be designed and implemented throughout this thesis, based on the Axiomatic Design approach. The machine is a prototype semiconductor wafer processing machine developed at MIT. The detailed explanations about the background of the machine are to be presented in the following sections.

Table 1.1 National Technology Roadmap for Semiconductors

Year of First Product Shipment- DRAM*	1997	1999	2001	2003	2006	2009	2012
Minimum feature size (μm)	250	180	150	130	100	70	50
Bits/chip- DRAM	64M	256M	1G	1G	4G	16G	64G
Chip size (mm^2)							
- DRAM	280	400	445	560	790	1120	1580
- Logic	300	340	385	430	520	620	750
- ASIC**	480	800	850	900	1000	1100	1300
Maximum Substrate Diameter (mm)	200	300	300	300	300	450	450
Number of metal levels							
- DRAM	2-3	3	3	3	3-4	4	4
- Logic	6	6-7	7	7	7-8	8-9	9
Maximum interconnect length- logic (meter/chip)	820	1480	2160	2840	5140	10000	24000
Planarity requirement within litho field for minimum interconnect CD*** (nm)	300	250	230	200	175	175	175
Minimum metal CD (nm)	250	180	150	130	100	70	50
Metal height/width aspect ratio- logic	1.8	1.8	2.0	2.1	2.4	2.7	3.0
Minimum contacted/non-contacted pitch (nm)							
- DRAM	550/500	400/360	330/300	280/260	220/200	160/140	110/100
- Logic	640/590	460/420	400/360	340/300	260/240	190/170	140/130
Metal effective resistivity (μΩ-cm)	3.3	2.2	2.2	2.2	2.2	< 1.8	< 1.8
Barrier/cladding thickness (nm)	100	23	20	16	11	8	6
ILD**** dielectric constant (κ)	3.0-4.1	2.5-3.0	2.0-2.5	1.5-2.0	1.5-2.0	≤ 1.5	≤ 1.5
Interconnect metal	Al	Al, Cu	Al, Cu	Al, Cu	Al, Cu	Al, Cu	Al, Cu

* DRAM: Dynamic Random Access Memory
** ASIC: Application Specific Integrated Circuits
*** CD: Critical Dimension
**** ILD: InterLayer Dielectric

2. Chemical-Mechanical Planarization (CMP)

Since the advent of Integrated Circuit (IC) technology, microelectronic devices have been fabricated on tiny pieces of semiconductor material, or on chips. A microelectronic chip is composed of numerous circuit components, or devices, such as transistors, diodes, resistors, and capacitors. Several layers of dielectric and metal materials are grown or deposited on top of the base silicon wafer, and these circuit features are created by a variety of methods such as diffusion, oxidation, lithographic patterning, etching, etc. A single wafer usually contains tens of 'dies', which are electronically complete units. Each die is sliced from the wafer, and then bonded and packaged to provide electrical connections and a more rugged support.

Each device is composed of many features, such as a metal line, a segment of doped silicone substrate and a portion of dielectric layer. The topographic combination of these features produce transistors, diodes, resistors, capacitors, etc. These devices then constitute a microelectronic chip. Feature sizes are getting smaller with the advance of integration. A typical feature size is less than a micron, while 0.15 μm is becoming the current generation. Table 1.1 is the National Technology Roadmap for Semiconductors, which shows the trends and requirements in the semiconductor industry for the next decade[1].

To meet the stringent requirements of submicron feature size, the layers deposited or grown during fabrication processes must be flat. The thickness variation of a layer over a device scale should understandably be much smaller than its feature size. Without this level of planarity, many fabrication techniques, especially lithography, fail to produce the desired submicron features.

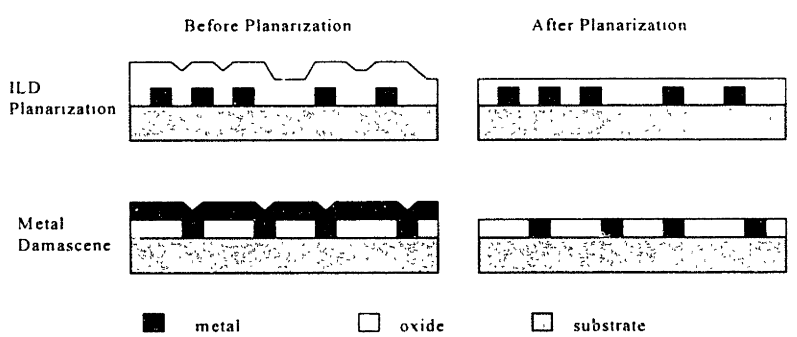
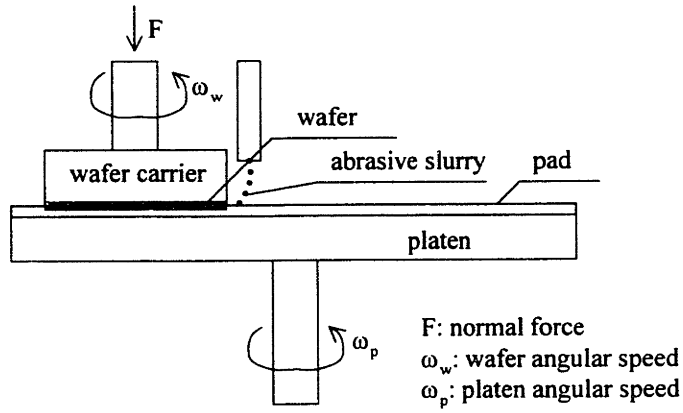
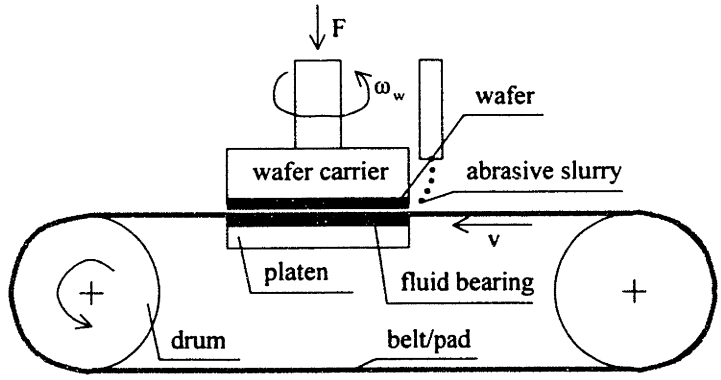


Figure 1.1 Schematic of Microelectronic Features before and after Planarization



Rotary kinematics



Belt kinematics

Figure 1.2 Two Types of Polishing Kinematics

Metal layers are planarized to expose the underlying microelectronic circuitry, whereas oxides are planarized to provide flat base layers for further depositions or growths of upper layers in the subsequent processes. Types of metals to be planarized include copper (Cu, dual damascene), tungsten (W, vias planarization) and aluminum (Al, alternative to Al etching). Figure 1.1 shows the schematic of the microelectronic features before and after planarization for both metal and oxide.

Planarization is usually accomplished by polishing a wafer mechanically against a pad in the presence of slurry, which is a mixture of loose abrasives and chemicals suspended in

water. For this reason, CMP is also interpreted as Chemical-Mechanical Polishing. A polishing process requires pressure and relative displacement between the two bodies in contact. Pressure is usually created by pneumatically pressing the wafer in a container (wafer carrier) against the pad on top of the platen. Relative displacements can be generated either by rotating the pad with the platen (rotary kinematics), or by the pad only with a stationary platen (linear or belt kinematics). In either case, the wafer is also rotated to ensure uniform polishing profile across the wafer surface and also to increase the relative speed seen from the wafer surface. Figure 1.2 shows the two types of polishing mechanism schematically.

3. CMP Performance Requirement

A CMP equipment should be able to process wafers to the desired specifications (effectiveness) in an efficient manner, as in any manufacturing machine. CMP specifications include uniformity, surface quality, defect density, etc. Throughput rate, cost of ownership, and tool capital cost can be the measures of a tool efficiency.

Uniformity can be classified into three levels- wafer, die, and feature level. On wafer level, non-uniformity (or variation in uniformity) is usually quantified by the standard deviation of a layer thickness divided by the mean thickness of the layer (σ/h). The lower is the thickness variation, the lower is the non-uniformity, meaning more planar surface. From the production point of view, wafer-to-wafer non-uniformity as well as within-wafer non-uniformity is important.

On die level, a difference in pattern (or feature) density leads to a difference in polishing rate and thus creates a non-uniformity. In metal polishing, the metal in high density area (i.e. which has more metal lines compared to the surrounding area) gets polished faster than the one in low density area, leading to overpolishing the underlying oxide layer. One solution to this problem is to use a selective slurry- the chemical in the slurry has a preferable etch rate to metal compared to oxide, thus minimizing oxide overpolishing.

On feature level, the cross section of an interconnect metal line tends to develop a hollow indentation after polishing, termed 'dishing.' Metal lines typically lose 20 % of their designed volume by the combination of dishing and overpolishing. A slight overpolishing is desirable to prevent any possible short circuit between the metal lines due to the metal residue over them.

Surface quality is closely related to defects. The polishing action can generate scratches comparable to the abrasive size used. The industry widely uses 50 to 300 nm size abrasives, which is close to the underlying feature size of the wafer surface. A streak of scratch can ruin many devices, considering it can be millimeters or centimeters long. The scratches should be minimized to the possible extent. Also the surface roughness should be less than 10 nm (R_a).

Mechanical abrasion and chemical etching during polishing can damage underlying features. Scratches are commonly observed. Weakly laid features can also be damaged by the contact with the abrasive particle, resulting in defects. A defect can also be created by an impurity particle embedded during polishing or attached after polishing. Buffing (polishing with light pressure and only with water after main polishing) and post-CMP cleaning are essential to remove any impurity from the wafer surface.

Porous polyurethane is typically used as a pad material in the conventional loose abrasive polishing, because of its chemical stability, low cost, and ease in tuning the

mechanical and physical properties. However as polishing continues, the surface of a pad plastically changes by the rubbing and plowing actions of abrasive particles. Pores are filled and the surface hardness changes (pad surface is glazed). These deformations result in the decrease in the material removal rate (MRR) and in the deterioration in the wafer uniformity and surface quality. To minimize this effect, the pad surface is continually regenerated during and/or between polishing by rubbing the pad surface with a fixed abrasive disk. This refreshing process is termed 'conditioning' in CMP industry. Diamond grits are typically used as the conditioning abrasives. Conditioning maintains a certain degree of pad roughness and keeps a number of pores open.

Table 1.2 2002 CMP Production Tool Performance Targets

	Attribute	Metrics for Dielectrics	Metrics for Metals
Equipment Parameters	Auto pad condition	Required	Required
	In-line metrology	Desired	N/A
	End point detector for STI application	Required	N/A
	In-situ thickness monitor and control or endpoint control	N/A	Desirable
	Integrated with post-CMP clean	N/A	Required
	Dry in-Dry out	Required	Required
Process Targets	CMP uniformity total variability (3σ)	10%	10%
	Head to head variation (3σ)	1%	1%
Defects (with in-situ CMP integrated post-clean)	On-film @ 0.20 μm	< 12.8/wafer (0.0186/cm^2)	< 12.8/wafer (0.0186/cm^2)
	On bare Si @ 0.09 μm	< 64.3/wafer (0.0919/cm^2)	< 64.3/wafer (0.0919/cm^2)
	Backside on Si @ 0.20 μm	< 200/wafer (0.30/cm^2)	< 200/wafer (0.30/cm^2)
Cost/Perform Targets	Throughput	75 wafers/hr	75 wafers/hr
	Tool capital cost	$1.9M	$1.9M
	Mean Time Between Failure(MTBF)	300 hrs	220 hrs
	Mean Time To Repair(MTTR)	2 hrs	2 hrs
	Preventive maintenance	6 hrs/wk	6 hrs/wk
	Consumables	< $4/wafer pass	< $4/wafer pass
	Area per tool	7.9 m^2	7.9 m^2
	Support area per tool	2.8 m^2	2.8 m^2
CoO Target	CoO objective	$5.56/wafer pass	$5.57/wafer pass

In the semiconductor industry, metrology is of the utmost importance. Metrology is required in almost all the process controls, yet measurements at nanometer level are quite challenging. A wafer can be measured during (in-situ) or at the end (ex-situ) of a process to check if the required physical changes have occurred. In-situ measurement is usually preferred, because the measured information can be fed to the controller for real-time process control purposes. But in-situ measurement is usually more difficult and costs more than ex-situ. Ex-situ measurement is applied where in-situ is not viable. Each wafer and/or die is tested and if it doesn't meet the process requirement, it is either sent back for re-work or discarded. Minimizing the amount of re-work and scrap is a big aim in the industry.

In CMP, the in-situ measurement is still not well established. The end of polishing (endpoint) is indicated by the amount of polishing time, which is set beforehand based on

the experiments and trial-and-error. To reduce the amount of rework, the wafers are usually overpolished slightly. It becomes essential to have an in-situ measurement capability to signal the endpoint at a right moment. Several methodologies are being used or under investigation: motor current sensing (friction force), film thickness measurement (acoustic, optical, electrical), etc.

Material removal rate (MRR) is of great concern, because it plays a major part in determining the throughput rate of a CMP tool. Thus a higher MRR is desirable. But increasing MRR beyond a certain point can reduce uniformity and surface quality. An optimum MRR increases the efficiency of the equipment, yet ensuring the effectiveness of the process it provides.

Throughput rate is not only affected by MRR, but also affected by other factors such as wafer handling and transportation time, cleaning time, conditioning time (ex-situ), number of heads and platens in a single machine, scheduling of individual process, etc. Throughput rate is a system issue indeed, and the request for a specific rate is to be met by considering those factors altogether.

There also are other measures of tool efficiency: tool capital cost, consumable cost, cost of ownership, tool size, ease of maintenance, etc. Table 1.2 shows the 2002 Production Tool Performance Targets from Semiconductor International magazine[2].

4. CMP α Machine

In order to meet the growing research need, a CMP machine has been developed at MIT by the CMP research group, as a successor of the simple test-bed machine developed two years ago. This α machine has a great flexibility and a high precision, and designed to excel other commercially available machines. Figure 1.3 shows the drawing of the machine.

The machine has one wafer carrier head and two platens, which enable multi-step polishing. The wafer carrier is mounted on the bracket, and the bracket can move up and down along the gantry Z axis. The gantry also provides X directional wafer carrier transportation. Two synchronized motors on each side of the gantry transmits x directional motion to the gantry via ball screws. The wafer carrier and the platens have their motors for rotation. The conditioner has one motor for the rotation of its head and also the other motor for the X direction transportation. Loading, unloading, and cleaning of a wafer are all performed at the load/unload/cleaning station. All of these components are mounted on top of the granite table, which has a good vibrational damping characteristic. The table is then mounted on the steel supporting structure.

There are several distinctive features in the machine. The machine is designed with precision in mind. Flatness of the granite table and the platens are ensured. Verticality of the gantry Z axis, the wafer carrier spindle, and the platen spindles are also double checked. All the translational motions will be controlled with less than 50 μm of precision, and all the rotational motions will have error bounds of ±0.1 rad/sec. The machine can provide wider range of process parameters. It can achieve 4 m/s of relative polishing speed, whereas other tools are typically limited to 1 or 2 m/s. It also provides polishing pressure of up to 20 psi,. However nominal polishing pressure is usually 7 psi.

Depending on the CMP process development, the machine may be asked to perform multi-step polishing. A wafer can be polished for a given amount of time on one platen with a certain set of process parameters- pressure, velocity, abrasive size and material, slurry

chemical, pad material and physical property, etc. Then it will be transferred to the other platen with a different set of process parameters. Usually a platen is dedicated to a specific slurry and pad, but other process parameters can be changed even within a single platen. A plausible scenario is two-step metal polishing: First step for major material removal with high speed and pressure, and large abrasive size; Second step for finishing and planarization with low speed and pressure, fine abrasive size, and selective chemicals. The machine is mainly designed for metal (Cu) polishing, but can also be used for oxide polishing and even for polymeric ILD polishing.

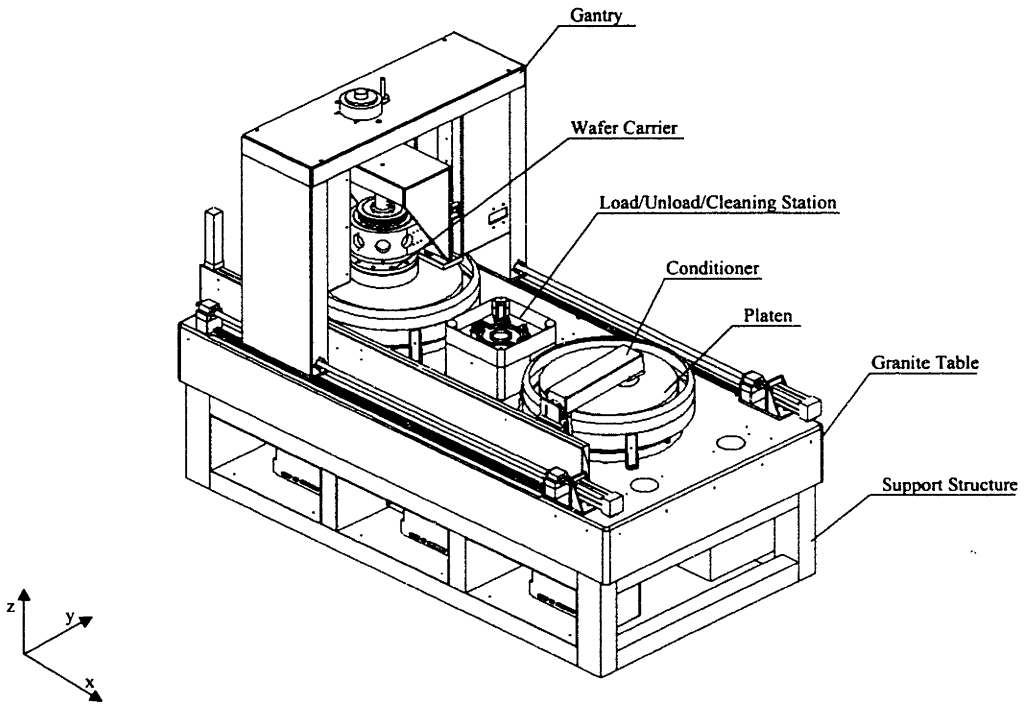


Figure 1.3 Overview of CMP α Machine

Two different types of slurry can be delivered to each platen at the same time. The slurry dispense lines can be used for deionized (DI) water flushing. A slurry can be dispensed either on top of the platen, or through the platen for a better slurry transportation. When a slurry is dispensed only on the top of the platen, it is often difficult for the slurry to flow to the middle of the polishing interface. The edge sees more slurry then the center, and thus it usually gets polished faster, contributing to the non-uniformity. By dispensing slurry through the holes in the platen and pad, an even distribution of slurry is achieved.

An optical sensor is embedded in the platen, which measures intensity of the reflected light, or reflectance, from the opposing surface, which is the wafer being polished. Metal has higher reflectance than oxides. Thus at the end of metal polishing, once underlying oxide layers are exposed, the reflectance value will drop significantly. By employing this principle, the reflectance measurement can be used for endpoint detection in metal

polishing. Signal acquisition methods and signal processing algorithms are being developed for Cu endpoint detection.

Many factors can affect the uniformity of the wafer surface- uneven pad profile, uneven slurry distribution, spatial difference of relative velocity over the polishing interface, etc. Due to the rotation of wafer, the circumferential layer thickness variation is usually less than the radial one. Typically, edge is being polished faster than the center, leading to a convex profile. In part to combat the problem, the pressure actuation compartment of the wafer carrier is divided into four concentric chambers, called Active Membrane Assembly (AMA), which can have different pressures. For example, to prevent the convex profile, the inner compartments may exert higher pressures, where as the outer compartments lower pressures. Another possibility is to actuate membranes dynamically from the reflectance feedback. Reflectance measurement can give the information about the remaining metal layer thickness (or if it is completely removed). The machine control system keeps track of the sensing position on the wafer. By combining film thickness metrology and spatial information, it is possible to actively actuate AMA during polishing. For example, if an area corresponding to the third compartment is detected to be polished faster than the rest, the process controller tells the AMA to reduce the pressure in the third compartment.

5. CMP α Machine Control System Overview

The machine control system needs to be designed to ensure the effectiveness of the CMP process and enhance the efficiency of the machine operation, by integrating all the intended system functions. The control system development includes the electrical wiring and interfacing of various actuators, sensors and the controller; the design and implementation of individual control algorithms (or controllers) for each drive and actuation component; creation of process steps, such as wafer loading, buffing, conditioning and polishing, which are generated by time-wise and conditional combinations of individual control actions; realization of machine operation, which allows a user to process sets of wafers to the desired specifications, using the various process parameters and sequences (recipe) in a fully automated environment.

The control system has a cabinet which houses most of the electrical components separate from the machine. The external processing system (a computer for system control) is located inside the cabinet. A 19" rack mountable, modular system, called ADwin, is selected as the real time process controller. ADwin has its own local 40 MHz RISC (Reduced Instruction Set Computer) DSP (Digital Signal Processor) and 32 MB of memory. Analog-to-Digital (AD), Digital-to-Analog (DA), Digital-In-Out (DIO), counter, relay modules can be mounted on the ADwin controller. The processor communicates with these modules via ADwin bus. For more information about ADwin, refer to its manual[12].

A host computer is required for the initial loading of the control program to ADwin and the communication and supervision while the controller is at work. A personal computer (PC) is selected for this purpose. The PC has a graphical user interface (GUI) to interact with a user, to save and edit recipes, and to monitor the machine operation. The PC has a Pentium III, 450 MHz CPU and runs under Windows NT 4.0 environment. It has a flat panel LCD (liquid crystal display) touch screen for an easy user interaction. Figure 1.4 gives the general idea of the CMP α machine control system.

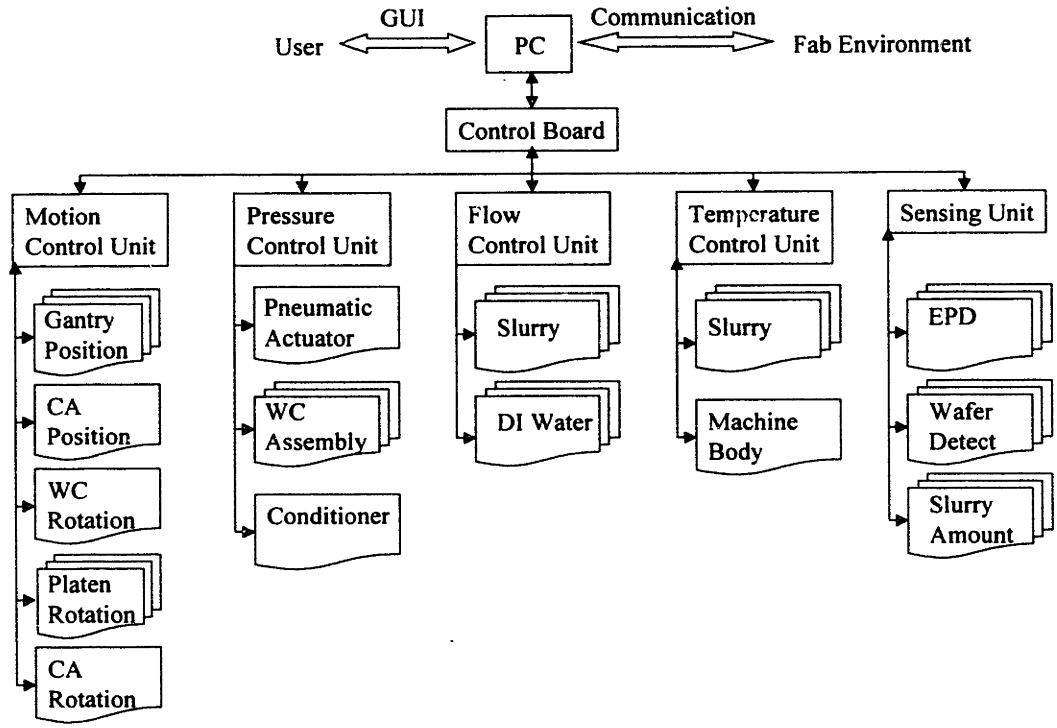


Figure 1.4 Overview of CMP α machine Control System

The machine has nine motors, with nine corresponding switching amplifiers. They all have incremental rotary encoders for position feedback. Two pumps are used to deliver slurries and DI water for each platen. Numerous valves are required for slurry and DI water transport control and for pneumatic control. Endpoint and wafer detection sensors are used for process control purposes. In addition, there are numerous relays for on/off type actuation. All these devices are located either in the control cabinet or on the machine and wired both in power and logic.

Wiring requires a careful planning and cautious implementation, because a single misplaced wire can ruin the whole control system. Wiring is also a greatly time consuming process. The biggest concern in wiring is to minimize noise pickup, generated from switching amplifiers and other electronic devices. Noise can interfere with the signal in/out (I/O) processing of the ADwin and the communication between the host PC and the ADwin. A small glitch can stop the whole machine and also undermine the safety of the machine operation. Certain methods will be employed to suppress the noise from propagation. But usually, they can not eliminate noise completely. The issue here is how to keep the noise below a certain level, so that the control system can perform its normal functions without being affected by the noise.

Once wiring is completed and proper signal I/Os for the ADwin are established, the next task is to design digital controllers. Digital controllers are implemented as software programs which run on the ADwin controller. Although confusing, the term controller is

used to mean both an external control circuit board (such as ADwin) and a specific control algorithm for a drive.

Relays and On/Off controls require just a single line of coding in control software. All the pneumatic and fluid valves will be controlled in open-loop, but they need to be calibrated so that the dictated value from the controller matches with the actual physical value in the valves. Analog sensors, such as the reflectance sensor and the strain gages need calibration, too.

For the CMP α machine, all the position and velocity components will be controlled in closed-loop. The gantry X axis, the gantry Z axis, the conditioner X axis and the wafer aligner angle are position controlled. Synchronization of two axes motors and providing fast enough speed for the heavy gantry mass (approx. 1100 Kg) are challenges of the gantry X axis position controller development. The weight of the wafer carrier (approx. 300 Kg) and maintaining the vertical position are the concerns in the gantry Z axis controller design.

Conditioner X position control is expected to be relatively easy because of its light dynamics. But coordinating its motion to avoid any collision with the wafer carrier is a demanding subject. Four arm wafer aligner is located in Loading/Unloading/Cleaning Station (LUCS), to center the wafer before the wafer pickup by the wafer carrier. A proper force control is required in order not to crush the wafer during aligning.

Velocity control is relatively easier than position control. The wafer carrier rotation, the two platen rotations and the conditioner rotation are all velocity controlled. The issue here will be maintaining a set velocity under the load, such as polishing or conditioning.

Having functional individual controllers at hand doesn't necessarily mean we can polish wafers. We have to create an intelligent set of the control actions, generated by the designed controllers, to embark any processing. For example, to move the wafer carrier from a position (X1, Z1) to (X2, Z2), we have to call both X and Z controllers with a certain sequence and possibly multiple times.

A small set of control actions can be designated as a 'sub-step.' A sub-step refers to a set of individual control actions, which is identifiable as a single unity during wafer processing. Wafer transport, wafer sweeping during polishing and wafer buffing can be the examples of the sub-step.

These sub-steps and other procedures can be combined to form a 'step,' which is a distinctive set of processes in the continuous stream of wafer processing. A step is an ingredient to form a recipe. By combining various steps, a process recipe can be designed. Loading, unloading, polishing and cleaning can be the examples of the step. The grouping is rather arbitrary, but performed in the viewpoint of a user, who has to design a recipe to obtain the desired product. Recipe is a set of process parameters and sequences to induce the desired physical change to a wafer which has a certain film layer (material, thickness, topography, etc.) to be planarized with underlying features.

The process control software will be designed to accommodate various possible recipes to run the machine. The process control software will send process commands to the machine control software. The machine side will recognize the commands from the process side, collect process data, perform required control actions, and update process parameters and machine status to the process side.

On the PC side, it will have a GUI, where the user can edit and retrieve recipes and operate the machine. Machine operation involves loading and unloading of control program to ADwin, loading and unloading of recipes, sending function commands (Run, Stop, etc.), supervising the work of ADwin control software, receiving process parameters and machine

status data from ADwin and updating on the GUI operation screen. Supportive features, such as set up, machine operation database (machine use log), and maintenance log should be included on the PC side.

6. Thesis Objective

The objective of this thesis is to develop the CMP α machine control system successfully, using the framework of Axiomatic Design. During the course, it will become evident that Axiomatic Design can provide a systematic design methodology for a complex control system design. Main hypotheses include:

1. Axiomatic Design can provide a systematic framework and guideline in developing a complex machine control system.
2. Axiomatic Design clearly identifies required control actions (FRs), corresponding controllers (DPs), and run-time control variables (PVs).
3. Modern control technology can be employed to realize the control action specified by AD during the decomposition with a reasonable tolerance (Controller design follows Axiomatic decomposition).
4. Axiomatic Design can be used to design the structure and the sequence of the process control software.
5. Hierarchical structuring and abstraction between the levels in a software improves readability, efficiency, and maintainability of the system, which are the natural consequence of software design based on Axiomatic Design .
6. The structure designed by AD is realizable by the current software technology (Programming follows Axiomatic decomposition).

7. Thesis Overview

This chapter has been the introduction, outlining the work involved in the CMP α machine control system development and presenting the objective of this thesis. In Chapter 2, the machine side control system is developed using the AD framework. The development continues in Chapter 3, which presents the design of the process control system. Chapter 4 deals with the design of a servo controller and its implementation. The entire system is integrated and implemented in Chapter 5. The conclusion is given in Chapter 6. The discussion about the future work follows in Chapter 7.

Chapter 2. Machine-level Control System

Axiomatic Design begins with identifying customer needs(Customer Attributes, CAs). In the case of the CMP α machine control system development, customers include MIT CMP research group(both the design and the process team), the research sponsor, and potentially the semiconductor industry as a market. Many of the needs have already been remarked in the introduction chapter. The following lists the most fundamental CAs.

- Provide the required precision.
- Minimize the non-processing time.
- Enable a fully automated wafer processing.
- Flexible recipe editing and its implementation.
- Provide an equipment use and maintenance log.
- Easy to operate.
- Safe to use.

These CAs can also be thought of as general guidelines or constraints throughout the overall system development process. These constraints will be explained more elaborately at each corresponding level.

Axiomatic decomposition often starts with a single objective or mission statement. In this case, the objective is clear: Integrate individual components to create an effective and efficient machine control system. Table 2.1 shows the mission statement, which serves as the highest level FR/DP set.

Table 2.1 Mission Statement (Highest Level FR)

Functional Requirement(FR)	Design Parameter(DP)
Coordinate the individual components of the CMP α machine in a systematic manner so that it can process wafers effectively and efficiently to the desired specifications.	Equipment control system

In the CMP α machine control system development, the issues are three fold: hardware interfacing, generating individual control actions, organizing those control actions to process wafers. Hardware interfacing belongs mostly to the domain of Electrical Engineering, employing the knowledge of circuit design, power electronics, signal conditioning, etc. Design of an individual controller also belongs to the realm of Control Engineering, with the indexes of tracking error, response time, robustness, optimization, etc. However, providing a framework for individual controllers and organizing the output of the controllers to achieve higher level goals are mostly system design issues. Thus, AD can be used from the beginning to structure the control system. AD can not design a controller itself. A controller is designed from the principles of Control Engineering. But it will identify the need of a certain type of controller to achieve FRs and place the controller as a DP, through the decomposition. Also, AD will be used to identify and layout the elements and methods of a controller, because a digital controller is essentially a block of coding running as a part of a greater control system software.

We define the equipment control system as a set of software designed to achieved the goal of coordinated wafer processing, with appropriate interfaces to the machine and the user. Thus it is natural to decompose the highest level FR into two sub FRs: Machine-level and Process-level. Table 2.2 is the decomposition of the mission statement.

Table 2.2 Decomposition of Mission Statement

Index:	Control System Design		
	Functional Requirements (FRs)		Design Parameters (DPs)
	Name	*Description*	*Description*
		Coordinate the individual components of the CMP a machine in a systematic manner so that it can process wafers effectively and efficiently to the desired specifications.	***Equipment control system***
1	Control	Generate control actions from each individual machine component.	Machine-level control system
2	Process	Organize the controlled outputs for wafer processing.	Process-level control system

$$\begin{Bmatrix} FR1 \\ FR2 \end{Bmatrix} = \begin{bmatrix} X & O \\ X & X \end{bmatrix} \begin{Bmatrix} DP1 \\ DP2 \end{Bmatrix}$$

Machine-level control system will concentrate on ensuring and structuring individual control actions of the machine, which are the basic ingredients of the overall equipment control system. Because it deals with the real time hardware control issues, it will be programmed for and running on ADwin external processing system. In contrast, Process-level control system deals with the issue of organizing these individual actions generated by the Machine-level to process wafers in desired manners. Thus it should supervise and communicate with the Machine-level, giving commands and taking statuses, interact with a user to accept user inputs and display machine parameters, and run the machine in a preprogrammed manner in an automatic wafer processing mode. Process-level control software naturally nests on the host PC.

Although not specified in this decomposition, we can think of a third design parameter, DP3: Operation-level control system. Our present goal is to process a single wafer with a given set of conditions in a fully automated manner. But in a real production environment, we are interested in operating the machine to process lots of wafers. For example, we may want to process two lots of 50 Cu wafers with 0.13 μm line width in the morning and a lot of 100 oxide wafers with 1 μm film thickness in the afternoon. Also, in a fab, a machine hardly works as a stand alone equipment. A CMP machine is usually located in a bay or cluster type configuration, with other equipments such as a cleaning station, an inspection station, and a wafer transporting and handling robot. Scheduling of the equipment, cooperation with other machines, and managerial supervision of 'operation' become important issues at this level. In most cases, this Operation-level control system will

run on a physically separate unit, such as a central processing and monitoring computer, connected via an intranet. DP3 addresses the necessity of a higher level control system and provides an interface for it. However, we will remain in the decomposition of DP1 and DP2, because the machine is more oriented to the research rather than the production.

It will simply be impossible to present every detail of the decomposition of all the branches and all the way down to the lowest levels. Rather, structuring of the system along the major branches will be described, with an occasional illustrative decomposition down to the required level where needs arise. We will first tackle the Machine-level control system. When we are comfortable with the clear picture of the Machine-level in mind, we will move on to the Process-level in the next chapter to zoom out.

DP1. Machine-level Control System

Machine-level control system is responsible for generating individual control actions of the machine in a systematic manner, so that they can be used as constituents in wafer processing and machine operation by higher level control systems. To meet the objective, a machine-level control system should have an In/Out(I/O) unit to handle communication with the machine and a higher level control system, an signal processing unit to condition incoming signals(mostly from the machine), a controller unit to command outputs to the machine and an overhead to structure and coordinate these units and the machine-level itself. Table 2.3 shows the decomposition of the Machine-level control system.

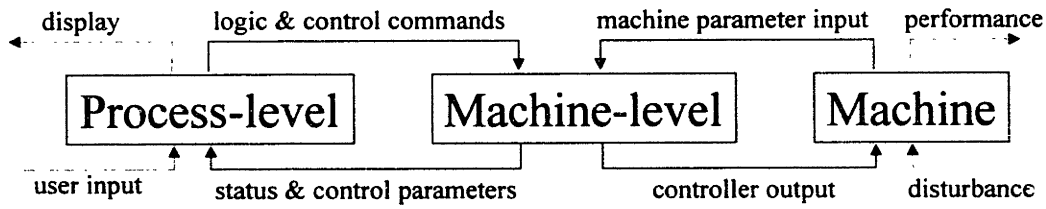
Table 2.3 Decomposition of Machine-level Control System

Index: 1.#	Control System Design		
	Functional Requirements (FRs)		Design Parameters (DPs)
	Name	*Description*	*Description*
		Generate control actions from each individual machine component.	***Machine-level control system***
1	Control	Receive inputs and send outputs.	I/O unit
2	Control	Process incoming signals.	Signal processing unit
3	Control	Produce controlled outputs.	Controller unit
4	Control	Structure the Machine-level control system.	Machine-level overhead

$$\begin{Bmatrix} FR11 \\ FR12 \\ FR13 \\ FR14 \end{Bmatrix} = \begin{bmatrix} X & O & O & O \\ X & X & O & O \\ X & X & X & O \\ X & X & X & X \end{bmatrix} \begin{Bmatrix} DP11 \\ DP12 \\ DP13 \\ DP14 \end{Bmatrix}$$

DP11. I/O Unit of the Machine-level Control System

The Machine-level control system basically interacts with the machine and the Process-level control system. In turn, the machine interacts with the environment and the Process-level interacts with a user and possibly an operation-level control system. Figure 2.1 conceptualizes the interactions of the Machine-level with other systems.



* Logic commands: function buttons, initialization
* Control commands: position, velocity, pressure, flow rate, valve and switch on/off, etc.

* Status parameters: error, operative status
* Control parameters: current position, velocity, sensor output, time, etc.

Figure 2.1 Interaction of Machine-level Control System with other systems

We decompose the I/O unit into two sub units by their corresponding counter parts- the machine and the Process-level, as shown in Table 2.4. The machine I/O unit deals with the interface with the machine to acquire the data for machine control and transmit the output to drive the machine to the desired states. Thus the machine I/O works as a bridge between the machine hardware and the control system software. As one can imagine, the formation of the machine I/O depends largely on the hardware configuration. Types of actuators and sensors and the natures of their signal dictate the forms of the required interface cards. Basically inputs and outputs are either analog(continuous) or digital(discrete), and interface cards are built to handle varieties of those basic signals.

Table 2.5 shows the decomposition of the machine I/O unit. A counter(CNT) card is required to decode the position information from the encoder signals installed on the mating motor shaft. An encoder typically gives two channels of TTL(transistor-transistor logic) level square wave signal. As described in Figure 2.2, one channel leads the other depending on the direction of the rotation. The state transition sequence of a counter-clockwise(CCW) rotation is different from the one of a clockwise(CW) rotation. Thus by observing the state sequence, a counter can determine in which direction the shaft is spinning, and it is reflected in counting(up or down). The counter decodes the position

information by counting the transitions of states(quadruple decoding). Thus an 1000 pulses/rev encoder gives 4000 counts/rev resolution in quadruple decoding. Starting from 0, it will count 4000 if the shaft makes one CCW revolution, -4000 if one CW revolution.

Table 2.4 Decomposition of I/O Unit

Index: 11.#	Control System Design		
	Functional Requirements (FRs)		Design Parameters (DPs)
	Name	*Description*	*Description*
	Receive inputs from and outputs to the machine		
1	Control	Handle inputs from and outputs to the machine.	Machine I/O unit
2	Control	Handle inputs from and outputs to the Process-level control system.	Process-level I/O unit

$$\begin{Bmatrix} FR111 \\ FR112 \end{Bmatrix} = \begin{bmatrix} X & O \\ O & X \end{bmatrix} \begin{Bmatrix} DP111 \\ DP112 \end{Bmatrix}$$

Table 2.5 Decomposition of Machine I/O Unit

Index: 111.#	Control System Design		
	Functional Requirements (FRs)		Design Parameters (DPs)
	Name	*Description*	*Description*
	Handle inputs from and outputs to the machine		Machine I/O unit
1	Control	Acquire encoder(position) signal	Counter(CNT) card
2	Control	Sense digital(Hi or Low) signal	Digital Input(DI) card
3	Control	Accept analog(continuous) signal	Analog-to-Digital Conversion(ADC) card
4	Control	Send On/Off(Hi/Low) signal	Digital Output(DO) card
5	Control	Output relay signal	Relay(REL) card
6	Control	Provide analog output	Digital-to-Analog Conversion(DAC) card

$$\begin{Bmatrix} FR1111 \\ FR1112 \\ FR1113 \\ FR1114 \\ FR1115 \\ FR1116 \end{Bmatrix} = \begin{bmatrix} X & O & O & O & O & O \\ O & X & O & O & O & O \\ O & O & X & O & O & O \\ O & O & O & X & O & O \\ O & O & O & O & X & O \\ O & O & O & O & O & X \end{bmatrix} \begin{Bmatrix} DP1111 \\ DP1112 \\ DP1113 \\ DP1114 \\ DP1115 \\ DP1116 \end{Bmatrix}$$

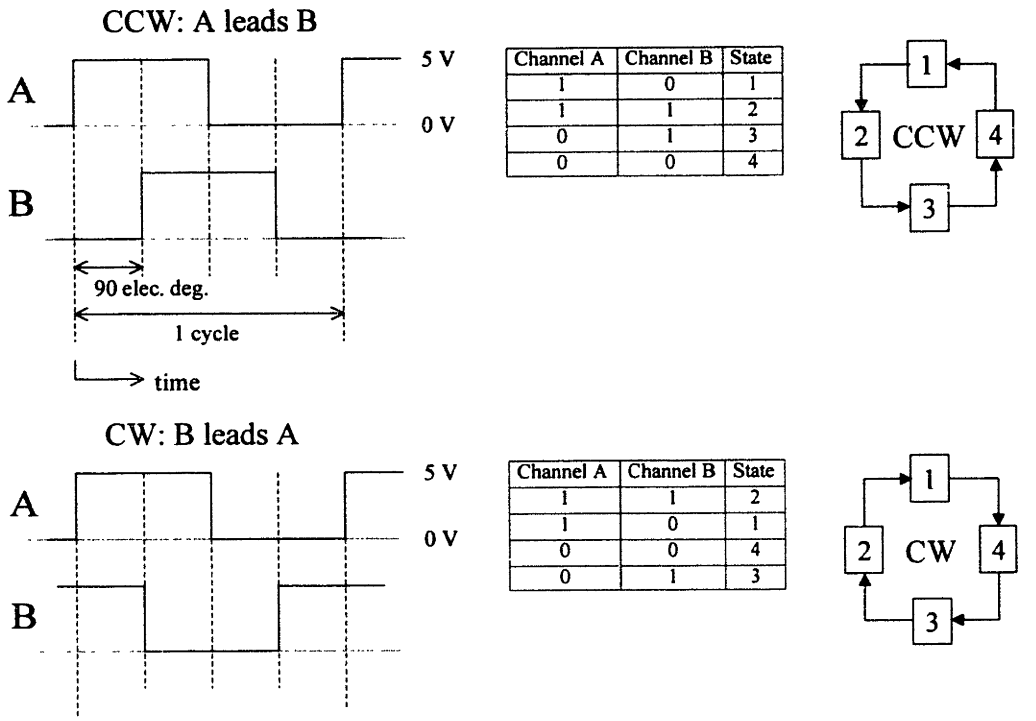


Figure 2.2 State Transitions of Two Channel Encoder

ADwin is equipped with 3 CNT-VR4 counter cards. Each CNT card has four counters with maximum input channel rate of 1.25 MHz(5 MHz quadruple count rate). Figure 2.3 shows the block diagram of a CNT-VR4 counter. Optionally each counter can be configured as a clock(single channel) and a direction(up or down) input. The output is stored as a 32 bit integer value in the register and transported to the CPU via ADwin-PRO bus.

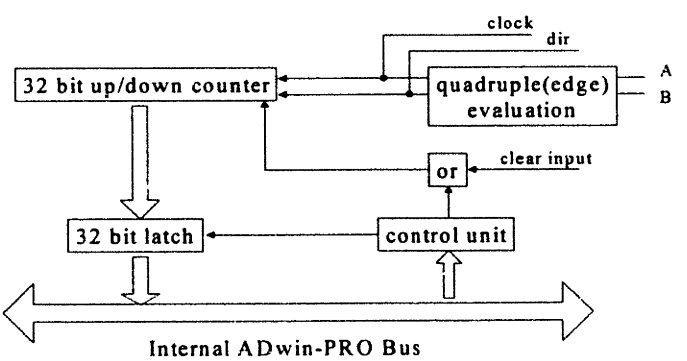


Figure 2.3 Block Diagram of CNT-VR4 Counter

Many on/off type sensors give TTL level signal, and a digital input(DI) card is required to capture the signal. Also a digital output(DO) card is required to transmit TTL level on/off signal either directly to an actuator or its amplifying circuit. ADwin has three Pro-DIO-32 cards with 32 digital input and output channels at TTL levels for each card. Each channel can be selected as input or output by software.

An analog-to-digital conversion(ADC) card is necessary to measure a continuous voltage signal. A Pro-AIn-8/16 card is equipped with a 16 bit ADC and 8 differential input channels, and has a maximum sampling rate of 50 kHz. ADC will be used to interface process sensors, such as a reflectance senor and strain gauges. Figure 2.4 shows the block diagram of the Pro-Ain-8/16 ADC card. Its output is a 16 bit integer variable for each channel.

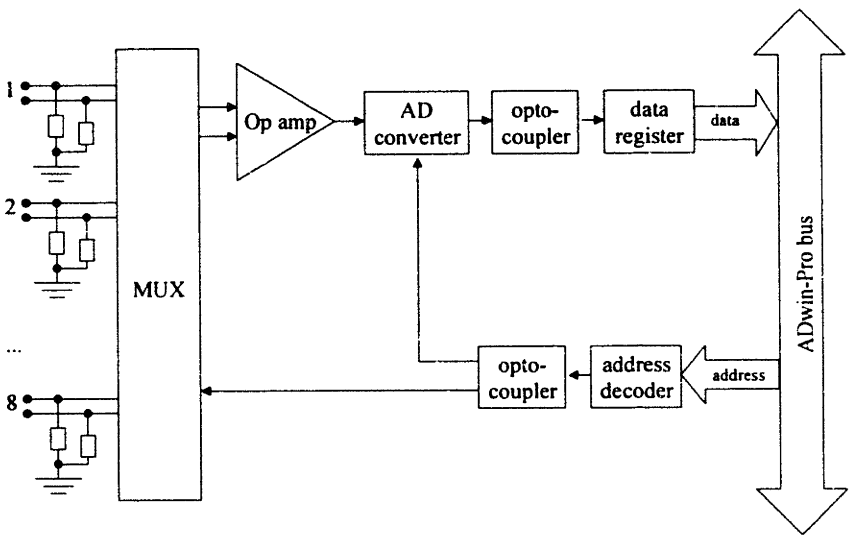


Figure 2.4 Block Diagram of Pro-Ain-8/16 ADC card

The Pro-REL-16 relay card has 16 isolated relay outputs. Each relay channel is driven to open or closed contact by the built in transistor switch, which is in turn controlled by a digital output. In that sense, it is a variation of DO card, with built in relays and transistor switches. Figure 2.5 shows a simple diagram of a relay output circuit. Each relay can switch up to 30 V AC/DC and 500 mA current.

Servo amplifiers and variable speed pumps usually accepts an analog voltage input as a command signal. In a computer controlled system, both open loop and closed loop controllers are implemented by a software program. Thus, to transmit a controller output to the corresponding plant, a digital-to-analog conversion(DAC) card is required. The Pro-Aout-8/16 has 8 channels of 16 bit DACs with fixed 1st order low-pass filters. Each channel can output up to ±10 V and 5 mA with 20 μsec of settling time. Figure 2.6 shows the schematic of the Pro-Aout-8/16 analog output module.

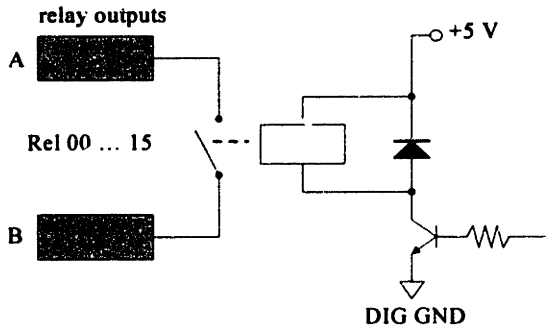


Figure 2.5 Circuit Diagram of Pro-REL-16 Relay

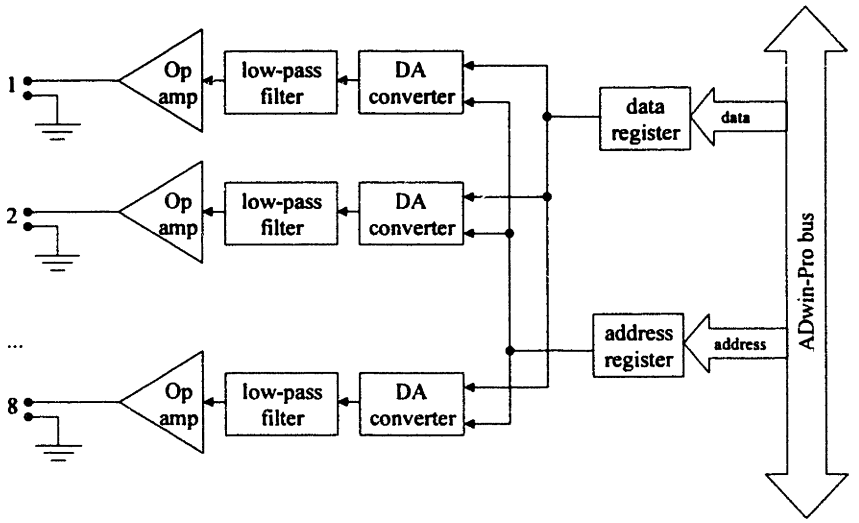


Figure 2.6 Block Diagram of Analog Output Module, Pro-AOut-8/16

The Machine-level control system should also communicate with the Process-level control system to receive command inputs and provide machine status and process data. The ADwin system is equipped with two Analog Device's 40 MHz ADSP 21062 digital signal processor(DSP) chips. Each processor has two types of memory: 256 KB of internal memory to load program and local variables; 16 MB of external dynamic memory(DRAM) for array variables which are typically used for data measurement and storage. ADbasic, the programming language of ADwin, provides drivers so that a host PC can access the memories of the ADwin processors. For more information about the ADwin processor memory structure and accessibility, please refer to the ADbasic manual[5].

We use the memories of DSPs to design the Process-level I/O unit of the Machine-level control system. Table 2.6 shows the decomposition of DP112. Command inputs

include both logic and control commands. Run, Stop, Reset, and Initialize are examples of logic commands, which mainly handles process controls. Control commands are instructions for desired control actions, such as position, velocity, and flow rate. The number of the command input is typically less than 100. Thus it is reasonable to use a section of the DSP internal memory writable by the host PC. By the same token, parameter outputs of the Machine-level control system are placed in a segment of the DSP internal memory, accessible by the host PC. Parameters include both status and control parameters. Status parameters indicate the status of the machine and the error code, if an error is set. Control parameters are machine parameters including, current positions, velocities, sensor values, time, etc.

A large amount of data(typically on the order of MB) is collected during wafer processing for the purpose of measurement and on- and off-line analysis. Reflectance of wafer surface during polishing is an example of process data. If the host PC requires the data for a higher level analysis and storage, the ADwin should be able to temporarily house the data until the host come and pick them up. 16 MB of external memory of DSP can be used for this purpose. However, one should be careful not to overwrite the memory.

Table 2.6 Decomposition of Process-level I/O Unit

Index: 112.#	Control System Design		
	Functional Requirements (FRs)		Design Parameters (DPs)
	Name	*Description*	*Description*
		Handle inputs from and outputs to the Process-level control system.	***Process-level I/O unit***
1	Control	Receive command inputs	Internal memory of DSP writable by the Process-level
2	Control	Send parameter outputs	Internal memory of DSP readable by the Process-level
3	Control	Send data outputs	External memory of DSP readable by the Process-level

$$\begin{Bmatrix} FR1121 \\ FR1122 \\ FR1123 \end{Bmatrix} = \begin{bmatrix} X & O & O \\ x & X & O \\ O & O & X \end{bmatrix} \begin{Bmatrix} DP1121 \\ DP1122 \\ DP1123 \end{Bmatrix}$$

DP12. Signal Processing Unit of the Machine-level Control System

A raw signal from the machine first needs to be converted to an appropriate unit used in the control system software, and then differentiated and integrated, if necessary. Typically, an incoming signal contains a lot of noise, superposed on the original signal. Proper filtering is quite essential in a high precision measurement and control. A controller output is also required to be converted to a required format by the I/O unit. Process date needs to be saved temporarily until requested by the Process-level control system.

Signal processing unit conditions incoming signals from the machine so that they can be used both by the Machine-level and the Process-level control system. Most signals are processed to be used by the Machine-level controllers, but some processed signals are fed directly to the Process-level for a higher level process monitoring and control. Wafer reflectance signal is an example of locally conditioned information, yet used for a higher level process control. Table 2.7 is the decomposition of the signal processing unit.

Table 2.7 Decomposition of Signal Processing Unit

Index: 12.#	Control System Design		
	Functional Requirements (FRs)		Design Parameters (DPs)
	Name	*Description*	*Description*
		Process incoming signals.	***Signal processing unit***
1	Control	Convert an input to the desired unit.	Input conversion unit
2	Control	Differentiate an input signal.	Differentiation unit
3	Control	Filter the input.	Filtering unit
4	Control	Integrate required signals.	Integration unit
5	Control	Convert controller outputs to the I/O unit units	Output conversion unit
6	Control	Save data internally.	Internal data save unit

$$\begin{Bmatrix} FR121 \\ FR122 \\ FR123 \\ FR124 \\ FR125 \\ FR126 \end{Bmatrix} = \begin{bmatrix} X & O & O & O & O & O \\ X & X & O & O & O & O \\ X & X & X & O & O & O \\ X & X & X & X & O & O \\ X & X & X & X & X & O \\ O & O & O & O & O & X \end{bmatrix} \begin{Bmatrix} DP121 \\ DP122 \\ DP123 \\ DP124 \\ DP125 \\ DP126 \end{Bmatrix}$$

An encoder counter gives a 32 bit integer value, but for control purposes it needs to be converted to an angle either in radian or degree. The following simple formula converts a count to a radian in an incremental manner. To decode an absolute rotary position(θ: $0 \sim 2\pi$), a modulo q operation can performed on n.

$\theta = 2\pi \times n/q$

θ: motor shaft angular position, [radian]
n: encoder count
q: number of counts per revolution

An ADC input is expressed by a 16 bit unsigned integer value. With bipolar 10 V input range, -10 V corresponds to 0 and +10 V to 65535. Suppose the reflectance sensor output has a range of 0 to 10 V corresponding to 0 to 100 % of reflectance. An ADC input n can be converted to the reflectance r by the following simple formula.

$r = m/V \times V/l \times (n - c)$

r: reflectance, [%]
m/V: reflectance ratio, 100/10, [%/V]
V/l: voltage ratio, 10/32768, [V]
n: 16 bit ADC value
c: integer offset, 32767

Also an input should be checked for the validity with respect to its value range. For example, if a slurry pump has the flow rate range of 0 to 1000 ml/min, the command input of 1500 ml/min is impossible, and if unchecked, it will either cause an error on the pump or saturate the DAC output line. The input range check will be performed on the Process-level after the user enters the command variable. However, it is still recommended to perform the second input range check on the Machine-level as a fail-safe, and also because a noise can be picked up during the transmission from the Process-level to the Machine-lelvel.

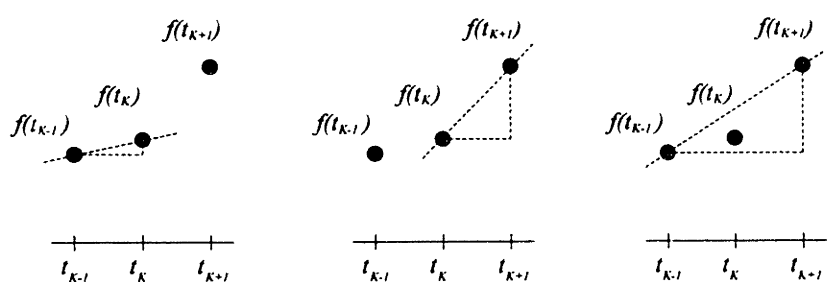


Figure 2.7 Three Difference Methods to Approximate $f'(t_k)$

Differentiation is often performed to extract the velocity and the acceleration from a positional information, in the condition that a complete time information is available to the signal processing unit. However in reality, the differentiation unit runs in a discrete time domain, where differentiation is emulated by a variety of difference techniques. Suppose a positional information is given by a (sampled) function $f(t_k)$, then the derivative of the function(velocity) at a point t_k is approximated by the slope of the tangent line at t_k using the values of the function at points near t_k. The approximation can be done in several ways. Figure 2.7 illustrates the most popular three methods.

In a backward difference approximation, the derivative at t_k is estimated by the slope of the line between $f(t_k)$ and $f(t_{k-1})$. Thus in a time domain, the derivative is determined by the present and the past value.

backward difference: $$f'(t_k) = \frac{f(t_k) - f(t_{k-1})}{t_k - t_{k-1}}$$

Forward difference approximates the derivative by the slope of the line between $f(t_k)$ and $f(t_{k+1})$, or the derivative is determined by the present and the future value.

forward difference: $$f'(t_k) = \frac{f(t_{k+1}) - f(t_k)}{t_{k+1} - t_k}$$

A combination of backward and forward difference is possible by using the slope between $f(t_{k-1})$ and $f(t_{k+1})$. This type of derivative approximation is called a central difference approximation.

central difference: $$f'(t_k) = \frac{f(t_{k+1}) - f(t_{k-1})}{t_{k+1} - t_{k-1}}$$

Central difference is the most accurate among the three, but backward difference is the most widely used in control and signal processing, because the differentiation is performed with the known(present and past) values, without the need to wait for future values. In other words, there is no computational delay involved in calculating derivatives. The backward difference scheme will be used in the differentiation unit of the Machine-level control system.

Incoming signal usually contains lots of noise picked up during the transmission. A digital(low or high) signal and its complementary one are usually filtered by a differential operational amplifier(op amp). An analog signal is often filtered by various types of low-pass filters. Hardware filters can be placed before the data acquisition cards of the ADwin. Some cards have built-in filters before and after sampling. However, even after A/D conversion, noises are still present on the measurement value. Further software(digital) filtering is required to attenuate the noise below an acceptable level. Differentiation often amplifies the superimposed noise on the original signal. Thus, filtering should be performed after differentiation, if necessary.

A digital filter has the following general form:

$$\hat{y}(kh) = -a_1\hat{y}[(k-1)h] - a_2\hat{y}[(k-2)h] - \ldots - a_n\hat{y}[(k-n)h] + b_0 y(kh) + \ldots + b_m y[(k-m)h]$$

where h is the sampling interval, $\hat{y}$ the filtered output and y the input measurement value. If all the a_is are zero, the filtered output is determined solely by the measurement inputs. This type of filter is called a moving average(MA) filter. If some of the a_is are non-zero, the filtered output is determined as a weighted sum of both the measurement values and the previous filtered outputs, which is called an autoregressive(AR) filter. The general form is called an autoregressive moving average (ARMA) time series.

Moving average and exponential filters are the most widely used low pass filters. For example, a 3-point MA filter can be constructed by the following simple formula.

$$\hat{y}(kh) = \frac{1}{3}\{y(kh) + y[(k-1)h] + y[(k-2)h]\}$$

This MA filter has a finite impulse response. Thus if an impulse is given at $t = 0$ then $\hat{y}$ becomes 0 after $t = 3h$.

An exponential filter is a first-order ARMA filter, given by:

$$\hat{y}(kh) = \alpha\ \hat{y}[(k-1)h] + (1-\alpha)y(kh) \qquad \text{or}$$
$$\hat{y}(kh) = \hat{y}[(k-1)h] + (1-\alpha)\{y(kh) - \hat{y}[(k-1)h]\}$$

where $0 \le \alpha \le 1$. The filtered value is computed as a weighted average of the previous filtered value and the present measurement. The exponential filter is a discrete form of a first-order continuous analog low pass filter. Suppose the following transfer function of a first order filter, with the time constant T.

$$G(s) = \frac{\hat{Y}(s)}{Y(s)} = \frac{1}{1+sT}$$

In a time domain,

$$T\frac{d\hat{y}(t)}{dt} = -\hat{y} + y$$

If we approximate the derivate by the backward difference, we obtain:

$$\frac{\hat{y}(t) - \hat{y}(t-h)}{h} = -\frac{1}{T}\hat{y}(t) + \frac{1}{T}y(t)$$

Rearranging the equation, we get:

$$\hat{y}(t) = \frac{1}{1+h/T}\hat{y}(t-h) + \frac{h/T}{1+h/T}y(t)$$

Comparison with the formula of the exponential filter reveals that:

$$\alpha = \frac{1}{1 + h/T}$$

Thus we verify that the exponential filter is a 1st order low pass filter. Higher order filters can be constructed by connecting first-order filters in series.

It is often necessary to integrate an incoming signal. The most common usage is the integration of the error signal in a PID type controller to minimize the tracking error. In a discrete time domain, integration is performed by summing the products of the measurement value and the sampling interval. The rectangular and the trapezoidal rule are widely used in numerical integration. Figure 2.8 shows the schematic of these two rules.

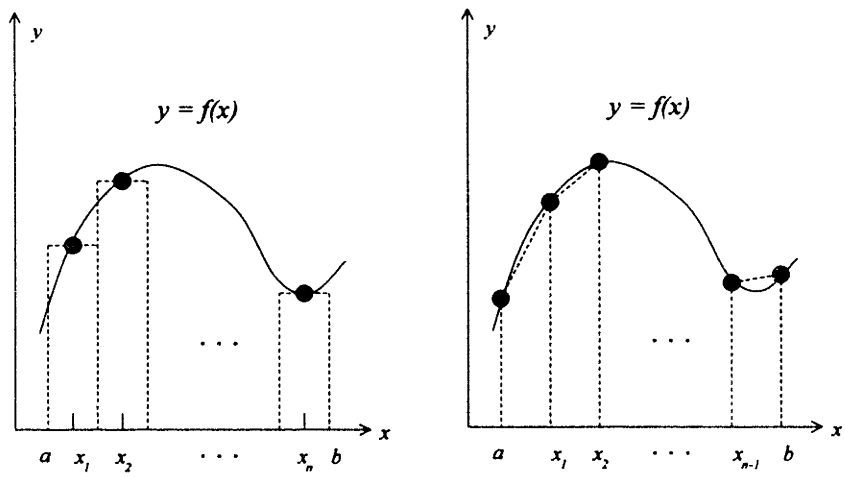


Figure 2.8 Two Methods of Numerical Integration

The rectangular rule is obtained by subdividing the interval of the integration $a \leq x \leq b$ into n subintervals of equal length $h = (b-a)/n$ and by approximating $f(x_i)$ at the middle of each interval. Then f is approximated by a series of step functions. The rectangular rule gives the following formula:

$$\int_a^b f(x)\,dx \approx h[f(x_1) + f(x_2) + \cdots + f(x_n)]$$

The trapezoidal rule is generally more accurate, and is obtained by the same subdivision as in the rectangular rule but with $f(x_i)$ approximated at each vertex of the

subdivision. Each area of the subdivision takes the form of a trapezoid. The trapezoidal rule can be expressed by:

$$\int_a^b f(x)\,dx \approx h[\frac{1}{2}f(a) + f(x_1) + f(x_2) + \cdots + f(x_{n-1}) + \frac{1}{2}f(b)]$$

The output of an controller needs to be converted to an appropriate format, so that the output unit can transmit to the corresponding actuator. For example, if a controller has ±2 Nm of torque command range for a servo motor with a torque constant of 1 Nm/A, we have to convert the torque command to an unsigned 16 bit integer value (0 ~ 65535(2^{16})), so that a DAC channel can transmit as a voltage command (-10 ~ 10 V) to the corresponding servo amplifier. The amplifier supplies an armature current proportional to the input voltage command. In a brushless DC servomotor, the output torque is proportional to the armature current. Thus by combining all of these scalings in a single equation, we can write an output conversion equation for the DAC output unit.

$$n = \frac{T}{K\alpha\beta} + c$$

n: 16 bit DAC value
T: required torque, [Nm]
K: motor torque constant, 1, [Nm/A]
α: armature current constant, 2/10, [A/V]
β: voltage constant, 10/32768, [V]
c: integer offset, 32767

The output needs to be checked against its limit. Especially, the output of a PID controller can be quite large to compensate the error when there is a large disturbance. However the large output can easily saturate the DAC line and the servo amplifier. Thus any output the DAC, n, should be checked for its maximum(65535). If it is greater than the allowable maximum, the output conversion unit should export the legal maximum, instead.

A few measurement data need to be saved temporarily to the memory of DSP, either for the Process-level data acquisition(large array) or for internal signal processing purposes, such as past position and velocity for filtering and differentiation. The internal data save unit handles these types of data storages.

DP13. Controller Unit of the Machine-level Control System

The controller unit is the backbone of the Machine-level control system, because all the control actions take place here, which is the main functional requirement of the Machine-level control system. The other three design parameters, in a sense, serve as supportive units for the controller unit. The controller unit can be categorized into three classes of controllers based on the types of the plants to be controlled, as shown in Table 2.8.

On-off controllers control the on-off(bang-bang) type devices. This type of controller is basically a switch to activate or deactivate the device. Lamps, valves, relay switches are the examples of on-off type devices. In ADwin, an on-off signal is generated either by a hi-low(5-0 V) signal of a DO channel or by an open-close contact of a REL channel. A DO signal is usually used to drive the switching circuit of a device. Switching circuits usually consist of a series of switching transistors. A REL signal is generally used to close or open directly the switching circuit.

Table 2.8 Decomposition of Controller Unit

Index: 13.#		Control System Design	
	Functional Requirements (FRs)		Design Parameters (DPs)
	Name	*Description*	*Description*
		Produce controlled outputs	***Controller unit***
1	Control	Handle on/off type devices	On-off controller
2	Control	Drive open-loop type components	Open-loop controller
3	Control	Manipulate closed-loop type plants	Closed-loop controller

$$\begin{Bmatrix} FR131 \\ FR132 \\ FR133 \end{Bmatrix} = \begin{bmatrix} X & O & O \\ O & X & O \\ O & O & X \end{bmatrix} \begin{Bmatrix} DP131 \\ DP132 \\ DP133 \end{Bmatrix}$$

Table 2.9 Decomposition of On-off Controller

Index: 131.#		Control System Design	
	Functional Requirements (FRs)		Design Parameters (DPs)
	Name	*Description*	*Description*
		Handle on/off type devices	***On-off controller***
1	Control	Create a general structure for the on-off controllers	On-off controller structure
2	Control	Control individual on/off type devices	Individual on-off controllers

$$\begin{Bmatrix} FR1311 \\ FR1312 \end{Bmatrix} = \begin{bmatrix} X & O \\ X & X \end{bmatrix} \begin{Bmatrix} DP1311 \\ DP1312 \end{Bmatrix}$$

A REL card has 16 channels of relays(REL00 ~ REL15). A 16 bit integer is used as a bitmap to command the type of the contacts for each channel. Suppose we want to close the relay 15, 10, and 0, with other relays remaining open, then the following bit map fed to the REL card via the output unit will do the job:

n = 1000001000000001B

n: 16 bit integer for a REL card, binary

We used the convention that LSB(Least Significant Bit) represents the lowest relay and that a high(1) bit represents a closed contact.

A DO card has 32 digital output channels(DO00 ~ DO31). These 32 channels are controlled by two 16 bit integer variables, $n1$ and $n2$. $n1$ controls the first 16 channels(DO00 ~ DO15), and $n2$ takes care of the rest(DO16 ~ DO31). As in the REL card bitmap, we can control each channel individually by setting the corresponding bit either high(+5 V output) or low(0 V output). If we want to turn on the device 10, 9 and 1, we set the following bitmap.

$n1$ = 0000011000000010B

$n1$: 16 bit integer for the first 16 channels of a DO card, binary

Again, we used the convention that LSB represents the lowest channel and that a high bit represents a high(+5 V) output signal.

A general structure of the on-off controller should be designed first, and this structure will be applied in designing individual controllers. The on-off controller can be decomposed as in Table 2.9. Because of the complex nature of the wafer processing, more than 50 on-off type devices are installed in the machine. Wafer polishing is essentially a combined art of electrical, mechanical, pneumatic, and fluidic actuations. Figure 2.9 enumerates the number and the type of individual on-off controllers.

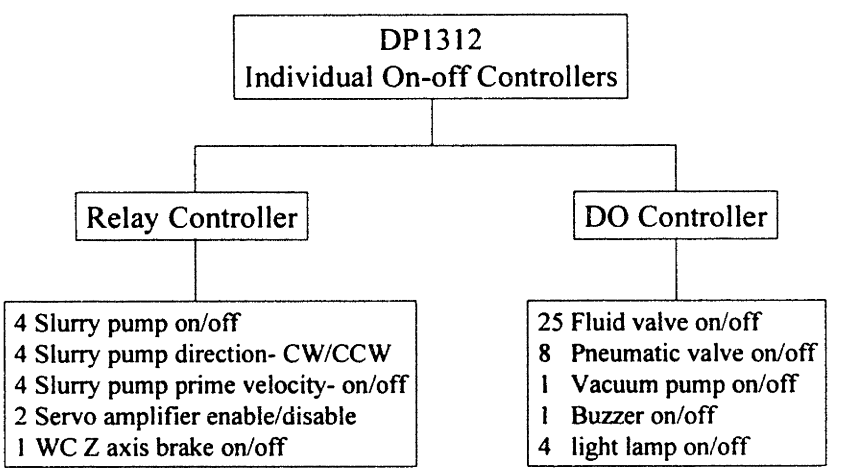


Figure 2.9 Quantities and Types of On-off Controllers

The general structure of an on-off controller is simple: Read → Convert → Write, as shown in Table 2.10.

Table 2.10 Decomposition of On-off Controller Structure

Index: 1311.#	Control System Design		
	Functional Requirements (FRs)		**Design Parameters (DPs)**
	Name	*Description*	*Description*
		Create a general structure for the on-off controllers	***On-off controller structure***
1	Control	Receive inputs	Read procedure
2	Control	Convert to an output	Convert procedure
3	Control	Send the output	Write procedure

$$\begin{Bmatrix} FR13111 \\ FR13112 \\ FR13113 \end{Bmatrix} = \begin{bmatrix} X & O & O \\ X & X & O \\ X & X & X \end{bmatrix} \begin{Bmatrix} DP13111 \\ DP13112 \\ DP13113 \end{Bmatrix}$$

Step 1: Read
Inputs to the on-off controller are both the output bitmap and the input command. The input command can be either a Boolean command or a bitmap received from the Process-level. Suppose we want to control the device 1, which is dictated by the bit 3 of the input bitmap command.

Step2: Convert
We have to extract the bit 3 from the input bitmap command.

b = shift_right((i AND 01000B), 3)

b: internal variable of the on-off controller, Boolean
i: incoming bitmap command, integer
AND: bitwise operation, 1 AND 1 = 1, otherwise 0

The AND operation applies the bitmask 01000B to the input bitmap to extract the bit 3. Then the result is shift by 3 bits to the right. Thus, if the bit 3 is set, b becomes 1(True), otherwise 0(False).

Step3: Write
As a final step, we write b to o, the output bitmap.

o = o OR shift_left(b, 1)

o: output bitmap, integer

OR: bitwise operation, 0 OR 0 = 0, otherwise 1

The output bitmap is initialized to 0 at the beginning of every event by the Machine-level overhead. The Boolean value *b* is shifted to the corresponding output bit position, and is written to *o* by the OR operation. The output bitmap is repeatedly fed to each device controller and, at the end of the event, is transferred to the output unit. Then the output unit converts the bitmap into the physical control actions.

Figure 2.10 shows the diagram of a signal tower light switching circuit, controlled by the output of the on-off controller. At 0 V DO output, no current flows through the bases of the dual transistor, thus there is no returning current through the LED. Once DO is changed to 5 V, the resulting base current turns on the transistors. Then the return line of the LED is connected to the DC ground and a current flows through the LED from the +24 VDC source, which in turn generates the light.

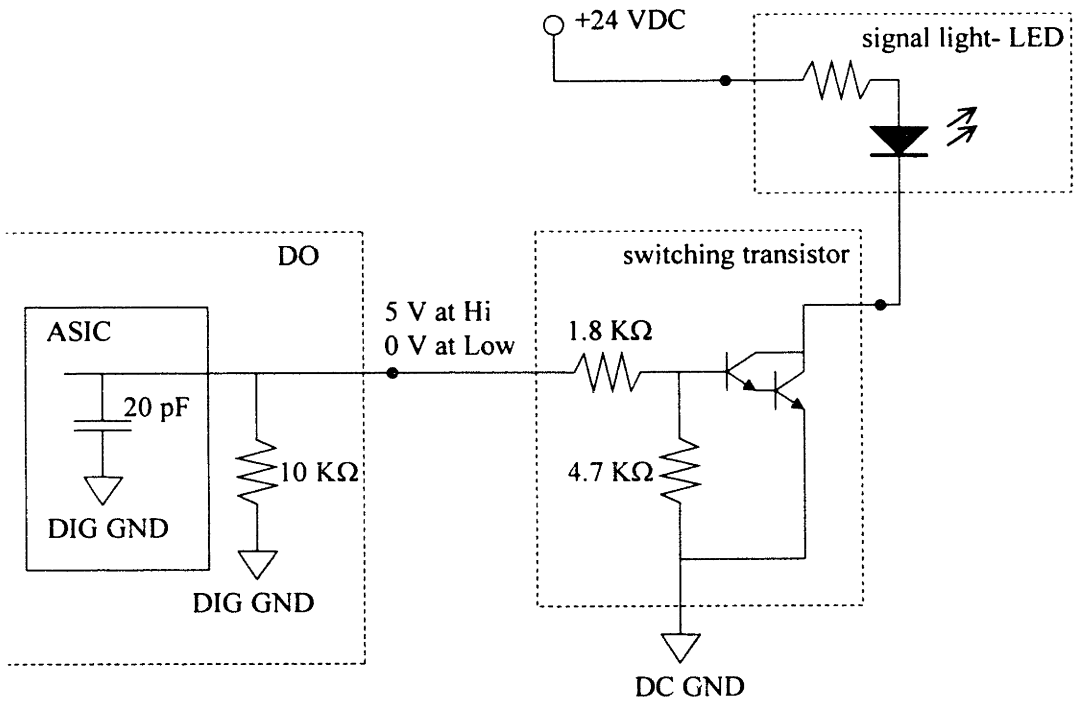


Figure 2.10 Switching Circuit of LED Signal Light

Open-loop controllers control open-loop type devices. The output is continuous(analog), compared to the binary output of the on-off controller. Also the output is directly proportional to the input command, because there is no feedback and error correction mechanism programmed. However many open-loop controlled devices have their own feedback loop built in. For example, a pneumatic pressure regulator sets its output pressure at a constant command value, despite fluctuations in the input pressure and the disturbances in the output line. In the system point of view, this type of localized closed-loop control(open-loop for the system) is a great benefit, because it can save lots of

computation time and program space for the main controller. However, open loop controllers usually don't provide fast and accurate responses. Open-loop control is usually used to control the slow-varying and low precision processes, such as pressure and flow rate.

As in the decomposition of the on-off controller, the open-loop controller can be decomposed into the general structure and the individual controllers. Table 2.11 illustrates this decomposition. Open-loop controllers can be grouped into the pressure controller and the flow rate controller. Figure 2.11 enumerates each individual controller.

Table 2.11 Decomposition of Open-loop Controller

Index: 132.#		Control System Design	
	Functional Requirements (FRs)		Design Parameters (DPs)
	Name	Description	Description
		Drive open-loop type components	*Open-loop controller*
1	Control	Create a general structure for the open-loop controllers	Open-loop controller structure
2	Control	Control individual open-loop type devices	Individual open-loop controllers

$$\begin{Bmatrix} FR1321 \\ FR1322 \end{Bmatrix} = \begin{bmatrix} X & O \\ X & X \end{bmatrix} \begin{Bmatrix} DP1321 \\ DP1322 \end{Bmatrix}$$

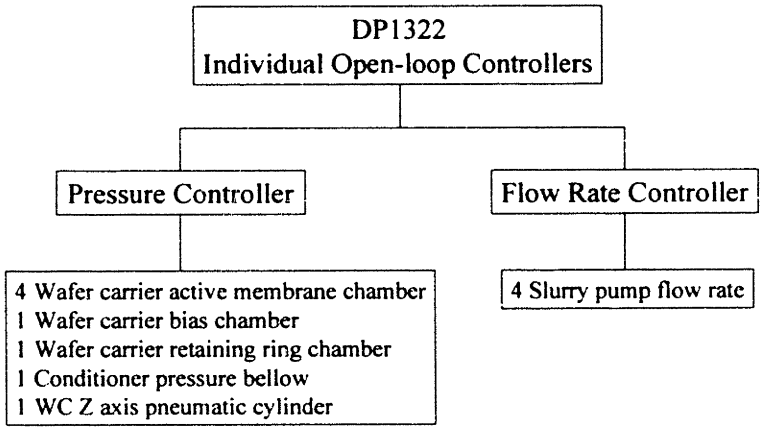


Figure 2.11 Types of Open-loop Controllers

Open-loop devices are often needs to be calibrated before the design of the open-loop controllers. A voltage input-pressure output relationship of a pressure regulator, for example, should be plotted before hand. Most open-loop devices have linear input-output

relationship with nonlinearity typically less than 1 %. In this case, we can find a first order relationship curve by the least square fit of the measurements.

Table 2.12 Decomposition of Open-loop Controller Structure

Index: 1321.#	Control System Design		
	Functional Requirements (FRs)		Design Parameters (DPs)
	Name	*Description*	*Description*
		Create a general structure for the open-loop controllers	***Open-loop controller structure***
1	Control	Receive inputs	Read procedure
2	Control	Convert to an output	Convert procedure
3	Control	Send the output	Write procedure

$$\begin{Bmatrix} FR13211 \\ FR13212 \\ FR13213 \end{Bmatrix} = \begin{bmatrix} X & O & O \\ X & X & O \\ X & X & X \end{bmatrix} \begin{Bmatrix} DP13211 \\ DP13212 \\ DP13213 \end{Bmatrix}$$

Open-loop controller structure can be decomposed to read, convert and write procedure, as in Table 2.12, following the same pattern as in the on-off controller structure decomposition. For example, a pressure regulator controller should read the command pressure p at the read procedure. The pressure command is converted to an integer value, and finally written to a DAC variable at the write procedure. Figure 2.12 summarizes this process in the block diagram.

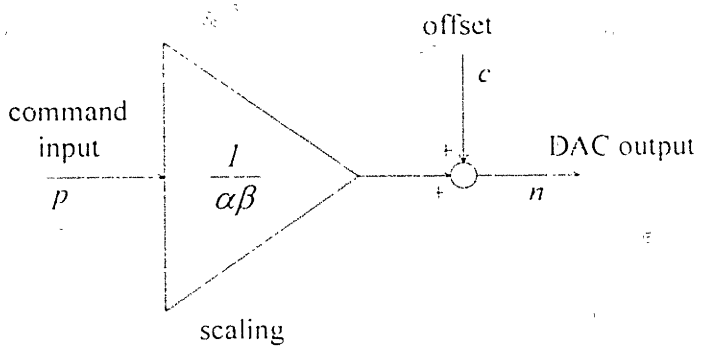


Figure 2.12 Block Diagram of the Open-loop Controller

$$n = \frac{p}{\alpha\,\beta} + c$$

n: 16 bit integer DAC variable
p: pressure command, [psi]
α: voltage constant, [V]
β: pressure constant, [psi/V]
c: integer offset, 32767

Closed-loop control is used where the high precision and the fast response are required. The motion(servo) control of a machine is usually done by the closed-loop control, because the motion control is directly related to the precision of the machine process and the safety of the machine operation. In the current Machine-level control system, the closed-loop controller is equivalent to the motion controllers. We can decompose the closed-loop controller into the general structure and the individual controllers as in Table 2.13.

Table 2.13 Decomposition of Closed-loop Controller

Index: 133.#	Control System Design		
	Functional Requirements (FRs)		Design Parameters (DPs)
	Name	Description	Description
		Manipulate closed-loop type plants	*Closed-loop controller*
1	Control	Provide a general structure for closed-loop controllers	Closed-loop controller structure
2	Control	Control individual closed-loop plants	Individual closed-loop controllers

$$\begin{Bmatrix} FR1331 \\ FR1332 \end{Bmatrix} = \begin{bmatrix} X & O \\ X & X \end{bmatrix} \begin{Bmatrix} DP1331 \\ DP1332 \end{Bmatrix}$$

The machine has 8 servo controlled components. Thus 8 closed-loop controllers need to be designed. However, the wafer carrier is transported in X direction by the motion of the gantry, and the gantry has two ball screw and motor actuation mechanism, one on each side. The total motor count becomes 9 and the WC X axis controller should synchronize two motors to provide a coordinated X motion to the gantry. Figure 2.13 lists the types of motion controllers.

A motion controller is either a position or a velocity controller, although both can be controlled at the same time in a limited manner. The velocity control is usually much easier and simpler than the position control, because typically a less precision is required in a velocity control than in a position control. However, the general structure of the controller will be the same in both cases.

The design of a closed-loop controller typically involves the following processes: Selection of the performance index(target) → Modeling of the physical plant → Parameter

Identification → Selection and design of the controller → Testing and the controller gain tuning → Implementation and further optimization. Rise time, maximum overshoot and steady state error are common performance indices. After setting a reasonable control target, the corresponding physical plant is modeled for the analysis and the controller design. Because not every parameter is known at the time of modeling, many experiments are performed after the modeling to identify the modeled parameters. For example, let's consider the design of the WC Z axis position controller.

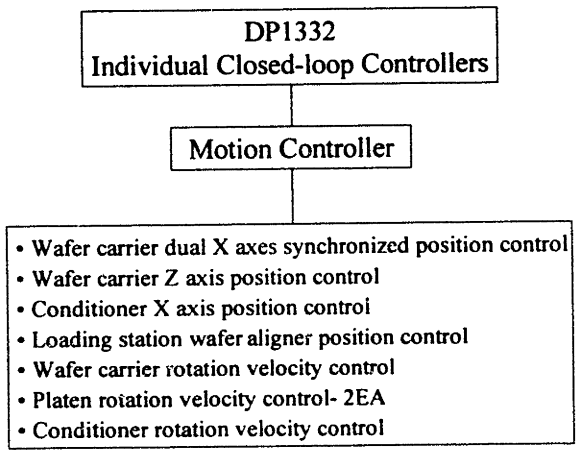
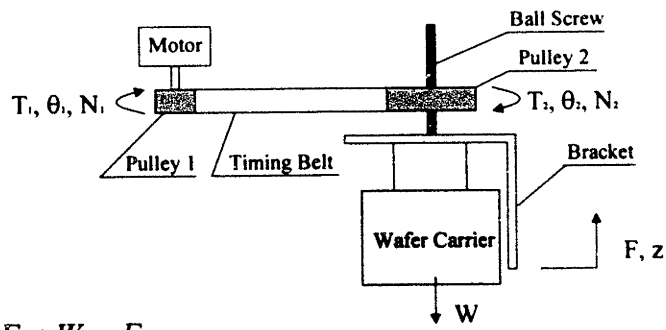


Figure 2.13 Types of the Closed-loop Controllers



$m_{wc}\ddot{z} + b_{wc}\dot{z} \pm F_f + W = F$

m_{wc} : Wafer Carrier Assembly Mass(Kg)

b_{wc} : Viscous Damping(Kg/sec)

F_f : Coulomb Friction(N)

W : Weight(N)

F : Force applied(N)

Figure 2.14 Schematic of WC Z Axis Mechanism

Figure 2.14 shows the schematic of the WC Z axis mechanism and the modeling of the system on the Z axis. However, the WC Z controller controls the motor and we are more interested in the input torque and the output position relationship on the motor shaft. After a further reduction, the following second order differential equation is obtained along the motor spindle.

$$J_1\ddot{\theta}_1 + b_1\dot{\theta}_1 \pm T_{f1} + T_w = T = Ku_d$$

T: torque generated by the armature current, [Nm]
K: torque constant, [Nm]
u_d: DAC variable, 16 bit integer
θ_1: motor shaft angle, [rad]
J_1: Total inertia viewed from the motor shaft, [Kg·m^2]
b_1: Total viscous damping viewed from the motor shaft, [Nm·sec/rad]
T_{f1}: Total friction torque viewed from the motor shaft, [Nm]
T_w: Torque induced by the weight of the wafer carrier assembly, [Nm]

The torque constant K is usually known to the designer. The total inertia can be calculated from the mass of the plant and the known constants(pulley ratio, ball screw pitch, etc.). The torque due to the dead weight can also be computed. However, the friction torque and the viscous damping are usually unknown during the modeling and they are need to be identified. The friction torque can easily be measured by supplying small amounts of torque to the system, increasing the supply until it starts to move. The torque applied at the verge of the motion is roughly equivalent to the friction torque. The viscous damping can be identified by supplying a certain amount of torque to the system and let the spindle spin at a constant speed. At the constant speed, the acceleration is zero and the supplied torque minus the friction and the weight torque is used to spin the system at a constant speed. Dividing the net torque by the angular speed gives b_1.

After the parameter identification, the whole system is usually modeled in the complex domain and represented in a block diagram, and the controller is designed by employing various control engineering techniques. Figure 2.15 shows the block diagram of the WC Z position control system as an example. The input z_{cmd} (mm) first needs to be converted to the motor command angle θ_{1cmd} dividing by ξ and N_1/N_2. ξ is the pitch(mm/rad) of the ball screw, and N_1/N_2 is the ratio of the number of gear teeth of each pulley. The error signal, $\theta_{1cmd} - \theta_1$, is fed to the controller $G_c(s)$ and the controller generates the control signal u_d. Although many different types of controllers can be designed, the proportional-integral-derivative(PID) controller and its variations are the most widely used in the industry. In the complex domain, the PID controller can be represented by the following transfer function.

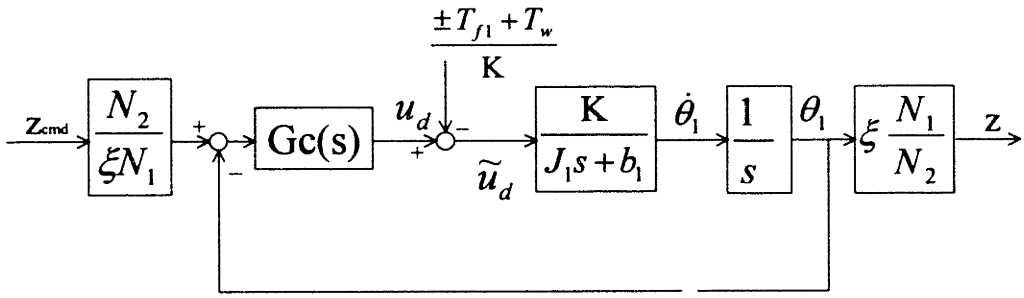
$$G_c(s) = k_p + k_i\frac{1}{s} + k_d s$$

k_p: proportional gain
k_i: integral gain

k_d: derivative gain

The controllers for the Machine-level control system will also be based on the PID controller. Because of the friction and the weight, only $\tilde{u}_d$ from u_d will be available for the system dynamics. The controller output minus the parasitic dynamics will eventually drive the plant and generate the output velocity and the output position, θ_1. By multiplying ξ and N_1/N_2, we extract the output z position of the wafer carrier.

The physical system is usually more complex than the simple modeling, in reality. The system behaves rather differently than designed and simulated. Thus it is quite often necessary to test the controller to tune the individual gains of the controller, and sometimes to tune the controller itself. After tuning, the controller is implemented in the overall control software and can be further optimized in conjunction with the overall control system.



$$J_1\ddot{\theta}_1 + b_1\dot{\theta}_1 \pm T_{f1} + T_w = \mathrm{K}u_d$$

$$\tilde{u}_d = u_d - (\pm T_{f1} + T_w)/\mathrm{K}$$

Figure 2.15 Block Diagram of WC Z Position Controller

The structure of a closed-loop controller is basically the same as the open-loop controller: input, computation, and output. But because of the nature of the servo control, two more procedures need to be built in the controller- homing and input shaping. Table 2.14 summarizes the decomposition of the closed-loop controller structure.

Homing is usually necessary in the position control system to find a reference position of the plant, unless it has an absolute encoder throughout its range of the motion. The rotary encoder of the motor can not provide the absolute position of the motor shaft, instead it provides incremental position information, unless its motion is confined to less than a single revolution. Homing typically involves the on-off signal from a homing sensor at a known reference position. During homing, the plant is driven toward the homing

sensor, until the homing signal is triggered. Once the signal is triggered the procedure calculates the position offset from the amount traveled and the known reference position.

Table 2.14 Decomposition of Closed-loop Controller Structure

Index: 1331.#	Control System Design		
	Functional Requirements (FRs)		Design Parameters (DPs)
	Name	*Description*	*Description*
		Provide a general structure for closed-loop controllers	***Closed-loop controller structure***
1	Control	Receive inputs	Read procedure
2	Control	Handle homing commands	Homing procedure
3	Control	Shape the command input	Input shaping procedure
4	Control	Perform closed-loop computations	Computation procedure
5	Control	Send outputs	Write procedure

$$\begin{Bmatrix} FR13311 \\ FR13312 \\ FR13313 \\ FR13314 \\ FR13315 \end{Bmatrix} = \begin{bmatrix} X & O & O & O & O \\ X & X & O & O & O \\ X & X & X & O & O \\ X & X & X & X & O \\ X & X & X & X & X \end{bmatrix} \begin{Bmatrix} DP13311 \\ DP13312 \\ DP13313 \\ DP13314 \\ DP13315 \end{Bmatrix}$$

The homing procedure is placed after the read procedure, so that it can override the normal input commands for the homing purpose. The homing is signaled by a Boolean variable, for example blnHome, which is normally False. blnHome is set to True by a higher level control program to home the corresponding plant, usually during the machine initialization. Once the homing is completed, the homing procedure sets blnHome back to False, so that the normal control action can be performed. A homing usually involves three steps- Start, Peek and Finish.

Step1: Start

In this step, the position command, the reference velocity and the controller output limit are set for the homing purpose. The position command is usually a large enough value towards the reference point to make the plant to travel all the way to the homing sensor. The reference velocity is usually set at a low value. The controller output is also limited to prevent any possible damage to the plant and the servo amplifier.

Step2: Peek

Once the homing motion is in steady state, the procedure looks for(peeks) the homing sensor signal.

Step3: Finish

Once the procedure detects the homing sensor signal(on), it calculates the position offset, θ_{offset}.

$$\theta_{offset} = \theta_{home} - \theta_{cnt}$$

θ_{offset}: position offset, [rad]
θ_{home}: home position, known, [rad]
θ_{cnt}: current encoder counter position, [rad]

From now on, adding the position offset to the counter position gives the absolute position.

$$\theta = \theta_{cnt} + \theta_{offset}$$

θ: absolute position, [rad]

The procedure resets the position command, the reference velocity and the controller output limit and sets blnHome to False, before it exits the loop

Input shaping is usually necessary to smooth out the input profile and the resulting output of the controller, otherwise the sudden change of a command input(a step command) will result in large controller outputs at the beginning, which result in large overshoots and sustained oscillations during the settling phase. The additional befit is that by shaping the input you can control the gradient of the input(the reference velocity in the case of a position control system or the reference acceleration in the case of a velocity control system). There can be many types of input profiles. But the ramp and the sinusoid are the most popular. Figure 2.16 illustrates two types of the input shaping- the ramp and the hybrid of ramp and sinusoid.

In the ramp input shaping, the slope(reference velocity) is first specified. At the moment of the input command change($t = T0$) from $x0$ to $x1$, the reset time($Treset$) is calculated. While $T0 \leq t \leq Treset$, the input shaped command x_cmd is computed by the following simple formula.

$$x\_cmd = slope \times (t - T0) + x0$$

After t passes $Treset$, x_cmd is set to $x1$, the desired command input. However there still is a sharp transition at $t = Treset$, which is not desirable in terms of minimizing the overshoot. The hybrid sinusoid is introduced to address this problem.

The hybrid ramp and sinusoid input shaping combines the merits of each input shaper: the velocity referencing in the ramp and the smooth transition in the sinusoid. At $t = T0$, it calculates the reset time, $Treset$, but in this case, the reset occurs at the half of the command step, $(x1 - x0) \div 2$, to make a transition to the sinusoidal profile. After t passes $Treset$, the input shaped command is given by the following formula.

$$x\_cmd = (x1 - x0) \div 2 \times (\sin((t - Treset)/T) + 1)$$

where $T = (x1 - x0) \div 2 \div slope$

Once $t > T \times (\pi/2 + 1) + T0$, x_cmd becomes $x1$.

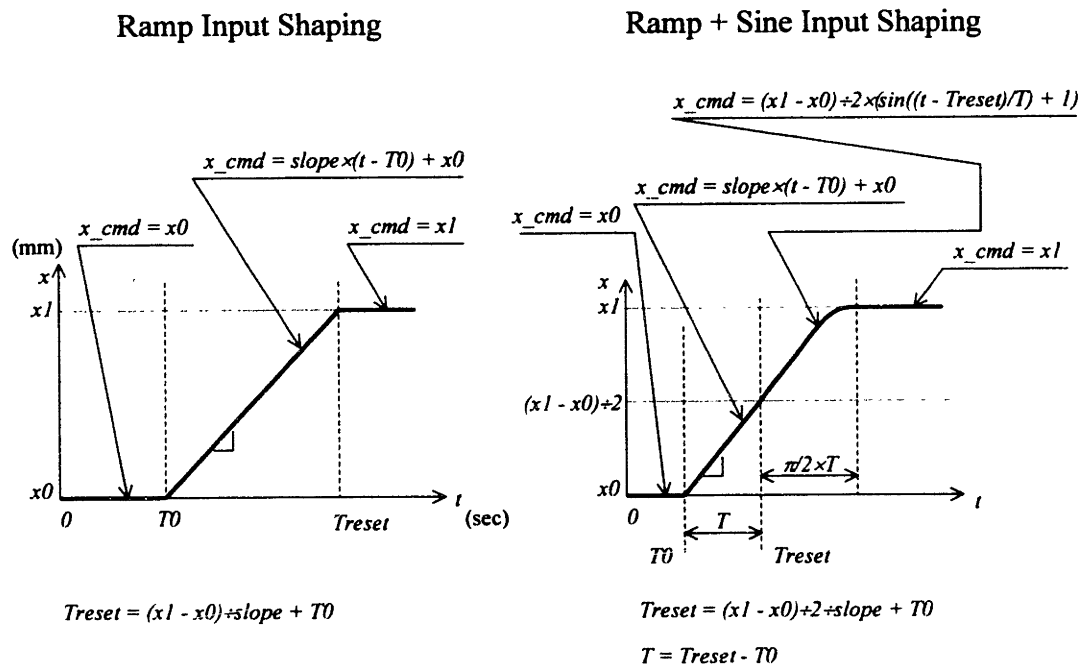


Figure 2.16 Examples of Input Shaping- Ramp and Ramp plus Sinusoid

The input shaping procedure will have various input shaping profiles available to a higher-level control system. The higher-level will select the profile and send the command to the input shaper. The procedure will generated the input shaped commands based on the selected profile, the slope and the time. The input shaped command is fed to the computation procedure of the closed-loop controller to produce the controller output. Finally, it is written to a DAC variable to be transmitted to the servo amplifier via the output unit.

DP14. Machine-level Overhead

The three units discussed so far, the I/O, the signal processing and the controller unit, are the essence of the Machine-level control system. They acquire and process the input signal, and generate and send the output signal, which are the fundamental functionalities to control a machine. However due to the large number of machine constituents to be controlled and the complexity involved in controlling them, it is necessary to organize those units and their sub units in a systematic way, to ensure the effectiveness and enhance the efficiency of the Machine-level control system. The Machine-level overhead deals with the issue of how to build the Machine-level control system from the existing units and other

available software procedures, and with the practical considerations and the design decisions to run the control software in real time to drive the 'real machine.'

The Machine-level overhead constructs the system software with the units, the sub units and other individual software procedures as its constituents. However, the Machine-level control system runs on the memories of the ADwin CPUs, commanding hardware interface cards to drive the physical machine. Naturally, there are physical limitations in running the control system. Among them, time and space are the most fundamental and important. Thus we have to make the time-wise and the space-wise decisions at the early stage of the overhead structure design. We decompose the Machine-level overhead as shown in Table 2.15.

Table 2.15 Decomposition of Machine-level Overhead

Index: 14.#	Control System Design		
	Functional Requirements (FRs)		Design Parameters (DPs)
	Name	*Description*	*Description*
		Structure the Machine-level control system	***Machine-level overhead***
1	Control	Organize the Machine-level control actions into a system.	Machine-level control system software
2	Control	Designate how often the Machine-level control actions occur.	Event frequencies
3	Control	Distribute the computational loads between and within CPUs.	Load division

$$\begin{Bmatrix} FR141 \\ FR142 \\ FR143 \end{Bmatrix} = \begin{bmatrix} X & O & O \\ X & X & O \\ X & X & X \end{bmatrix} \begin{Bmatrix} DP141 \\ DP142 \\ DP143 \end{Bmatrix}$$

The event frequency is often called the sampling frequency, because the required signals are sampling at every periodic control event. For a closed loop servo control, not less than 1 KHz of event frequency is required. Theoretically, the higher is the sampling rate, the better is the closed loop controller performance, because a sampled discrete system can better emulate a continuous system with a higher sampling frequency. However, a higher frequency demands more resources from the CPU and the other hardware devices. The event block of the Machine-level control program is repeatedly executed at the beginning of each event interval, for a finite time span, called the duty interval. Thus a duty interval can not exceed an event interval, otherwise a fatal error will occur (either the rest of the program will not be executed or the CPU will simply crash). The ratio of the duty interval to the event interval is called the CPU duty ratio. The duty ratio should always be less than 1. For a given duty interval (which is typically the case), the duty ratio can change by changing the event interval. Figure 2.17 illustrates this point. The event frequency needs to be selected to keep the duty ratio less than 1, but as close to 1 as possible to make a maximum use of the available hardware resources.

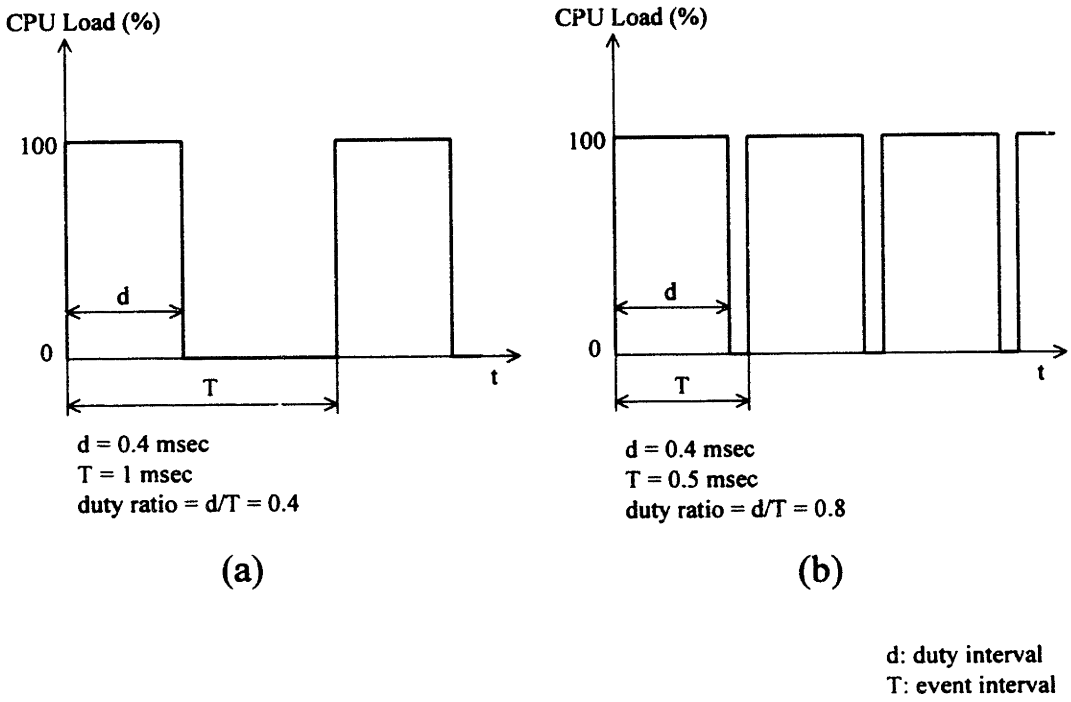


Figure 2.17 CPU Duty Ratios as Function of Duty Interval and Event Interval

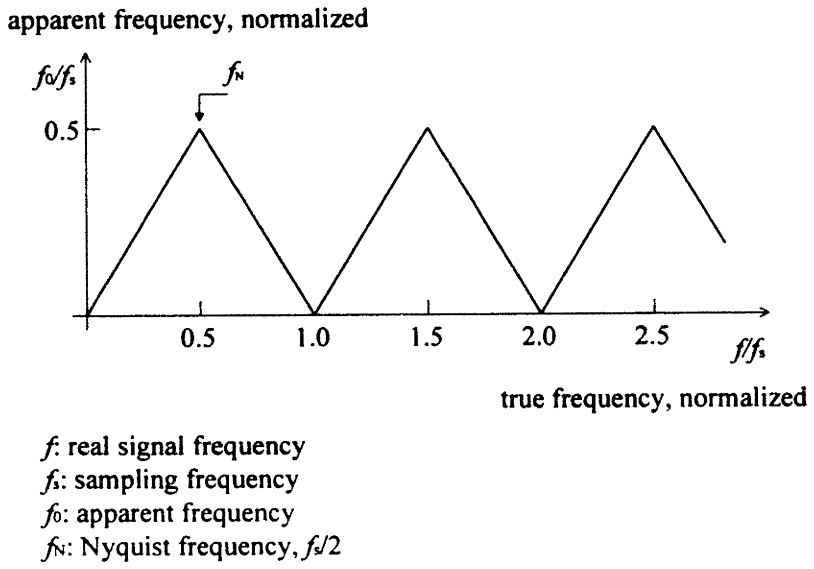


Figure 2.18 Apparent frequency f_0 as function of true frequency f and sampling frequency f_s

Another concern in the selection of a sampling frequency is the prevention of the aliasing. If the sampling frequency is too small compared to the signal frequency, a false frequency (an alias frequency) appears. Theoretically, for a given sampling frequency f_s, any signal which has frequency components higher than $f_N = f_s/2$ can not be reconstructed from the sampled values. The frequency f_N is called the Nyquist frequency. Higher frequency components will appear as waves of lower frequency. Figure 2.18 shows the apparent frequency f_0 as a function of the true frequency and the sampling frequency.

Thus, it is required to sample at least twice faster than the highest predictable frequency component in the signal. However, it is still recommended to sample five to ten times faster than the signal frequency, so that the sampled (discrete) waveform can well approximate the continuous (analog) input signal. In a highly noisy environment, noise signals with bandwidths of a few KHz to hundreds of MHz can be picked up by the analog signal. Maximum sampling rate is usually limited to a few hundred KHz by the hardware capacity. AD converters and multiplexers typically have the settling time of a few microseconds. It is simply impossible to achieve a MHz of sampling rate. Instead, it is a common practice to place a low pass filter with the bandwidth of 1/5 to 1/10 of the sampling frequency before the AD conversion to eliminate any potential high frequency noise signal.

1 KHz is a good starting point for both the closed-loop control and the signal processing. Event frequencies will be increased if the hardware permits. The goals are 3 KHz for the closed-loop control and 5 KHz for the signal processing.

The division of the computational load is required to make a maximum use of two CPUs of the ADwin system. Two limiting factors need to be considered in the load division.

- Program size: There is a limit in the program size, which is loaded on the CPU's memory. ADSP21062, the processor of the ADwin system, can load up to 75 Kbytes of program size. Also the duty cycle is roughly proportional to the program size with the given sampling frequency.
- Communication between the CPUs: The CPUs in the ADwin system can not share the memory nor the bus. The communication between the CPUs is possible via the host PC. The host PC can not guarantee a real time data transmission. Thus, any time critical event procedure should be performed within a single CPU.

It is recommended for a single processor to handle the machine control and the required signal processing for its own. Analog signals (strain gauge, reflectance sensing) are usually not required for the Machine-level control, but for the process control in a higher level. We dedicate the CPU 1 for the machine control and the CPU2 for the analog signal processing with its own ADC card.

ADwin further allows the division of the load in a single processor. Instead of running a single program, several programs can be written and run with their own unique sampling frequency. This capability is quite useful when it is required to sample signals at different frequencies, depending on the input channels. In the given application, the reflectance signal can be sampled at 5 KHz, while other slow varying analog signals are sampled at 1 KHz.

The Machine-level control software can be grouped into three parts- the software initialization, the event, and the software termination block. The software initialization configures the hardware cards, defines the numerical constants, etc. so that the event block

can perform the routine jobs. The software initialization is different from the machine initialization, which requires a series of control commands and hence can not be completed in a single software initialization event. The machine initialization will be handled by the event block. The event block is repeatedly executed with the designated event frequency for the machine control and the signal processing. The termination block reconfigures the hardware, assign a nominal signal for each output channel, etc. before the finish of the Machine-level control. Table 2.16 shows the decomposition of the software structure.

Table 2.16 Decomposition of Software Structure

Index: 141.#	Control System Design		
	Functional Requirements (FRs)		Design Parameters (DPs)
	Name	*Description*	*Description*
		Organize the Machine-level control actions into a system	***Machine-level control system software***
1	Control	Perform necessary initializations.	Software initialization
2	Control	Designate event sequences.	Event sequence
3	Control	Set the controller at off states before termination	Software termination

$$\begin{Bmatrix} FR1411 \\ FR1412 \\ FR1413 \end{Bmatrix} = \begin{bmatrix} X & O & O \\ X & X & O \\ O & O & X \end{bmatrix} \begin{Bmatrix} DP1411 \\ DP1412 \\ DP1413 \end{Bmatrix}$$

Table 2.17 shows the further decomposition of the software initialization block. The bus in the back plane of the ADwin needs to be separated as a first procedure in the software initialization, if multiple CPUs are to be used. Even though it is possible to run two processors at the same time without the bus separation, it is strongly recommended to split the bus to prevent for the two processors to access the same channel of an interface card, which will result in the crash of the system. Bus separation is required for the stability and reliability of the control system.

Some interface cards of the ADwin requires software configurations before use. The counter card requires the selection of either a dual channel quadruple evaluation or a single channel clock mode. Then the card is enabled and its counter is cleared to zero. Also, the selected channels of the DIO card need to be configured as outputs, because by default all DIO channels are booted as inputs.

A large amount of constants need to be defined during the initialization. It may be any of a purely numeric constant, such as π, a geometric constant, such as the distance from the loading/unloading station center to the platen center, a hardware constant, such as the number of encoder pulses per a platen revolution, or a control system constant, such as the conditioner X controller proportional gain.

Sometimes it is necessary to read sensor values during the initialization for various purposes. One example is the gantry X linear encoder reading during the software initialization. Because the linear encoder provides the absolute X position of the gantry, the

homing of the gantry can be skipped in the event block, once the X position offset variable is set after the linear encoder reading.

Software variables are initialized after the acquisition of the necessary sensor values. Position and velocity commands, error integrals, and the buffers for temporary storage are the examples of the variables need to be initialized. The appropriate indicators of the system, mostly the LEDs of the ADwin, are turned on at the last procedure, marking the end of the software initialization block.

Table 2.17 Decomposition of Software Initialization

Index: 1411.#		Control System Design	
	Functional Requirements (FRs)		Design Parameters (DPs)
	Name	*Description*	*Description*
		Perform necessary initializations.	***Software initialization***
1	Control	Split the bus of the controller for each CPU, if necessary	Bus separation procedure
2	Control	Configure each counter channel for its respective input.	CNT configuration procedure
3	Control	Configure necessary DIO channels (for output).	DIO configuration procedure
4	Control	Assign numerical values to the declared constants.	Constant assignment procedure
5	Control	Pick up initial sensor values, if necessary	Initial sensor reading procedure
6	Control	Initialize variables.	Variable initialization procedure
7	Control	Turn indicators on	Indicators-on procedure

$$\begin{Bmatrix} FR14111 \\ FR14112 \\ FR14113 \\ FR14114 \\ FR14115 \\ FR14116 \\ FR14117 \end{Bmatrix} = \begin{bmatrix} X\,O\,O\,O\,O\,O\,O \\ X\,X\,O\,O\,O\,O\,O \\ X\,O\,X\,O\,O\,O\,O \\ O\,O\,O\,X\,O\,O\,O \\ X\,X\,X\,O\,X\,O\,O \\ X\,X\,X\,X\,X\,X\,O \\ X\,O\,O\,O\,O\,O\,X \end{bmatrix} \begin{Bmatrix} DP14111 \\ DP14112 \\ DP14113 \\ DP14114 \\ DP14115 \\ DP14116 \\ DP14117 \end{Bmatrix}$$

The decomposition of the event sequence is a bit more demanding because of the size and the nature of the event block, which is the main body of the Machine-level control system. Table 2.18 proposes a possible decomposition of the event sequence.

At the beginning of every event, which is cyclically called with the event frequency, various machine parameters are updated by the machine input handler. The machine input handler calls the machine input units and assigns their output to the machine parameters. An encoder count, a digital input bit, and a 16 bit ADC value are the examples of machine

parameters. Then the signal processing handler calls the numerous signal processing units to process the machine parameters to the appropriated forms and values.

Table 2.18 Decomposition of Event Sequence

Index: 1412.#	Control System Design		
	Functional Requirements (FRs)		**Design Parameters (DPs)**
	Name	*Description*	*Description*
		Designate event sequences	*Event sequence*
1	Control	Update machine parameters.	Machine input handler
2	Control	Perform signal processing.	Signal processing handler
3	Control	Take appropriate actions if an error is set	Error handler
4	Control	Update inputs from the Process-level	Process-level input handler
5	Control	Adjust the command inputs based on the selected machine modes	Mode handler
6	Control	Perform control processing	Control processing handler
7	Control	Update outputs to the machine	Machine output handler
8	Control	Save data	Data save handler
9	Control	Update outputs to the Process-level	Process-level output handler
10	Control	Update clock	Clock handler

$$\begin{Bmatrix} FR14121 \\ FR14122 \\ FR14123 \\ FR14124 \\ FR14125 \\ FR14126 \\ FR14127 \\ FR14128 \\ FR14129 \\ FR141210 \end{Bmatrix} = \begin{bmatrix} X & O & O & O & O & O & O & O & O & O \\ X & X & O & O & O & O & O & O & O & O \\ X & X & X & O & O & O & O & O & O & O \\ O & O & O & X & O & O & O & O & O & O \\ X & X & X & X & X & O & O & O & O & O \\ X & X & X & X & X & X & O & O & O & O \\ X & X & X & X & X & X & X & O & O & O \\ X & X & X & X & X & X & X & X & O & O \\ X & X & X & X & X & X & X & X & X & O \\ O & O & O & O & O & O & O & O & O & X \end{bmatrix} \begin{Bmatrix} DP14121 \\ DP14122 \\ DP14123 \\ DP14124 \\ DP14125 \\ DP14126 \\ DP14127 \\ DP14128 \\ DP14129 \\ DP141210 \end{Bmatrix}$$

Error handling is quite important in a machine control system, because it is directly related to the safety of the machine. The error handler should be able to detect an error signal in a real time and stop any further motion of the motion. Removal of the power stage of the actuators is further recommended, if possible. However power-off switchings are usually accomplished by hard wiring. On/off type sensing devices are used in conjunction with the relays and/or the logical enable circuits. Figure 2.19 illustrates the external enable circuit of a servo amplifier connected to an optical limit switch, which provides an NPN

transistor output. Normally the transistor is off, and the sense-out carries +5 V signal. Because there is no current on the opto-diode, the opto-coupler of the external enable circuit of the amplifier is off, which gives the +5 V of enable logic signal. Once sensing, the sense out is pulled down to the ground, and as a result a current flows through the opto-diode and the opto-coupler is on. The logic carries 0 V of disable signal, which is used by the servo amplifier to remove the power stage from itself. Thus once the sensor is triggered, the corresponding amplifier is first disabled by the external wiring.

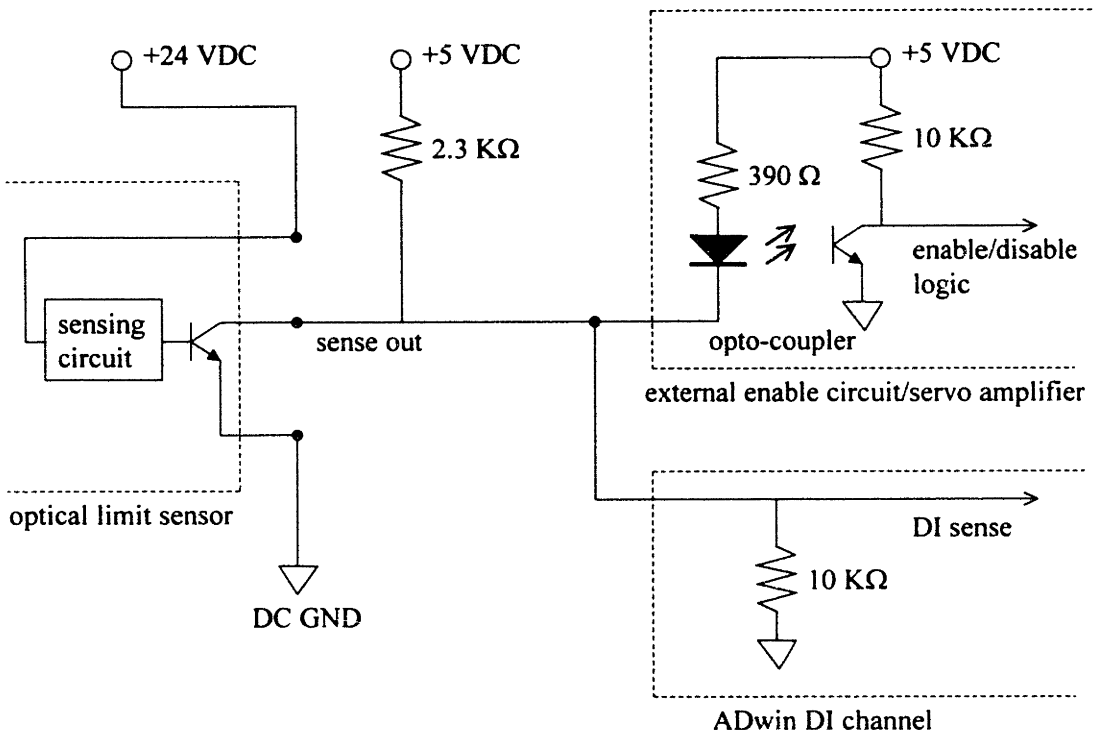


Figure 2.19 External Enable/Disable Circuit of Servo Amplifier

The sense out is also connected to an ADwin DI channel. The error handler of the event block will recognize the trigger of the limit sensor, remove the controller output and set an error for the Process-level control system. The following are the examples of the Machine-level errors.

- Emergency off(EMO) switch engagement.
- Gantry X travel limit.
- Conditioner X travel limit.
- Gantry X/Conditioner X relative clearance.
- Wafer carrier wafer retention sensor.

Inputs from the Process-level will be updated by the process-level input handler, which calls the Process-level input unit in conjunction with the corresponding input

conversion unit. Some of the control commands are required to be updated as a batch. For example, a position command and a reference velocity command for a servo controller need to be updated at the same time, otherwise a fatal error can occur during input shaping. Because it takes a finite amount of time to transmit a packet of control commands from the Process-level to the Machine-level, there is no guarantee that all the commands are updated within a single event. A simple solution is to send an update flag at the end of the packet transmission. The process-level input handler updates the control commands only after it receives the update flag, by that time all the control commands will have been received and saved in the buffer of the Process-level input unit.

In most cases, it is not required to adjust the commands once updated by the Process-level input handler. However, during the machine initialization and the machine termination(parking), which are set by the corresponding modes, the mode handler assumes the role of the Process-level control system and manipulates the control commands in a preprogrammed manner. Four components require homing during the machine initialization: the wafer carrier Z axis, the conditioner X axis, the wafer aligner angular position, and the platen B angular position. At each homing a max or min command is generated by the mode handler, so that the corresponding actuator can travel to the position of the homing. The homing is usually sensed by a digital sensor or the presence of a mechanical stop. Figure 2.20 illustrates the sequence of the initialization mode.

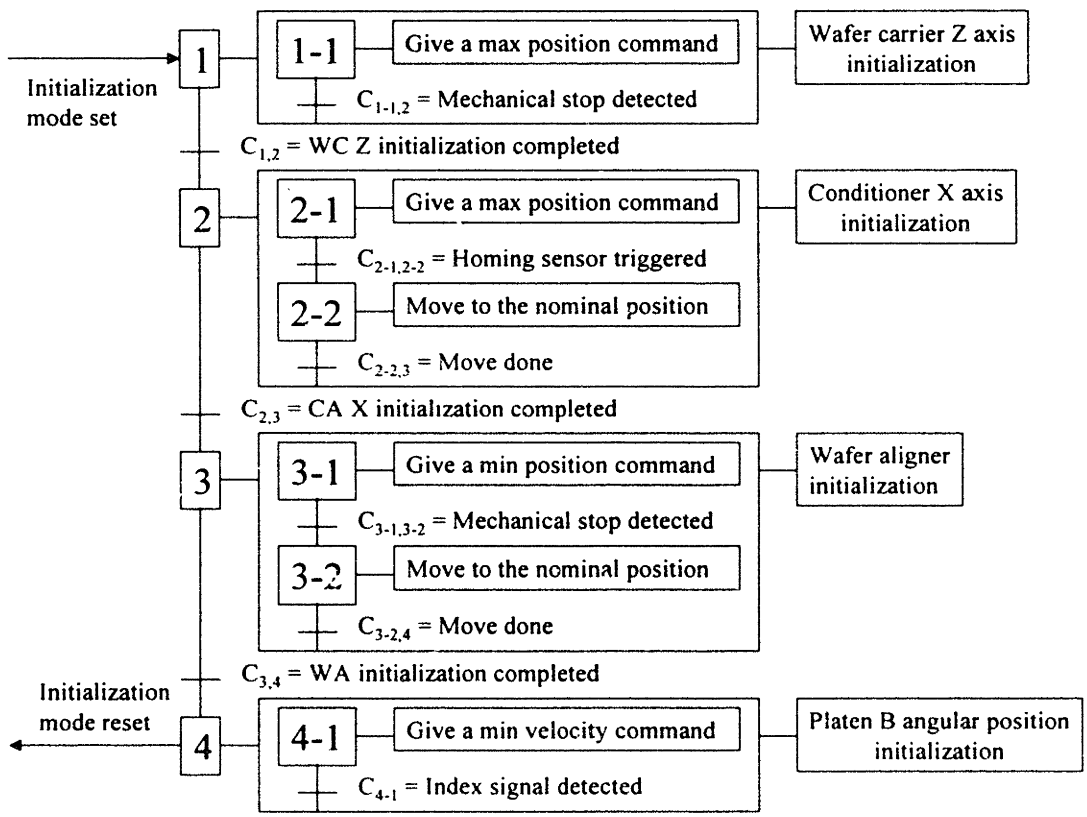


Figure 2.20 Sequential Functional Diagram of Initialization Mode

The control commands generated either by the Process-level input handler or the mode handler are fed to the control processing handler, which calls the various controller

units. The output of the controller units are picked up by the machine output handler, which sends the output to the machine via the machine output unit.

Many data are internally saved for signal processing purposes. Current position and velocity are typically saved for the filtering and the differentiation purposes. The data save handler performs the specified data save function by calling the internal data save unit.

The Machine-level control system should update the machine status at the end of every event to the Process-level. The Process-level output handler posts the machine parameters (position, velocity, controller output, etc.) and the status parameters (error, running indicator, etc.) to the Process-level output unit, which writes to the domain of the DSP memory, accessible by the Process-level.

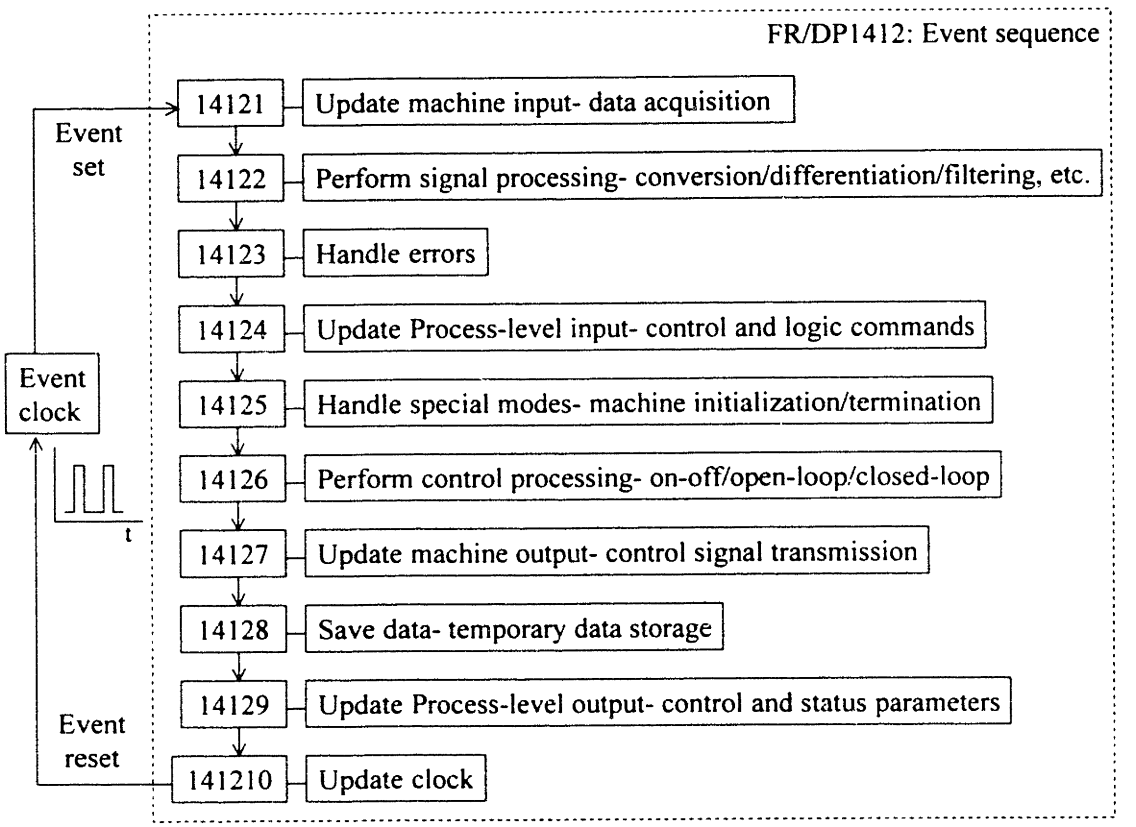


Figure 2.21 Sequence of Event Block

Time is the most important and widely used variable in the event block. Time needs to be updated before the end of the event procedures. The following simple procedure is used for the clock, because the event is guaranteed in the real time control system. ADwin also offers a timer function.

$t = t + T$

t: time (sec)
T: event interval (sec)

Once t is initialized to zero in the software initialization, t is incremented by the event interval at the end of every event. Thus t is the measure of time since the control system is first run.

Figure 2.21 shows the summary of the event block sequence in a block diagram format. Each event is triggered by the internal event clock of the ADwin CPU.

Table 2.19 Decomposition of Software Termination

Index: 1413.#	Control System Design		
	Functional Requirements (FRs)		Design Parameters (DPs)
	Name	*Description*	*Description*
		Set the controller at off states before termination	***Software termination***
1	Control	Reset DIO channels (for input).	DIO reset procedure
2	Control	Assign nominal off-state values to each output channel.	Output reset procedure
3	Control	Reset the variables.	Variable reset procedure
4	Control	Turn indicators off	Indicators-off procedure
5	Control	Cancel the bus split of the controller.	Bus unification procedure

$$\begin{Bmatrix} FR14131 \\ FR14132 \\ FR14133 \\ FR14134 \\ FR14135 \end{Bmatrix} = \begin{bmatrix} X & O & O & O & O \\ X & X & O & O & O \\ O & O & X & O & O \\ O & O & O & X & O \\ X & X & O & X & X \end{bmatrix} \begin{Bmatrix} DP14131 \\ DP14132 \\ DP14133 \\ DP14134 \\ DP14135 \end{Bmatrix}$$

The finish and the subsequent unload of the Machine-level software is triggered by the Process-level control system, such as a 'Quit' or 'Exit' button. The Process-level transmits the finish command and the ADwin executes the termination block before unloading the program. Table 2.19 shows the decomposition of the software termination.

The DIO channels which have been configured as outputs need to be reset to inputs to drive the corresponding actuators to the nominal off states. All the relays are switched to the nominal open contacts, and the nominal outputs (normally 0 V) are assigned to the DAC channels.

Global variables of the ADwin have the permanent addresses in the CPU memory. The global variables will keep (memorize) the values assigned to themselves even after the termination of the program. If another program is loaded without rebooting the ADwin, these values can be picked up the program, which may result in errors. Thus it is recommended to reset all the global variables to zero.

The ADwin indicators are turned off and the split bus is reunified before the end of the software termination. Then the ADwin unloads the program. The machine is in a safe 'off' state.

The Machine-level control system has been designed in this chapter. As a summary, the design parameter tree of the Machine-level control system is attached in Figure 2.22, which shows the major constituents of the system.

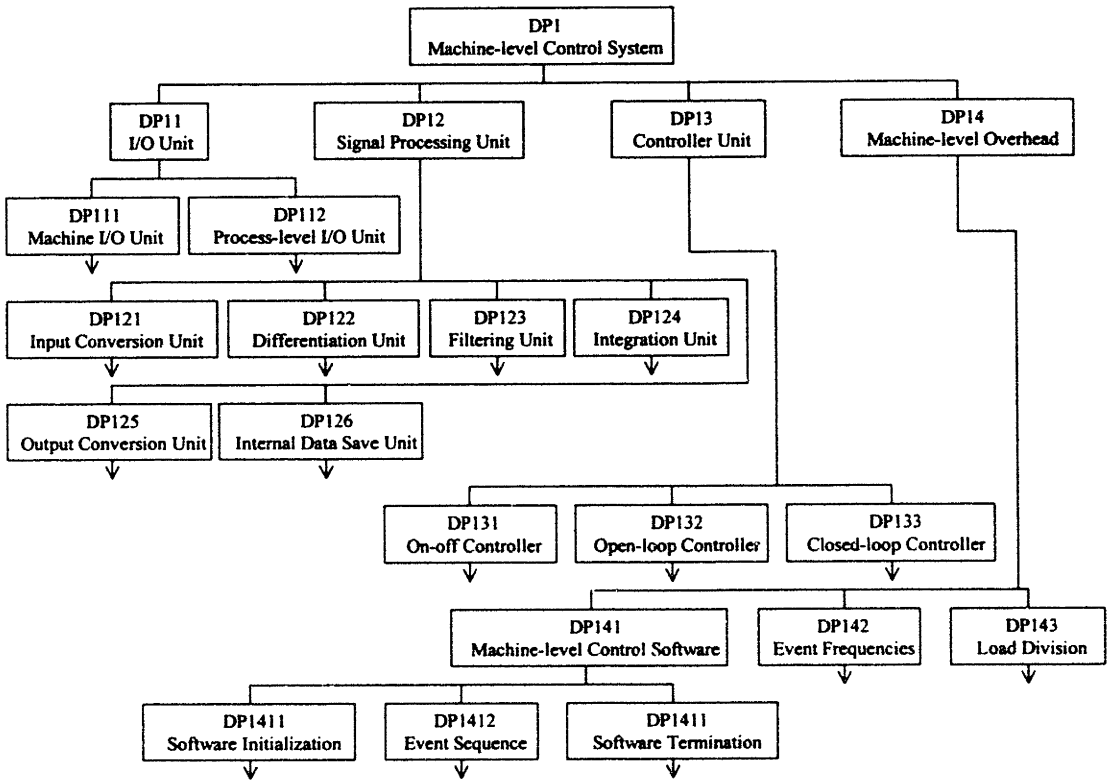


Figure 2.22 DP Tree of Machine-level Control System

Chapter 3. Process-level Control System

The processing of a wafer requires a series of the coordinated machine control actions. For example, to polish a wafer, the wafer carrier first picks up the wafer, move to the polishing position, the wafer carrier and the platen start to rotate, the slurry pump is turned on, pneumatic pressures are applied to the wafer carrier membranes, etc. The individual control actions are provided by the Machine-level control system, which is thoroughly designed in the previous chapter. Having the machine control actions at its disposal, the Process-level control system dictates the Machine-level in a concerted manner to achieve the desired wafer processing with the user interaction. The Process-level control system is placed between the Machine-level and the user (and also the higher level control systems), and acts both as a messenger and a coordinator (and possibly as a manager). Running the machine for wafer processing and supporting the processing are the basic functionalities of the Process-level control system. Table 3.1 shows the decomposition.

Table 3.1 Decomposition of Process-level Control System

Index: 2.#	Control System Design		
	Functional Requirements (FRs)		Design Parameters (DPs)
	Name	*Description*	*Description*
		Organize the controlled outputs for wafer processing.	***Process-level control system***
1	Process	Provide supportive features for wafer processing.	Supportive unit
2	Process	Process wafers.	Process unit

$$\begin{Bmatrix} FR21 \\ FR22 \end{Bmatrix} = \begin{bmatrix} X & O \\ X & X \end{bmatrix} \begin{Bmatrix} DP21 \\ DP22 \end{Bmatrix}$$

The process unit is used to process wafers to the desired specifications. The unit may include manual and/or auto mode wafer processing. The supportive unit contains various utilities used by the process unit and useful for the equipment run and maintenance. Let's first decompose the supportive unit

DP21. Supportive Unit

The supportive unit can be decomposed into the recipe builder, the equipment database, and the Set Up mode as shown in Table 3.2.

Although not essential for an α stage or research grade machine, the equipment database is quite important and necessary beyond the β stage toward the production grade machine. The data logged in the database can be grouped in two categories.

Operation database: handles the data related with the equipment use. At each run (single wafer or batch), the database may record time, user, recipe, lot number, type and number of

wafers processed, type and amount of used consumables, etc. The operation database can be used to keep track of the equipment use history and the wafers which have been processed by the equipment. The database can provide a useful data to improve the efficiency of the equipment operation and to optimize the process performed by the equipment.

Maintenance database: deals with the data concerning the equipment maintenance. They may fall in two categories- machine and consumables. The machine maintenance database will handle the data of the machine itself- lubrication, tubing, fitting, fastening, etc. The consumable database will log the data related to pad, slurry and other utilities. Each database saves the time and the type of the service performed on the equipment and let the user know when is the next scheduled service.

The database can belong either to the Process-level control system or to the Operation-level control system, depending on the configuration. However, the database development is not critical at this stage. We will come back to the database design once we accomplish the process unit design and implementation.

Table 3.2 Decomposition of Supportive Unit

Index: 21.#	Control System Design		
	Functional Requirements (FRs)		Design Parameters (DPs)
	Name	*Description*	*Description*
		Provide supportive features for wafer processing.	***Supportive unit***
1	Process	Provide recipes for wafer processing	Recipe builder
2	Process	Provide a database for the machine use and maintenance	Equipment database
3	Process	Set up and calibrate the machine for wafer processing	Set up mode

$$\begin{Bmatrix} FR211 \\ FR212 \\ FR213 \end{Bmatrix} = \begin{bmatrix} X & O & O \\ O & X & O \\ O & O & X \end{bmatrix} \begin{Bmatrix} DP211 \\ DP212 \\ DP213 \end{Bmatrix}$$

It is often necessary to readjust the set points and the calibration factors for the sensors and actuators. Also the maintenance and the repair require a direct access to run each component individually. The set up mode is required for these non-process machine run purposes. The set up mode also contains the machine initialization and the machine termination procedures to drive the machine to the on and the off state. These procedures can be called by the process units to initialize and terminate the machine before and after the wafer processing.

DP. 211 Recipe Builder

The recipe builder is required to process a wafer in an auto mode. The recipe builder defines the recipe and its file structure for storage, and provides a recipe editor for recipe handling. The recipe editor should be able to create, edit, and save a recipe, and retrieve the existing one. The auto mode will have an adequate link to load the recipe data to its process parameter buffer. The design of a flexible and easy-to-use recipe editor is quite essential to ensure the effectiveness of the automatic wafer processing and to enhance the processing efficiency. Table 3.3 shows the decomposition of the recipe builder.

Table 3.3 Decomposition of Recipe Builder

Index: 211.#		Control System Design	
	Functional Requirements (FRs)		Design Parameters (DPs)
	Name	*Description*	*Description*
		Provide recipes for wafer processing	***Recipe builder***
1	Process	Define the recipe structure	Recipe classes
2	Process	Define the recipe file structure	Recipe file
3	Process	Manipulate recipes	Recipe editor

$$\begin{Bmatrix} FR2111 \\ FR2112 \\ FR2113 \end{Bmatrix} = \begin{bmatrix} X & O & O \\ X & X & O \\ X & X & X \end{bmatrix} \begin{Bmatrix} DP2111 \\ DP2112 \\ DP2113 \end{Bmatrix}$$

The design of the recipe builder starts with the design of the recipe classes (templates). A recipe class defines the required data (process parameters) and tags (identification parameters) and its associated interaction methods to communicate with the environment. Based on the recipe classes, defined at the design time, the recipe editor program will create recipe objects at each run time.

The recipe object acts as an active data carrier between the editor window (front end, user side) and the recipe file (back end, hardware side) stored in a computer hard disk. It extracts the data from a recipe file and house them in a predefined manner, then it provides the data to the editor. The editor displays the data according to the user interface layout. The edited data is then fed to the recipe object and the object updates its data content. Finally the data is stored in a recipe file. Figure 3.1 illustrates the recipe object concept and related data flow.

The presence of the recipe objects reduces the complexity involved in the direct file I/O from the editor window, and enhances the readability, the flexibility, and the maintenanceablity of the editor program. The recipe objects make it easy to desigt. the recipe file structure and the recipe editor.

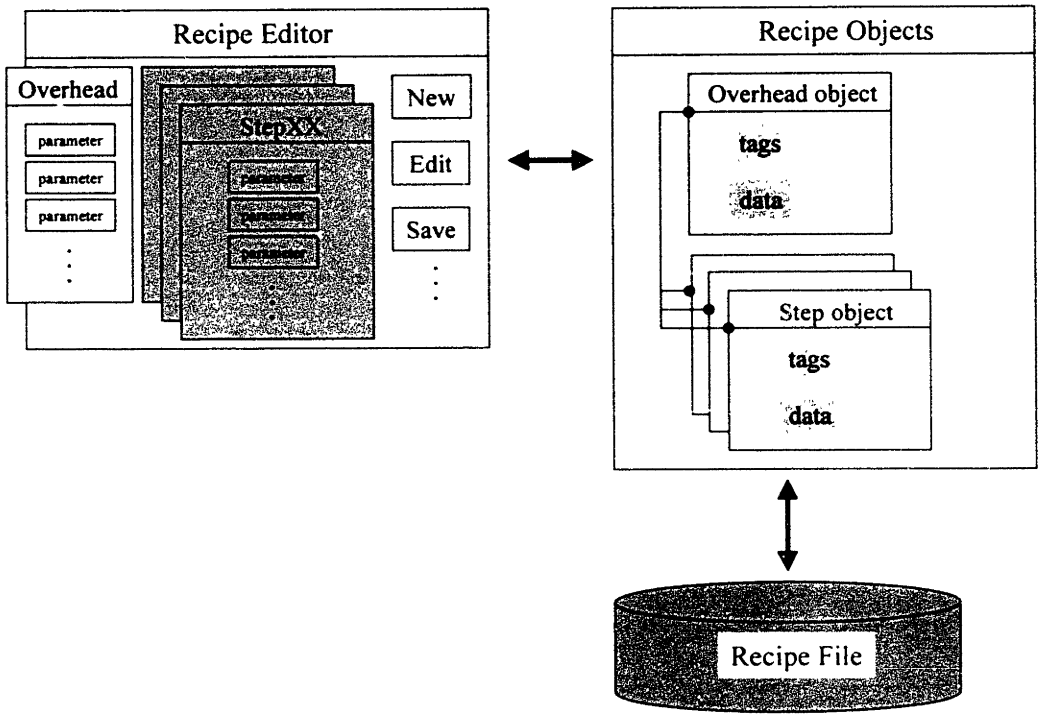


Figure 3.1 Recipe Objects and Data Flow

The recipe for an automated wafer processing is composed of many distinctive recipe steps. The step is a series of control actions, which can be uniquely identified and grouped in the (continuous) process flow. As an example, we can process a wafer using the following assembly of recipe steps:

Load → Polish → Clean → Unload → Condition

Table 3.4 shows the decomposition of the recipe classes into the individual ones. Step_LoadReady is used as a waiting step between each wafer count, and is the first step in any wafer processing recipe. Step_Condition and Step_CondClean are non wafer processing steps but required to insert the conditioner related activities into a recipe. A condition recipe composed solely of the conditioner steps can be created, but can be executed as a single run only in the run time. In contrast, wafer processing recipes can be executed as a multi run. A single recipe can process a stack of wafers repeatedly one by one.

These individual steps can be placed arbitrarily in a user-created sequence during editing to complete a recipe. However, certain rules must be observed. For example, you can not polish a wafer after you unload it. Also some steps can have pluralities in a single recipe. A user may want to polish a wafer first at the platen A and move to the platen B for the second polishing step. Polishing, cleaning, conditioning, and conditioner cleaning steps have plurality. A recipe should be able to indicate how many steps it has, the order of the steps,

and the plurality of each step. Thus it is quite necessary to have a type of carrier to hold this recipe-wide, or overhead information.

Table 3.4 Decomposition of Recipe Classes

Index: 2111.#	Control System Design			
	Functional Requirements (FRs)		Design Parameters (DPs)	
	Name	*Description*	*Description*	
		Define the recipe structure	Recipe Classes	
1	Process	Provide a structure for the recipe overhead	Class Recipe_Overhead	
2	Process	Provide a structure for the load ready step	Class Step_LoadReady	
3	Process	Provide a structure for the condition step	Class Step_Condition	
4	Process	Provide a structure for the load step	Class Step_Load	
5	Process	Provide a structure for the polish step	Class Step_Polish	
6	Process	Provide a structure for the clean step	Class Step_Clean	
7	Process	Provide a structure for the unload step	Class Step_Unload	
8	Process	Provide a structure for the conditioner clean step	Class Step_CondClean	

$$\begin{Bmatrix} FR21111 \\ FR21112 \\ FR21113 \\ FR21114 \\ FR21115 \\ FR21116 \\ FR21117 \\ FR21118 \end{Bmatrix} = \begin{bmatrix} X & O & O & O & O & O & O & O \\ X & X & O & O & O & O & O & O \\ X & O & X & O & O & O & O & O \\ X & O & X & X & O & O & O & O \\ X & O & X & O & X & O & O & O \\ X & O & X & O & O & X & O & O \\ X & O & X & O & O & O & X & O \\ X & O & O & O & O & O & O & X \end{bmatrix} \begin{Bmatrix} DP21111 \\ DP21112 \\ DP21113 \\ DP21114 \\ DP21115 \\ DP21116 \\ DP21117 \\ DP21118 \end{Bmatrix}$$

Recipe_Overhead class is designed to process the overhead information. As in any class design, the design of the Recipe_Overhead class should describe what to contain (attributes) and how to behave (methods).

Axiomatic Design rigorously specifies the design of an object. Object behaviors are stated as FRs, appropriate attributes are chosen as DPs to meet the behavior demands, and design matrix elements are selected to construct methods to realize the intended behaviors. For more detail about the object oriented programming based on Axiomatic Design approach, refer to Chapter 5 of Suh[4].

Attributes are the information encapsulated in a class. The attributes can be grouped in two types.

◆ Tags: Members designed for identification purposes. For example, the recipe name and the edit time are tags for the Recipe_Overhead.

♦ Data: Members designed to hold process parameters. The number of steps included, the sequence of step IDs are the example of data for the class Recipe_Overhead

Suppose a recipe is composed of the following sequence of process steps (the number in a parenthesis indicates the corresponding step ID).

LoadReady(0) → Load(2) → Polish(3) → Clean(4) → Unload(5) → Condition(1)

Then the number of steps field contains the value of 6 (Recipe_Overhead.StepN = 6). The step ID sequence field holds the string of 0-2-3-4-5-1 (Recipe_Overhead.StepSqc = 023451).

The methods dictate the behavior of a class. The methods of the Recipe_Overhead class should provide means of interaction with the recipe file and the recipe editor. Following methods are required:

♦ WriteToFile: writes the attributes to a recipe file in a predefined manner and sequence.
♦ ReadFromFile: reads the data from a recipe file and assigns them to the corresponding attributes.
♦ WriteToBuffer: writes the (data) attributes to the I/O buffer of the recipe editor.
♦ ReadFromBuffer: reads the data from the editor I/O buffer and writes them to the attributes.

More methods may be added with the advance of the recipe builder/the Process-level control system development.

The attribute and method design of the step classes will follow the same pattern. Each step class will have similar members and behaviors compare to each other, differing only in the specific data types and contents. Generally a step class will have the following attributes:

♦ Public: step name, step ID, edit time, edit flag (default or edited), plurality, value string (to hold either a Boolean, integer, or float variable), value index (for value sting I/O), number of Boolean data fields, number of integer data fields, number of float data fields, starting index of the Boolean data, starting index of integer data, starting index of float data, etc.
♦ Private: Array of value strings (to hold Boolean, integer, float data arrays), array index, etc.

Public attributes are data members accessible by the environment; private attributes are not accessible. Private members are for the internal use only. Ultimately a step has three types of process parameters: Boolean (on/off of an actuator, etc.), integer (number of sweeps, etc.), and float (position and velocity of an actuator, etc.). A step will have three arrays to hold these parameters according to their variable type. However, every variable is written as a text in the recipe file to be stored in a computer hard disk. The step class will have a single string (text) array to hold every type of parameters. The parameters will be converted to their corresponding data type once picked up by the I/O buffer of the recipe editor.

One practical concern is that in Visual Basic, the programming language of the Process-level control system, an array can not be a public member of a class. Thus it is impossible to send an array of data to the I/O buffer at a single time. One solution is to

create a private array and its index (local copy) and a public string and its index (public copy). To read a variable from the private array, the public index is first set, which is copied by the local index. The request of the public string will pick up the value in the private array located by the local index, and assign it to the public string. By advancing the public index one by one and requesting the public string one at a time, the whole content of the private array can be read by the I/O buffer of the editor. A simple I/O stream function can be written to handle this process. Figure 3.2 illustrates the private array data out process of a step class. The data in process follows the similar pattern (set index → set public string → locate → assign).

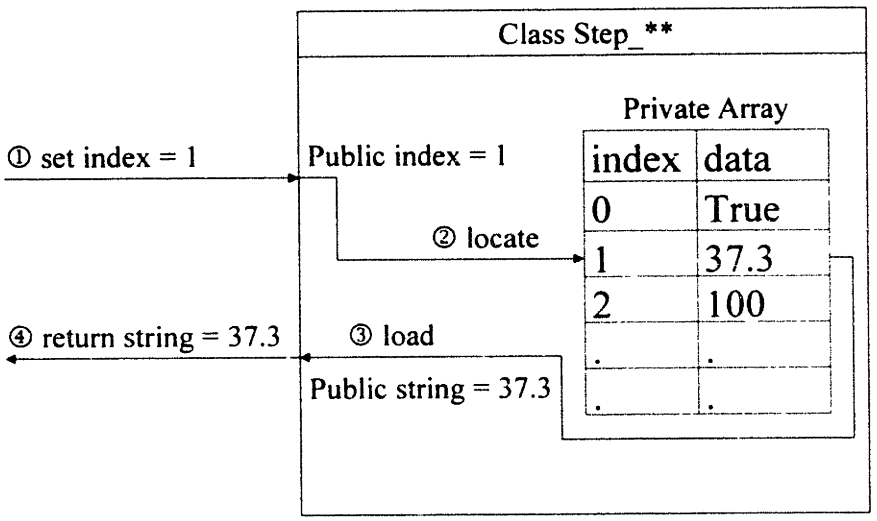


Figure 3.2 Private Array Data Out Procedure of Step Class

Table 3.5 Decomposition of Recipe File

Index: 2112.#	Control System Design		
	Functional Requirements (FRs)		Design Parameters (DPs)
	Name	Description	Description
		Define the recipe file structure	*Recipe file*
1	Process	Store the overhead information	Overhead section
2	Process	Store the individual recipe steps	Step section

$$\begin{Bmatrix} FR21121 \\ FR21122 \end{Bmatrix} = \begin{bmatrix} X & O \\ X & X \end{bmatrix} \begin{Bmatrix} DP21121 \\ DP21122 \end{Bmatrix}$$

Even though a recipe consists of many step objects, it is required to be stored in a single recipe file. Otherwise, a recipe will generate several recipe files and the bookkeeping

and maintenance of them can be problematic. A recipe file contains two sections: overhead and step. The overhead section is the file header. It contains the information from the Recipe_Overhead class. The step section is composed of a series of process steps following the order labeled in the overhead section. In run time, the editor first reads the information in the overhead section and create the appropriate type and number of step objects. Table 3.5 is the decomposition of the recipe file structure.

Each step part is composed of the tag fields and the data fields. The data fields hold the process parameters, originally Boolean, integer, or float variables, converted to the texts. Figure 3.3 illustrates the structure of a recipe file.

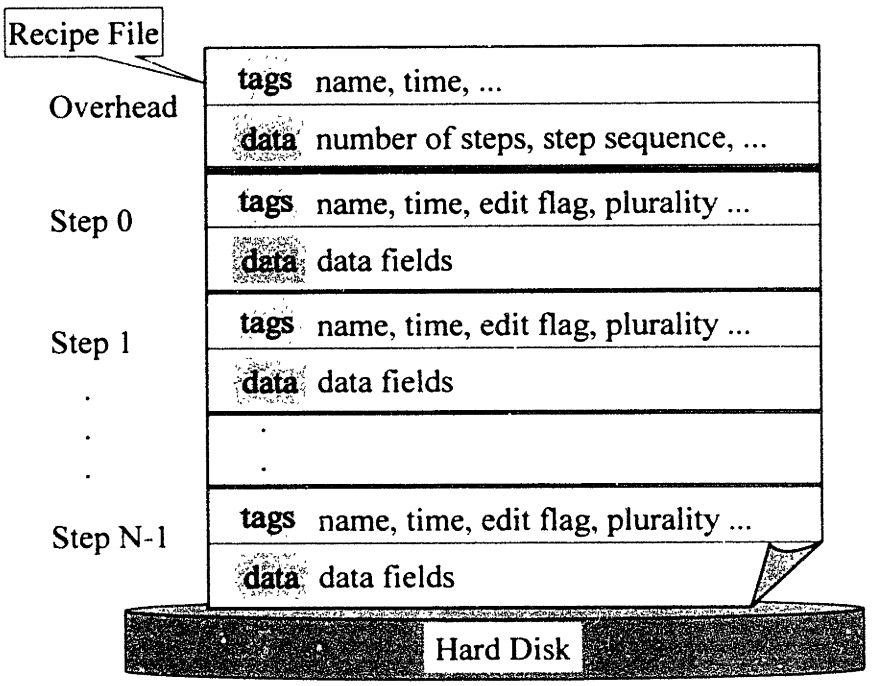


Figure 3.3 Structure of Recipe File

A recipe editor should provide a user interface (front end) and contain functionalities (back end) to handle recipes in and out of the recipe files via the recipe objects. The recipe editor can be decomposed as shown in Table 3.6

The file I/O handler takes care of the data transmission between a file and the corresponding recipe objects. Because the objects have the built-in file I/O methods, the file I/O handler simply opens a file, calls the object file I/O methods passing the file name as an argument, and close the file. The physical data transfer is executed by the object file I/O methods.

The editor I/O handler performs the data transmission between the editor and the objects. In a saving event, the user generated data are first stored in the I/O buffer and written to the objects. In a open event, the data from the objects are fed to the I/O buffer and the data in the buffer are displayed on the editor user interface. Because the objects do not have editor I/O methods, the editor I/O handler should have the I/O stream functions to read from and write to the private array of a recipe object. Figure 3.4 illustrates the I/O handling from the recipe editor.

Table 3.6 Decomposition of Recipe Editor

Index: 2113.#	Control System Design		
	Functional Requirements (FRs)		Design Parameters (DPs)
	Name	*Description*	*Description*
		Manipulate recipes	Recipe editor
1	Process	Link recipe objects to a recipe file	File I/O handler
2	Process	Link recipe objects to the recipe editor	Editor I/O handler
3	Process	Interact with a user	Editor user interface

$$\begin{Bmatrix} FR21131 \\ FR21132 \\ FR21133 \end{Bmatrix} = \begin{bmatrix} X & O & O \\ O & X & O \\ X & X & X \end{bmatrix} \begin{Bmatrix} DP21131 \\ DP21132 \\ DP21133 \end{Bmatrix}$$

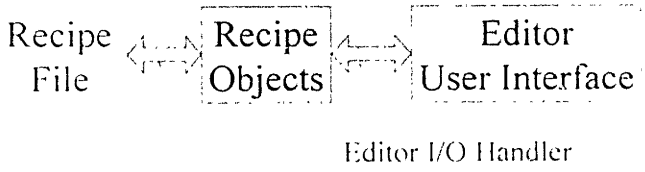


Figure 3.4 Object I/O Handling from Recipe Editor

An editor should provide the required functionalities (methods) to process the information and an appropriate window lay out both for the data display and the input controls. The editor user interface can be decomposed as in Table 3.7.

The editor methods are a set of procedures to link the recipe information between the editor window and the back end handlings. The methods will be accessible both by the pull down menus under the editor title bar and the tool bar buttons. The following are the minimum set of methods to edit a recipe.

- New: Create a default recipe
- Open: Open an existing recipe
- Save: Save the edited recipe
- Save As: Save the edited recipe under a new name
- Close: Close the current recipe

◆ Exit: Exit the recipe editor

Other methods, such as Print, may be added if needs arise.

Table 3.7 Decomposition of Editor User Interface

Index: 21133.#	Control System Design		
	Functional Requirements (FRs)		Design Parameters (DPs)
	Name	*Description*	*Description*
		Interact with a user	***Editor user interface***
1	Process	Process recipes	Editor methods
2	Process	Edit recipe steps	Step editor
3	Process	Edit a recipe	Main editor

$$\begin{Bmatrix} FR211331 \\ FR211332 \\ FR211333 \end{Bmatrix} = \begin{bmatrix} X & O & O \\ O & X & O \\ X & X & X \end{bmatrix} \begin{Bmatrix} DP211331 \\ DP211332 \\ DP211333 \end{Bmatrix}$$

Each methods works by calling I/O handlers and executing its own procedures. Open method, for example, first calls the file I/O handler to extract and create recipe objects from a specified recipe file, calls the editor I/O handler to load the data to the I/O buffer, and executes the parameter loading procedures to display the recipe information to the editor window.

The step editor is displayed on a portion of the main editor. A single step editor will be used to edit every type of process steps for the program simplicity and the coding efficiency. The step editor will display the step specific information (tags) and the data view and editing section.

The main editor houses the step editor and the editor methods buttons. It will display the recipe file path and its overhead information, and step editor control buttons. Figure 3.5 shows an example of the recipe editor user interface. The main editor shows the recipe file directory path and the step-wise information in a table-like format. The user highlights the step and click Edit button to load the step to the step editor window. Once finished, the user clicks Update button to save the step parameters to the step objects. A new step can be inserted and the existing step can be deleted. Clicking the step editing buttons result in the data transfer to and from the recipe objects only. Data transfer to and from a recipe file occurs only by clicking the methods buttons of the main editor.

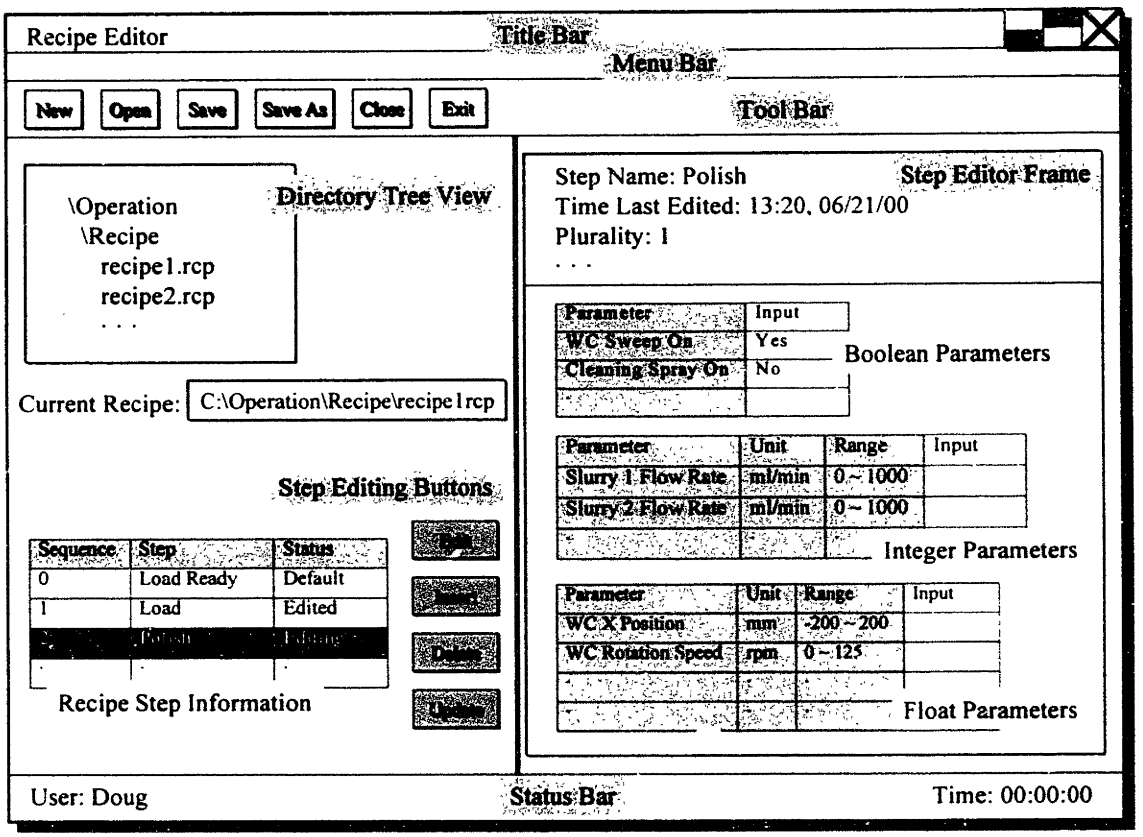


Figure 3.5 Recipe Editor User Interface

DP22. Process Unit

A wafer is processed by the equipment under one of the modes in the process unit. On one extreme, the wafer can be treated in a purely manual manner. A user has to click the corresponding button at every sequence. The user types in the wafer carrier X position and click Move button, to move the wafer carrier to the nominal x polishing position, and then punch in the wafer carrier rpm to rotate it at the polishing speed desired, and so on. On the other end, the user simply edits a recipe and load it in a fully automated environment. Now the user only has to click Run button to complete the recipe based wafer processing.

Many different levels of automation can exist between the two extremes. However, the two modes are the most fundamental and necessary- manual and auto. Many intermediate (or semiauto) modes can be designed by combining the manual and the auto mode in a certain proportion, along the path of the control system development if needs arise. The process unit is decomposed to the manual and the auto mode as shown in Table 3.8.

Table 3.8 Decomposition of Process Unit

Index: 22.#	Control System Design		
	Functional Requirements (FRs)		Design Parameters (DPs)
	Name	*Description*	*Description*
		Process wafers	***Process unit***
1	Process	Provide wafers manually.	Manual mode
2	Process	Process wafers automatically.	Auto mode

$$\begin{Bmatrix} FR221 \\ FR222 \end{Bmatrix} = \begin{bmatrix} X & O \\ X & X \end{bmatrix} \begin{Bmatrix} DP221 \\ DP222 \end{Bmatrix}$$

DP221. Manual Mode

Running a machine in a manual mode is a fairly straight forward job. A user types in command variables and click the corresponding command buttons. Then the commands are transferred to the Machine-level control system, which generates the control outputs in response to the request. Then, the machine is driven to the desired status by the control outputs. In the manual mode, the user has a one-to-one access to the machine actuators and the commands are transferred at each user generated events (Move button click, for example). The command transfers are unpacked in space and asynchronous in time.

The design of the manual mode is also straight forward. A user interface is required to interact with a user, and a Machine-level interface to communicate with the Machine-level control system. Then, a type of overhead control is required to specify and coordinate the two interfaces. Table 3.9 shows the decomposition of the manual mode.

Table 3.9 Decomposition of Manual Mode

Index: 221.#	Control System Design		
	Functional Requirements (FRs)		Design Parameters (DPs)
	Name	*Description*	*Description*
		Process wafers manually.	*Manual mode*
1	Process	Structure the manual mode software	Manual mode overhead
2	Process	Interact with a user	User Interface
3	Process	Communicate with the Machine-level software	Machine-level interface

$$\begin{Bmatrix} FR2211 \\ FR2212 \\ FR2213 \end{Bmatrix} = \begin{bmatrix} X & O & O \\ X & X & O \\ X & O & X \end{bmatrix} \begin{Bmatrix} DP2211 \\ DP2212 \\ DP2213 \end{Bmatrix}$$

The manual mode overhead should specify the I/O frequencies of the manual mode and design the handlers to deal with the two interfaces. Table 3.10 is the decomposition of the manual mode overhead.

The user input and the subsequent output to the Machine-level are transferred at a user generated event. In a Visual Basic terminology, this corresponds to a control (button) 'click' event. Thus the user input and the Machine-level output are asynchronous in nature, and do not have a specified event frequency. On the other hand, the update from the Machine-level input and subsequent user display should occur in a periodic manner, because the mating Machine-level control system runs with at least 1 KHz of event frequency. Because all the closed-loop controls and the safety related actions are contained in the Machine-level control system, there is no need for the manual mode to sample as fast as the Machine-level. For the monitoring and supervision, tens of Hz of sampling frequency is usually good enough. Microsoft operating systems (including Windows NT) provide up to 18 Hz of timer frequency. The manual mode will update inputs from the Machine-level and display them on the user interface with 18 Hz of event frequency accessible by the Visual Basic timer control.

The user input handler extracts the values from the user interface input fields, perform the necessary checks on them, and assigns them to the internal command variables. The input checks range from a simple limit check to a complicated collision avoidance logic. Suppose the wafer carrier has the travel range of –800 (sncWCXCmdMin) to +750 (sncWCXCmdMax) mm in X direction. The following procedure will safeguard the command input against its limits.

```
sngWCXCmd = CSng(txtWCX.Text)
If (sngWCXCmd > sncWCXCmdMax) Then
  sngWCXCmd = sncWCXCmdMax
End If
If (sngWCXCmd < sncWCXCmdMin) Then
  sngWCXCmd = sncWCXCmdMin
```

End If

The command input in the text box is first converted to a single (4 byte float, in Visual Basic) value and assigned to the sngWCXCmd command variable. Then the command variable goes through the maximum and the minimum check, and is bounded by them.

Table 3.10 Decomposition of Manual Mode Overhead

Index: 2211.#		Control System Design	
	Functional Requirements (FRs)		Design Parameters (DPs)
#	Name	Description	Description
		Structure the manual mode software	***Manual mode overhead***
1	Process	Designate how often inputs and outputs are updated	I/O frequencies
2	Process	Process user inputs	User input handler
3	Process	Send outputs to the Machine-level	Machine-level output handler
4	Process	Update Machine-level inputs	Machine-level input handler
5	Process	Update the machine information to the user	User display handler
6	Process	Initialize the manual mode	Manual mode initialization procedure
7	Process	Terminate the manual mode	Manual mode termination procedure

$$\begin{Bmatrix} FR22111 \\ FR22112 \\ FR22113 \\ FR22114 \\ FR22115 \\ FR22116 \\ FR22117 \end{Bmatrix} = \begin{bmatrix} X & O & O & O & O & O & O \\ X & X & O & O & O & O & O \\ X & X & X & O & O & O & O \\ X & O & O & X & O & O & O \\ X & O & O & X & X & O & O \\ X & O & X & X & O & X & O \\ O & O & O & O & O & O & X \end{bmatrix} \begin{Bmatrix} DP22111 \\ DP22112 \\ DP22113 \\ DP22114 \\ DP22115 \\ DP22116 \\ DP22117 \end{Bmatrix}$$

A more complicated logical sequence is involved to avoid collision due to the geometrical constraints. The loading/unloading station and each platen have guard walls to prevent splashing of slurries and water sprays. Thus when the wafer carrier is within each guarded zone, if it doesn't have enough clearance in the Z direction and the X direction command is greater than each zone limit, the wafer carrier will collide with the wall, if unchecked. Figure 3.6 illustrates this WCXCmd collision avoidance logic. The wafer carrier Z position is first checked against its transportation clearance. If the Z position is greater than the clearance, the new X command is OK. If not the X position is checked to see if the

wafer carrier is within either the Loading/Unloading station, the Platen A, and the Platen B. If it is within a zone, the X command is bounded by the travel limits of each zone.

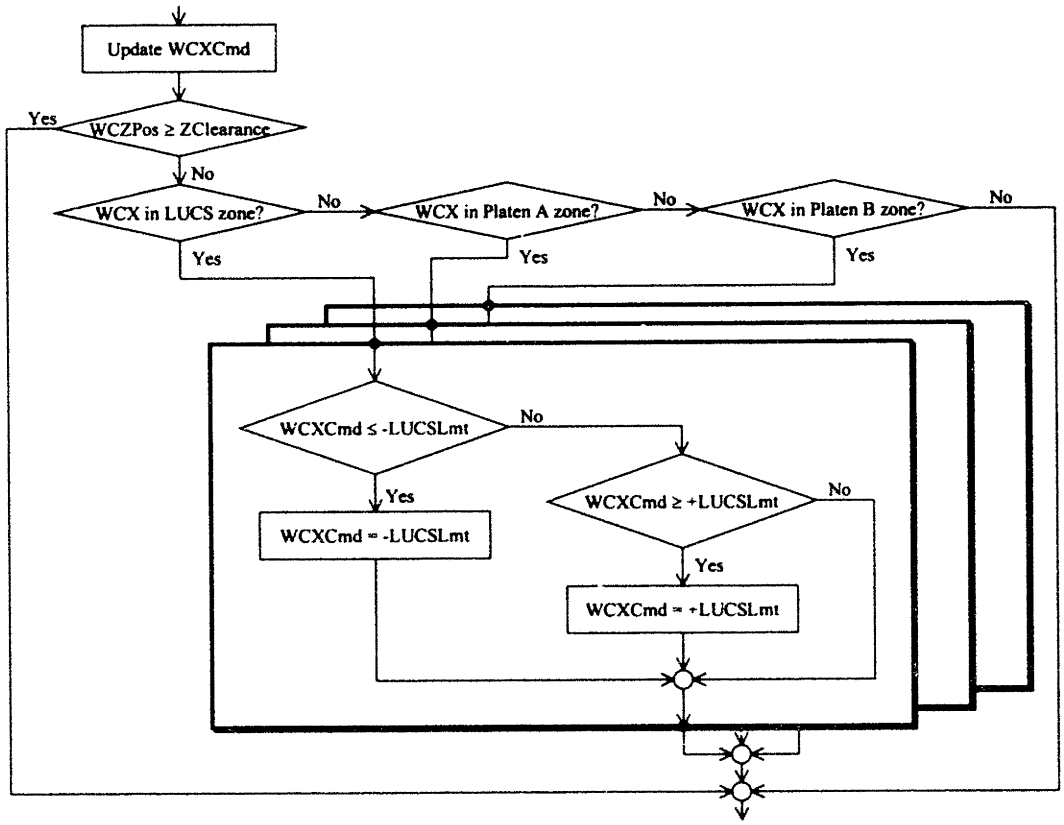


Figure 3.6 Wafer Carrier X Command Collision Avoidance Logic Sequence

The command variables processed by the user input handler are transferred to the Machine-level output handler. The output handler places the commands in certain formats specified by the Machine-level interface. Then the command packets are transferred to the Machine-level control system by the interface. The Machine-level interface may want some servo commands to be placed in a single array packet, so that it can send them at once. Another example is the bitmap transfer.

Figure 3.7 shows the example of the wafer carrier active membrane assembly(AMA) vacuum bit carrier handling. Suppose a user selected to apply vacuum to the membrane except the compartment 3. The corresponding command variables processed by the user input handler have 'True' Boolean values, while AMA 3 holds 'False.' The bitmap is initialized on zero and goes through four conditional statements. If the statement is true, the corresponding bit is set to 1, otherwise it will remain 0. Note that the addition operation is used instead of the bitwise OR operation and that the suffix -B denotes a binary number (100B = 4). The Machine-level interface requires the bitmap format, because sending one bitmap is much more efficient than sending four Boolean commands.

The Machine-level input handler receives the parameter and status inputs from the Machine-level via the Machine-level interface. The handler unpacks the inputs, analyze them,

and assigns them to the user display handler. The user display handler then displays the inputs to the designated sections of the user interface window.

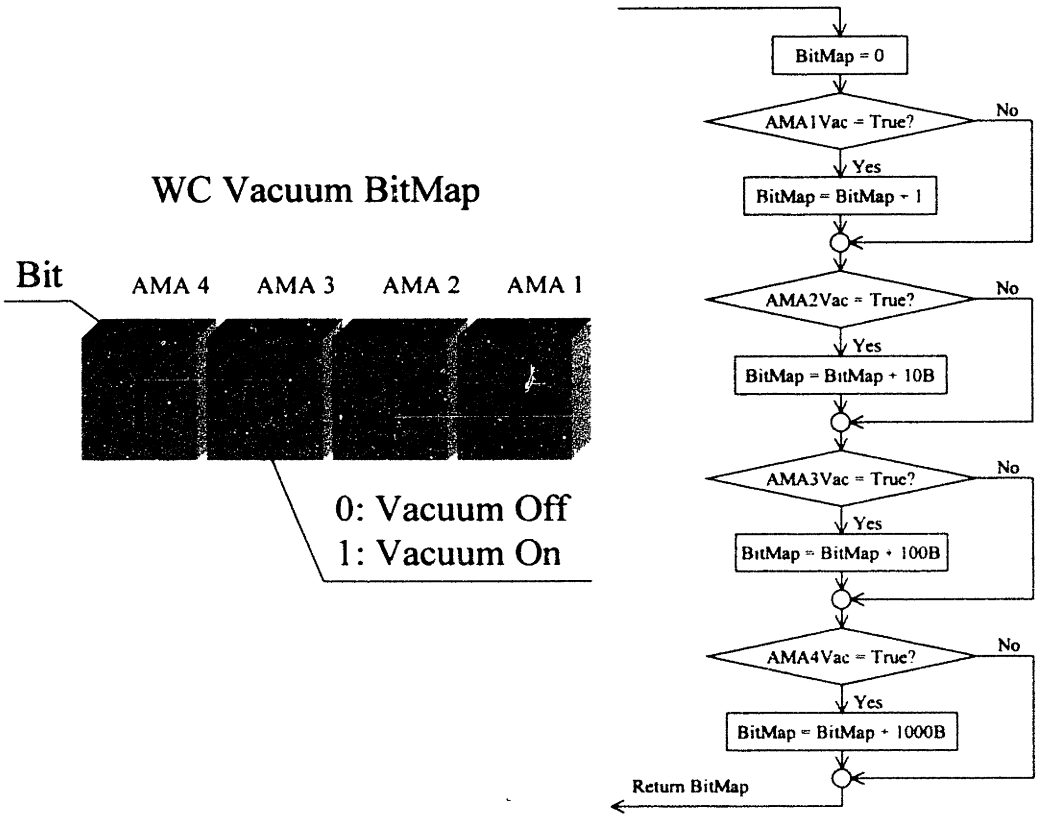


Figure 3.7 Wafer Carrier Vacuum BitMap Handling

Of the prime importance is the error handing in the Machine-level input handling. Because all the safety related actions are already taken by the Machine-level, no immediate action is required. The error handler sets and displays the error status, prevents further command update, and asks the user if he or she wants to continue to run the equipment. If yes, the user has to reset the error status. If it is successful, the machine becomes the 'running' status again. If the error reset is not successful, the error status is set again at the next timer event. If the user chose to stop the equipment run, the error handler calls the manual mode termination procedure. Figure 3.8 summarizes the error handling procedure.

The manual mode initialization procedure is executed when the manual mode is loaded to the Process-level control system. It simply assigns the constants, initialize the variables, calls the machine initialization routine of the set up mode if the machine was not initialized before the manual mode call, and loads the user interface to the screen. Figure 3.9 illustrates the procedure.

The manual mode termination procedure simply unloads the user interface from the screen. Optionally, it can perform the machine termination procedure. The machine termination procedure will move the machine components to the parking positions and then stop the ADwin program, or stop ADwin immediately if requested by an error set.

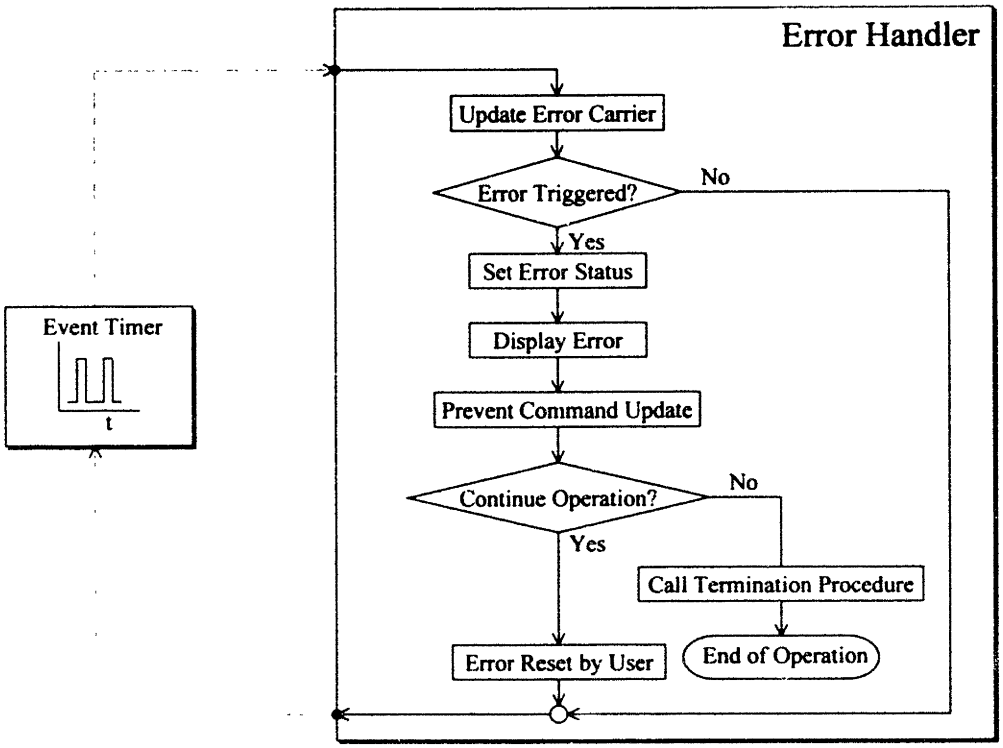


Figure 3.8 Error Handling of Manual Mode Overhead

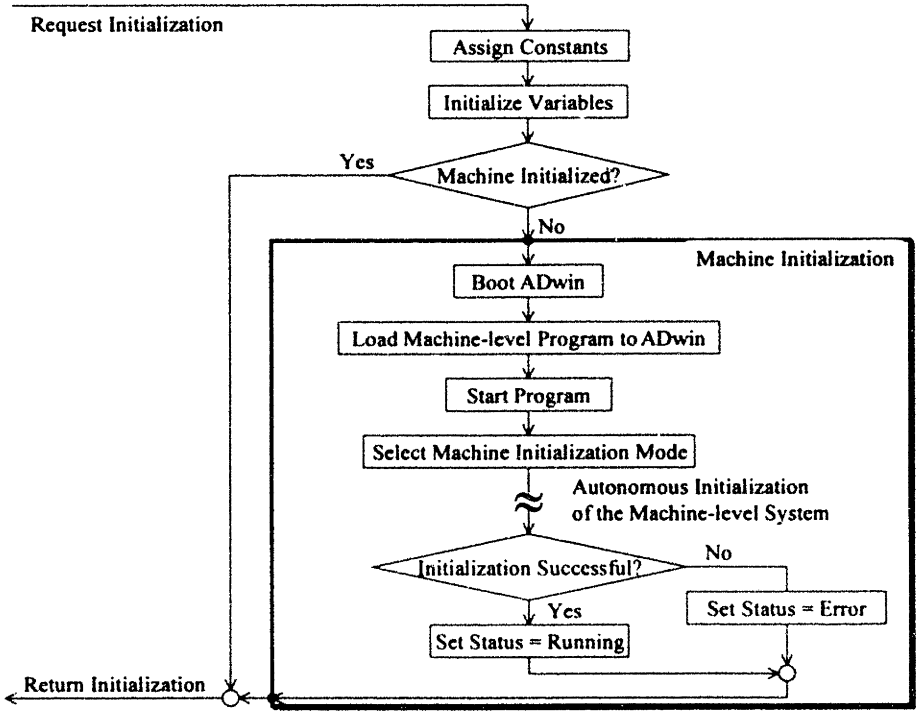


Figure 3.9 Initialization Procedure of Manual Mode

The manual mode user interface should be able to display the machine parameters and the status in real time and accept the command updates at user generated events. Table 3.11 suggests the decomposition of the user interface.

Table 3.11 Decomposition of User Interface

Index: 2212.#	Control System Design		
	Functional Requirements (FRs)		Design Parameters (DPs)
	Name	*Description*	*Description*
		Interact with a user	***User Interface***
1	Process	Provide a numeric input device for the touch screen environment	Number pad
2	Process	Accept command inputs	Command input boxes
3	Process	Accept mode inputs	Mode selector
4	Process	Provide command update controls	Function buttons
5	Process	Provide a means to remove error interlocks	Error reset button(s)
6	Process	Display the machine parameters	Parameter display
7	Process	Display the machine status	Status display
8	Process	Display the Process-level status	Status bar

$$\begin{Bmatrix} FR22121 \\ FR22122 \\ FR22123 \\ FR22124 \\ FR22125 \\ FR22126 \\ FR22127 \\ FR22128 \end{Bmatrix} = \begin{bmatrix} X & O & O & O & O & O & O & O \\ X & X & O & O & O & O & O & O \\ O & O & X & O & O & O & O & O \\ O & O & O & X & O & O & O & O \\ O & O & O & O & X & O & O & O \\ O & O & O & O & O & X & O & O \\ O & O & O & O & O & O & X & O \\ O & O & O & O & O & O & O & X \end{bmatrix} \begin{Bmatrix} DP22121 \\ DP22122 \\ DP22123 \\ DP22124 \\ DP22125 \\ DP22126 \\ DP22127 \\ DP22128 \end{Bmatrix}$$

Because the Process-level control system has the touch screen environment and the user needs to type in commands while the manual mode is running, it is required to provide an input device on the screen. For alphanumeric inputs, a virtual keyboard is required; for numeric inputs, a number pad is good enough. A simple number pad layout is shown in Figure 3.10. The user interface will display the number pad once the user clicks a command input box to enter a number. Other execution and selection commands can be handled by the command buttons, the option (radio) buttons, etc.

The user selects a mode, such as Transport, Polishing, Cleaning, etc., using the mode selector in the manual wafer processing. Function buttons control the transfer of the motion commands. The motion commands will be updated to the Machine-level once Run button is clicked. Stop button click triggers the stop sequence in the Machine-level, which stops all the

actuators and removes the controller outputs. Error Reset button is used to clear the error status if the user chooses to continue the machine operation after an error is triggered.

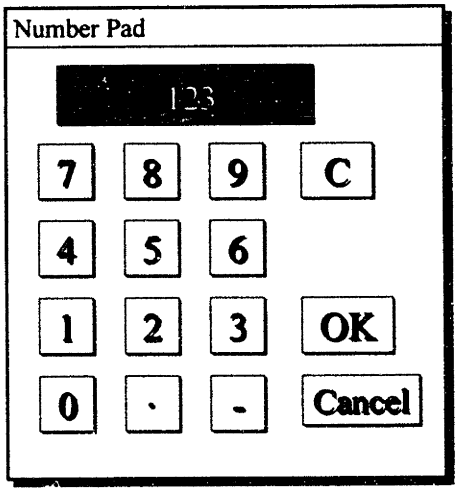


Figure 3.10 Number Pad for Touch Screen

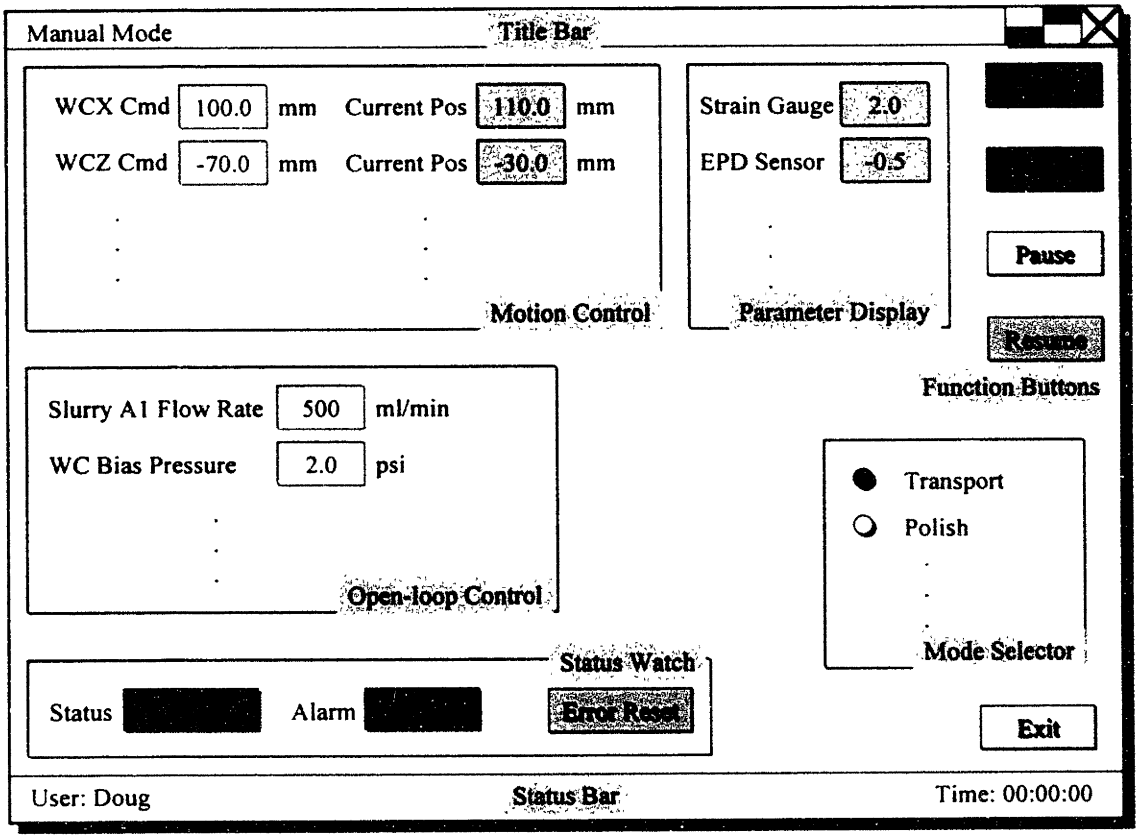


Figure 3.11 Manual Mode User Interface Screen

Machine parameters related to the motion control are displayed in the motion control section. Other parameters will be displayed in the Parameter Display section. The output from sensors, such as the strain gauge, the reflectance sensor, etc., are the examples of the parameters displayed.

The status parameters are displayed in the Status Watch window. The window will show the machine status (Running, Stopped, Initializing, etc.) and the alarm status(Normal, Error, etc.). The Error Reset button is housed in the Status Watch.

The interface may have a status bar to display the information related to the Process-level control system. User name, current time, time since login, and process time are examples of the Process-level status parameters which can be displayed. Figure 3.11 shows a layout of the manual mode user interface screen.

Table 3.12 Decomposition of Machine-level Interface

Index: 2213:#		Control System Design	
	Functional Requirements (FRs)		Design Parameters (DPs)
	Name	Description	Description
		Communicate with the Machine-level software	Machine-level interface
1	Process	Send command outputs	Command output procedure
2	Process	Send logic commands	Logic output procedure
3	Process	Receive parameter inputs	Parameter input procedure
4	Process	Receive status inputs	Status input procedure
5	Process	Receive data inputs	Data input procedure

$$\begin{Bmatrix} FR22131 \\ FR22132 \\ FR22133 \\ FR22134 \\ FR22135 \end{Bmatrix} = \begin{bmatrix} X & O & O & O & O \\ X & X & O & O & O \\ O & O & X & O & O \\ O & O & O & X & O \\ O & O & O & X & X \end{bmatrix} \begin{Bmatrix} DP22131 \\ DP22132 \\ DP22133 \\ DP22134 \\ DP22135 \end{Bmatrix}$$

Machine-level interface deals with the low level serial communication between the manual mode and the Machine-level control system. Fortunately, the ADwin provides a list of library functions for the communication between the PC and itself. Thus, to send a set of commands, the machine-level interface only have to receive the data packet from the machine-level I/O handler and call an appropriate library function to transmit the packet.

The output from the manual mode can be grouped into: Command output (control commands) and Logic output (functional commands). For example, to update a set of servo commands, the command packet is first transferred and then the Run command is transferred. By the time the Machine-level receives the Run command, all the servo

commands are already secured. Thus it can update all the command at a single event. Otherwise some of the commands may have not been received at the update event.

All the inputs to the manual mode are checked at each timer event. In reality, at every timer event, the machine-level interface library functions request the inputs from an ADwin CPU, the global variables designated as 'accessible' in the Machine-level control system.

Each parameter or status input is carried by a single global and then by a single transmission variable. A data input is a (time) series of parameters, which is sent as a whole. The time wise measurement of a reflectance sensor is an example of the data input. An array is used to house the series in place, and transmitted as a single packet to the manual mode. The machine-level interface then receives the data and assign them to a corresponding data array of the manual mode.

DP. 222 Auto Mode

The automatic wafer processing is, in a sense, an extension of the manual wafer processing. The same set of control actions are used to induce a desired physical change on the wafer. The auto mode clicks the command buttons, on behalf of a user. The process commands are transferred in a preprogrammed manner meeting the built-in time and logic conditions in the auto mode, whereas the commands are transferred asynchronously when the user clicks the command buttons in the manual mode. In other words, the auto mode transfers the commands in serial events; the manual mode at user generated events.

Table 3.13 Decomposition of Auto Mode

Index: 222.#	Control System Design		
	Functional Requirements (FRs)		Design Parameters (DPs)
	Name	*Description*	*Description*
		Process wafers automatically	***Auto mode***
1	Process	Structure the auto mode software	Auto mode overhead
2	Process	Connect to the recipe editor	Recipe editor link
3	Process	Interact with a user	User interface
4	Process	Communicate with the Machine-level	Machine-level interface
5	Process	Designate distinct steps to process wafers in auto mode	Process steps

$$\begin{Bmatrix} FR2221 \\ FR2222 \\ FR2223 \\ FR2224 \\ FR2225 \end{Bmatrix} = \begin{bmatrix} X & O & O & O & O \\ X & X & O & O & O \\ X & X & X & O & O \\ X & O & O & X & O \\ X & O & O & O & X \end{bmatrix} \begin{Bmatrix} DP2221 \\ DP2222 \\ DP2223 \\ DP2224 \\ DP2225 \end{Bmatrix}$$

The design of the auto mode is also based on the platform of the manual mode. Many manual mode procedures can be reused by the auto mode with a slight modification in details. However each mode contains all the necessary procedures to process a wafer on its own way. They are independent from each other.

Procedures similar to those of the manual mode will be briefly mentioned during the decomposition. Procedures unique to the auto mode will receive more attention and undergo detailed explanations. Table 3.13 shows the first decomposition of the auto mode.

Table 3.14 Decomposition of Auto Mode Overhead

Index: 2221.#	Control System Design		
	Functional Requirements (FRs)		Design Parameters (DPs)
	Name	Description	Description
		Structure the auto mode software	***Auto mode overhead***
1	Process	Designate how often the input to and the output from the auto mode are updated	I/O frequency
2	Process	Process user input	User input handler
3	Process	Process recipe data	Recipe nandler
4	Process	Send outputs to the Machine-level	Machine-level output handler
5	Process	Update inputs from the Machine-level	Machine-level input handler
6	Process	Update the process information to the user	User display handler
7	Process	Initialize the auto mode	Auto mode initialization procedure
8	Process	Terminate the auto mode	Auto mode termination procedure

$$\begin{Bmatrix} FR22211 \\ FR22212 \\ FR22213 \\ FR22214 \\ FR22215 \\ FR22216 \\ FR22217 \\ FR22218 \end{Bmatrix} = \begin{bmatrix} X & O & O & O & O & O & O & O \\ X & X & O & O & O & O & O & O \\ X & X & X & O & O & O & O & O \\ X & X & X & X & O & O & O & O \\ X & O & O & O & X & O & O & O \\ X & O & X & O & X & X & O & O \\ X & O & O & X & X & O & X & O \\ O & O & O & O & O & O & O & X \end{bmatrix} \begin{Bmatrix} DP22211 \\ DP22212 \\ DP22213 \\ DP22214 \\ DP22215 \\ DP22216 \\ DP22217 \\ DP22218 \end{Bmatrix}$$

Recipe editor link and Process steps are new to the auto mode. The recipe editor link connects the auto mode to the recipe editor so that it can load and unload an edited recipe. Also it invokes the editor screen, if a user decides to edit a recipe at run time. Process steps are unique to the auto mode. They house sets of sequential procedures to process wafers in a fully automatic manner in conjunction with a recipe. In an auto run, the auto mode

repeatedly gives process commands contained in a recipe, at each sequential stage specified by the process steps.

Table 3.14 shows the decomposition of the auto mode overhead. In auto mode, the user inputs are confined to logic commands (Run, Stop, Error Reset, etc.). The user input handler extracts these commands from the user interface and pass them to the Machine-level output handler at user generated events. Other outputs to the Machine-level are sent periodically at each serial stage of the loaded process step, using the PC timer with the event frequency of 18 Hz. The input from the Machine-level and the subsequent user display are updated at every timer event as in the manual mode.

The recipe handler extracts the process commands from the I/O buffer of the recipe editor link and places them in the packets following the format required by the Machine-level output handler. The output handler then sends the packets to the Machine-level control system. Other design parameters in Table 3.14 perform similar functions as those in the manual mode overhead.

Table 3.15 Decomposition of Recipe Editor Link

Index: 2222.#	Control System Design		
	Functional Requirements (FRs)		Design Parameters (DPs)
	Name	*Description*	*Description*
		Connect to the recipe editor	***Recipe editor link***
1	Process	Load an existing recipe	Load method
2	Process	Unload the current recipe	Unload method
3	Process	Call the recipe editor to edit a recipe	Edit method

$$\begin{Bmatrix} FR22221 \\ FR22222 \\ FR22223 \end{Bmatrix} = \begin{bmatrix} X & O & O \\ O & X & O \\ O & O & X \end{bmatrix} \begin{Bmatrix} DP22221 \\ DP22222 \\ DP22223 \end{Bmatrix}$$

The recipe editor link provides a link to the recipe editor and the recipe files. It should be able to load and unload the saved recipe, and invoke the recipe editor to modify an existing recipe or to create a new recipe at run time without exiting the auto mode. Table 3.15 shows the decomposition of the recipe editor link.

Load method opens up a recipe file as 'read-only' to prevent any accidental modification to the file data. An recipe overhead object is created by default, and the file header information is transferred to the overhead object using the file read method of the object. Then various step objects can be created based on the header information. Each step object accesses the step section of the recipe file to acquire each step data. Then the recipe data is loaded to the I/O buffer of the recipe editor link to be used for the wafer processing. The load method closes the recipe file before finish. Figure 3.12 illustrates the procedures of the load method in a format of the sequential functional diagram. Unload method simply empties the I/O buffer and destroy the once created recipe objects.

Load and Unload methods of the recipe editor link access a recipe file directly, creating and destroying recipe objects, with its own I/O buffer. The methods are

independent from the recipe editor. As a result, the loaded recipe is treated as 'read-only' and the user can not edit the loaded recipe. He or she can only view the recipe data using the user interface. On the other hand, the user should be able to call the recipe editor in the auto mode, otherwise he or she has to terminate the auto mode to access the editor. The user may have forgotten to edit a recipe in advance or found a need to modify an existing recipe.

Edit method calls the recipe editor and launches its screen on top of the auto mode user interface. Once the user finishes editing, he or she can go back to the auto mode user interface, and load the just edited new recipe.

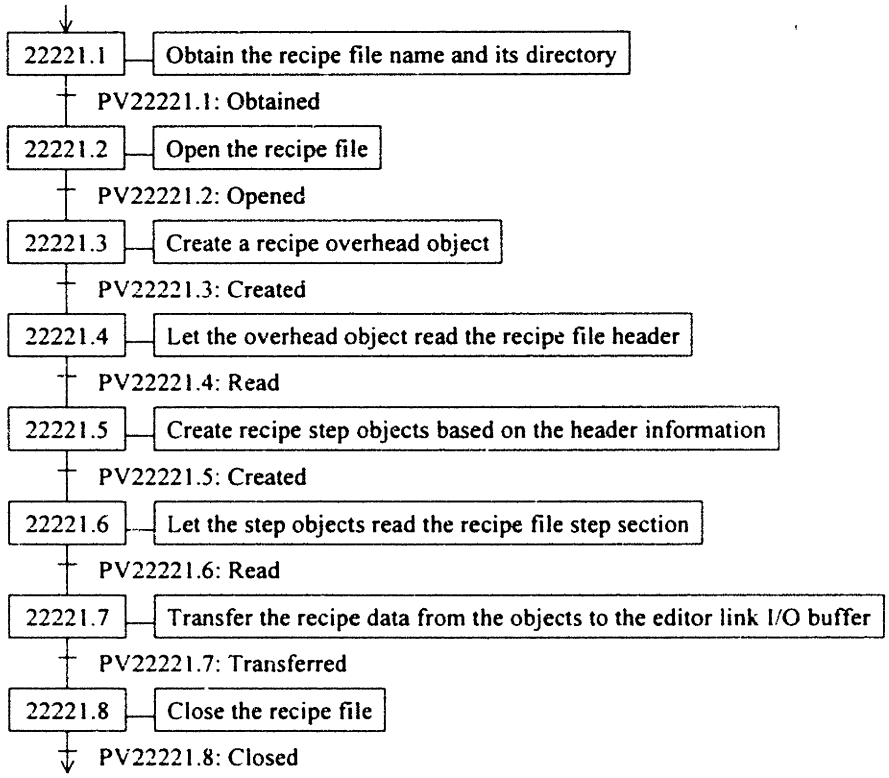


Figure 3.12 Sequential Functional Diagram of Load Method of Recipe Editor Link

An economical and logical design of the user interface is quite essential, because a great amount of information needs to be viewed in the auto mode. Table 3.16 shows the decomposition of the auto mode user interface.

The administrative information includes title, data and time, user name, etc. They will be displayed on the top portion of the interface screen. In an automated wafer processing, the interface should show the wafer related information. The wafer tracking section will display the lot number, the wafer ID (if known), and the wafer count (number of wafer processed). Recipe name, current recipe step, and run time are the examples of the process related information. The status display and the function buttons are similar to those in the manual mode.

In an automated machine run, it is often quite convenient to view the motion of the equipment in a graphical display. The interface will have a small picture box, which shows a

simple animation of the machine movement. The motions of the wafer carrier and the conditioner, along with the wafer flow will be played in real time.

Table 3.16 Decomposition of User Interface

Index: 2223.#	Control System Design		
	Functional Requirements (FRs)		**Design Parameters (DPs)**
	Name	*Description*	*Description*
		Interface with a user	*User interface*
1	Process	Show the administrative information	Administration display
2	Process	Show the wafer related information	Wafer tracking
3	Process	Show the process related information	Process information display
4	Process	Show the machine and operation statuses	Status display
5	Process	Provide means of process control	Function buttons
6	Process	Visualize machine movements	Graphical machine movement display
7	Process	Visualize process monitoring	Process monitoring graph
8	Process	Display current machine parameters	Parameter display
9	Process	Display current commands	Command display
10	Process	Allow the user to see the recipe	Recipe View
11	Process	Provide access to other modes	Navigation buttons

$$\begin{Bmatrix} FR2223.1 \\ FR2223.2 \\ FR2223.3 \\ FR2223.4 \\ FR2223.5 \\ FR2223.6 \\ FR2223.7 \\ FR2223.8 \\ FR2223.9 \\ FR2223.10 \\ FR2223.11 \end{Bmatrix} = \begin{bmatrix} X & O & O & O & O & O & O & O & O & O & O \\ O & X & O & O & O & O & O & O & O & O & O \\ O & O & X & O & O & O & O & O & O & O & O \\ O & O & O & X & O & O & O & O & O & O & O \\ O & O & O & O & X & O & O & O & O & O & O \\ O & O & O & O & O & X & O & O & O & O & O \\ O & O & O & O & O & O & X & O & O & O & O \\ O & O & O & O & O & O & O & X & O & O & O \\ O & O & O & O & O & O & O & O & X & O & O \\ O & O & O & O & O & O & O & O & O & X & O \\ O & O & O & O & O & O & O & O & O & O & X \end{bmatrix} \begin{Bmatrix} DP2223.1 \\ DP2223.2 \\ DP2223.3 \\ DP2223.4 \\ DP2223.5 \\ DP2223.6 \\ DP2223.7 \\ DP2223.8 \\ DP2223.9 \\ DP2223.10 \\ DP2223.11 \end{Bmatrix}$$

Although the interface has the parameter display to view the various machine parameter values, it is helpful to view the parameter in a graphical form to catch the time wise and the space-wise evolution of the parameter signals. The process monitoring graph can show different types of signals- wafer reflectance, strain gauge, current sensor, etc.

The parameter and command displays are standard view panels of the auto mode for informative purposes. The navigation buttons provide access to other modes, such as manual mode, recipe editor, database, and set up mode. Figure 3.13 shows the example screen of the auto mode interface.

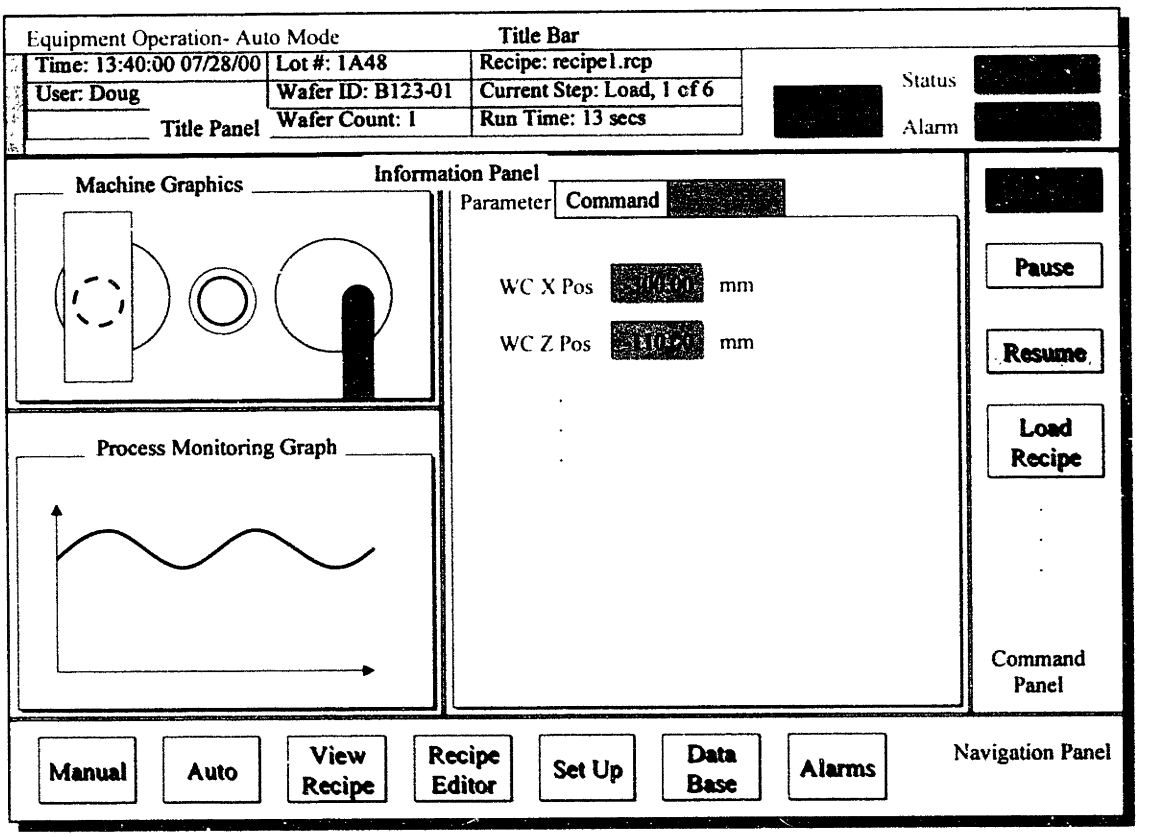


Figure 3.13 Auto Mode User Interface

The user interface layout in Figure 3.13 follows the general guideline of the semiconductor industry standard, SEMI E95-0200: Specification for Human Interface for Semiconductor Manufacturing Equipment[6]. The title panel on the top portion of the screen houses administration display, wafer tracking, process information display, and status display. The navigation panel at the bottom of the screen contains the navigation buttons, which provide links to the other modes and utilities. The two panels will always be present on the interface screen, regardless of the mode chosen. The command panel houses function buttons. The list of the function buttons will change depending on the mode or utility chosen. The information panel at the center of the screen will display the related information of the selected mode. If a user clicks View Recipe button, the information panel will change its display to the recipe related parameters. Manual mode launched from the auto mode interface will display the manual mode interface within the information panel, with a slight modification from the original manual mode interface design.

Table 3.17 shows the decomposition of the Machine-level interface of the auto mode. The procedures are basically the same as those of the manual mode, differing slightly in details. Thus we skip the detailed explanation.

Table 3.17 Decomposition of Machine-level Interface

Index: 2224.#		Control System Design	
	Functional Requirements (FRs)		Design Parameters (DPs)
	Name	*Description*	*Description*
		Communicate with the Machine-level	Machine-level Interface
1	Process	Send command outputs	Command output procedure
2	Process	Send logic commands	Logic output procedure
3	Process	Receive parameter inputs	Parameter input procedure
4	Process	Receive status inputs	Status input procedure
5	Process	Receive data inputs	Data input procedure

$$\begin{Bmatrix} FR22241 \\ FR22242 \\ FR22243 \\ FR22244 \\ FR22245 \end{Bmatrix} = \begin{bmatrix} X & O & O & O & O \\ X & X & O & O & O \\ O & O & X & O & O \\ O & O & O & X & O \\ O & O & O & X & X \end{bmatrix} \begin{Bmatrix} DP22241 \\ DP22242 \\ DP22243 \\ DP22244 \\ DP22245 \end{Bmatrix}$$

Processing of a wafer demands hundreds of individual control actions to take place both in serial and parallel. Thus it is quite necessary to organize these actions so that they can have a meaning. For example, the wafer carrier motion from x_1 to x_2 doesn't have any meaning. But if she is driven from x_1 to x_2 and then back to x_1 with a fixed frequency, it can be a sweeping motion of the wafer carrier during polishing. Small sets of control actions can be organized as 'sub steps' to be used by the bigger 'steps.' The process steps are a sequence of control actions and sub steps, which can uniquely de distinguished in the stream of wafer processing. Then a user can construct his or her own process recipe by choosing the individual steps in a desired sequence, with the parameters edited for each step. Designing steps and sub steps to assemble them to create a new recipe is much like the old good design technique of 'Divide and Conquer.'

Seven different process steps can be conceived as shown in Table 3.18. They are the minimum set of steps to create every possible recipe configuration. More steps may possibly be conceived, such as buffing or wafer carrier transportation. But the buffing is always conducted after wafer polishing and the wafer carrier transportations are executed at the beginning and the end of each step. In other words, they are not truly independent and distinctive steps to form a recipe. Rather, they can be designed as sub steps to be called within a step.

Step_LoadReady is a waiting step between the wafer unloading and the loading in a continuous wafer processing. A sequence of the recipe steps are executed from the wafer loading step to the wafer unloading step, or possibly to the conditioner steps upon user selection. The ready step simply waits the next wafer to be placed on the loading station

Either Run button click by a user or the automatic detection of wafer placement will trigger the run of the recipe based automatic wafer polishing, beginning with Step_Load.

Table 3.18 Decomposition of Process steps

Index: 2225.#	Control System Design	
	Functional Requirements (FRs)	**Design Parameters (DPs)**
	Name	*Description*	*Description*
		Designate distinct steps to process wafers in auto mode	***Process steps***
1	Process	Create sets of control actions repeatedly used in process steps	Sub-steps
2	Process	Provide a waiting step in automatic wafer processing	Step_LoadReady
3	Process	Condition polishing pads	Step_Condition
4	Process	Load a wafer	Step_Load
5	Process	Polish a wafer	Step_Polish
6	Process	Clean a wafer	Step_Clean
7	Process	Unload a wafer	Step_Unload
8	Process	Clean the conditioner head	Step_CondClean

$$\begin{Bmatrix} FR22251 \\ FR22252 \\ FR22253 \\ FR22254 \\ FR22255 \\ FR22256 \\ FR22257 \\ FR22258 \end{Bmatrix} = \begin{bmatrix} X & O & O & O & O & O & O & O \\ X & X & O & O & O & O & O & O \\ X & O & X & O & O & O & O & O \\ X & O & X & X & O & O & O & O \\ X & O & X & O & X & O & O & O \\ X & O & X & O & O & X & O & O \\ X & O & X & O & O & O & X & O \\ X & O & O & O & O & O & O & X \end{bmatrix} \begin{Bmatrix} DP22251 \\ DP22252 \\ DP22253 \\ DP22254 \\ DP22255 \\ DP22256 \\ DP22257 \\ DP22258 \end{Bmatrix}$$

Load, Clean, and Unload are fairly simple steps and easy to imagine. Step_Polish is a bit demanding. It is the most important step in wafer processing. The desired physical change of a wafer surface occurs in this step. In some sense, other steps can be viewed as supporting steps for the polish step. The detailed decomposition of Step_Polish will be given after the discussion of the sub-steps. Because of the limitation in space, we will take a look at Step_Polish only, as an example of the step decomposition and design. Other steps can be designed in a similar fashion by using the techniques to be presented.

Step_Condition and Step_CondClean are conditioner related steps and can be distinguished from the rest of the steps, which are wafer (carrier) related ones. The conditioning step can possibly cause a coupling to other wafer steps, because conditioning may be performed in parallel to them. However, if we include the conditioning as an

dependent procedures to each wafer related step, we can eliminate the coupling. Either way is expected to produce an acceptable result, depending on how much emphasis is placed on the conditioning (If you weigh conditioning more and want to achieve a flexible conditioning, it is better to design a condition step which can be executed in parallel with other wafer steps).

A sub-step is a set of control actions, which can be identified within a step. It is written because it is repeatedly used by many steps or can hide details within a step. For example, the wafer carrier transportation occurs at every step. Instead of writing procedures to move the wafer carrier from (x_1, z_1), to $(x_1, 0)$, then from $(x_1, 0)$ to $(x_2, 0)$, and finally from $(x_2, 0)$ to (x_2, z_2) at every place in steps where transportation is required, we can design a sub-step (or a function in programming sense), which has built-in procedures to drive the wafer carrier from (x_1, z_1) to (x_2, z_2). The calling step doesn't have to know the detail, the sub-step will simply return a success flag, once the movement is completed. Then the step moves on to the next stage. This technique improves the efficiency, reusability, and readability of program codes. Table 3.19 shows the decomposition of the sub-steps.

Table 3.19 Decomposition of Sub-steps

Index: 22251.#	Control System Design		
	Functional Requirements (FRs)		Design Parameters (DPs)
	Name	*Description*	*Description*
		Create sets of control actions repeatedly used in process steps	***Sub-steps***
1	Process	Move the wafer carrier and the conditioner	Transport sub-steps
2	Process	Provide means for wafer polishing	Wafer polishing sub-steps
3	Process	Substantiate pad conditioning	Conditioning sub-steps

$$\begin{Bmatrix} FR222511 \\ FR222512 \\ FR222513 \end{Bmatrix} = \begin{bmatrix} X & O & O \\ O & X & O \\ O & O & X \end{bmatrix} \begin{Bmatrix} DP222511 \\ DP222512 \\ DP222513 \end{Bmatrix}$$

The transport sub-steps are used by many steps to move the wafer carrier and/or the conditioner. They are the most frequently used sub-steps in an automated wafer processing (although hidden to a user). The wafer polishing sub-steps are a collection of functions solely used by the polishing step. Similarly, the conditioning sub-steps are for the conditioning step. We will first take a look at the transport sub-steps. Table 3.20 shows the decomposition.

There are three possible scenarios in moving the wafer carrier (WC) and the conditioner arm (CA). Move WC only, move CA only, and move WC and CA at the same time. Each of the three sub-steps in Table 3.20 is designed to respond to each case. The sub-steps are mutually exclusive. They can not occur at the same time and no new transportation sub-step can be initiated before the end of a current transport step. We will take a detailed look at Sub-step Move_WC_CA, because it is the most involved and the most general case.

Table 3.20 Decomposition of Transport Sub-steps

Index: 222511.#		Control System Design	
	Functional Requirements (FRs)		Design Parameters (DPs)
	Name	*Description*	*Description*
		Move the wafer carrier and the conditioner	*Transport sub-steps*
1	Process	Move WC	Sub-step Move_WC
2	Process	Move CA	Sub-step Move_CA
3	Process	Move both WC and CA	Sub-step Move_WC_CA

$$\begin{Bmatrix} FR2225111 \\ FR2225112 \\ FR2225113 \end{Bmatrix} = \begin{bmatrix} X & O & O \\ O & X & O \\ O & O & X \end{bmatrix} \begin{Bmatrix} DP2225111 \\ DP2225112 \\ DP2225113 \end{Bmatrix}$$

Because WC and CA share the same X axis, it is simply impossible to give position commands to each actuator and hope nothing to happen. They will surely collide with in a minute. Thus the sub-step Move_WC_CA should contain a series of movements to dissolve the spatial coupling of the actuators by differential motion dispatching in time. Table 3.21 shows the decomposition of the sub-step.

Table 3.21 Decomposition of Sub-step Move_WC_CA

Index: 2225113.#		Control System Design	
	Functional Requirements (FRs)		Design Parameters (DPs)
	Name	*Description*	*Description*
		Move both WC and CA	*Sub-step Move_WC_CA*
1	Process	Represent the state of the sub-step	Status indicator
2	Process	Receive inputs from the higher level program	Argument procedure
3	Process	Generate command values for WC and CA movements	Sequential algorithm
4	Process	Return command values and the status to the higher level	Return procedure

$$\begin{Bmatrix} FR22251131 \\ FR22251132 \\ FR22251133 \\ FR22251134 \end{Bmatrix} = \begin{bmatrix} X & O & O & O \\ X & X & O & O \\ X & X & X & O \\ X & X & X & X \end{bmatrix} \begin{Bmatrix} DP22251131 \\ DP22251132 \\ DP22251133 \\ DP22251134 \end{Bmatrix}$$

As in any function design, the input and output and its unique processing (algorithm) should be specified in the sub-step design. Sequential algorithm is the processing of the sub-step: it generates motion commands along the time line. The status indicator is the value returned by the sub-step and indicates the internal stage of the sequential algorithm. For example, it can have values 1, 2, 3, and so on, which corresponds to the sequential stage 1, 2, 3, etc. of the sub-step. It may return 0 once all the movements are completed. A higher-level program (step, in this case) will watch the returned status indicator value and decide if the movements are done and if not at which stage the sub-step is. Table 3.22 shows the decomposition of the sequential algorithm of the sub-step Move_WC_CA.

Table 3.22 Decomposition of Sequential Algorithm

Index: 22251133.#		Control System Design	
	Functional Requirements (FRs)		Design Parameters (DPs)
	Name	*Description*	*Description*
		Generate command values for WC and CA movements	***Sequential algorithm***
1	Process	Check the final points of WC and CA	Collision check procedure
2	Process	Move WC all the way up	WC up procedure
3	Process	Move CA to the up position	CA up procedure
4	Process	Move CA to the desired x position	CA x move procedure
5	Process	Move CA to the down if specified	CA down procedure
6	Process	Move WC to the desired x position	WC x move procedure
7	Process	Move WC down to the desired z position	WC down procedure

$$\begin{Bmatrix} FR222511331 \\ FR222511332 \\ FR222511333 \\ FR222511334 \\ FR222511335 \\ FR222511336 \\ FR222511337 \end{Bmatrix} = \begin{bmatrix} X & O & O & O & O & O & O \\ X & X & O & O & O & O & O \\ X & O & X & O & O & O & O \\ X & X & X & X & O & O & O \\ X & X & X & X & X & O & O \\ X & X & X & X & O & X & O \\ X & X & X & X & X & X & X \end{bmatrix} \begin{Bmatrix} DP222511331 \\ DP222511332 \\ DP222511333 \\ DP222511334 \\ DP222511335 \\ DP222511336 \\ DP222511337 \end{Bmatrix}$$

The algorithm first checks the final positions of WC and CA to see if it is safe to move to the destinations. If either X or Z axis relative distance between the two falls less than the specified safety clearance, the collision check fails and the rest of the algorithms are not executed. The indicator returns an error to the calling step.

The machine is designed such that when WC is all the way up, CA can pass underneath WC. CA has to be in the up position to make an X motion because of the splash guard walls of the platens and the loading station. However, CA Z position is controlled by the pneumatic pressure of its bellow. Thus it is impossible to do a precise position control in

Z direction for CA. Rather, we can designate two positions for CA- Up and Down. As a first (motion) stage in the algorithm, both WC and CA are moved to the up positions. Each up procedure is independent from each other. Thus they can occur parallel in a single stage. The decoupled nature between FR222511332 and FR222511333 is shown by the corresponding 'O's in the design matrix. However they can occur only after the collision check is successful. In other words, they are coupled to FR222511331. The first column of 'X's in the design matrix illustrates this coupling of the rest of procedures to the initial check. Thus after reading the first three rows of the design matrix, we can design a sequential logic:

Stage 1. Check collision
If OK go to Stage 2.
Else Return Error

Stage 2. Move WC Up AND Move CA Up

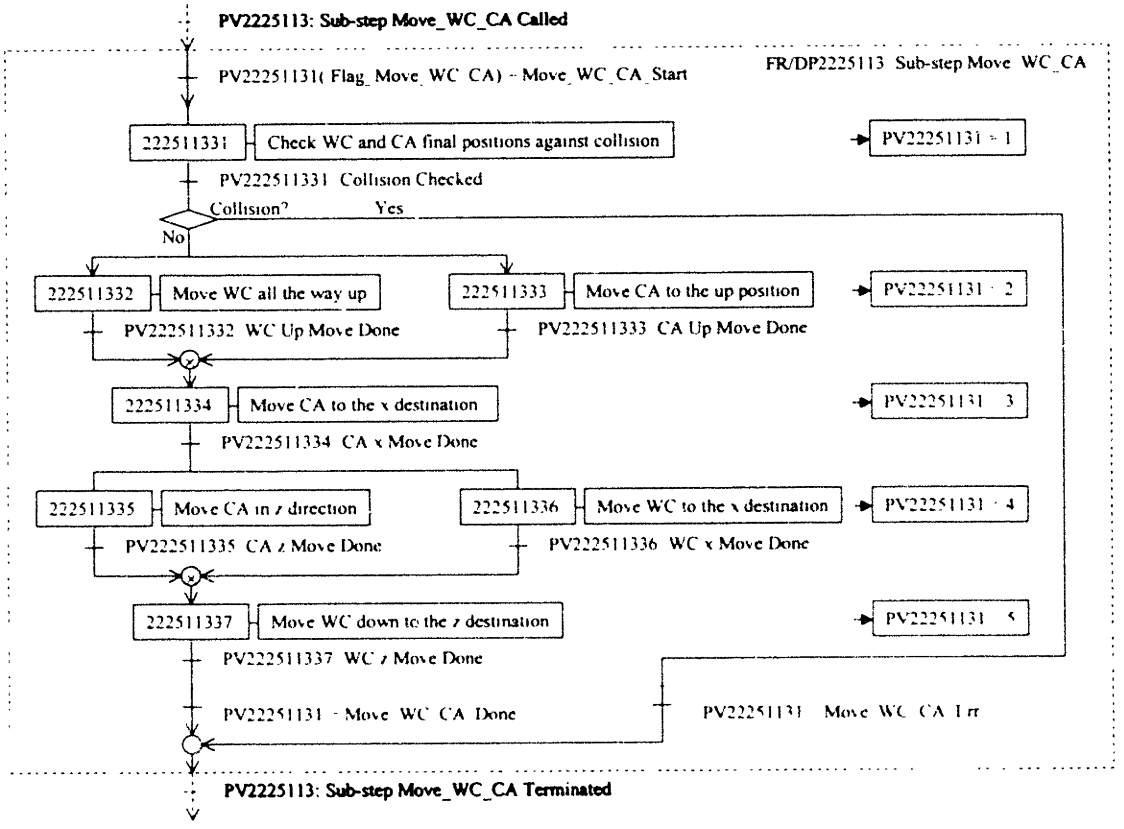


Figure 3.14 Sequential Functional Diagram of Sub-step Move_WC_CA

Traditionally, sequential logics of a control system have been designed by using a 'sequential functional diagram.' The sequential functional diagram is a graphical representation of a sequential algorithm in which each control stage is connected serially to one another with a transition condition between the two stages. We can also form a sequential functional diagram of the sub-step Move_WC_CA based on its design matrix,

because the matrix contains the information of its sequential logic. Figure 3.14 shows the sequential functional diagram constructed from the design matrix.

The FR/DP number serves as a stage number. The process variable represents the corresponding stage transition condition. Thus in the WC move up procedure (DP), the process variable checks if the condition (WC all the way up?) is met. If satisfied, it returns True and signals the transition to the next stage, otherwise it gives False and the transition is prohibited. The 'AND' junction (⊗) represents the transition from the parallel processes. Every transition condition should be true to advance to the next stage. The status of each stage is monitored by the status indicator (DP/PV22251131). In this case, the design parameter and the process variable are the same.

After WC and CA move up procedures are done, CA is moved to its X destination position (CA is cleared to move in X direction). Then WC is moved to its X destination position and CA either remains up or is moved down depending on the Z destination as a parallel stage. Finally WC is moved to its Z destination. Once the WC Z motion is completed, the status indicator returns 'Success' and exits the sub-step. Figure 3.15 illustrates these four motion stages graphically.

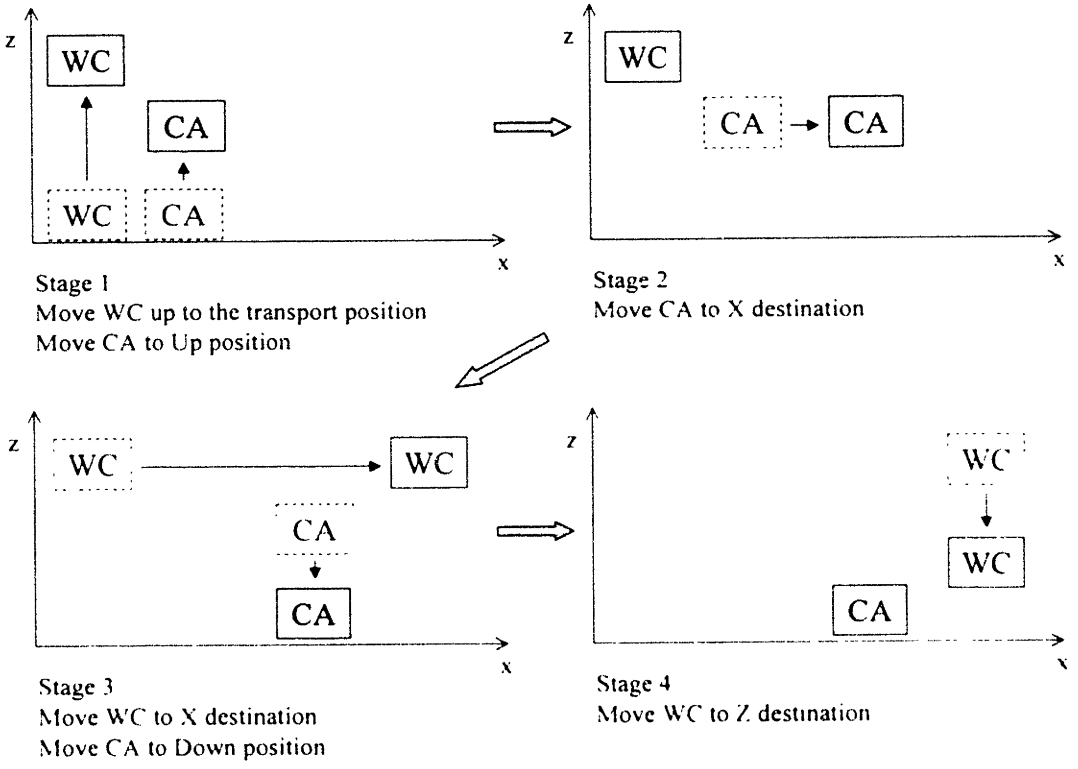


Figure 3.15 Four Motion Stages of Sub-step Move_WC_CA

A wafer polishing sub-step is a set of procedures, which can be identified to have a single purpose, used by the polishing step. Because lots of control actions are involved in the polishing step, it is quite convenient to organize sub steps before and simply call them as single functions, which eases the design of the polishing step and reduces its complexity. Table 3.23 shows the decomposition of the wafer polishing sub-steps.

Table 3.23 Decomposition of Wafer Polishing Sub-steps

Index: 222512.#		Control System Design	
	Functional Requirements (FRs)		Design Parameters (DPs)
	Name	*Description*	*Description*
		Fundamental Wafer Polishing	Wafer Polishing Sub-steps
1	Process	Sweep WC during polishing	Sub-step Sweep_WC
2	Process	Profile AMA pressure during polishing	Sub step Profile_AMA
3	Process	Profile linear velocity during polishing	Sub step Profile_Vel
4	Process	Detect the end point of polishing	Sub-step Detect_EndPoint
5	Process	Control AMA pressure based on the reflectance sensing of the wafer	Sub-step Control_AMA
6	Process	Buff wafer after polishing	Sub step Buff_Wafer

$$\begin{Bmatrix} FR2225121 \\ FR2225122 \\ FR2225123 \\ FR2225124 \\ FR2225125 \\ FR2225126 \end{Bmatrix} = \begin{bmatrix} X & O & O & O & O & O \\ O & X & O & O & O & O \\ X & O & X & O & O & O \\ X & O & X & X & O & O \\ X & X & X & X & X & O \\ X & O & O & O & O & X \end{bmatrix} \begin{Bmatrix} DP2225121 \\ DP2225122 \\ DP2225123 \\ DP2225124 \\ DP2225125 \\ DP2225126 \end{Bmatrix}$$

Material removal rate (MRR) of a CMP process is often described by Preston equation

$$MRR = K_p P v$$

MRR: material removal rate (nm/min)
K_p: Preston Constant (kPa)
P: Pressure (kPa)
v: Velocity (m/sec)

Thus, the material removal rate is directly proportional to the product of pressure and velocity. Preston constant depends on the set up of the equipment (slurry chemistry, abrasive, etc.), pad (elastic modulus, porosity, etc.), tool configuration (wafer carrier head design, platen design, etc.). It can also depend on pressure and velocity beyond a certain region.

In an advanced process handling, it is desirable to have a capability to adjust pressure and velocity during polishing instead of keeping single ones through out the processing. We may design a velocity profiler which adjusts rotational speeds of the head and the platen depending on the X position of the head to give a uniform relative velocity on the wafer

surface. Also we may want to give a slight ramp profile of velocity. For example, 0.7 m/sec at the beginning, 1.0 m/sec by the end of a polishing with linear increase in between. The sub-step Prof_Vel will adjust the velocity components of the machine to create a user specified velocity profile. Similarly, Prof_AMA is used to implement a user specified pressure profile, both in time and space (radial variation), during polishing.

The duration of polishing is usually preset for a certain amount of time, with the assumptions that the material removal rate is known and that the material removal rate does not change during polishing. But the material removal rate usually varies a little with each run, and as a result the time based wafer polishing is not accurate. In industry, they usually adjust polishing time based on the inspection of the test wafers after a batch of wafer run. Fluctuation of the material removal rate result in either an overpolished wafer or an underpolished wafer. If overpolished too excessively, it is scraped. If underpolished, it goes back to the equipment and goes through recycle. Both reduce the productivity of the tool significantly.

It is quite essential to have a sensing and subsequent control technique, which can detect the end of polishing while the polishing is actually taking place. Various sensing techniques can be used- optical, acoustic, electrical, etc. In the α machine, an optical sensor which measures the reflectance of the wafer surface is installed as a primary end point detection sensor. A current sensor, which measures the current input to the platen motor amplifier complements the end point detection. The current is proportional to the torque generated by the motor. At the end of polishing, there usually is a change in the friction coefficient between the pad and the wafer, which result in the change of motor torque. The sub-step Detect_EndPoint actually is an interface to a huge program which use signal processing, statistics, database, and model based decision to single out an endpoint. A whole new thesis may be written about this subject. Due to the limitation in space, we will omit the detailed explanation of the endpoint detection.

The reflectance sensor can also provide quantitative (though limited) information about the layer thickness being polished on a wafer. Thus it is possible to adjust the head membrane chamber pressures during polishing to enhance the uniformity of the surface. It simply reduces the pressures of a section which shows a faster polishing than the rest. The sub-step Control_AMA is a part of the advanced process handling, which actively controls the chamber pressure based on the real-time surface profile obtained from the reflectance sensor to enhance the polishing uniformity. We stop further discussion at this point because of its involvement.

The wafer carrier usually goes through a cyclic radial motion, or sweeping, during polishing. Sweeping is normally performed to even out the pad wear (and hence increase pad life), and possibly to help slurry transport to the polishing interface. The design of the sweeping sub-step is a little different from the design of the sequential sub-steps. It is periodic in nature. The sweeping motion should occur repeatedly throughout the whole polishing step, possibly in parallel with other sub-steps and procedures. The sweeping sub-step requires a 'cyclic' algorithm. Table 3.24 shows the decomposition.

The Sweep_WC sub-step also has a status indicator which represents the internal stage of itself. The inputs to the sub-step are sweep amplitude, cycle time, current time, input shaping selection, previous status indicator, etc. The WC X command is also transferred to the sub-step by reference, so that any update in the sub-step can be delivered to a higher step. The sub-step returns the value of the status indicator at the end of its execution. The sub-step can not terminate itself. Once at steady state in the cyclic algorithm, it repeatedly

gives sweep commands with the specified cycle time. The higher level program simply stops calling the sub-step to terminate sweeping.

Table 3.24 Decomposition of Sub-Step Sweep_WC

Index: 2225121.#	Control System Design		
	Functional Requirements (FRs)		Design Parameters (DPs)
	Name	*Description*	*Description*
	[illegible]	[illegible]	[illegible]
1	Process	Represent the internal state of the sub-step	Status indicator
2	Process	Receive inputs from a higher level step	Input procedure
3	Process	Generate a reference velocity based on the inputs	Reference velocity computation procedure
4	Process	Generate cyclic position commands based on the inputs	Cyclic position command generator
5	Process	Update cycle time information	Cycle time updater
6	Process	Return outputs	Output procedure

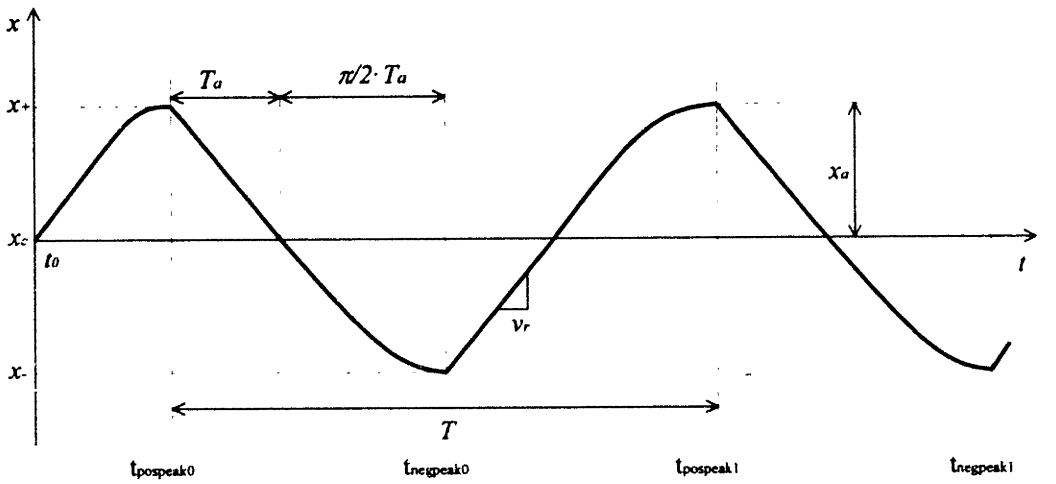
$$\begin{Bmatrix} FR22251211 \\ FR22251212 \\ FR22251213 \\ FR22251214 \\ FR22251215 \\ FR22251216 \end{Bmatrix} = \begin{bmatrix} X & O & O & O & O & O \\ X & X & O & O & O & O \\ X & X & X & O & O & O \\ X & X & O & X & O & O \\ X & X & O & X & X & O \\ X & X & X & X & X & X \end{bmatrix} \begin{Bmatrix} DP22251211 \\ DP22251212 \\ DP22251213 \\ DP22251214 \\ DP22251215 \\ DP22251216 \end{Bmatrix}$$

A user simply selects the amplitude, the cycle time, and possibly the input shaper of a sweeping motion. Then the sub-step first calculates the reference velocity based on the amplitude and the cycle time for a specific input shaper chosen. As shown in Figure 3.16, the reference velocity is computed by the following formula, in the case of the ramp plus sinusoid input shaping:

[illegible]

v: reference velocity (mm/sec)
x: amplitude (mm)
t: cycle time (sec)

The reference velocity and X position commands generated by the sub-step are then transferred to the Machine level control system to produce the sweeping motion.



$(1 + \pi/2)T_a = T/2$
$1/T_a = (2 + \pi)/T$
$v_r = x_a/T_a = (2 + \pi)\cdot x_a/T$

Figure 3.16 Wafer Sweep Command Profile with Ramp Plus Sinusoid Input Shaper

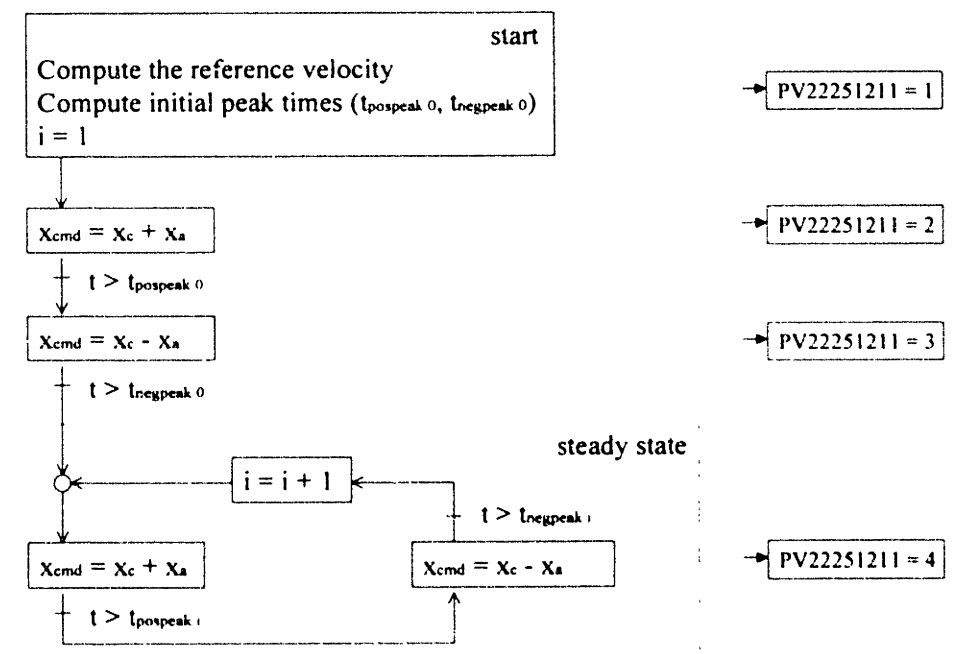


Figure 3.17 Cyclic Position Command Generation Algorithm of sub-step Sweep_WC

The cyclic position command generator alternatively generates the positive peak command (x_+) and the negative peak command (x_-) with the interval of the half cycle time between them. Figure 3.16 illustrates this alternative peak command generation at $t_{pospeak\ i}$ or $t_{negpeak\ i}$. The interval between $t_{pospeak\ i}$ and $t_{pospeak\ i+1}$ is the sweep cycle time.

The cyclic algorithm first calculates the reference velocity and the first positive and negative peak times. Then it generates the positive peak position command (x_+). Once the current time passes the positive peak time, it starts to give the negative peak command (x_-). Again once the time passes the negative peak time, it sends the positive peak command. Then it reaches the steady state. It gives the positive peak command until the time reaches the positive peak time, then the negative peak command until the negative peak time. This cycle is repeated. At each transition, the next peak time is calculated by the following simple formula.

$$t_{pospeak\ i} = t_{pospeak\ 0} + i \cdot T \quad \text{or} \quad t_{negpeak\ i} = t_{negpeak\ 0} + i \cdot T$$

Figure 3.17 summarizes the cyclic algorithm.

Table 3.25 Decomposition of Sub-Step Buff_Wafer

Index: 2225126.#		Control System Design	
	Functional Requirements (FRs)		Design Parameters (DPs)
	Name	*Description*	*Description*
		Buff wafer after polishing	***Sub-step Buff_Wafer***
1	Process	Dispense DI water	DIW dispense procedure
2	Process	Spin WC and Ptn at a buffing speed	WC/Ptn spin procedure
3	Process	Apply buffing pressures to AMA	AMA pressurization procedure
4	Process	Set buffing timer	Timer set procedure
5	Process	Sweep WC if selected	Sub-step Sweep_WC
6	Process	Signal the end of buffing	Timer up procedure
7	Process	Stop DIW	DIW off procedure
8	Process	Turn sweep off	Sweep off procedure

$$\begin{Bmatrix} FR22251261 \\ FR22251262 \\ FR22251263 \\ FR22251264 \\ FR22251265 \\ FR22251266 \\ FR22251267 \\ FR22251268 \end{Bmatrix} = \begin{bmatrix} X & O & O & O & O & O & O & O \\ X & X & O & O & O & O & O & O \\ X & O & X & O & O & O & O & O \\ X & X & X & X & O & O & O & O \\ X & X & X & X & X & O & O & O \\ X & X & X & X & X & X & O & O \\ X & X & X & X & X & X & X & O \\ X & X & X & X & X & X & O & X \end{bmatrix} \begin{Bmatrix} DP22251261 \\ DP22251262 \\ DP22251263 \\ DP22251264 \\ DP22251265 \\ DP22251266 \\ DP22251267 \\ DP22251268 \end{Bmatrix}$$

Wafer buffing is performed at the end of polishing with deionized (DI) water only, to finish and clean up the wafer surface. The wafer is usually buffed with a higher velocity and a lower pressure than the ones in polishing. In the viewpoint of process design, the buffing sub-step is a small cousin of the polishing step with reduced number of sequential stages involved. Table 3.25 shows the decomposition of the sub-step.

The polishing step brings the wafer carrier to the nominal polishing position and stops the slurry dispensation before it calls the buffing sub-step. The sub-step starts itself by dispensing DI water to the platen. Then, the wafer carrier and the platen are ramped up to the buffing speed (rpm), and the membrane compartments in the wafer carrier are pressurized to the buffing pressures in parallel. Once the velocities and the pressures reach the specified values, the software timer is set to the specified buffing time.

Sweeping is also usually performed during buffing. To sweep, the sub-step simply calls the sub-step Sweep_WC, which we have just designed. The end of buffing is signaled by the timer. The sub-step stops DI water dispensing and wafer sweeping, before returning 'finish' to the polishing step. Figure 3.18 shows the sequential functional diagram of the buffing sub-step.

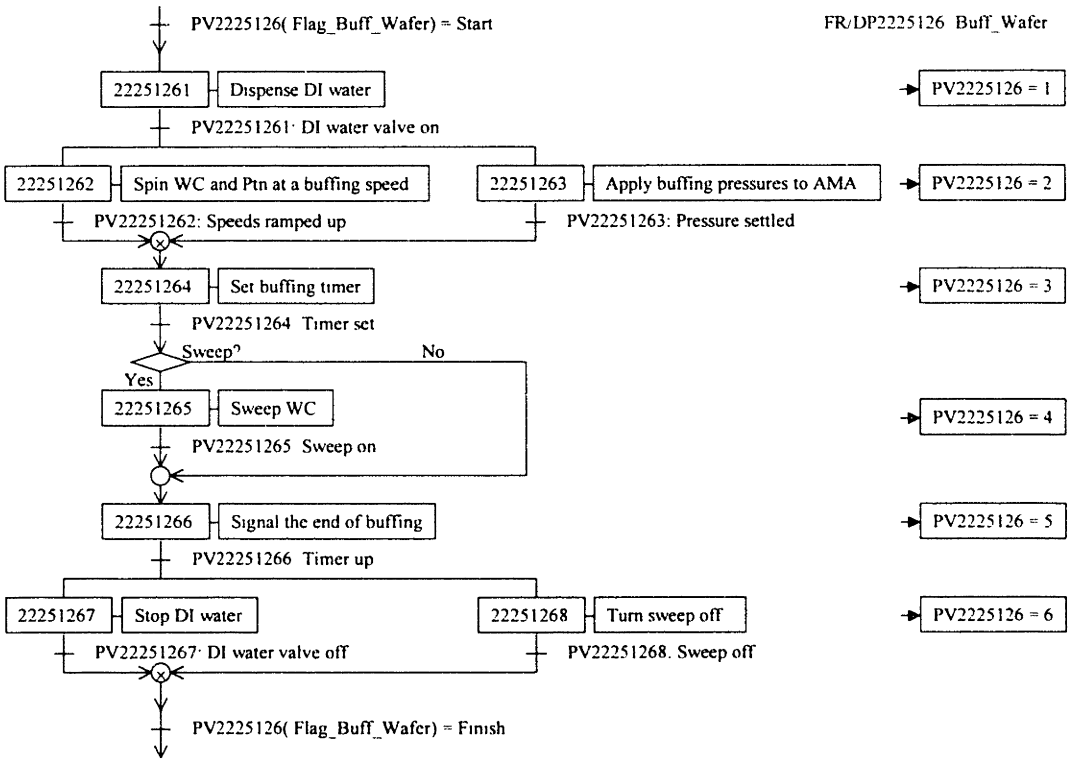


Figure 3.18 Sequential Functional Diagram of Sub-Step Buff_Wafer

Conditioning requires bringing the conditioner to the condition position, spinning CA and the platen at condition speeds, applying pressure to CA, dispensing DI water (and slurry), etc. In terms of control sequence design, the condition step is much similar to the

polishing step. The conditioning sub steps are reduced sets of the polishing sub steps. Table 3.26 shows the conditioning sub step decomposition.

Table 3.26 Decomposition of Conditioning Sub steps

Index: 222513.#	Control System Design		
	Functional Requirements (FRs)		Design Parameters (DPs)
	Name	*Description*	*Description*
1	Process	Sweep the conditioner during conditioning	Sub step Sweep C A
2	Process	Profile the conditioner head pressure during conditioning	Sub step Profile C APres
3	Process	Profile the conditioner head velocity during conditioning	Sub step Profile C AVel

$$\begin{Bmatrix} FR2225131 \\ FR2225132 \\ FR2225133 \end{Bmatrix} = \begin{bmatrix} X & O & O \\ O & X & O \\ O & O & X \end{bmatrix} \begin{Bmatrix} DP2225131 \\ DP2225132 \\ DP2225133 \end{Bmatrix}$$

Table 3.27 Decomposition of Step Polish

Index: 22255.#	Control System Design		
	Functional Requirements (FRs)		Design Parameters (DPs)
	Name	*Description*	*Description*
1	Process	Represent stages of the step	Status indicator
2	Process	Receive inputs	Input procedure
3	Process	Produce process commands for each stages in the step	Sequential algorithm
4	Process	Send command and status outputs	Output procedure

$$\begin{Bmatrix} FR222551 \\ FR222552 \\ FR222553 \\ FR222554 \end{Bmatrix} = \begin{bmatrix} X & O & O & O \\ X & X & O & O \\ X & X & X & O \\ X & X & X & X \end{bmatrix} \begin{Bmatrix} DP222551 \\ DP222552 \\ DP222553 \\ DP222554 \end{Bmatrix}$$

The conditioner is swept during conditioning to cover the entire pad surface, and hence to generate a uniform wear profile. The conditioner is typically 4 inches in diameter, where as the platen is at least 24 inches in diameter. The wear of the pad during conditioning

is also a function of pressure and velocity. Profiling of pressure and velocity is then necessary to create an even pad surface. We will skip the detailed explanation of the conditioning sub-steps because they are much similar to the polishing sub-steps in nature.

With the requested sub-steps at our disposal, now we can go forward to design the process steps. However, due to the volume and complexity involved in step design, we will select Step_Polish as an example and take a detailed look at it. The polishing step is the longest and complex, yet the most important of the steps because the core physical process occurs in this step. The other steps can be designed in a similar fashion.

Table 3.28 Decomposition of Sequential Algorithm (Step_Polish)

Index: 222553.#	Control System Design		
	Functional Requirements (FRs)		Design Parameters (DPs)
	Name	*Description*	*Description*
		Produce process commands for each stages in the step	***Sequential algorithm***
1	Process	Move WC to the nominal polishing position and CA to the home position	Sub step Move_WC_CA
2	Process	Spin WC and Platen at a touch down speed	WC/Ptn spin procedure
3	Process	Supply slurry	Slurry supply procedure
4	Process	Move WC to the z polish position	WC z polish acquisition procedure
5	Process	Turn spray and drain valves on	Spray/drain valves on procedure
6	Process	Change AMA to positive(low) pressures	AMA pressurization procedure
7	Process	Spin WC and Platen at a full polishing speed	WC/Ptn spin procedure
8	Process	Apply full polishing pressures to AMA	AMA pressurization procedure
9	Process	Set timer for polishing	Timer procedure
10	Process	Sweep WC if selected	Sub step Sweep_WC
11	Process	Detect end point of the polishing	End point detection procedure
12	Process	Turn sweep off	Sweep off procedure
13	Process	Turn slurry off	Slurry off procedure
14	Process	Buff wafer if selected	Sub step Buff_Wafer
15	Process	Bring WC to the nominal x polish position	WC x move procedure
16	Process	Reduce WC and Platen rotations to a lift up speed	WC/Ptn spin procedure
17	Process	Change AMA to vacuum to pick up wafer	AMA vacuuming procedure
18	Process	Move WC up to the z clearance position for transportation	WC z move procedure
19	Process	Turn spray and drain valves off	Spray/drain valves off procedure
20	Process	Move WC to the finish position	Sub step Move_WC

Table 3.28 Decomposition of Sequential Algorithm (Step_Polish), continued

$$
\begin{Bmatrix} FR222553.1 \\ FR222553.2 \\ FR222553.3 \\ FR222553.4 \\ FR222553.5 \\ FR222553.6 \\ FR222553.7 \\ FR222553.8 \\ FR222553.9 \\ FR222553.10 \\ FR222553.11 \\ FR222553.12 \\ FR222553.13 \\ FR222553.14 \\ FR222553.15 \\ FR222553.16 \\ FR222553.17 \\ FR222553.18 \\ FR222553.19 \\ FR222553.20 \end{Bmatrix} =
\begin{bmatrix}
X & O & O & O & O & O & O & O & O & O & O & O & O & O & O & O & O & O & O & O \\
X & X & O & O & O & O & O & O & O & O & O & O & O & O & O & O & O & O & O & O \\
X & O & X & O & O & O & O & O & O & O & O & O & O & O & O & O & O & O & O & O \\
X & X & X & X & O & O & O & O & O & O & O & O & O & O & O & O & O & O & O & O \\
X & X & X & O & X & O & O & O & O & O & O & O & O & O & O & O & O & O & O & O \\
X & X & X & X & X & X & O & O & O & O & O & O & O & O & O & O & O & O & O & O \\
X & X & X & X & X & X & X & O & O & O & O & O & O & O & O & O & O & O & O & O \\
X & X & X & X & X & X & O & X & O & O & O & O & O & O & O & O & O & O & O & O \\
X & X & X & X & X & X & X & X & X & O & O & O & O & O & O & O & O & O & O & O \\
X & X & X & X & X & X & X & X & X & X & O & O & O & O & O & O & O & O & O & O \\
X & X & X & X & X & X & X & X & X & X & X & O & O & O & O & O & O & O & O & O \\
X & X & X & X & X & X & X & X & X & X & X & X & O & O & O & O & O & O & O & O \\
X & X & X & X & X & X & X & X & X & X & X & X & X & O & X & O & O & O & O & O \\
X & X & X & X & X & X & X & X & X & X & X & X & X & X & O & O & O & O & O & O \\
X & X & X & X & X & X & X & X & X & X & X & X & X & X & X & O & O & O & O & O \\
X & X & X & X & X & X & X & X & X & X & X & X & X & X & X & X & O & O & O & O \\
X & X & X & X & X & X & X & X & X & X & X & X & X & X & X & X & X & O & O & O \\
X & X & X & X & X & X & X & X & X & X & X & X & X & X & X & X & X & X & O & O \\
X & X & X & X & X & X & X & X & X & X & X & X & X & X & X & X & X & X & X & X
\end{bmatrix}
\begin{Bmatrix} DP222553.1 \\ DP222553.2 \\ DP222553.3 \\ DP222553.4 \\ DP222553.5 \\ DP222553.6 \\ DP222553.7 \\ DP222553.8 \\ DP222553.9 \\ DP222553.10 \\ DP222553.11 \\ DP222553.12 \\ DP222553.13 \\ DP222553.14 \\ DP222553.15 \\ DP222553.16 \\ DP222553.17 \\ DP222553.18 \\ DP222553.19 \\ DP222553.20 \end{Bmatrix}
$$

The design of the polishing step is simple to begin with. It requires a status indicator to represent the internal stage, the input and the output procedures, and the sequential algorithm. The sequential algorithm is unique to Step_Polish. Actually the process steps differ only by their sequential algorithms. They all have their status indicators and the input and the output procedures, which are more or less similar to one another. Table 3.27 shows the decomposition of the polishing step.

The sequential algorithm of Step_Polish is decomposed in Table 3.28. Although demanding in the number of stages involved, each stage once decomposed is fairly easy to grasp and implement. Any polishing first starts by moving the wafer carrier to the nominal polishing position and moving the conditioner to its safe home position to avoid any possible collision with the wafer carrier. To accomplish this, the step simply calls the sub-step Move_WC_CA.

The wafer carrier and the platen are rotated at a slow speed so that when the wafer makes contact with the pad, less friction can develop. Making a static contact and then ramping up the rotational velocity can induce a significantly high initial friction coefficient, which is not desirable. The slurry is dispensed in advance to wet the pad surface enough.

Because of the wear of the pad surface and the retaining ring of the wafer carrier, it is impossible to set a fixed Z polishing position. Instead, at the beginning of each polishing,

the Z polishing position needs to be acquired. For this purpose, the retaining ring has strain gauges imbedded. Moving the Z position against the pad result in compressing the retaining ring and hence the strain change in the strain gauges. With the known relationship between the gauge strain and the retaining ring pressure, it is possible to find the Z position so that the retaining ring pre-compresses the pad with the specified pressure. Reading the change of the gauge strain, the Z position acquisition procedure 'hunts' the polishing position. The drain and the DI water spray valves are turned on in parallel.

The wafer is held by applying vacuum to the wafer carrier membrane chambers during transportation. A low pressure is applied to the chambers so that the wafer can make a partial contact with the pad surface. Once the low pressure is settled, the wafer carrier and the platen are spun at the full polishing speeds. The chamber pressures are also ramped up to the polishing pressures.

The timer is set for the specified polishing time. Even in the case of the end point detection based on the reflectance and current sensing, an override time is set to prevent an infinite polishing. The endpoint detector may fail to send the end signal. The step calls the sub-step Sweep_WC, if the sweep is selected. The end of polishing is signaled either by the timer or the end point detector. Then the step turns the slurry pump off and halts the sweeping sub-step.

Buffing is optional to the user. If buffing is selected, the step calls the sub-step Buff_Wafer. All the stages for buffing are taken care of by the sub-step. The step watches the status indicator of the buffing sub-step. Once it returns 'Finish,' it moves to the next stage.

Because of the sweeping, the wafer carrier at the end of either polishing or buffing may not be at the nominal polishing position. The step moves the wafer carrier back to the nominal position to prepare the transportation. The wafer carrier and the platen may optionally spun at a low speed during the wafer 'lift.' There exists a large amount of adhesion force between the wafer and the pad, because of the surface tension of water. Spinning at a low speed usually helps to reduce the surface tension. After the rotation, vacuums are applied to the membrane chambers to pick up the wafer from the pad surface. Once the vacuum is settled, the wafer carrier slowly moves up to the Z clearance position to lift the wafer. The spray and the drain valves are turned off.

After the wafer carrier reaches the Z clearance position for transportation, the step calls the sub-step Move_WC to transport the wafer carrier to the specified finish position, which is typically the top of the loading/cleaning station. However, user may select to move to the other platen for a multi step polishing. The completion of the movement signals the end of the polishing step. The step returns 'end' to a higher level program, the auto mode in this case The auto mode terminates the step and calls the next step based on the recipe loaded. Figure 3.19 shows the sequential functional diagram of Step_Polish.

The Process-level control system has been designed throughout this chapter. The major design parameters are shown in Figure 3.20 as a summary of the chapter.

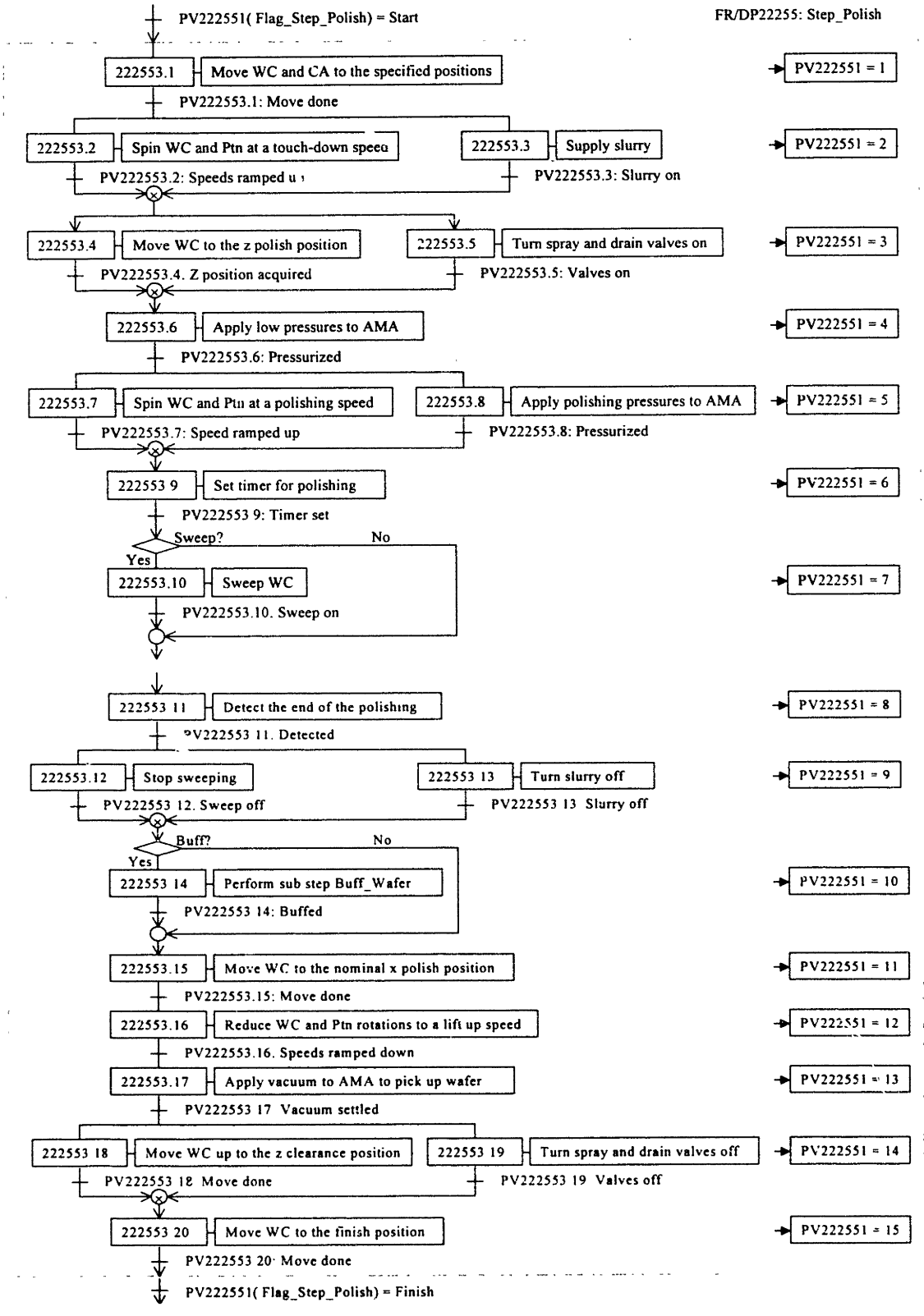


Figure 3.19 Sequential Functional Diagram of Step_Polish

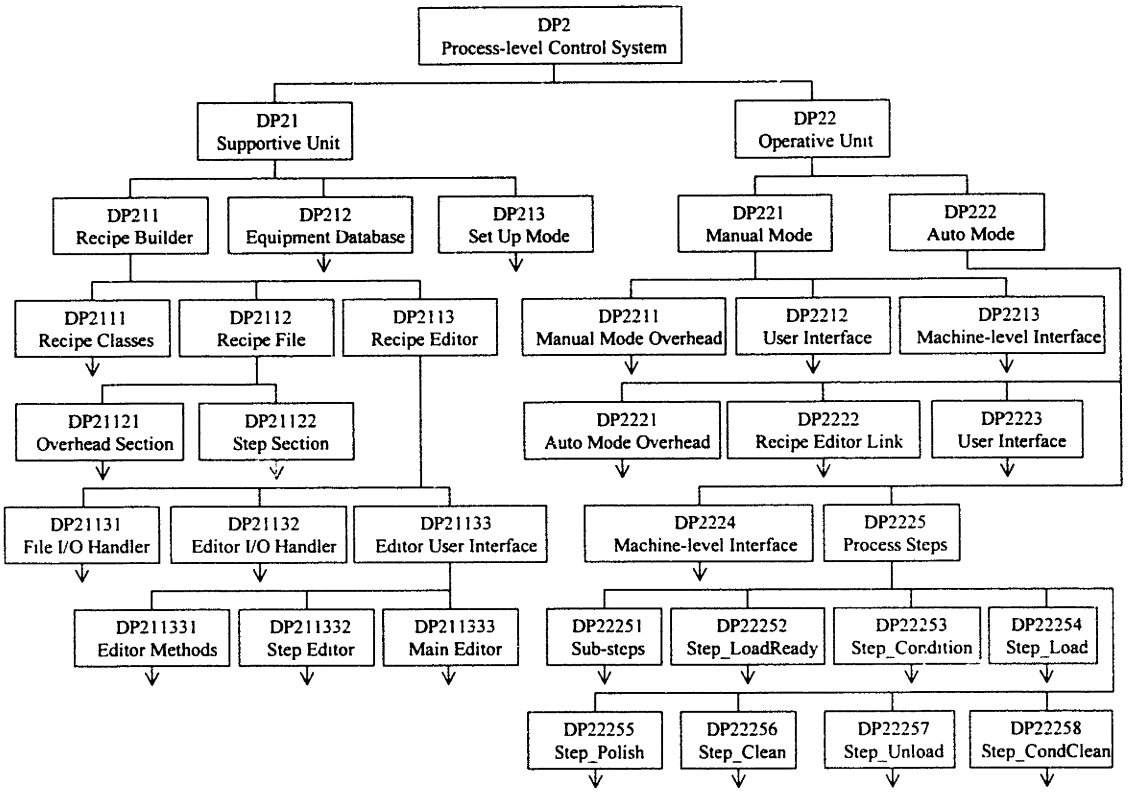


Figure 3.20 Design Parameter Tree of Process-level Control System

Chapter 4. Servo Controller Design

The CMP α machine has eight servo controllers. Four of them are position servos with reference velocity input: wafer carrier X, wafer carrier Z, conditioner X, and wafer aligner θ. The rest of them are velocity servos with reference acceleration input: wafer carrier ω, platen A ω, platen B ω, and conditioner ω. A general guide for digital servo controller design was given in the closed-loop controller design part in Chapter 2. In this chapter, we go into the details of designing and implementing a digital servo for a specific drive. The eight servo drives of the machine all have interesting characteristics, and the design of mating controllers are equally demanding.

Showing the details of all of eight controllers will take simply be too much. Instead, we will focus on a single controller to examine thoroughly the issue of designing and implementing a digital controller. As the example, we select the wafer carrier X controller, which has the interesting characteristic of the dual axes motion synchronization. The wafer carrier is attached to the gantry frame and its transportation is achieved by the gantry motion. The wafer carrier X control is a synonym to the gantry X control. Because of its construction, the gantry has two ballscrew axes, one at each supporting column, each powered by a brushless DC servo motor with gear reduction. The two axes need to be perfectly synchronized, otherwise a great amount of torque unbalance and subsequent distortion on the gantry structure will result. The positioning accuracy of the wafer carrier is not guaranteed without a harmonious coordination of two axes.

The design of gantry X controller requires not only an accurate position servo but also a harmonious synchronization between the two axes. The design first begins with modeling the system. A controller is designed in the continuous domain based on the model. The controller is then transferred to the digital domain with modifications and improvements in the digital domain. Finally, the controller is implemented and its performance is evaluated.

1. System Modeling

Figure 4.1 shows the simple schematic of the gantry X axes drive mechanism. The output speed from each motor is reduce by the gearhead and transmitted to the gantry by the ballscrew. The ball housing on each side of the gantry then converts the rotary motion of the ballscrew to the linear motion of the gantry. We are interested in the system dynamics viewed from the motor side, because it is the servo motor which we control. However modeling first starts from the linear motion of the gantry and will be reduced to the equivalent rotary motion observed by the motor.

For simplicity, we model the gantry as a second order mass-damper-spring system. Although a higher order dynamics may present, the second order is good enough because the gantry has a high mass (1100 Kg) and a high structural rigidity (welded stainless steel tubular structure). The following differential equation describes the linear dynamics of the gantry (in X direction).

$$m_g\ddot{x} + b_g\dot{x} + k_g x \pm F_f = F \qquad (1)$$

m_g: gantry mass [Kg]
b_g: viscous damping [N/(m/sec)]
k_g: spring constant [N/m]
F_f: Coulomb friction [N]
F: force applied to the gantry [N]

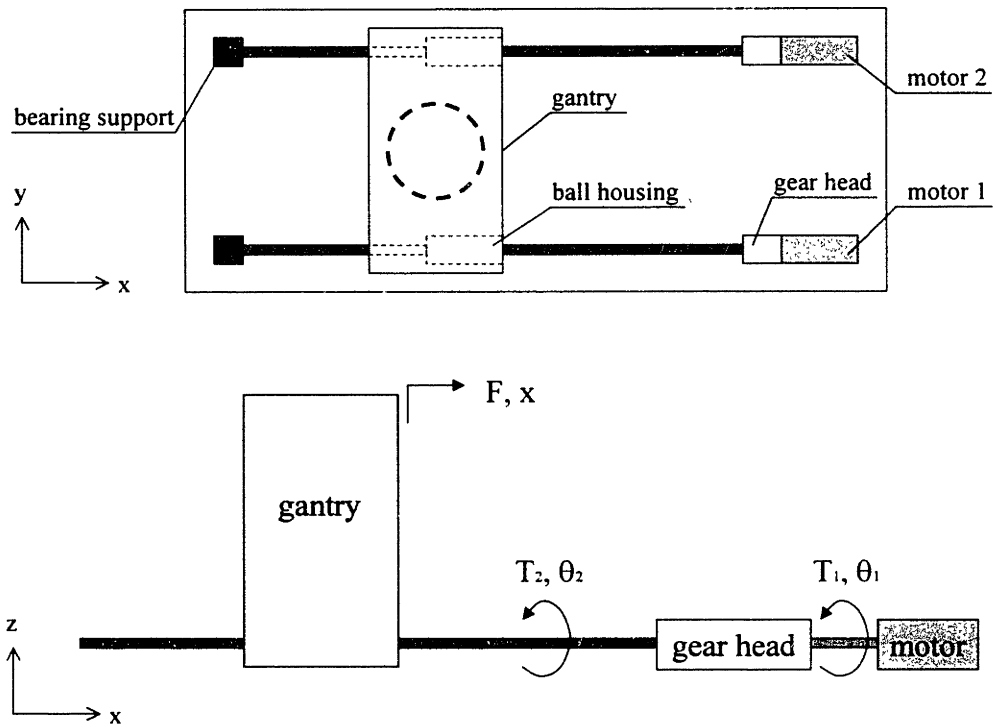


Figure 4.1 Schematic of Gantry X Axes Drive

We note that there exist two stages of energy transformation from the motor to the gantry- the gearhead and the ballscrew. The gearhead has the reduction ratio of n:1, where n = 5. Thus the angular position ratio of the gearhead output shaft to the motor shaft is given by the reciprocal of the reduction ratio.

$$\frac{d\theta_2}{d\theta_1} = \frac{1}{n}$$

The ballscrew convers the angular input to the linear output. The ratio of the output linear position to the input rotary position is called a pitch (ξ) of the ballscrew.

$$\xi = \frac{dx}{d\theta_2}$$

Suppose T_2 is the torque transmitted to the gearhead. T_2 is then used to supply the force F to the gantry and to overcome its own parasitic dynamics. We can form the following integral equation from the energy conservation argument.

$$\int T_2\, d\theta_2 = \int F\, dx + \int (J_2\ddot{\theta}_2 + b_2\dot{\theta}_2 + k_2\theta_2 \pm T_{f2})\, d\theta_2$$

By differentiation both sides with respect to θ_2, we get:

$$\begin{aligned} T_2 &= F\frac{dx}{d\theta_2} + (J_2\ddot{\theta}_2 + b_2\dot{\theta}_2 + k_2\theta_2 \pm T_{f2}) \\ &= F\xi + (J_2\ddot{\theta}_2 + b_2\dot{\theta}_2 + k_2\theta_2 \pm T_{f2}) \end{aligned} \qquad (2)$$

J_2, b_2, θ_2, and T_{f2} are the rotary dynamics of the gearhead and the ballscrew.

By similar argument, the output torque from the motor T_1 is used to supply T_2 and overcome its parasitic dynamics.

$$\int T_1\, d\theta_1 = \int T_2\, d\theta_2 + \int (J_1\ddot{\theta}_1 + b_1\dot{\theta}_1 + k_1\theta_1 \pm T_{f1})\, d\theta_1$$

$$\begin{aligned} T_1 &= T_2\frac{d\theta_2}{d\theta_1} + (J_1\ddot{\theta}_1 + b_1\dot{\theta}_1 + k_1\theta_1 \pm T_{f1}) \\ &= T_2\frac{1}{n} + (J_1\ddot{\theta}_1 + b_1\dot{\theta}_1 + k_1\theta_1 \pm T_{f1}) \end{aligned} \qquad (3)$$

Again, J_1, b_1, θ_1, and T_{f1} are the rotary dynamics of the motor.

We insert the equation (1) to the equation (2) and then insert the resulting equation to (3). Using the relationships $\theta_2 = (1/n)\theta_1$ and $x = \xi\theta_2 = \xi(1/n)\theta_1$, we obtain the following system equation viewed from the motor shaft.

$$T_1 = \mathrm{K}\, u_d = J\ddot{\theta}_1 + b\dot{\theta}_1 + k\theta_1 \pm T_f \qquad (4)$$

$$J = J_1 + \frac{J_2}{n^2} + \xi^2\frac{m_g}{n^2}$$

$$b = b_1 + \frac{b_2}{n^2} + \xi^2\frac{b_g}{n^2}$$

$$k = k_1 + \frac{k_2}{n^2} + \xi^2\frac{k_g}{n^2}$$

$$T_f = T_{f1} + \frac{T_{f2}}{n} + \xi\frac{F_f}{n}$$

J, b, k, and T_f are the equivalent inertia, viscous damping, spring constant, and friction torque, respectively, viewed from the motor side. K is the numerical torque constant of the motor and u_d is the 16 bit integer output of the digital controller.

The numerical values for the system constants can be either calculated or measured. The inertia J can be computed from the known mass and inertias. The viscous damping b and the friction torque T_f are usually unknown and need to be estimated by testing. The spring constant k is also unknown, but we can assume it negligible because the ballscrew drive is fairly rigid.

The 16 bit numerical output of a controller is first converted to the DAC voltage with the voltage gain K_v. The servo amplifier converts the command voltage to the armature current with the current gain K_a. The motor finally transforms the electrical current to the mechanical torque with the torque constant K_T. The numerical torque constant is the product of these three constants and is the direct gain from the 16 bit integer controller output to the motor torque.

$$K = K_T \cdot K_a \cdot K_v$$

K: numerical torque constant [Nm]
K_T: torque constant [Nm/A]
K_a: armature current gain [A/V]
K_v: DAC voltage gain [V]

The numerical torque constant can easily be computed from the three known constants.

The friction torque T_f is measured by increasing the output u_d from zero to the point that the gantry starts to move. The output multiplied by the numerical torque constant gives the estimation of the friction torque. The viscous damping b is obtained by giving a constant u_d to the system and measuring the rotational speed at steady state. At steady state, the acceleration is zero. Thus the governing equation reduces to:

$$K u_d = b\dot{\theta}_1 + T_f$$

Once we measure the speed ω_1, we can compute b with other known quantities.

$$b = \frac{K u_d - T_f}{\omega_1}$$

To obtain the numerical values of the system constant, we assume that the load of the gantry is equally distributed to the two axes and that each axis has the identical dynamics. We assume the mass of each axis is the half of the gantry mass. Table 4.1 shows the numerical values of the system both computed and measured.

Figure 4.2 shows the system dynamics modeled in block diagram for each axis. From now on, the subscript 1 represents the axis 1; 2 represents the axis 2. In the next session, we will see how to design a controller, which generates u_d, the input to the plant, and how to fill up the voids of the block diagram by crisscrossing inputs and outputs.

Table 4.1 Numerical Values of Gantry X Control System Constants (for each axis)

Parameter	Symbol	Unit	Numerical Value
Effective Gantry Mass	m_g	Kg	0.5×1100
Screw/Gearhead Inertia	J_2	Kg·m^2	4.07×10^{-4} (SC) + 5.0×10^{-4} (GH)
Motor Inertia	J_1	Kg·m^2	13.9×10^{-5}
Effective Inertia	J	Kg·m^2	5.348×10^{-4}
Effective Viscous Damping	b	N·m/(rad/sec)	4.51×10^{-3}
Effective Friction Torque	T_f	N·m	5.61×10^{-1}
Numerical Torque Constant	K	N·m	2.23×10^{-4}

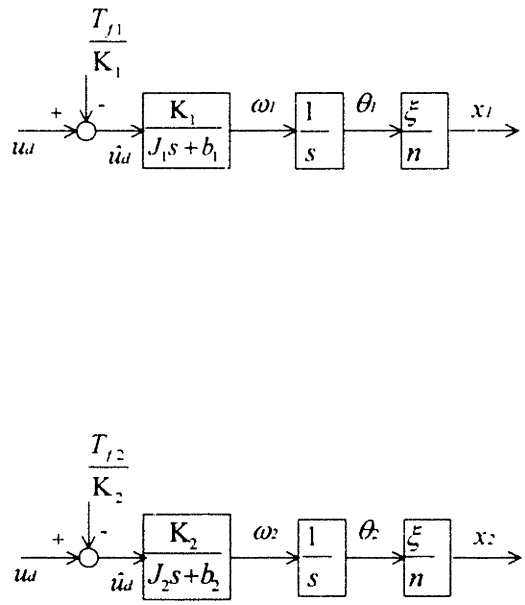


Figure 4.2 Block Diagram Model of Each Gantry X Axis

2. Controller Design

Synchronized motion control of multiple axes has been an interesting problem in the industry. Paper, sheet metal, textile, and other industries which manufacture products from a flexible web of material have been benefited from the motion synchronization of multiple axes. In a typical manufacturing process, a web of material is stretched over a multitude of rollers with its position, velocity and tension controlled by them. In early days, the rollers were connected by mechanical devices- shafts, gears, chains, etc., in part to transmit the

mechanical energy and in part to achieve the synchronization. With the advance of the electronics and the advent of the microprocessors, the mechanical coupling was replaced by electric motors and drives, individually placed at each roller. Since then, the motion synchronization has been a challenging subject.

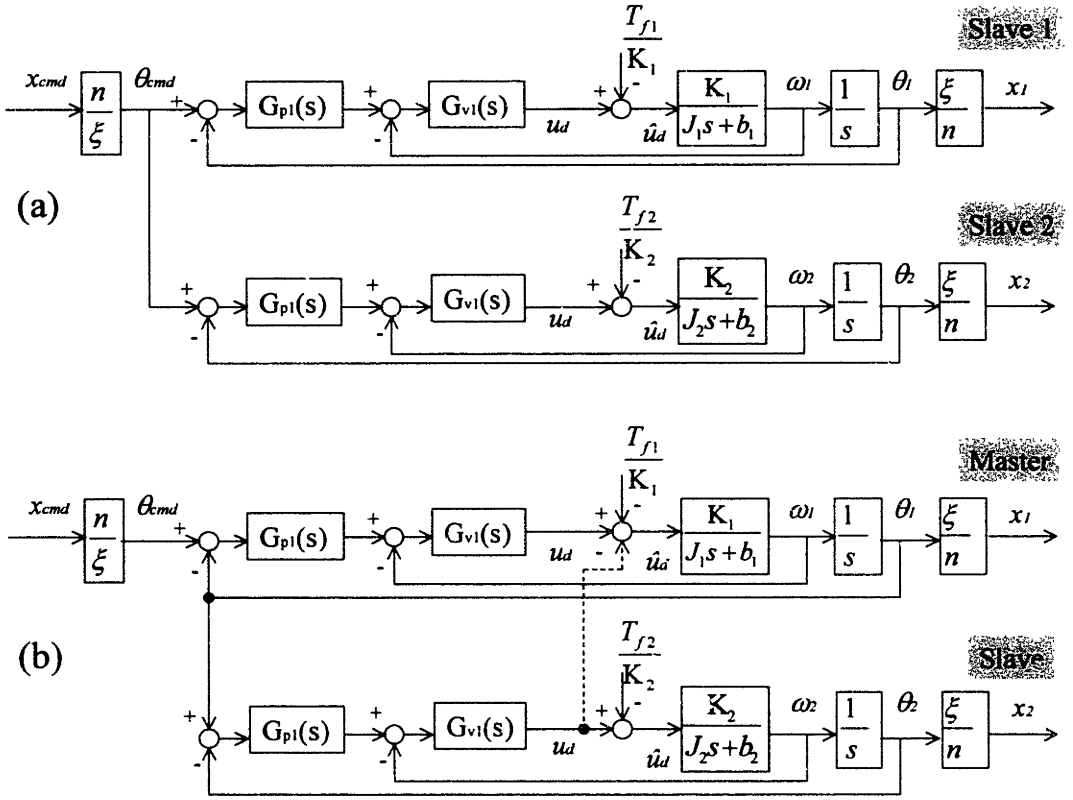


Figure 4.3 Block Diagram of Synchronization Topologies Applied to Gantry Twin Axes with Velocity Minor and Position Major Control Loop
(a) Synchronized Master Reference Topology
(b) Master-Slave Topology (the dotted line represents the relative stiffness feedback)

In general, there are two synchronization techniques widely used in the industry: Synchronized Master Reference (SMR) topology and Master-Slave topology[7]. SMR uses a synchronized master command generator to create and transmit a motion command to multiple slave axes simultaneously. Error tracking is left to each slave axis. In construction, each axis will perfectly be synchronized with the reference, but the relative errors between the axes are not addressed. SMR works best when the slave axes have an identical dynamics.

Master-Slave topology is typically used where minimizing relative errors between the axes are important. A slow moving, some times open-loop controlled axis is used as a master motion generator. A fast moving, high performance slave axis tracks the motion of the master axis. The relative error between the master and the slave reduces significantly. But there is a large amount of tracking error and a significant delay between the input command and the slave output, because of the master who works as a 'middleman.'

An improvement from the Master-Slave topology is the relative-stiffness or the virtual shaft topology[8], in which a virtual torque feedback is connected from a slave to the master. The output from a slave controller, which is proportional to the torque being generated at the slave axis, is fed to the master axis. If the slave axis generates more output than the master, the master will feel the 'stiffness' of the slave axis and reduce its output. The reduced master output is then fed to the slave as an input, which eventually reduces its output. It is analogous for two axes being connected by a 'virtual' shaft with a certain torsional stiffness. However, there still are delays for the master to feel and adjust its output, for the output to be transmitted to the slave, and for the slave to adjust its output based on the input transmitted.

Figure 4.3 shows the block diagram of each technique applied to the gantry axes with a velocity minor and a position major control loop in place.

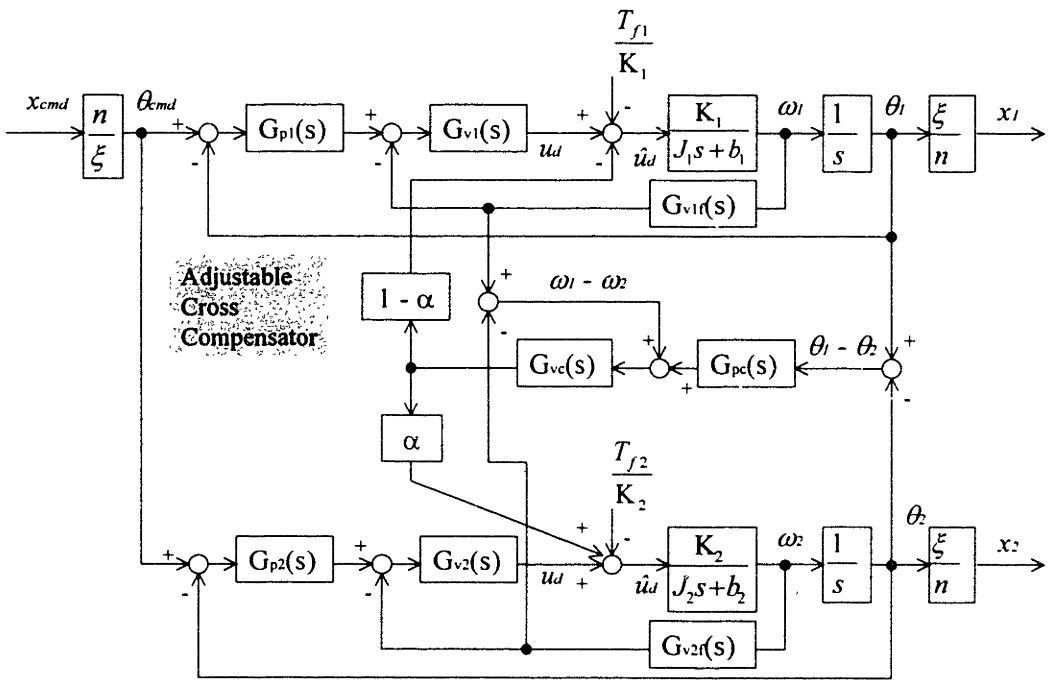


Figure 4.4 Block Diagram of Adjustable Cross Compensation for Motion Synchronization Applied to Gantry Twin Axes

Instead of choosing between the slave/slave or the master/slave topology, we propose a new scheme: the Adjustable Cross Compensation for dual axes synchronization, or the master/master topology. It is the combination of the command referencing in SMR topology and the reciprocal output referencing in the extended Master-Slave topology. A servo command is referenced by both axes, and at the same time the relative output between two axes is amplified by the cross compensator (controller) and then directed to each axis with adjustable gains. Figure 4.4 shows the block diagram of the Adjustable Cross Compensation scheme applied to the gantry twin axes.

The command reference θ_{cmd} is directed to the both axes, and each position and velocity controllers do a routine job to drive itself toward the reference. The position and the velocity outputs are also sampled by the compensator and the relative errors are fed to its own position and velocity controllers. The output from the compensator is adjusted by the gain α ($0 \leq \alpha \leq 1$) and added to the reference controller output. The sum is transmitted to the plant. By varying the gain α, we can configure the system from the master/slave to the master/master. If we set $\alpha = 1$, all the compensation is directed to the axis 2 resulting in Axis 1- Master/Axis 2- Slave configuration. If $\alpha = 0.5$, the equal amount of compensation is directed to each axis, resulting in Master/Master set up. If $\alpha = 0$, the axis 2 becomes the master.

The Adjustable Cross Compensation minimizes the relative error by the cross compensation, an advantage over the SMR topology, and reduces the tracking delay by the direct referencing of each output, an advantage over the Master/Slave topology. If the axis 1 experiences a disturbance, at the next sampling event the axis 2 will respond by reducing its controller output while the axis 1 will increase its output to overcome the disturbance. The cross compensation reduces the relative errors during the transient period or under a disturbance, where command reference eventually drives the axes to the desired final state.

The cross compensation is ideal for the twin axes system where the dynamics are the same. But the compensation technique still can do a good job where the dynamics are not equal by adjusting the gain α. If one axis has a faster dynamics than the other, the gain is increased for the axis so that the axis can respond more readily to the change. The other axis, on the other hand, doesn't see much change because of the muted gain and simply follows the command reference.

The Adjustable Cross Compensation technique can be extended for a system having more than two axes. Two options may be possible: centralized compensator with a weighted error; distributed compensators with a single error for each compensator. The centralized compensator computes the compensation based on the weighted error. Suppose we have a three axes system. The error can be given by the following formula.

$$e = a\theta_1 + b\theta_2 + c\theta_3$$

a, b, and c are arbitrary constants tunable based on the system dynamics. We may choose (1, -0.5, -0.5) for example.

The distributed compensator scheme has a compensator for every possible axes pair. For three axes, we need three compensators for three relative errors.

$$e_1 = \theta_1 - \theta_2$$
$$e_2 = \theta_2 - \theta_3$$
$$e_3 = \theta_3 - \theta_1$$

For a multiple axes system, the number of the distributed compensator can grow quickly. For a system with more than five or six axes, the central compensation with a weighted error will be more suitable.

The performance target or specification needs to be set up before a controller can be designed. Although many indices can be used, we employ the following three.

♦ Rise time (or bandwidth): A measure of how quickly the system responds to a change. The rise time is roughly proportional to the system bandwidth.
♦ Maximum overshoot: A percentage deviation of the motion output from the reference input during the transient period. It can be conceived as a transient tracking error.
♦ Steady state error: The deviation of the motion output from the reference. If the reference is the command input, it is the tracking error. If the reference is the other axis, it is the relative error.

Ideally, it is desirable to have an infinite bandwidth, so that the system has an almost zero rise time. In reality, the drive (amplifier) quickly saturates and can not provide an infinite output required for the infinite bandwidth. Instead of shooting a fixed number of bandwidth for the gantry controller, we will design the controller which achieves the maximum bandwidth without saturating the drives. The overshoot needs to be minimized as small as possible. The overshoot is particularly problematic when the object is traveling to its maximum and there exist a barrier at the end. Even though the command is less than the travel maximum, if the overshoot is large enough, the object will collide to the barrier during the transient period. We will minimize the overshoot through the input shaping and other available techniques.

The steady state error needs to be zero. But in a real system, there can exist an offset. In a digital servo system, the positioning accuracy is quantized by the resolution of the encoder. The gantry system has the 4000 counts/rev quadrature counter for each axis. The position quantum Δ is given by the following formula.

$$\Delta = \delta \cdot \xi / n$$

δ: angular position quantum, $2\pi/4000$ rad
ξ: ballscrew pitch, 4.04×10^{-3} m/rad
n: gear ratio, 5
Δ: position quantum computed, 1.27 μm

Thus, it is impossible to control the position with an accuracy less than Δ. In reality, we set the target to control the position within a fewΔ. We set the following performance target for the gantry controller design.

1. Maximum bandwidth without amplifier saturation
2. Maximum overshoot $<$ 0.5 %
3. Steady state error (tracking and relative) $<$ ±5 μm
4. Relative error during transient period $<$ ±50 μm

We first design the controllers for the velocity minor loop and the position major loop and move on to the design of the compensator. A filter $G_{vf}(s)$ can be placed on the velocity feedback line. The velocity filter is especially useful in a sampled system, where the sampling and the difference approximation of differentiation cause a noise amplification and a velocity quantization. However for the design in the continuous domain, we set $G_{vf}(s)$ is 1 for each velocity feedback loop.

Although many different types of controllers can be designed for each loop, we design proportional controllers for each loop. We set $G_p(s) = k_p$ and $G_v(s) = k_v$ for each axis. The proportional controller on the velocity loop can be viewed as a type of differential controller from the position loop. We will add an integrator on the position loop during the digital implementation.

A velocity minor loop is typically designed to have a bandwidth 3 to 7 times greater than that of the position major loop. We propose 90 rad/sec for the velocity loop and 30 rad/sec for the position loop. The crossover frequency of a loop transfer function is a good approximation of the closed loop transfer function bandwidth. Thus we design for the crossover frequencies.

The loop transfer function of the velocity minor loop $G_{vLTF}(s)$ can be obtained from the block diagram (Figure 4.4).

$$G_{vLTF}(s) = k_v \frac{K}{Js + b}$$

K, J, and b are know constants from Table 4.1. From the condition that $|G_{vLTF}(\omega_{cr}j)| = 1$ at $\omega_{cr} = 90$ rad/sec, we obtain $k_v = 218$. The whole velocity loop can be reduced to a transfer function $G_{vminor}(s)$.

$$G_{vminor}(s) = \frac{k_v \cdot \frac{K}{b}}{\frac{J}{b}s + 1 + k_v \cdot \frac{K}{b}}$$

The loop transfer function of the position major loop $G_{pLTF}(s)$ is given by the following formula.

$$G_{pLTF}(s) = k_p \cdot G_{vminor}(s) \cdot 1/s$$

With $\omega_{cr} = 30$ rad/sec and $|G_{pLTF}(\omega_{cr}j)| = 1$, we get $k_p = 34.2$.

The output of the controller begins to saturate the amplifier at 32768 (2^{15}). The cascaded position and velocity controller ($k_p \cdot k_v$) will saturate at approximately 5 rad step input command, which corresponds to 4 mm of linear step input. Due to the inputshaping the controller will not see this much of step height. With 1 msec of sampling interval, this corresponds to 4 m/sec of input command slope. The machine will typically run with 0.07 m/sec and with an approx maximum 0.3 m/sec, which is at least one order of magnitude less than the saturation velocity. The design is acceptable. We set

$$k_{v1} = k_{v2} = k_v$$
$$k_{p1} = k_{p2} = k_p$$

The compensator dynamics is a linear superposition of each axis dynamics ($\theta_1 \cdot \theta_2$). Thus it is assumed that the compensator behavior is similar to the axis dynamics. Thus instead of designing whole new controllers for the compensator, we simply copy the axis controllers with a scaling factor. We set

$G_{pc}(s) = k_{pc} = \beta \cdot k_p$
$G_{vc}(s) = k_{vc} = \beta \cdot k_v$

We choose $\beta = 1.5$. With the selection of the adjustable gain $\alpha = 0.5$, the compensator has a slightly less dynamics than each axis controller ($\alpha \cdot \beta < 1$). The overall gantry X controller composed of each axis controller and the compensator is expected to meet the performance target balancing the command reference tracking and the relative error compensation.

3. Digital Implementation

In old days, the controllers were implemented by hard wired electrical circuits. Operational amplifiers and supporting RC networks often constructed first or second order controllers and filters. However, nowadays, with the introduction of cheap microprocessors and the advance of ASICs, the controllers are implemented in the digital domain as software programs which run on powerful computers with appropriate peripherals (A/D, D/A converter, etc.). Digital controllers provide more versatility and flexibility compared to the analog controllers. But there are special concerns because of its discrete nature in dealing with signals and representing them as quantized values in software. In this section, we implement the digital gantry X controller designed in the continuous domain in the previous section. We will take a look at a few issues involved in implementing the controller in the discrete domain.

Without a tachometer feedback, the velocity of a motor can be computed by differentiating the position signal. Any of the difference techniques mentioned in Chapter 2 can be used in the discrete domain to emulate the differentiation in the continuous domain. Concerns are that the differentiation (difference) usually amplifies the imposed noise signal and that the velocity is quantized. In the previous section, we saw the position quantum Δ of the gantry X controller is 1.27 μm. If we are sampling at 1 KHz, the velocity quantum λ is 1.27 mm/sec (1.27 μm/1msec). A faster sampling rate will increase the magnitude of λ. The velocity quantum is especially problematic, when we want to set a low reference velocity. Suppose we give 1 mm/sec of reference velocity, the velocity loop controller output will swing from zero to something as the computed velocity ripples between zero and 1.27 mm/sec.

A digital filter is required to reduce the noise and to reduce the effect of the velocity quantum. We install a 3-point moving average filter on the velocity loop.

$$\hat{\omega}(kh) = \frac{1}{3}\{\omega(kh) + \omega[(k-1)h] + \omega[(k-2)h]\}$$

Due to the averaging, the velocity quantum reduces to $\lambda/3 = 0.423$ mm/sec. The averaging filter will inherently cause a delay in signal (velocity) transmission. With the 3-point averaging, an average delay of 1.5 sampling interval is expected. With 1 KHz sampling rate, this corresponds to 1.5 msec. The decision concerning the amount of averaging is a trade off between filtering (smoothness in signal) and delay (responsiveness to signal). Because it is

rare to have a low velocity referencing, 0.423 mm/sec of velocity quantum will be smooth enough for the gantry X controller, and 1.5 msec of delay is expected to be tolerable.

Inputshaping was introduced in Chapter 2 as means of preventing sudden changes in the input and the resulting controller and plant output, and providing a way to indirectly control the velocity. We add an input shaper after the input command conversion ($x_{cmd} \rightarrow \theta_{cmd}$). Although many different input profiles are possible, we will use the ramp plus sinusoid inputshaper for the gantry X controller.

An integrator is often added to the position loop to reduce the steady state error. But the integrator often contributes to the large amount of overshoot during the transient period, because it adds up the error signal and gives an inertial effect to the system behavior. Thus it makes sense to design an integrator which engages during the steady state period only, because its main functionality is to minimize the settling error. It is an example of digital controller's flexibility, because you simply enable and disable the integrator code in the program based on time. In analog controllers, you may have to install a relay to connect and disconnect the integrator signal. We program the 'time differential integrator' so that it engages after the inputshaper starts to give a constant command input, i.e. at the very end of the sine part of the input form.

Another concern in the digital integrator is the windup. If there is an error signal for a prolonged time because of a disturbance in the plant, the integrator simply adds up the error and accumulates a large amount of controller output. Once the disturbance disappears, the integrator output is released, often more than enough to bring the plant to the command reference. As a result it will make large swings back and forth until it finally settles on the set point. In the analog controllers, the controllers also saturate beyond certain points, thus the amount of 'windup' is typically bound by the hardware limit. In software, there virtually is no limit in the controller output. A four byte float variable can hold a value up to 3.40×10^{38} and a four byte integer up to 2 billion, whatever their units are. Thus it is quite necessary to limit the integrator output so that remains reasonable and does not saturate the amplifier. This limitation scheme is called an 'anti-windup.'

We set ±0.07 rad·sec of anti-windup limit with 200000 of the integrator gain (k_i) for the fast error correction purpose. Then the integrator output will be bound at 14000, which is slightly less than the half of the maximum acceptable controller output (32768). The integrator is designed not to overwhelm other controllers. The time differential anti-wound integrator is solely designed for the steady state error correction.

Another practical concern for the digital integrator comes from the quantization in the position signal. Suppose the command reference (x_{cmd}) of 1 μm is delivered to the system. Because of the position quantization (Δ = 1.27 μm), the position feedback will report either x = 0 μm or 1.27 μm. Even though the position can not settle on 1 μm, the integrator will keep giving an output to the system by integrating the error signal.

$$\int (x_{cmd} - x) dt$$

As a result, the output will swing between 0 and 1.27 μm and never settle on 1 μm with frequency depending on the integrator gain. In worst case, it can lead to a vibration or an instability when linked to an unknown or minor dynamics.

The singing of the integrator due to the position quantum can be prevented by equally quantizing the command input. Also it is desirable to perform the integration at the

encoder counter level (integer), because the conversion to position (float) undermines the precision of the variable. The position x is represented by the original encoder counter c_{enc}. The command x_{cmd} is converted and rounded to a corresponding counter variable c_{cmd} which is an integer. The integrator does its job at the counter level.

$$\int (c_{cmd} - c_{enc})dt$$

The output is multiplied by an appropriate gain to convert to [rad·sec] dimension.

As in the anti-windup of the integrator, the resulting output of the controller needs to be limited. Otherwise the combination of the integrator output, the position and the velocity loop controller output, and the compensator output can exceed the DAC output limit (32768) and saturate the card and the mating drive. We add an output clipping with the limit of ±32768 after the controller. Figure 4.5 shows the block diagram of the gantry X controller with added features for the digital implementation.

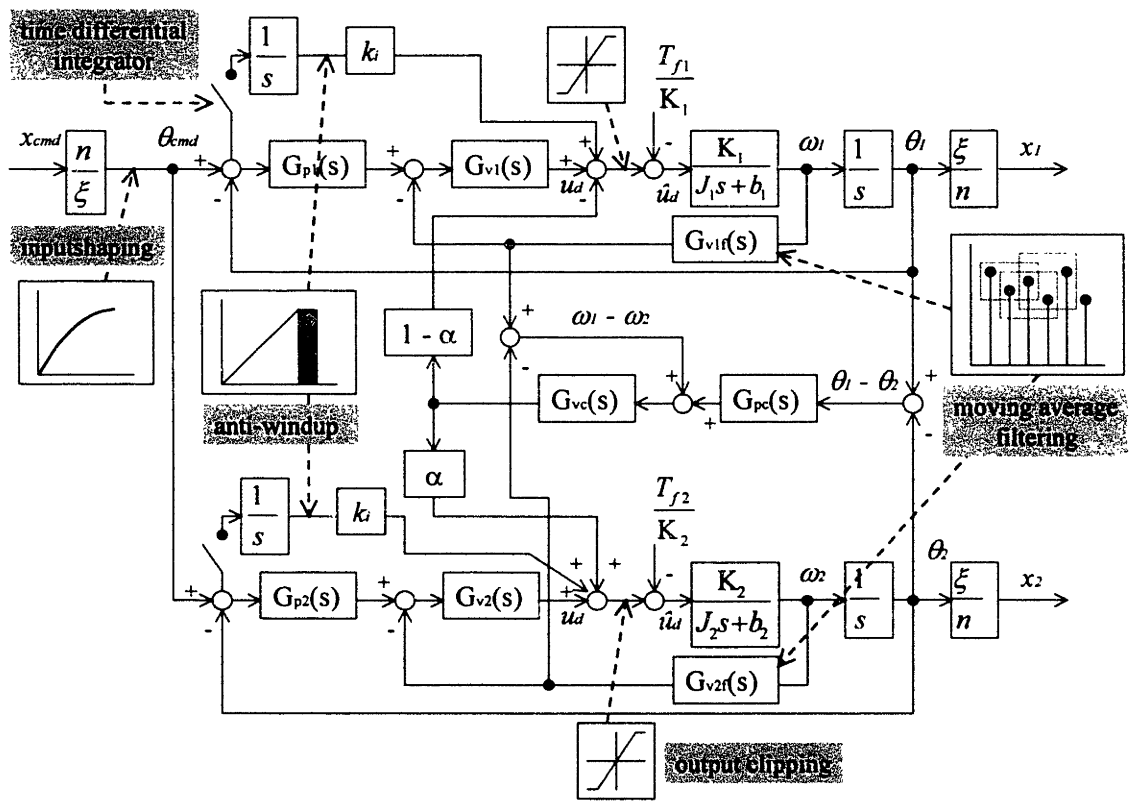


Figure 4.5 Block Diagram of Gantry X Controller Implemented in Discrete Domain

4. Performance Evaluation

Having written the program for the gantry X control with the parameters and the methods designed from the sections 2 and 3, we now have to evaluate the performance of the controller. The gantry typically runs at 70 mm/sec of reference velocity. The transient

response is expected to depend on how fast it is ramped up, whereas the steady state response is expected to be independent of the reference velocity. We select two reference velocity conditions: 50 mm/sec and 100 mm/sec. We will compare the transient responses in two cases and verify that both the transient responses and the steady state responses satisfy the performance target set up at the section 2.

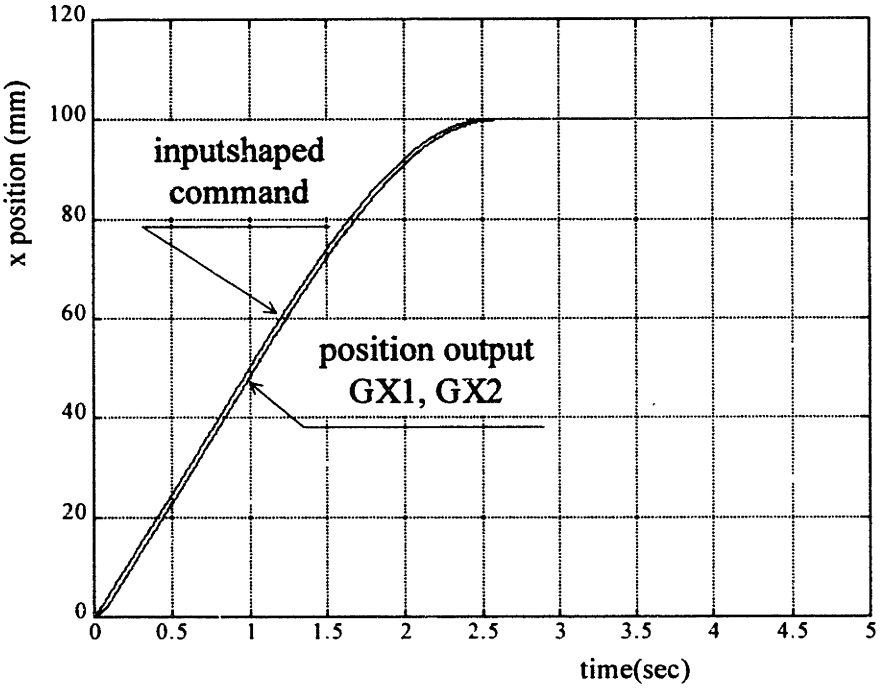


Figure 4.6 Inputshaped Command and Each Axis Position Output of Gantry X Axes with 100 mm Step Command and 50 mm/sec Reference Velocity

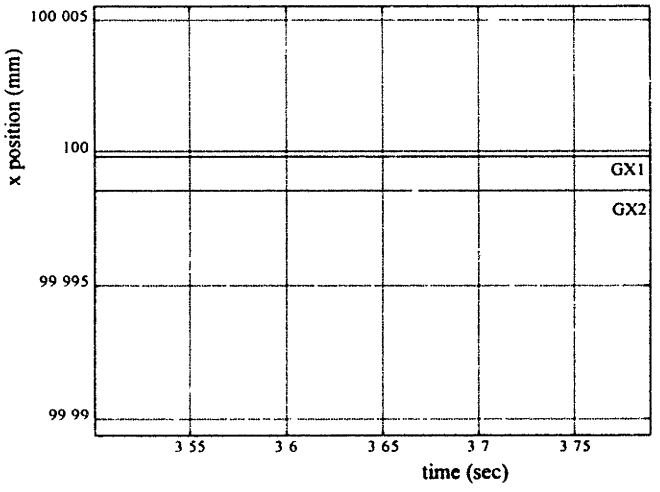


Figure 4.7 Steady State Response of Gantry X Axes with 100 mm Step Command and 50 mm/sec Reference Velocity

Figure 4.6 shows both the inputshaped command to the controller and the each axis position output at 100 mm step command with 50 mm/sec reference velocity. The ramp plus sinusoid input profile is used. It is the ramp profile up to 50 mm (1 sec) and then the sine profile follows. The input profiler reaches the steady state command of 100 mm after approximately 2.6 sec ((1 + π/2)·1 sec). We can see from the graph that the position outputs follow the command profile well and that there virtually is no overshoot because of the smooth sine profiling at the end of the transient period.

Figure 4.7 shows the exploded view of Figure 4.6 around t = 3.6 sec to examine the steady state error. We can see that GX1 (axis 1) is at 100 mm and GX2 (axis 2) is off only by 1.27 μm, which is the position quantum of the system.

Figure 4.6 also suggests that the relative error between GX1 and GX2 will be quite small both in transition and steady state. Figure 4.8 displays the relative error between the two axes. The relative error is bounded by 35 μm during transition, and once the integrator is engaged after t = 2.6 sec the error is reduced significantly and settles within a few position quanta. The error graph is biased on the positive side, which means the axis 1 almost always lead the axis 2. To balance the dynamics, the compensator gain to the axis 1 may be increased slightly, although acceptable as it is. The graph shows a little singing after t = 4.5 sec. But it is within two position quantum, and disappeared as time went by without creating any vibration problem in the physical observation.

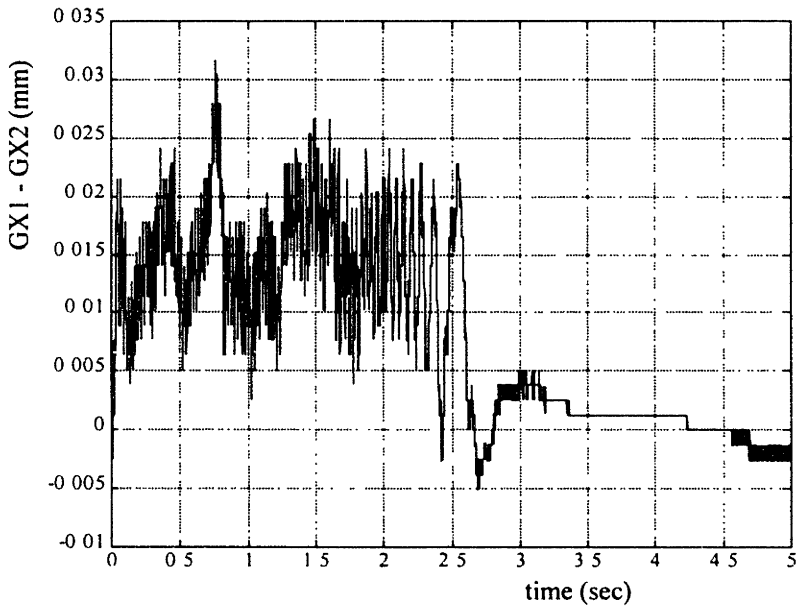


Figure 4.8 Relative Error between Two Axes of Gantry with 100 mm Step Command and 50 mm/sec Reference Velocity

Figure 4.9 shows the trajectory when 200 mm step command with 100 mm/sec reference velocity is applied to the gantry X controller. Although the reference velocity is increased by twofold, the output response is almost identical to the case of 50 mm/sec reference velocity.

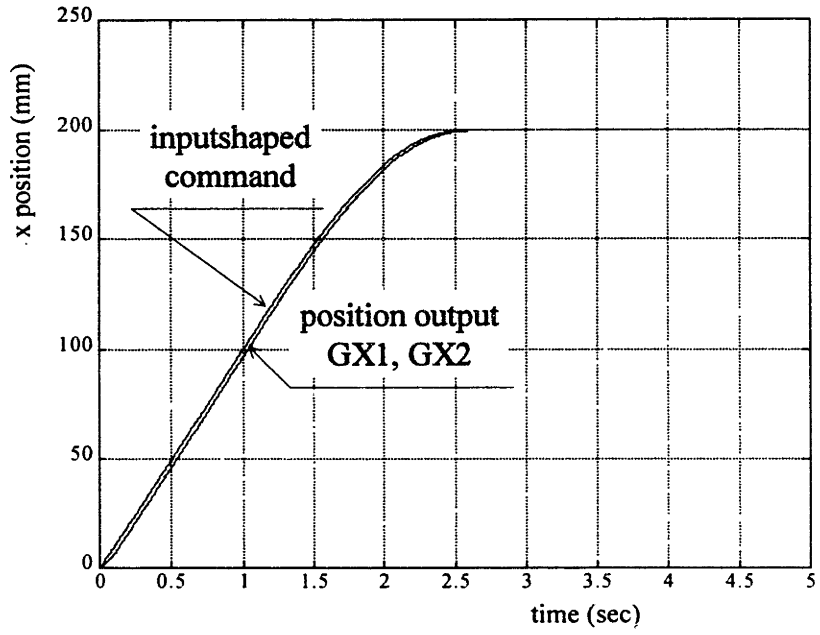


Figure 4.9 Inputshaped Command and Each Axis Position Output of Gantry X Axes with 200 mm Step Command and 100 mm/sec Reference Velocity

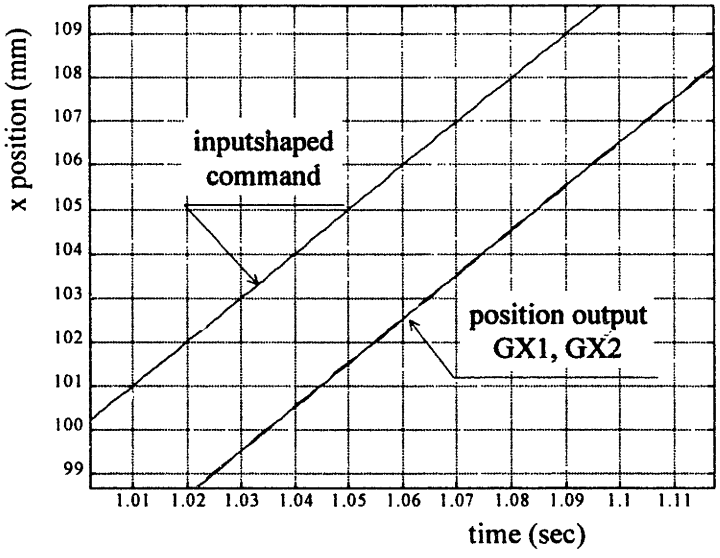


Figure 4.10 Exploded View of Command and Output Trajectory from Figure 4.9 (200 mm Step Command and 100 mm/sec Reference Velocity)

Figure 4.10 is the exploded view of Figure 4.9 around x_{cmd} = 105 mm. The position outputs lag the command by about 3.5 mm. In time scale, the lagging is approximately 0.034 sec. Interestingly, the inverse of the time lag gives 1/0.034 = 29.4 rad/sec, which is very

close to the position major loop bandwidth of the gantry X controller. This can be an indication that the modeling is very close to the real physical nature of the system and that the controller design is reasonably performed.

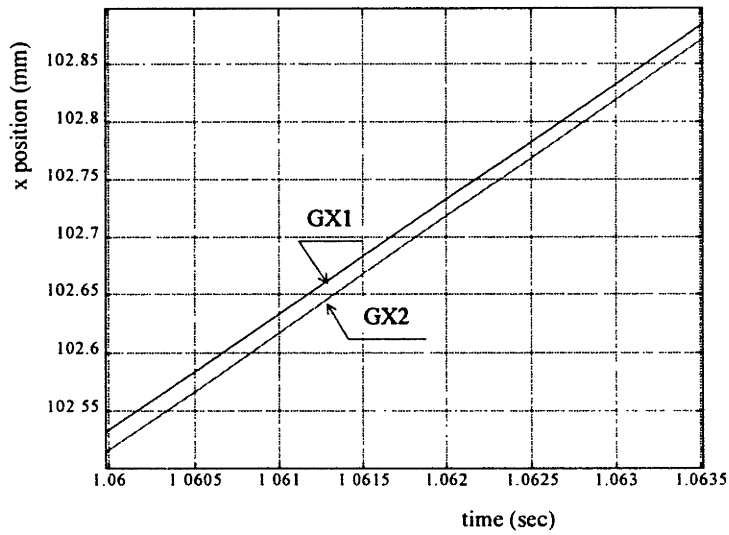


Figure 4.11 Exploded View of Gantry X Position Trajectory from Figure 4.10 (200 mm Step Command and 100 mm/sec Reference Velocity)

Figure 4.11 is again the exploded view of Figure 4.10 around t = 1.062 sec to see the position outputs in detail. The graph reveals that the axis 1 is leading the axis 2 during the transient period by approximately 20 μm and that this phenomenon seems to be well sustained throughout the period. The relative error of 20 μm is within the performance target.

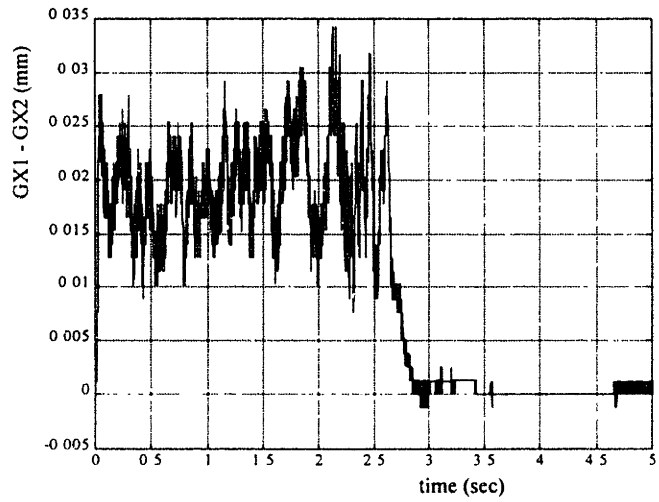


Figure 4.12 Relative Error between Two Axes of the Gantry with 200 mm Step Command and 100 mm/sec Reference Velocity

The time evolution of the relative error is shown in Figure 4.12. The overall trend is the same as in the 50 mm/sec reference velocity case. The error seems to be a bit increased but still bound by 0.035 mm and falls within a few position quanta once the integrator error correction loop is engaged after t = 2.6 sec. Hence we verify again that the relative error between the two gantry axes satisfies the performance target (±50 μm for transient period, ±5 μm at steady state)

The gantry X controller has been designed in this chapter through system modeling, individual axis controller design, and adjustable cross compensator design for the twin axes synchronization. The controller is implemented in the discrete domain with augmentative digital techniques for the performance enhancement. From the several trajectory graphs presented in this section, we confirm that the controller meets the performance targets- Maximum bandwidth, Virtually zero overshoot, Steady state error less than ±5 μm, and Transitional relative error less than ±50 μm.

The design of other position servos can be done following the similar fashion as in the gantry X controller design, except the adjustable cross compensator. Table 4.2 lists a few relevant numerical values for the position servo controllers. GX represents the gantry X, GZ the gantry Z, CX the conditioner X, and WA the wafer aligner angle.

Table 4.2 Numerical Values of Position Servo Controllers

Parameter(Symbol)	Unit	GX	GZ	CX	WA
Inertia(J)	$Kg \cdot m^2$	5.348×10^{-4}	3.13×10^{-3}	1.17×10^{-4}	5.20×10^{-6}
Viscous Damping(b)	N·m/(rad/sec)	4.51×10^{-3}	3.98×10^{-3}	8.92×10^{-4}	2.63×10^{-5}
Friction Torque(T_f)	N·m	5.61×10^{-1}	6.10×10^{-1}	1.74×10^{-1}	2.44×10^{-2}
Torque Constant(K)	N·m	2.23×10^{-4}	2.39×10^{-4}	1.20×10^{-4}	9.77×10^{-6}
Position Gain(k_p)		34.2	15.5	44.6	81.3
Velocity Gain(k_v)		218	981	116.7	152.1

Chapter 5. System Integration

All the components necessary to constitute the CMP α machine control system have been designed in the previous chapters- the Machine-level control system in Chapter 2, the Process-level control system in Chapter 3, and servo controllers in Chapter 4. With all the constituents designed, we are ready to implement them. This chapter deals with the system integration and the embodiment of the control system. We will first take a look at how the system is wired up, which is the physical foundation of the control system. Then we will see a few snapshots of the machine with the control system in action.

1. Hardware Interface

In the CMP α machine, all the required powers are transmitted to the machine by the means of electricity. Every information in the control system is communicated as a form of electrical signal. Thus in the viewpoint of the control system development, hardware interfacing means electrical wiring of the system components. Power wiring is often performed first to draw the electrical energy and distribute it to the individual components. Signal (logic) wiring then follows to connect each component to the control system.

The control system has a separate cabinet apart from the machine to house ADwin, servo amplifiers, DC supplies, filters, switch panels, etc. The operator console (touch screen) and the host PC are located outside the cabinet, close to the machine.

The cabinet has a contact box which houses heavy duty contactors to relay the AC input to the cabinet and subsequently to the machine. The contactors are controlled by the master on and off switches in the front panel of the cabinet, and also by the emergency off (EMO) switches located on the machine.

The electrical power relayed by the contact box is distributed to each components at the AC switch panel located at the lower front of the cabinet. The panel has individual circuit breakers for each component (amplifier, pump, etc.). The circuit breaker protects the corresponding component from the overloading and also works as a switch to turn on and off it individually.

From the AC switch panel, the AC power is delivered to motor amplifiers, pumps, DC supplies, etc. The amplifiers generate DC bus voltages to drive the brushless DC servo motors. All the amplifiers are pulse-width modulation (PWM) type drives. The DC voltages generate from DC supplies are fed to pressure regulators, valves, indicators, sensors, filters, etc. which run at DC voltages.

All these components are connected to the ADwin via a network of signal wiring. The host PC and the ADwin exchange information by serial communication. Figure 5.1 shows the general overview of the control system interface.

One concern in the wiring of a measurement and control system is to provide a strong ground immune to any noise or fluctuation. Because the noise from the PWM drives is particularly problematic, the good grounding system is a must. The cabinet has a ground bus made of ½" × ¾" copper bar. The ground bus is connected to the earth line of the AC input on one end, and various components reference the bus on the other end. As shown in Figure 5.1, the signal ground of ADwin and the DC ground of the DC supplies are separate from the AC ground. But they are connected to the same ground bus at the end. The separation of the ground and connection at the very front of a sink (earth) is required for

noise immunity. Any noise on the AC ground line will find a low impedance path to the eatrh instead of propagating through the signal ground line. The Ac ground and the frame ground are virtually the same.

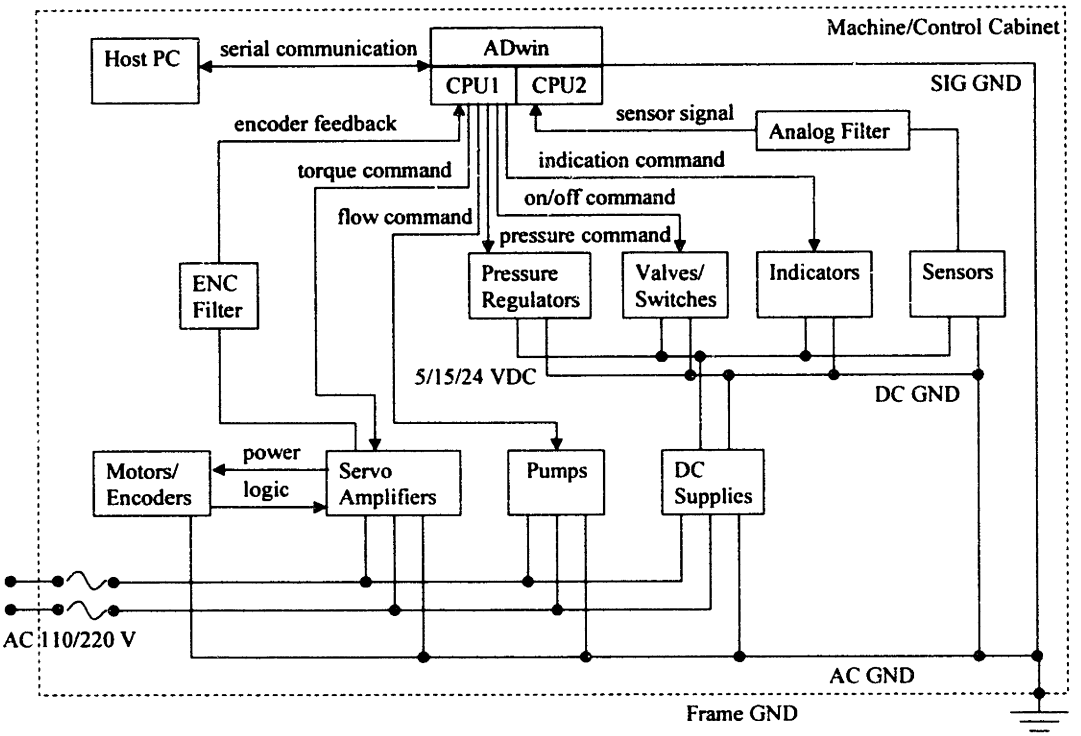


Figure 5.1 Overview of CMP α Machine Control System Interface

The contact box, the on/off switch panel, the EMO switches, and the AC switch panel forms the electrical power distribution system. Figure 5.2 shows the schematic of the distribution system. The first three also composes the safety system, which disconnects the electrical power to the load components once any of the EMO switch is pushed. Note one 110 VAC line of 20 A rate bypasses the contact box. This line is used as an uninterruptible power source for the logic components of the system- the host PC and the ADwin.

Machine tools typically have a single 480 VAC of power line and have various types of transformers to step down to whatever voltages desired. However, the machine requires 3 phase 208 VAC to power up the platen amplifiers and various types of single phase 110/220 VAC lines are readily available at the lab where the machine is installed. Based on the circumstance, the contact box is designed to accommodate various types of AC line inputs and distribute them accordingly.

The contact box accepts four line inputs- one line of 3 phase 208 VAC, 100 A; one line of 1 phase 220 VAC, 20 A; two lines of 1 phase 110 VAC, 30 A. 208 VAC is used to power up the two platen amplifiers. Each can consume up to 50 A. Three amplifiers for the gantry motion, GX1 (gantry X1), GX2 (gantry X2) and GZ (gantry Z), can accept either 110 VAC or 220 VAC. Because they can consume a great amount of current, it is preferable to connect them to the 220 VAC line (higher voltage can transmit the same amount of power

with lower current). One of the 110 VAC lines is used by the rest of amplifiers- CX (conditioner X), CR (conditioner rotation), WR(wafer carrier rotation) and WA (wafer aligner). The other 110 VAC line supplies power to the rest of control system components- slurry pumps, DC supplies, fans, etc.

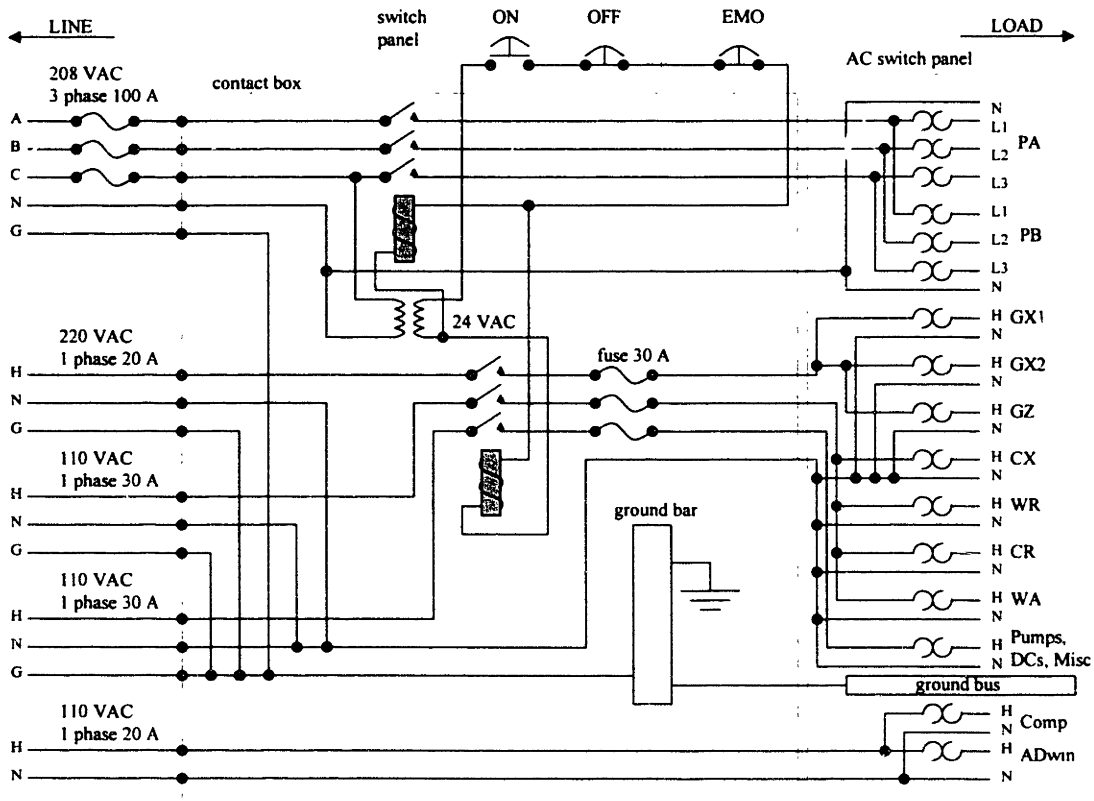


Figure 5.2 Schematic of Electrical Power Distribution of CMP α Machine Control System

Active lines are connected to a set of relay contacts, which are powered by the internally transformed 24 VAC and controlled by the On/Off/EMO switches. Figure 5.2 also shows how ON and OFF switches activate and deactivate the solenoids which in turn connect and disconnect the contact relays. Pressing ON switch (normally open) closes the 24 VAC circuit and the solenoids in turn close the contacts supplying power to the AC switch panel. Pressing either OFF switch or EMO switch (both normally closed) opens the circuit and solenoids breaks the contact cutting the power supply to the AC switch panel. In the actual system, more latching relays are involved in conjunction with the switches. But the above schematic is good enough to give an idea how the switch panel works.

Passive (neutral, ground) lines simply bypass the contact box (electrically) and shows up at the AC switch panel. The single phase neutral lines are tied together, because they are supposed to have the same potential. The neutral line is then distributed again at the switch panel. The ground lines are also tied together at the internal ground bar, which is connected to the ground bus of the control cabinet via a heavy duty wire (6 AWG). The ground lines are connected to the earth at the line side.

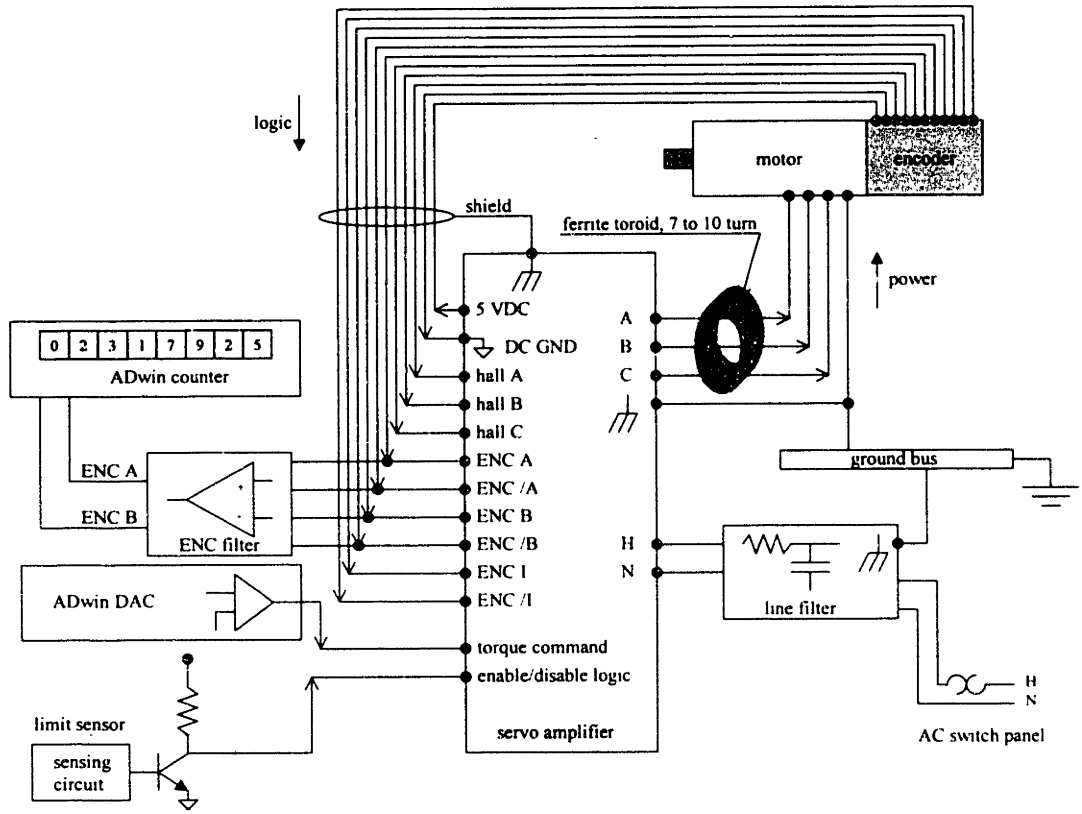


Figure 5.3 Schematic of Brushless DC Motor/Encoder/Amplifier Wiring

The electricity is then delivered to each load component from the AC switch panel. Among them, motor drives consume most of the electrical energy. The servo amplifier then generates a DC bus voltage for its mating motor and DC voltages to power up its own circuit and the mating encoder. The motor/encoder/amplifier wiring is a mix of power and logic wiring. Figure 5.3 shows the example of a brushless DC servo motor wiring.

A line filter is placed between the AC switch panel and the amplifier to prevent the PWM noise to propagate through the power line. Unfiltered, the high energy noise will affect other components, such as DC supplies, connected via the power line. The filter dumps the attenuated noise signal to its ground line. The ground line is connected to the ground bus, which is also connected to the frame grounds of the motor and the amplifier.

A PWM amplifier creates a DC bus voltage to drive the mating brushless DC servo motor. The bus voltage ranges from 40 to 350 VDC. The amplifier drives the motor by sending 3 phases of pulse train usually at 20 KHz. The height of a pulse is its DC bus voltage. The width of a pulse is determined by the input torque command. The higher is the command, the larger is the width. The pulse-width modulation refers to this change in the pulse width to achieve the change in the torque output. The inductance of the motor coil then converts the DC voltage train to the armature current, which in turn generates the output torque in the presence of the magnetic field. For more detail about how brushless DC motors and PWM drives work, refer to Moreton[9].

The switching from the DC bus voltage to zero typically occurs within 50 nsec with 20 KHz of switching frequency. This switching can generate electromagnetic interference (EMI) noise with several mega hertz bandwidth. In addition to the shielding and grounding of wires to protect their signal from the attack of EMI noise, ferrite cores are often placed on the motor power line close to the amplifier. The power leads are turned 7 to 10 times along the toroid to create a small inductance. This inductance in conjunction with the motor lead resistance forms a type of LR filter, which is quite effective in suppressing the EMI noise.

Encoders are usually powered up by the onboard 5 VDC supply of amplifiers. The encoder then sends the hall effect signals for commutation, and the square wave encoder signals to the amplifier and to the counter board. Hall effect signals are square waves which are in phase with the rotor position. Based on the hall effect sequence (HA-HB-HC, HA-HC-HB, etc.), the amplifier decides which phase (A, B or C) of the armature coil needs to be energized. The amplifier energizes (commutates) each phase in synchronization with the hall effect feedback. Some amplifiers use the encoder signal to interpolate the hall effect signal and hence to perform a finer commutation. With the hall effect signal only, there are six commutation steps per one magnet pole pair in a motor. Each step is separated by 60°. With the encoder feedback, the amplifier can refine the step resolution, depending on the encoder resolution. Typical motors/encoders have four magnet pole pairs and 4000 counts/rev, which allows the amplifier to refine the commutation step resolution down to 0.36°. Compared to the six-step commutation, the refined one is called the 'sinusoidal commutation.'

Encoders give a complimentary signal (/A) to each (A) of its channel outputs. When the channel A is high, the channel /A is low, and vice versa. The complimentary signal is often used to filter out the noise signal imposed on the encoder signal. If A and /A has the same routing, they are supposed to have picked up the same amount of noise. By subtracting /A from A to obtain the difference between the two, we can eliminate the noise. Differential operational amplifiers with appropriate resistor and transistor networks are used for this purpose. In the control system wiring, the encoder filter built from the differential op amps is placed between the amplifier and the ADwin counter card. It accepts differential inputs and sends a single ended output for each channel.

Torque command is typically an analog voltage signal ranging from −10 V to 10 V. The ADwin DAC card is used to send the torque commands to the amplifiers. Servo amplifiers typically have an enable/disable logic input, which removes power stages from the drives. It may require logic high or low to disable the amplifier depending on the configuration. The enable line is typically connected to a limit sensor or a limit switch, which sends the appropriate active output once triggered.

Wiring other components is relatively easy and the whole control cabinet can be wired up without any great difficulty. Figure 5.4 shows the picture of the control cabinet completely assembled and wired. The AC power lines are from the lab ceiling and directed to the back side of the cabinet. Although not shown in the picture, the contact box is located on the back side. The contact box relays the AC power to the AC switch panel located at the lower front portion of the cabinet. The AC power is distributed to amplifiers and DC supplies located on the back panel The front panel houses the DC distribution panel which is the terminal of various DC supplies to ease the connection to the various DC components.

Figure 5.4 Photograph of Control Cabinet Assembled & Wired

The encoder filter is housed inside an aluminum alloy box and mounted on the 19" rack on top of the DC panel. Figure 5.5 shows the detailed view of the cabinet top portion, where the switch panel, the ADwin and some of the servo amplifiers are located. The switch panel has the ON switch and the OFF switch, which turns on or off the whole machine. Each switch has a light bulb inside as an status indicator. The ADwin is also mounted on the 19" rack and acts as a brain and nerve of the Machine-level control system. Every signal wire is linked to the ADwin. Processed outputs are sent to the machine via the output module of the ADwin, and processed signals are sent to the PC via the serial link.

The servo amplifiers are shown in detail in Figure 5.6. Seven (one hidden) amplifiers are located on the top portion of the back panel. The two heavy amplifiers for two platens are located at the bottom portion of the back panel, because of their weight. The figure shows the line filters placed on the AC lines. Toroids can be found along the power lines. Amplifiers typically have D25-Sub connectors for the logic interface. Mating D-Sub connectors and cables were fabricated to connect to the encoders and the ADwin.

Having wired up the control cabinet, we established the physical layer of the equipment control system. Now we are ready to implement the control system.

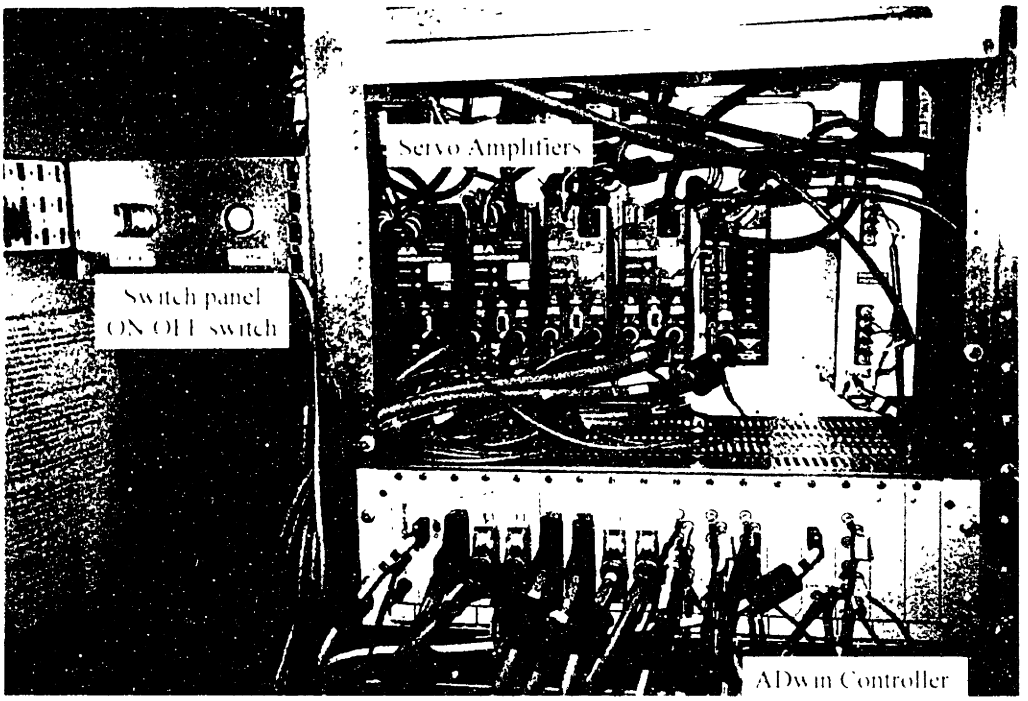


Figure 5.5 Photograph of Switch Panel & ADwin

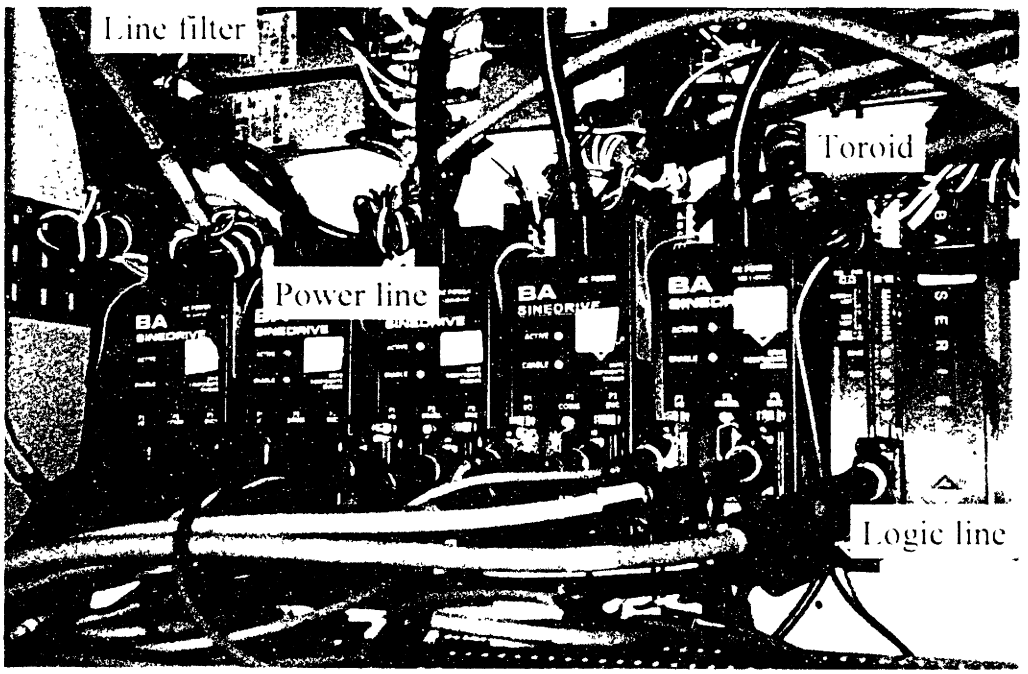


Figure 5.6 Photograph of Servo Amplifiers

2. System Implementation

Once the completion of wiring, the software program for each level of the control system has been written and tested. The Machine-level control system software is complete and fully functional. However, a few minor upgrades, such as incorporating more process measurement sensors, can be expected. Due to the time constraint, the Process-level software is still under development. The manual mode is in place. Thus moving the individual machine components and processing wafers in a primitive fashion are possible. To ease testing, a semiauto mode has been developed. It is based on the manual mode, but has a link to Step_Polish. It has the polish step editor. After a user edits its parameter and click OK button, it loads the parameter to the semiauto mode buffer. Once the click of START button, it performs the sequential stages of Step_Polish.

It is an automation in a very limited scale. Other steps, such as loading, conditioning, etc., are not included. And the step parameters are volatile. It does not have a capability to save in them as a file. With the design in Chapter 3, the development of recipe editor and auto mode is still in process.

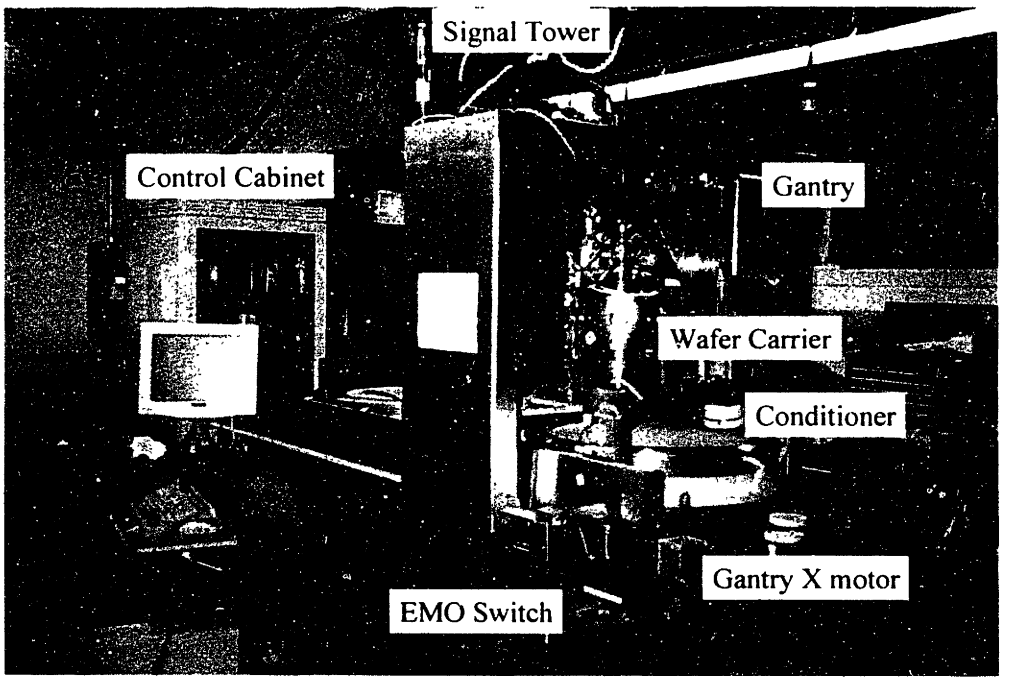


Figure 5.7 Photograph of CMP α Machine, oblique view

Figure 5.7 shows a photograph of the machine. The control cabinet is placed to a side of the machine with its front facing the machine. The picture shows one of the emergency off (EMO) switch located at the corner of the machine. The machine has a signal tower (Red/Yellow/Green/Blue) to indicate its status. Green means 'running,' yellow 'initializing' or 'warning,' red 'error,' and blue indicates the 'ready' status in auto mode. The gantry, which houses the wafer carrier, is shown with its X motor and ballscrew drive mechanism. The conditioner is moved next to the gantry.

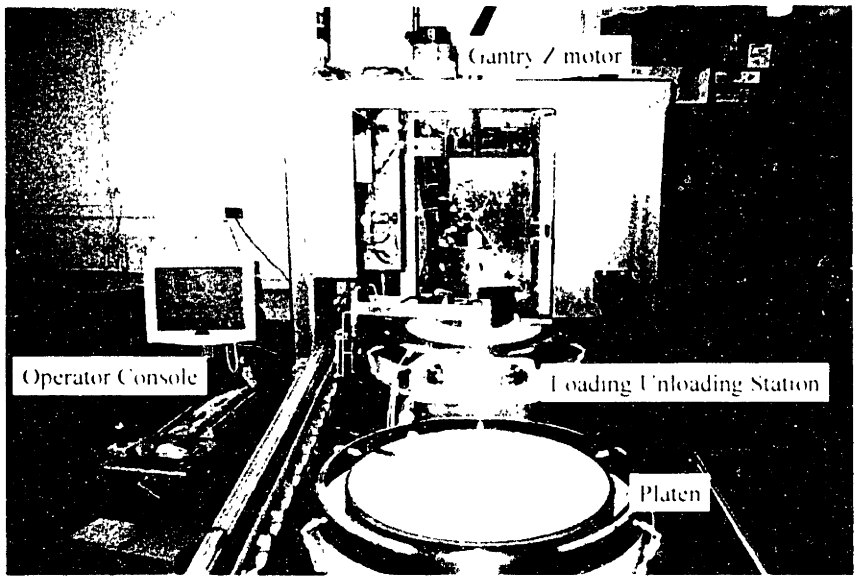


Figure 5.8 Photograph of CMP α Machine, front view

Figure 5.8 is another picture of the machine, viewed from a different angle. The operator console with the touch screen and the keyboard is located at the left side of the machine. It shows the gantry Z axis motor located on the top of the gantry, and its ballscrew. The two platens and the loading/unloading station are also visible.

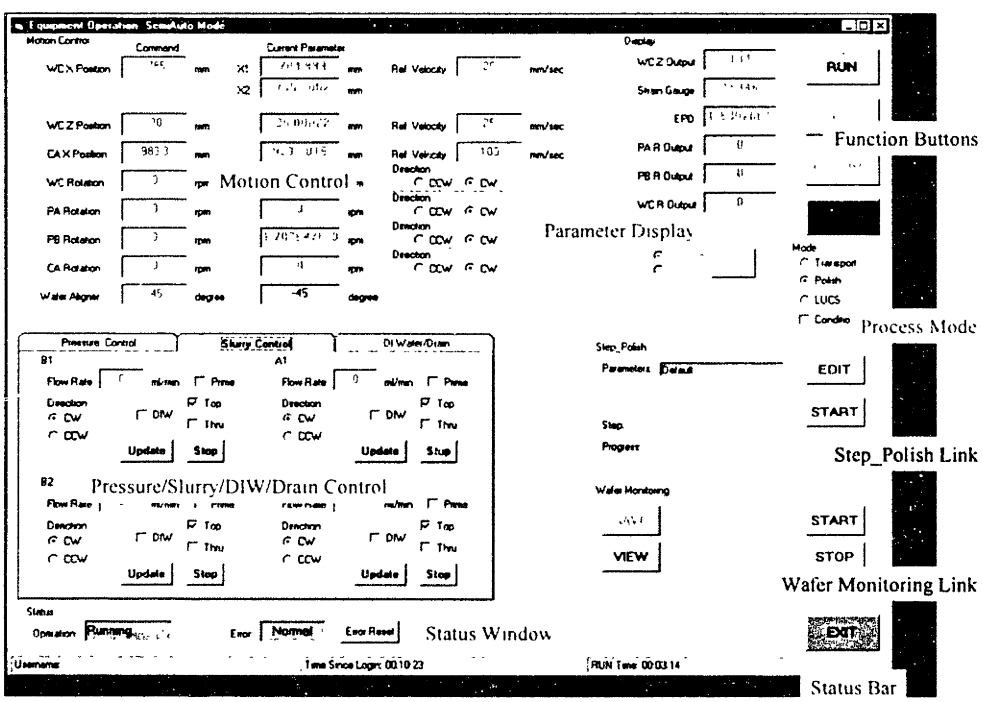


Figure 5.9 Semiauto Mode User Interface

The semiauto user interface is developed based on the manual mode interface design in Chapter 2, with a slight modification to accommodate features specific to the semiauto mode. Figure 5.9 shows the image of the semiautomode user interface taken while the machine is running.

All the servo command boxes and the servo parameter displays are located inside Motion Control frame. Users can type in position and velocity commands, and also can choose the rotational direction. Motion commands are delivered to the Machine-level by clicking RUN button. Clicking STOP button triggers the stop routine in the Machine-level, which resets the servo commands and removes all the controller outputs.

All the open-loop controls- pressure, slurry, DI water and drain, are located inside the Open-loop Control tab. Here users can type in or select commands. Open-loop controls have their own update and stop buttons independent from the function buttons.

Process Mode frame provides a way for a user to select among Transport, Polish and LUCS, which adjusts the wafer carrier Z axis travel limit. Conditioning can also be selected from the Process Mode which activates the conditioner pneumatic pressure control.

Machine parameters, other than servo control parameters, can be seen from Parameter Display. It displays the reflectance sensor reading, the strain gauge reading and the outputs of a few controllers. Users can watch the controller outputs to see if the amplifiers are getting saturated.

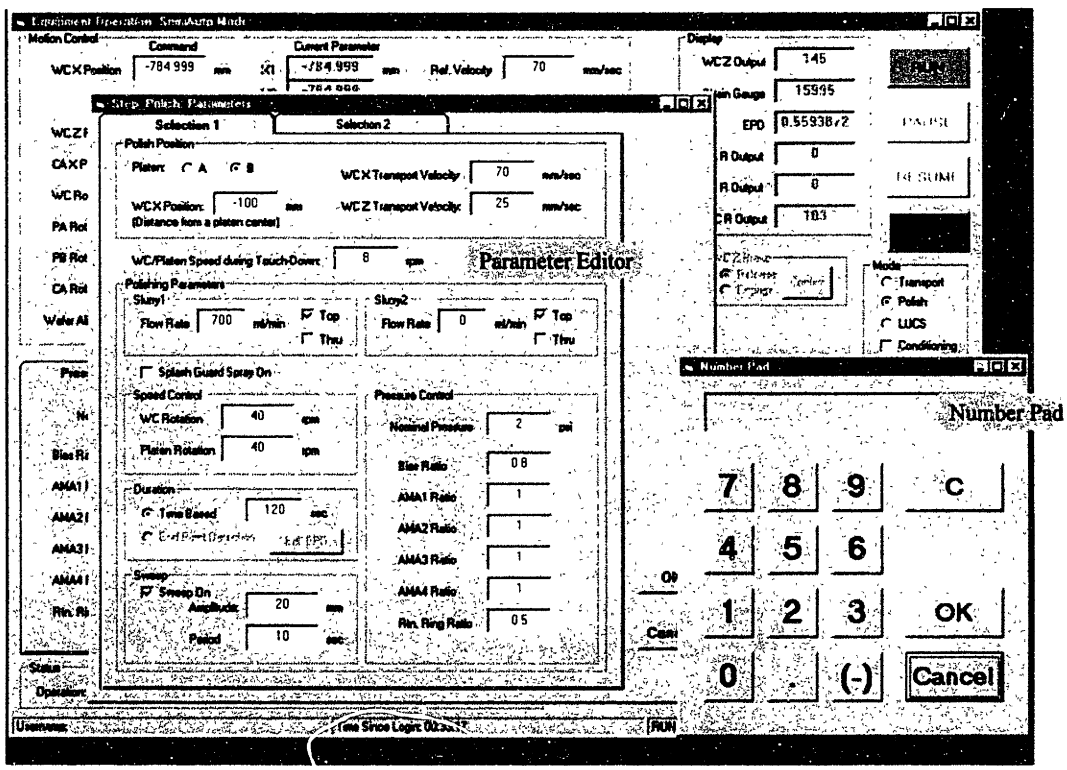


Figure 5.10 Step_Polish Editor and Number Pad

Status window shows the status of the machine (Running, Initializing, Stopped, etc.). The status color is synchronized with the signal tower lights of the machine. The error status

shows if an error is set. Once an error is triggered, it will change to red and shows an error message box. Depending on the type of the error, the user will have an option to restore the machine operation by clicking Error Reset button or simply will have to terminate the operation.

The screen components mentioned so far constitutes the manual mode user interface. Wafer Monitoring link and Step_Polish link are the additions to the manual mode to upgrade to the semiauto mode. Clicking START button in Wafer Monitoring link initiates the wafer reflectance measurement and signal processing routine in the second processor of the ADwin. The routine calculates the reflectance vs. the wafer radius, zone averages, mean and standard deviation of the reflectance signal, etc, and sends the information to the host PC. Clicking VIEW button displays the wafer monitoring window which shows the various information obtained from the routine.

Step_Polish link has EDIT and START button. Clicking EDIT button will display the polish parameter editor shown in Figure 5.10. There the user enters polishing parameters- platen selection, slurry flow rate, pressure, velocity, sweeping selection, etc. The figure also shows the number pad, which is automatically displayed once the user touches a command input field. Clicking OK of the number pad updates the content of the command input field with the number pad text. Clicking Cancel simply unloads the number pad from the screen without changing the current value of the command input field.

Clicking OK button of the polish editor loads all the polish data input to the semiauto mode process parameter buffer. Clicking Cancel unloads the editor without changing the content of the buffer. Clicking START button initiates the polishing step (changes the operation mode from manual to semiauto). It first disables all the user interface controls except STOP and EXIT button. Based on the preprogrammed sequential algorithm, the polishing step sends the process commands from the parameter buffer step by step. The user can always abort the polishing step by clicking STOP button. Clicking it terminates the polishing step and restores the manual mode. At the end of polishing, the step terminates itself and returns the process control to the manual mode.

Using the interface introduced so far, a user can drive the machine the way he or she wants to process wafers and to support wafer processing. Figure 5.11 shows the picture of the machine, performing pad conditioning prior to wafer polishing. The conditioner is brought to the top of the platen, and both of them start to rotate. The dispense tube starts to flush DI water to lubricate the interface and carry the debris away. The user enters the conditioner pressure to bring it down to the pad surface. Then the conditioner can be moved radially either constantly or intermittently zone by zone to cover up the whole pad surface.

Figure 5.12 shows the machine engaged in polishing. The wafer is first loaded to the wafer carrier and held by vacuum. The wafer carrier is brought to the nominal polishing position, and starts to hunt the Z polishing position by measuring the change in the strain gauge reading. Once the Z position is settled, the wafer carrier and the platen starts to spin at low speed, slurry is being dispensed, the wafer carrier releases the vacuum and apply low positive pressures to the membrane. Once the low pressure is settled, the rotational speeds are ramped up to the polishing speeds and at the same time the membrane pressures are increased to the polishing pressure. The polishing is fully engaged and the timer starts to tick. Sweeping the wafer carrier radially may optionally be performed.

At the end of polishing, the wafer carrier and the platen are slowed down to the pick up speed, and slurry dispensing is stopped. The pressures are released and vacuum is applied to the membranes to hold the wafer. Once the vacuum settles the wafer carrier picks up the

wafer by moving upward to the point it can clear the transportation clearance. Then the wafer carrier is transported to the specified finish position.

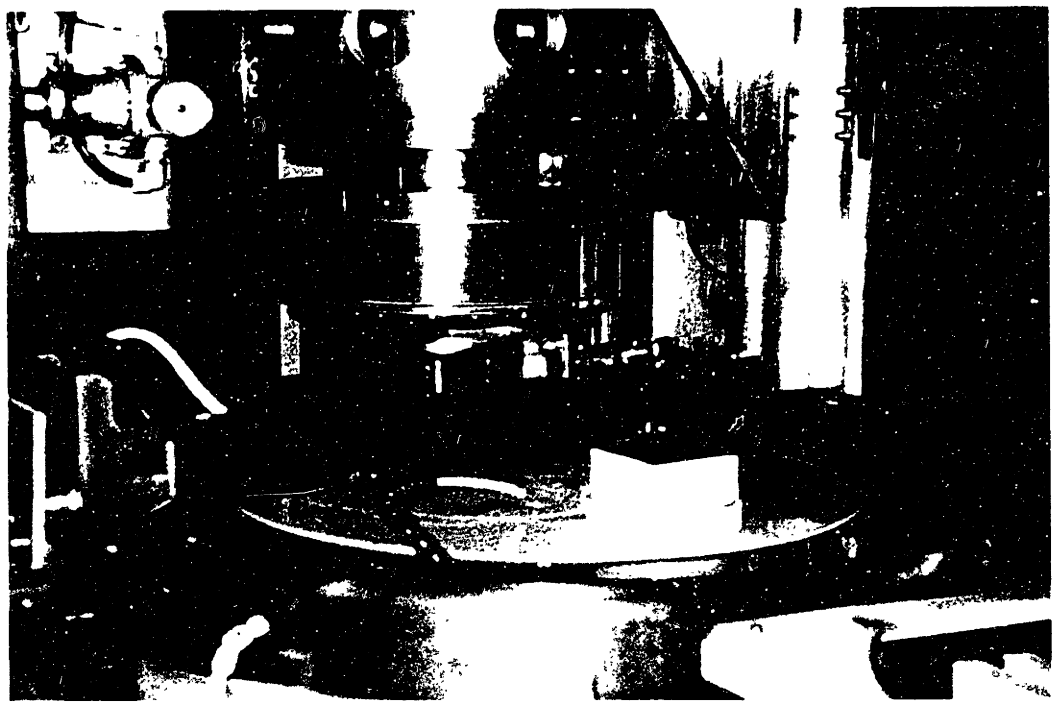


Figure 5.11 Photograph of CMP α Machine in Conditioning

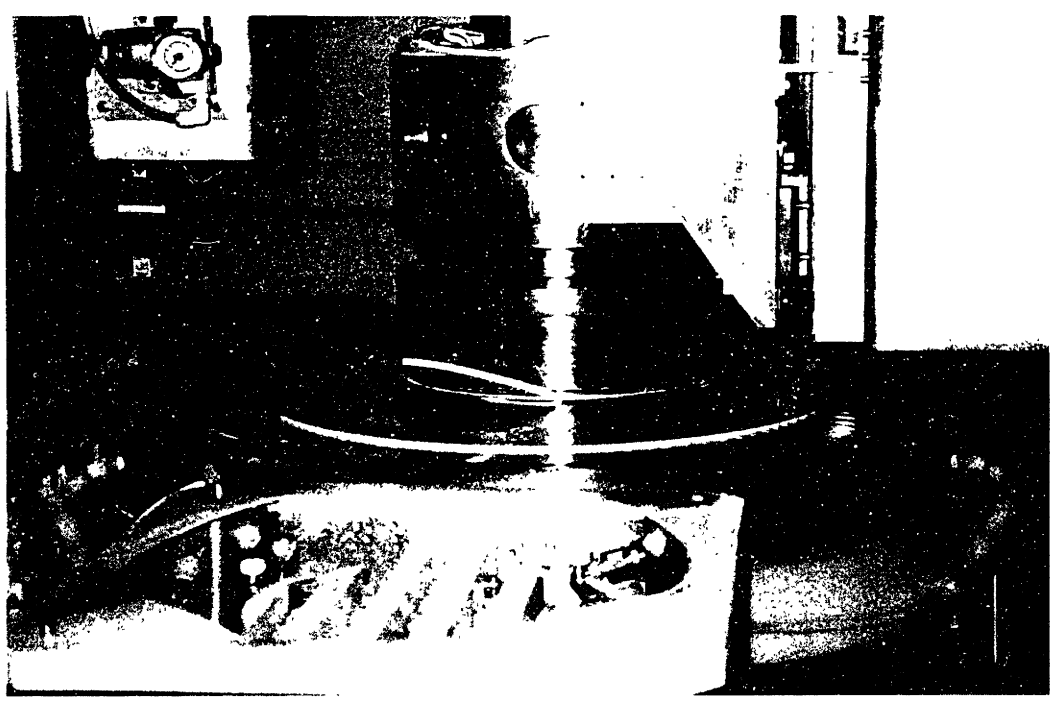


Figure 5.12 Photograph of CMP α Machine in Polishing

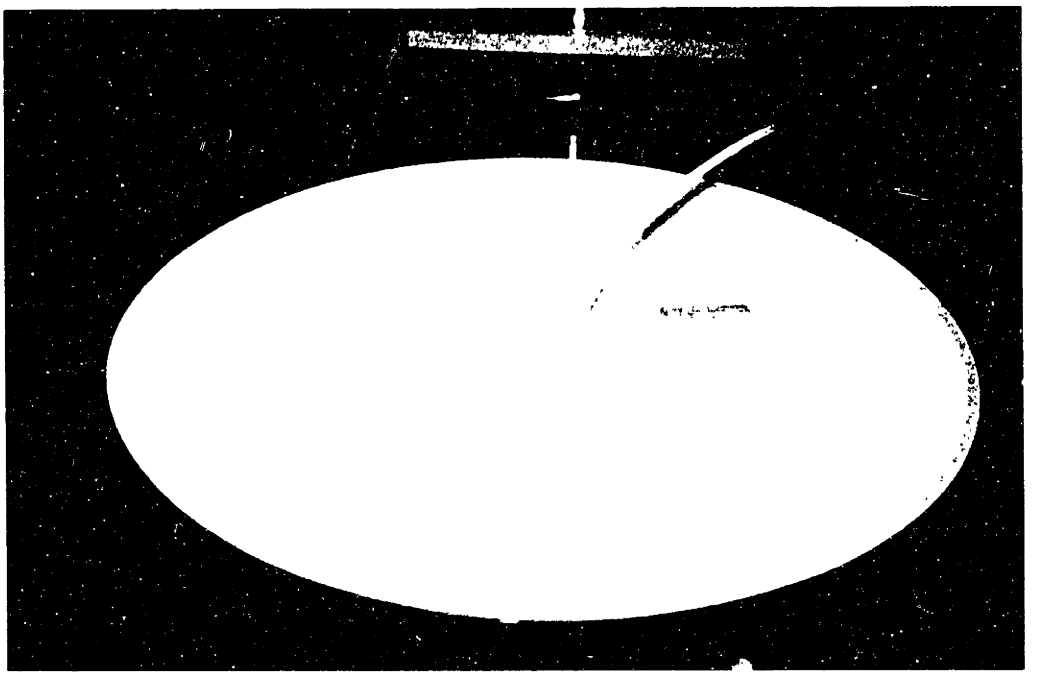


Figure 5.13 Photograph of CMP α Machine in Platen Cleaning

Polishing leaves lots of slurry and wear particles from the wafer on top of the platen. Platen usually needs to be cleaned after polishing, unless a reasonable amount of buffing has been done. During platen cleaning, DI water is dispensed to the center of the platen, which spins at a very high speed (~100 rpm). The centrifugal flow of DI water will clean up the pad surface. With approximately 2000 ml/min of flow rate, a platen can be cleaned within 20 seconds. Figure 5.13 show the platen being cleaned.

This chapter dealt with the wiring of the individual control system components to establish the hardware interface and the programming and implementation of the control system. As we have seen from the photographs of the machine in action. The system is fully integrated and successfully implemented.

Chapter 6. Conclusion

The CMP α machine control system has been developed successfully using the framework of Axiomatic Design. Although the development is not fully completed, the machine is functional to be able to process wafers in manual mode. A limited scale of automation is also in place.

The wiring of the control system components has been completed, ensuring the safety of the machine operation and minimizing the effect of noise. The interface to the ADwin is well established. It can detect every signal correctly and send outputs promptly. Signal processing is robust, and open-loop controllers and sensors work precisely due to the good calibration. The design and implementation of closed-loop controllers has been successful. All the servo controllers meet the design specification and ensure the precision and effectiveness of the machine operation.

The development on the host PC side also has been satisfactory. It can load, start, stop and unload the Machine-level software. The Process-level control system provides access to every control elements of the machine to the user both for the maintenance and the operation purpose. It acts as a supervisor of the Machine-level control system and as a messenger to the user. It has provided a solid basis for a further development in process control, including the fully automated wafer processing.

Axiomatic Design has been very effective in structuring the overall control system. Starting from the highest level functional requirements, appropriate design parameters were selected, and based on the selected design parameters, lower level functional requirements were established enabling subsequent decomposition. Higher level design parameters are identified with the higher level systems, such as the Machine-level control system and the Process-level control system. Subsequent decompositions specify the necessary subsystems, such as the recipe editor and the user interface. The Axiomatic decomposition is the very process to structure the whole control system, which simply starts from the root at the mission statement.

Axiomatic Design has been effective not only in designing the system at higher level but also in designing a small constituent. For example, Axiomatic Design specifies the general structure of closed-loop controllers.

Axiomatic Design has also been used to design the sequence of a software program and the sequential algorithm of a recipe step for automatic wafer processing. The initialization, the event and the termination procedure sequences of the Machine-level software are clearly enumerated by Axiomatic Design. The sequential stages of the polish step are explicitly stated in the design equation. The sequential functional diagram can easily be constructed from the design equation. The above example shows that Axiomatic Design can be used to design sequential procedures of a software or a control program as well as the structure of them. Axiomatic Design can expand its horizon not only in space (structure) but also in time (sequence).

A good software design often involves grouping a certain class of functionalities and forming an instance (object) to substantiate its associated functionalities. Related details are hidden to the viewer (abstraction). Different levels of functionalities are conceived and the objects are formed with in a level. The mess of functionalities, specifications, methods and technique are transformed to a building of objects, with its floor identified by associated functionalities. A hierarchically well structured software program is easy to visualize, understand and maintain.

Axiomatic Design is by nature hierarchical. A software design based on Axiomatic decomposition identifies its layers by stating functional requirements. The layers are realized by forming their classes (templates of objects), which are design parameters to be sought. The layer based on its functionality is substantiated at run time by the objects created from the classes. Axiomatic decomposition of software innately leads to the hierarchical structuring and abstraction of details, producing classes and objects to represent each layer along the way, which eventually guides to the good software design.

Although Axiomatic Design is used to develop a specific machine control system in this work, the decompositions and methodologies presented so far can universally be applied to any type of machine control system. Higher level decompositions, such as the Machine-level control system, the Process-level control system and the recipe editor, are not intrinsic to a CMP machine. The same structure can be applied to develop control systems for other types of machines, with possible modifications in lower level decompositions.

The design of a sequential functional diagram based on Axiomatic decomposition is a general methodology applicable to any sequential control process design. Any software sequence can also be designed by Axiomatic design. The structure of the recipe editor proposed in Chapter 3 can be used to design an editor for any different type of wafer processing machine.

A universal template of the machine control system has been designed in this thesis, based on Axiomatic Design.

Chapter 7. Future Work

Even though the machine is functional and simple wafer polishing tests can be performed on it, there still are many upgrades required. The auto mode development has the highest priority. The designs of the auto mode and the accompanying recipe editor are already completed in Chapter 3. Based on the design, small working models will be developed first to see if they can perform all the required functionalities. Then the full scale systems will be developed with possible modifications in design parameter selection depending on the results of the model testing.

The existing semiauto mode also needs minor upgrades such as adding more measurement sensors for process control and improving force and position control of wafer carrier polishing interface.

Advanced process handling is also an issue in the Process-level. End point detection based on the reflection and the motor current sensing needs to be implemented. Active adjustment of wafer carrier membrane pressure from the reflectance sensing feedback is an interesting in-situ process control issue. The process control software also should be able to adjust its parameters from the measurement of processed wafers (ex-situ). Prescanning of a wafer surface before polishing will also allow to adjust the process parameters (pre-situ). Ultimately, the process control software will have a database to store the previous and current measurement results and a decision model to direct when and how the parameters are adjusted. We may call it an intelligent process control system.

Higher level system integration is also an interesting subject. In an actual production environment, the machine hardly works as a stand alone device. It is usually grouped with several identical equipments and has either shared or its own wafer transfer robot, cleaning station, measurement station, etc. The grouping often result in cluster or bay type configuration. The equipment control system now has to collaborate with others. We may call it an operation-level control system. It may reside in the same hardware as in the machine host PC or in a different central computer.

Eventually there is a fab or factory level control system which supervise each equipment group and schedules their operation. How to integrate an individual machine control system to a group control system and then to a higher level management system is a subject worth to investigate. It will invoke Axiomatic Design to decompose the whole system, which will simply cite the CMP α machine control system as one of its lower level design parameters.

Reference

1. The National Technology Roadmap for Semiconductors, Semiconductor Industry Association(SIA), San Jose, CA, 1997.

2. Semiconductor International, Des Plaines, IL, November 1998.

3. N. P. Suh, The Principles of Design, Oxford University Press, New York, NY, 1990, ISBN 019-504346-6.

4. N. P. Suh, Axiomatic Design: Advances and Applications, to be published by Oxford University Press, New York, NY, 2000.

5. ADbasic, Jäger Computergesteuerte Meßtechnik GmbH, Rheinstraße 4, 64653 Lorsch, Deutschland, August 1998

6. SEMI E95-0200: Specification for Human Interface for Semiconductor Manufacturing Equipment, Semiconductor Equipment and Materials International(SEMI), San Jose, CA, 2000

7. Kevin Payette, "The Virtual Shaft Control Algorithm for Synchronized Motion Control", Proc. of the American Control Conference, 1998, vol. 5, pp. 3008-3012.

8. R. D. Lorenz and P. B. Schmidt, "Synchronized Motion Control for Process Automation", IEEE-IAS Conf. Rec. 1989, pp. 1693-1698.

9. Peter Moreton, "Industrial Brushless Servomotors", Newnes, Oxford, England, 2000

10. Gustaf Olsson, "Computer Systems for Automation and Control", Prentice Hall, Englewood Cliffs, NJ, 1992, ISBN 013-457581-4.

11. Katsuhiko Ogata, "Modern Control Engineering", Prentice Hall, Englewood Cliffs, NJ, 1990, ISBN 013-598731-8

12. ADwin-Pro: System and Hardware Manual, Jäger Computergesteuerte Meßtechnik GmbH, Rheinstraße 4, 64653 Lorsch, Deutschland, September 1998

THESIS PROCESSING SLIP

FIXED FIELD: ill. _____ name _____

index _____ biblio _____

▶ **COPIES:** Archives Aero Dewey Eng Hum

Lindgren Music Rotch Science

TITLE VARIES: ▶☐ _____

_____

_____

NAME VARIES: ▶☐ _____

_____

IMPRINT: **(COPYRIGHT)** _____

_____

▶ **COLLATION:** 140l

_____

▶ **ADD: DEGREE:** _____ ▶ **DEPT.:** _____

SUPERVISORS: _____

_____

_____ __ _____

_____ __ _____

_____ __ _____

_____ __ _____

_____ __ _____

_____ __ _____

_____ __ _____

NOTES:

_____ cat'r: _____ date:

page:

▶ **DEPT:** M.I.

▶ **YEAR:** _____ ▶ **DEGREE:** _____

▶ **NAME:** _____

_____