Full Software AC Servo Controllers with Dynamic Pulse Width Modulation

by

Hyoseok D. Yang

B.S., Mechanical Engineering
Massachusetts Institute of Technology, 1997

SUBMITTED TO THE DEPARTMENT OF MECHANICAL ENGINEERING IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE IN MECHANICAL ENGINEERING
AT THE
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

FEBRUARY 1999

Signature of Author [illegible]
Department of Mechanical Engineering
January 15, 1999

Certified by [illegible]
Haruhiko Harry Asada
Professor of Mechanical Engineering
Thesis Supervisor

Accepted by [illegible]
Ain A. Sonin
Professor of Mechanical Engineering
Chairman, Department Committee on Graduate Students

Full Software AC Servo Controllers with Dynamic Pulse Width Modulation

by

Hyoseok D. Yang

Submitted to the Department of Mechanical Engineering
On January 15, 1999 in Partial Fulfillment of the
Requirements for the Degree of Master of Science in
Mechanical Engineering

ABSTRACT

Full software AC servo controllers have been developed with special real time software in the Windows NT operating system. The full software AC servo controllers are able to control multiple AC servomotors with the CPU of a personal computer. The controller handles $100\mu s$ commutation and current feedback, $200\mu s$ velocity feedback, and $1ms$ position feedback. The full software AC servo controllers provide the ultimate flexibility, low cost, and powerful graphic user interface in Windows NT. They can also take advantage of all the Windows NT features, such as networking.

The software alone performs all the calculations in the CPU of the host computer in the full software AC servo controller. The software control makes the implementation of any algorithm much easier. Advanced control algorithms, such as d-q axis control and decoupling control, are implemented utilizing the flexibility of the controllers. These control algorithms generate accurate torque current and less iron loss in the AC servomotors. They also cancel the nonlinear terms of the motor equations to make the motor dynamics steady with changing angular velocity by feed forward control.

A new renovated pulse width modulation (PWM) method, called "Dynamic PWM" has been designed and formulated. The Dynamic PWM eliminates most of the delay problems associated with the current PWM. The effectiveness of the dynamic PWM was verified through simulation in the Matlab software. High frequency and instability analyses were done in the simulation. The dynamic PWM showed about one PWM period less phase lag than the regular PWM. It was also able to handle a much higher gain in the current feedback than the regular PWM. The future implementation algorithm and procedures using the full software AC servo controllers were also developed.

Thesis Supervisor: Haruhiko Harry Asada
Title: Professor of Mechanical Engineering

Acknowledgements

I would like to thank my thesis supervisor, Professor Haruhiko Harry Asada, for his encouragement and support during this thesis work. He made it both challenging and exciting. I would also like to thank Daewoo Heavy Industries, Co. and Shin Nippon Koki, Co. for sponsoring the project.

I am truly grateful for my advisor, Dr. Booho Yang, for his invaluable insights and friendship. He contributed to many aspects of this thesis. I am also grateful for Dr. Kuowei Chang for his advice and encouragement during the challenging times of this project. With his all-encompassing knowledge in electronics, he helped me overcome many obstacles.

I would like to thank my labmates in d'Arbeloff Laboratory for their friendship and assistance. I would like to express my appreciation especially to Sokwoo Rhee for being a pioneer in this field and for paving a way for future work.

My deepest love and appreciation goes to my fiance, Unhyi, for her dedicated love and care. Her patience and understanding brought me through times of sleepless lab hours. I'm forever grateful.

My sincere gratitude goes to my parents for their sacrificial love and support. Their continued devotion is the very reason that I could complete my work at MIT.

Finally, I praise my Lord Jesus Christ for the grace he has poured upon me. All glory and honor are his forever and ever.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1
Introduction

There is a great need for intelligent and flexible numerical controllers, which could replace the traditional controllers exclusively provided by only a few NC controller manufacturers. The conventional controllers do not provide flexibility and open architecture because they use the proprietary logic circuits with microprocessors or DSP chips. The users must follow the strict procedures and the control methods provided by the controller manufacturers. The dedicated chips are neither flexible nor as powerful as the central processing unit (CPU) of personal computers. They also require their own graphic user interfaces (GUI), which vary from one manufacturer to another unlike the personal computer (PC) industry where users can benefit from familiar GUI regardless of the PC manufacturer. However, the NC machine and the robotic industries still depend on the few manufacturers because the controllers must be able to perform in real time with many specifications. On the other hand, there are many advantages to replacing those exclusive traditional controllers with PC's. In order to use a PC as a controller, it must be able to handle all the controls in real time fulfilling all the specifications of the industries. The use of PC's would bring a new era to the NC machine and robotics industries.

The advantages to replacing those exclusive traditional controllers with PC's are manifold. The PC based controllers could provide the open architecture and flexibility. Users could have the flexibility to use their own control methods and specific implementation technology. Many new technologies in the NC machine and robotics industries could result from the flexible and open architecture controllers.
There will be vast opportunities for the motion control industry to grow. PC based controllers will also provide a user friendly GUI. However, the PC based controllers should not be a mere user interface for the traditional controllers, which still have to use some intelligent chips for actual control. Using those chips are not cost effective and limit the flexibility. Therefore, those chips should be replaced completely by software alone. Full software servo controllers give the ultimate flexibility because all the controls

and interfaces are done by the software alone. They make use of the advanced PC operating systems such as Windows NT and allow users to easily incorporate isolated machines into a network based system. Moreover, a variety of application software and other hardware peripherals for PC's could be used as well. However, the problem is that the real-time control and the time-critical operations are not supported under the advanced GUI operating systems. The current practice is to use dedicated motion control cards with DSP chips to off-load the burden of real-time computation and interrupt handling. The dedicated motion control cards not only limit the flexibility of the system but also increase the cost.

The rapid progress of CPU's computing power may eliminate the need for dedicated motion control cards and replace them by software alone. Real-time operating systems play more important roles for the complex real-time control applications. A real-time operating system in the feedback loop of such a control system must respond to periodic external interrupts consistently within a certain time limit called "hard deadline." [6]. If the delay in the operating system exceeds the hard deadline, the system behaves unexpectedly and may cause instability. A similar phenomenon also can be observed if the interrupt sampling in the feedback loop is fluctuated. To meet the deadline and the temporal consistency requirements for time-critical applications, the real-time operating system must have a microscopic, consistent interrupt latency.

In the past years, many algorithms have been developed to handle the external interrupts and the computation under the above timing-related constraints. To name a few, the worst case execution time estimate approach [7,8,9], the queuing spin lock algorithm [10, 11] and the integrated inter-process communication and scheduling scheme[12] are recent results. These approaches have already been implemented and tested on the high-end platforms such as UNIX workstations. However, in spite of the recent explosive improvement of performance and reliability of PC's, the available operating systems for PC's are not real time operating systems by themselves. If the high-performance real-time functionality is appended to general-purpose operating systems (OS) such as Windows NT, PC's will be the power motion controllers in the next century.

In chapter 2, the development of such full software real time controllers in the Windows NT operating system is presented. Windows NT has the capability to provide fast response time with powerful networking and graphic user interface capabilities. However, it is not deterministic under the preemptive multi-tasking architecture. Therefore, it is thought to be not suitable for the time-critical real-time operations such as the current feedback of AC servomotors with 50μs to 100μs sampling rates. However, by adding the real-time OS feature to Windows NT, it turns into the foundation for the deterministic and powerful real time controller. The goal of this thesis is to present the development of the software for handling interrupts and performing time-critical computations under the Windows NT operating system and issues involved.

One major issue in the full software servo controller is the time budget because the controller uses the CPU from the host computer. The number of controllable axes is determined by the CPU usage at the current sampling rates. The controller should be able to control multiple axes at the high current sampling frequency in order for it to be used in the robotics or NC machine industries. In chapter 4, the time budget and the new interface board are presented. The new board improves the time budget by decreasing the number of I/O accesses. The estimation and the actual time budget with the new board are also presented to show that the full software servo controllers are adequate for the robotics and the NC machine industries.

The objective of AC servo control is to have fast bandwidths of the system. In order to have fast bandwidths in velocity or position, the current controller must provide the desired torque current accurately. In order to determine the best algorithm to obtain the desired torque with fast bandwidth, various control algorithms can be implemented easily with the software AC servo controller because of the flexible architecture. The algorithms can be implemented just by compiling different programming source codes. Three different control algorithms are discussed and the experimental data using those control algorithms are presented. First, the three phase local feedback control, the conventional algorithm used by industry, is compared to the d-q axis feedback control. The d-q axis algorithm [2 3], derived from the Clarke and Park transformation, actually controls the torque current and the iron loss directly to provide the accurate torque current. The decoupling control with Back EMF compensation is also implemented. The decoupling control eliminates the nonlinear terms due to the mutual inductance of the AC servomotors. It also maintains the torque current at the desired value regardless of the angular velocity of the motor.

The full software servo controllers enable the control architecture to change easily. The flexibility of the controller prompts the search for technological breakthroughs in fundamental issues such as pulse width modulation (PWM), which is a method of powering the AC servomotors. Chapter 6 of this thesis discusses a new pulse width modulation (PWM) algorithm designed by the author. The new PWM algorithm is called "Dynamic PWM." The problems with the current PWM algorithms are discussed. The dynamic PWM is presented with the simulation data to prove the effectiveness of the algorithm.

Chapter 2

Full Software AC Servo Control

2.1 Overview

Full software AC servo real time controllers in the Windows NT platform using personal computers have been developed. Full software AC servo control systems provide flexible open architecture, powerful graphic user interface, low cost, and offers the availability of using other peripheral devices. The Windows NT operating system also provides an easy networking feature for factory automation and other robotics and machine tool applications. In this chapter, the architecture of the full software AC servo controllers is presented.

2.2 Overall Architecture

In traditional PC-based motion controllers, it has been the standard to use DSP chips for intense real time control tasks. The CPU's task, in this kind of controller, is to give out trajectory command and to monitor the overall operation. The reason that most motion controllers cannot get away from this kind of structure is the inadequate speed of the CPU. If the CPU were to carry out the motion control tasks in detail, for example, the computation for all the feedback, it would be almost impossible for the CPU to do even the most basic tasks such as monitoring or displaying. But with the introduction of Intel's fast Pentium Processors and Windows NT operating system with a new user-friendly interface, real time motion control without any dedicated hardware became possible. Figure 2.2.1 shows the overall structure diagram of the motion control system with Windows NT.

The periodic interrupts are generated by a counter/timer board plugged in the slots of the PC. It generates two periodic interrupts for the faster and the slower sampling rates. With the faster interval interrupt, the current feedback of the AC servomotor is carried out. Other feedback for position control and velocity control is done in the slower interrupt routine. Also the trajectory data update and networking is done on the application program level which lies on the lowest priority. The position and velocity feedback are done based on the position information read from the encoder board. Approximately half the CPU capacity is used for interrupt service routines, which carry out multiple feedback, and the rest is used for application programs including trajectory data update and networking, or other non-control related tasks.

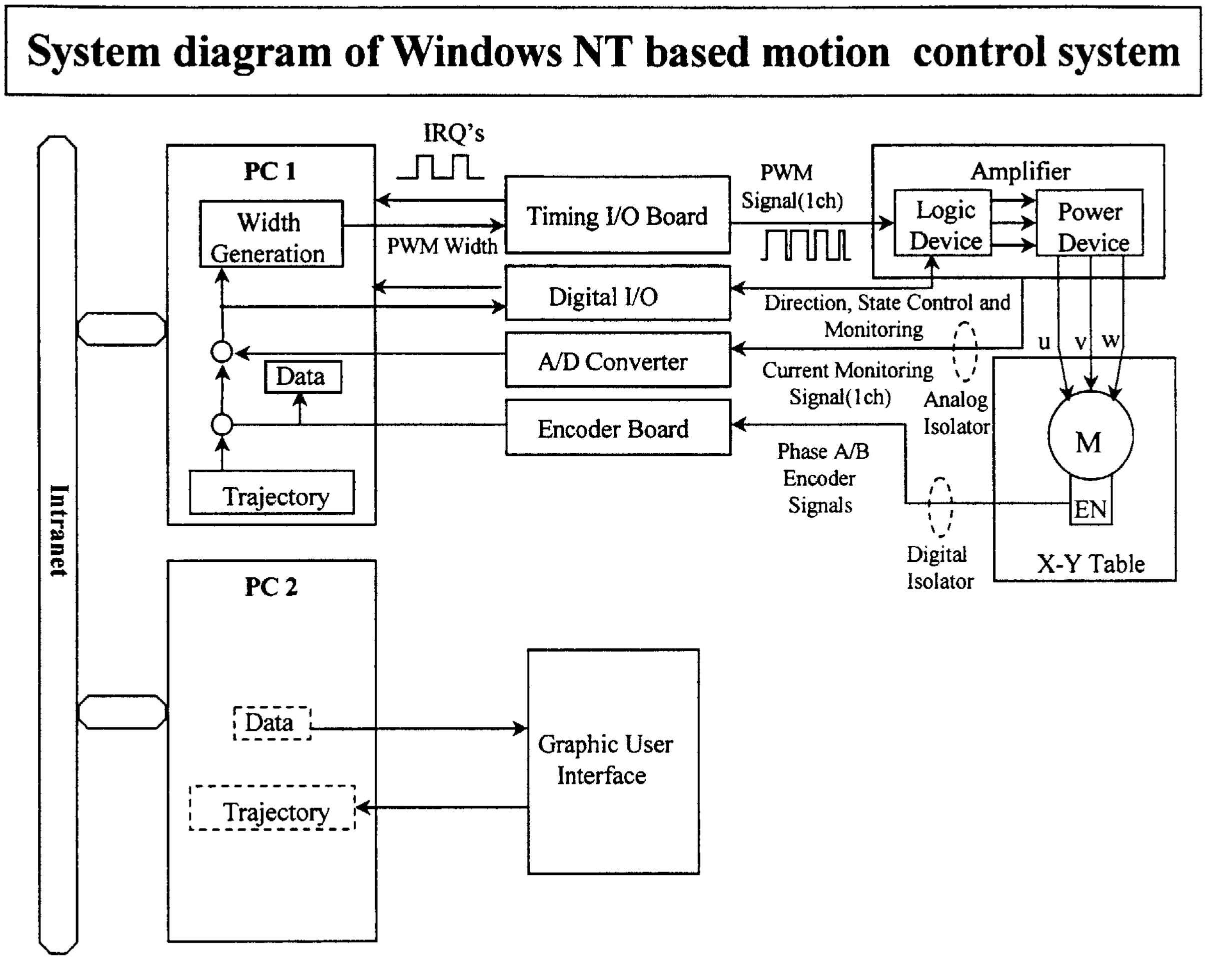


Figure 2.2.1: Structure Diagram of Windows NT based Motion Control System

2.3 Windows NT-based full Software AC Servo System

In this section, the recent development of the Windows NT-based full software AC servo system is presented. To emphasize the advantages of the system, the conventional industrial practice of AC servo motor control is reviewed. Also, the previous development of the PC-based AC control system is briefly reviewed. The new Windows NT-based AC servo control system provides a competitive control performance with great flexibility at a minimum cost.

2.3.1 Structure of an Industrial AC Servo System - Traditional Method

In order to control an AC servomotor with optimal efficiency, the control device has to generate a magnetic flux perpendicular to the current at all times. Designated hardware has been used to regulate the current. To control the current that flows into the motor, the PWM signal (Pulse Width Modulation) is generated by the electronic circuit. This circuit generates three-phase analog sine waves for commutation of the AC servomotor. These three sine wave signals have a phase difference of 120 degrees to each other. These analog signals go into the PWM generation circuit and are converted into corresponding digital signals, which are pulse width modulated. These TTL signals turn on and off the switching devices. The power inputs to motor are also turned on and off concurrently. The functions of these circuits are explained in detail in the following sections.

Sine Wave Generation Circuit

This circuit generates sine waves according to the rotor position. It is composed of ROM with sine value data written in it. When the position information enters the ROM as a form of address, the ROM gives out the corresponding sine value according to the position of the rotor. There are three sine waves with a phase difference of 120 degrees because the AC servomotor is a three-phase motor. In practice, phase V can be estimated by a simple analog operation through the equation $V = -(U + W)$. Therefore, only phases U and W have to be generated by ROM.

DC-SIN Conversion Circuit

The sine wave circuit generates two-phase sine waves synchronized with rotor position. In the DC-SIN conversion circuit, these values are multiplied by the reference input to increase or decrease the amplitude. This is how the current input to motor is controlled.

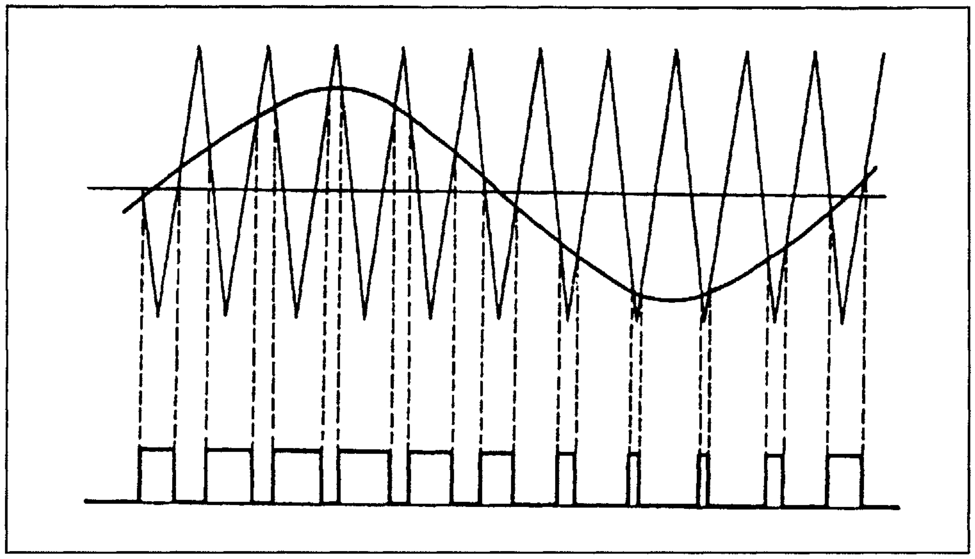


Figure 2.3.1: Traditional Pulse Width Modulation Method

PWM Generation Circuit

The sine wave current flows into the motor. To accomplish this, we can directly give continuously varying current using the analog feature of the switching devices. However, this will cause an enormous power loss and will eventually result in overheating the motor. Therefore, we have to cause the current flow by pulse to reduce the power loss. This method is called PWM (Pulse Width Modulation).

In the PWM method, the generated sine waves are compared with triangular waves with a fixed frequency. Figure 2.3.1 shows the procedure to generate the PWM with the triangular and the sine waves. The frequency of a triangular wave is around 10 ~ 20 kHz when FETs are used as switching devices. The switching device is turned off when the triangular waves have higher values than the sine-shaped current signal values. On the contrary, during the duration that the current signal values are larger than the triangular wave, the switching device is turned on, so that the current flows into the motor. By changing the duty ratio, the overall current that flows into the motor can be controlled.

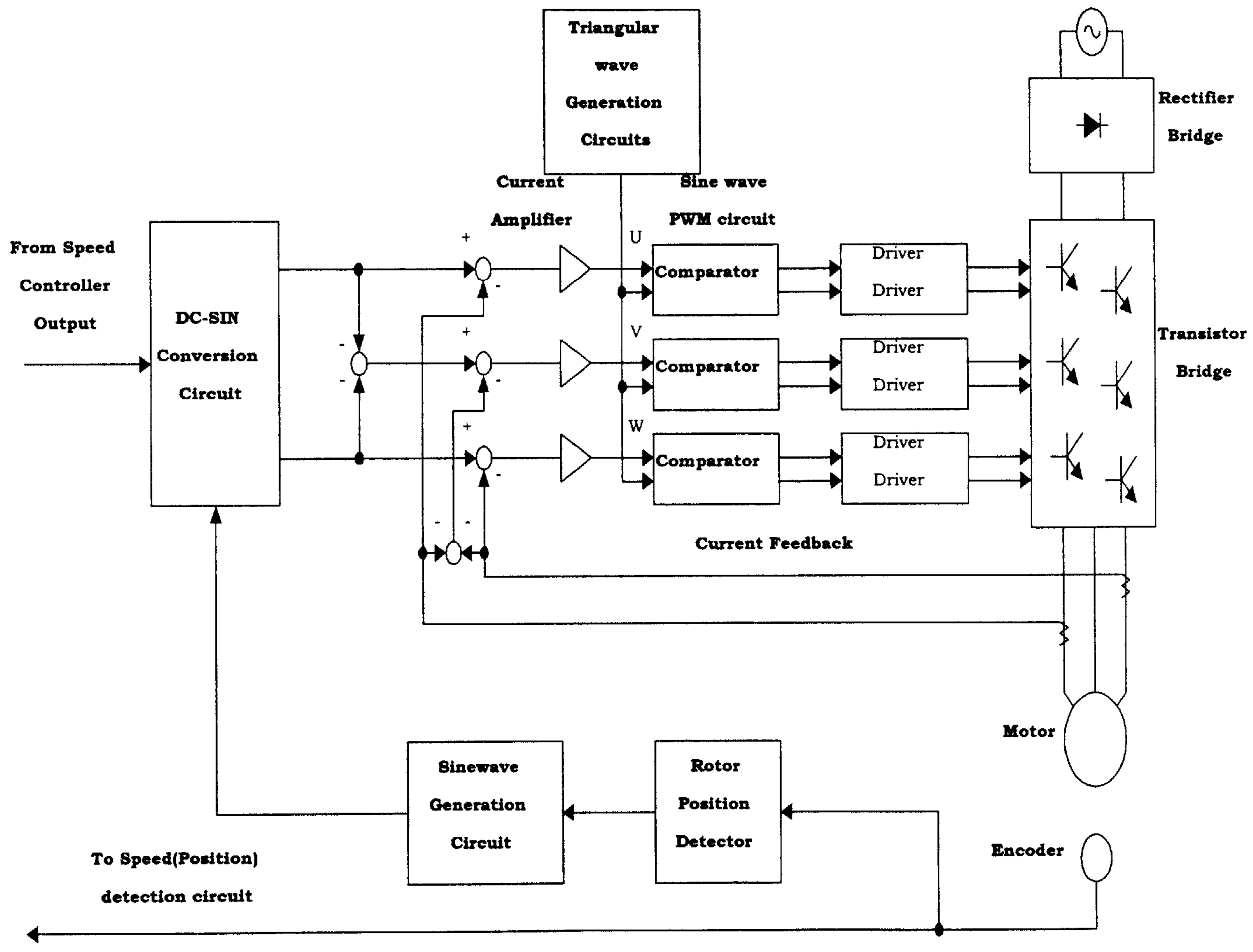


Figure 2.3.2: Block Diagram of Traditional AC Servomotor Control System

2.3.2 Windows NT-Based Software AC Servo Control

Digital AC Servo - PWM Generation by Software

The traditional method uses electrical circuits including comparators to generate the PWM signal as shown in Figure 2.3.2. Analog input signals are needed for the comparators. The electronic circuit generates analog signals. Therefore, DA conversion is needed to send out the reference value to the circuit when a digital computer is used as a controller. The converted analog signal goes through the PWM circuit. After going through the PWM circuit, the analog signal is converted back into the digital PWM signal. It is not a very efficient process.

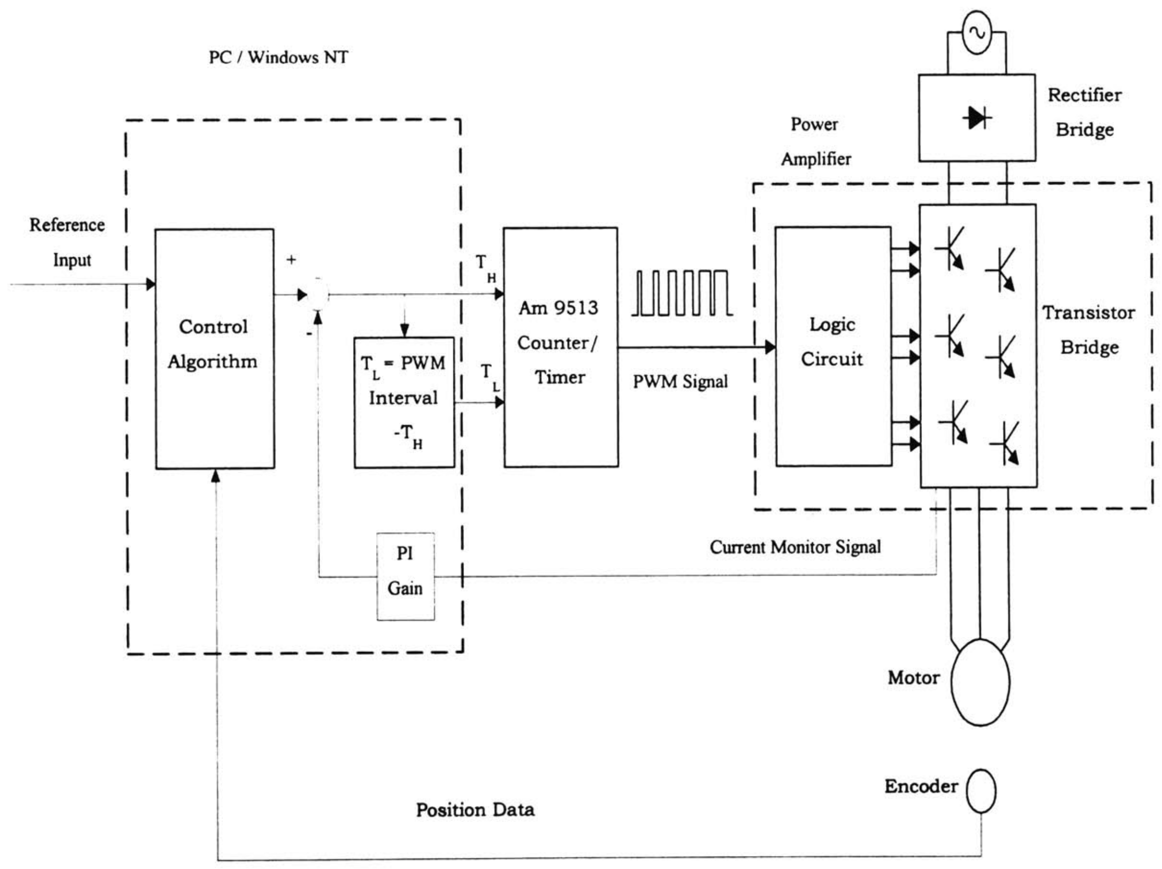


Figure 2.3.3: Diagram of AC Servomotor Control System with PWM by Counter

This tedious process is the needed for the PWM signal generation due to use of the comparator. The efficiency would improve if the PWM signals are generated directly from the digital reference input. The current industrial practice uses dedicated hardware for the system shown in Figure 2.3.3, besides the transistor bridge which is also called the power block. This report explains how we could replace most of the above electronic circuits by software. We can dramatically reduce the cost and obtain unlimited flexibility in motor control even at the current feedback level when we use the software based control system.

Previous Digital AC Servo System

In our previous system, we were already using a counter/timer board to initiate interrupt request signal generation as shown in Figure 2.3.3. This Am9513 timer/counter chip was good for our purpose in that it had a mode that could be used for generating a PWM signal by setting the high duration and low duration on the chip. By adopting this method, we were able to replace many electronic circuits by software. However, only one PWM signal was generated in the previous experimental setup. The PWM signal was manipulated in the logic circuit in the amplifier to generate all three PWM signals. The block diagram for the previous setup is shown in Figure 2.3.3.

2.3.3 PWM Inverter

In the AC servo motor control, it is required that the amplitude and the phase of each of the three phase sinusoidal currents must be precisely controlled at high speed. To satisfy the requirement, a method called PWM has been widely used. In this method, the current of a motor is converted into a controlled pulse of width proportional to the amplitude of the sine wave.

As shown in Figure 2.3.4, a three-phase PWM inverter consists of two transistors and diodes for each phase. The two transistors cannot be turned on simultaneously to avoid a short circuit. However, since the switching latency of the transistor is longer when it is off than when it is on, a short circuit might occur when both transistors are turned off simultaneously. To avoid the problem, a dead time is introduced in the inverter, as shown in Figure 2.3.4.

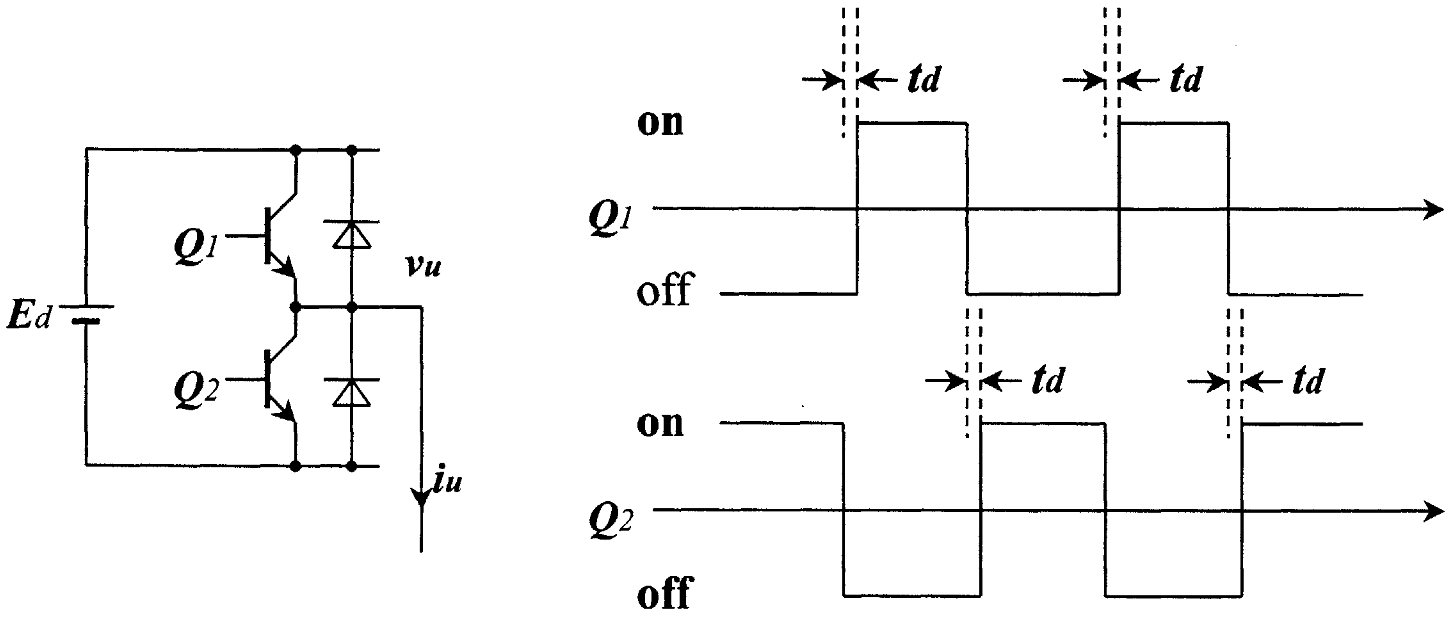


Figure 2.3.4: PWM pulse waves with a dead time t_d

In an ideal PWM inverter, output voltage V of a phase is $-E_d/2$ when the current is negative and $E_d/2$ when the current is positive. However, the output is perturbed due to the dead time. The output error can be approximated as a sinusoidal voltage and its amplitude ΔV is expressed as:

$$\Delta V = E_d\, t_d f_c$$

where t_d is the dead time and f_c is the carrier frequency of the PWM inverter. The phase of ΔV is opposite to the output current. When the motor operates at a high speed, ΔV is negligible compared with the sinusoidal output of each phase. However, when it is operating at a low speed, the current waves of the motor are distorted and torque ripples appear. Consequently, the servo control performance is deteriorated.

In this project, we first implement a dead-time compensating algorithm as shown in Figure x. In the algorithm, the direction of the motor current is detected, and ΔV is added to the voltage command when the direction is positive and $-\Delta V$ is added when the direction is negative. With this algorithm, the actual output of the PWM inverter becomes equal to the ideal output and the control performance can be maintained.

2.3.4 Advanced Full Software AC Servo

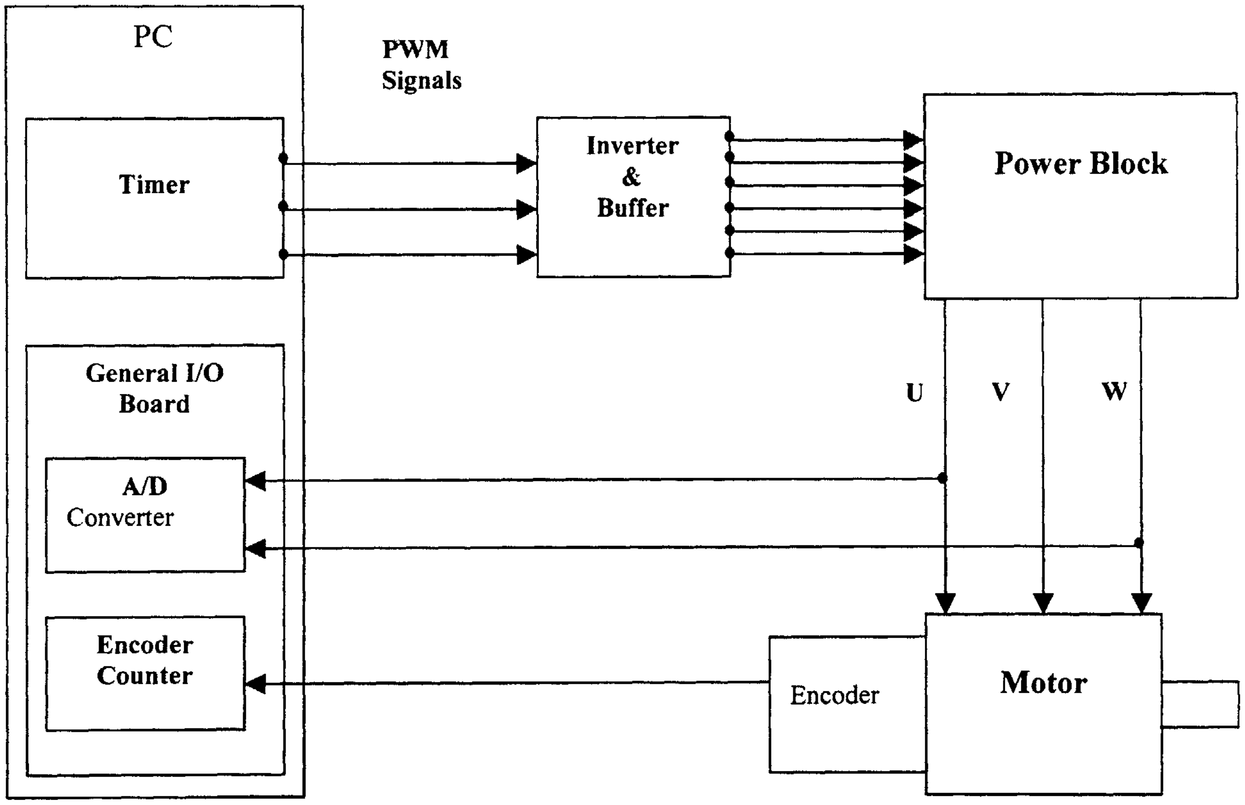


Figure 2.3.5: Structure diagram of full digital AC servo control setup

As it was shown before in Figure 2.3.3, three-phase inputs should be generated to run an AC servomotor. Each of these three signal is generated by the separate PWM circuit, in the shape of sine waves shifted 120 degrees to each other. In addition, to improve the motor performance, a current feedback circuit is also necessary, with fast periodical feedback.

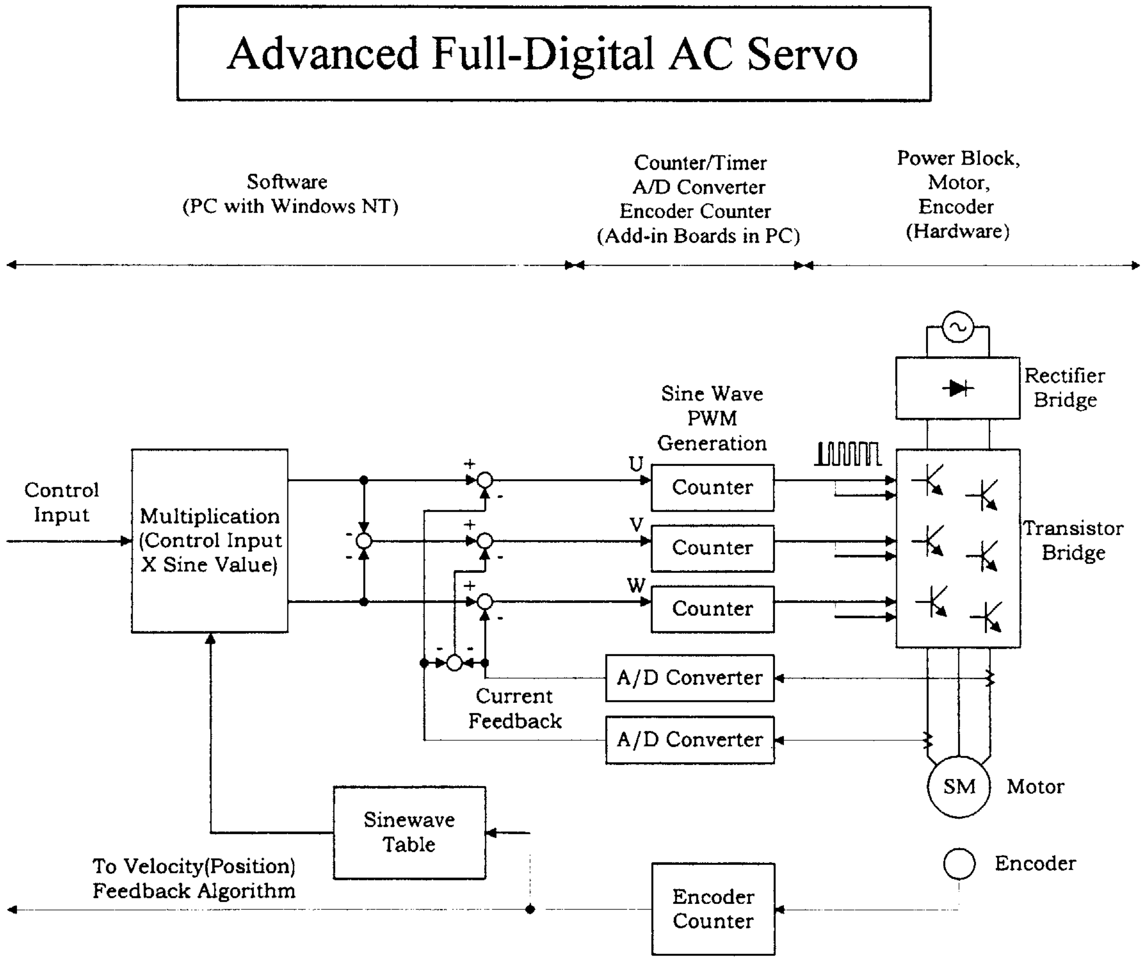


Figure 2.3.6: Block diagram of software AC servo motor control with three phase PWM generations

In the industry, the dedicated control board with DSP is used to offload the burden on those kinds of jobs, including commutation and current feedback. If the computer CPU could do commutation and current feedback with the software, there is no need for the expensive dedicated boards. In addition, there is no analog signal involved in the entire control process. Therefore, it is more immune to noise even in the harsh field environment.

In this chapter, the full software AC servo controllers that have been developed based on the Windows NT real-time operating system are discussed. Figure 2.3.6 shows the architecture of the system, while Figure 2.3.5 shows the hardware components and the connection of the servo system. As shown in Figure 2.3.6, a simple standard counter/timer board in this current setup generates the PWM signals. The logic circuit in the previous setup is no longer used because all three PWM signals are generated simultaneously by the software. The A/D converters are used for monitoring two phase currents. The sine table in the software replaces sine wave generation circuits. The software also operates the current feedback control. After the current feedback, the computed values are sent out directly to the counter to generate proper PWM signals to drive the motor. Consequently, all hardware can be replaced by software except the high voltage power amplifier.

Chapter 3

Reconfiguring Windows NT to a Real Time Operating System

3.1 Overview

As the real time control system based on Windows NT is developed, it is necessary to investigate Windows NT from the viewpoint of the internal system characteristics. In this chapter, the characteristics of Windows NT as a real time operating system are rigorously evaluated with the overall system description. One important aspect of the real time control system is to guarantee the periodic sampling without fail. It is important to find out the kinds of situations that must be avoided to guarantee real time performance. For example, there is a situation where the system control might fail if the interrupt requests are ignored because of other peripheral devices, such as a CD-Rom driver. The full software must guarantee the robustness of interrupt performance when a large amount of data is transferred from the hard disk or other data storage drivers. Some systems may not be as robust, depending on the different interface bus.

In this chapter, problems that might occur with the software real time controller during heavy data transfer are investigated and the solution is suggested. It is shown that the robustness of the system highly depends on the bus system of the PC, EIDE or SCSI drivers. An investigation and experiment was done to find out the conditions that would guarantee real time performance. It was concluded that the SCSI drive does not contribute to the interrupt disappearance while the EIDE drive does. However, a recent development of a new EIDE disk controller with a bus-mastering DMA chipset like the SCSI controller enables the interrupts to go through without disturbance from the hard disk controller. Therefore, any disk controllers with DMA compatibility could be used for the software AC servo controller. The feasibility test of using the EIDE disk controller with bus-mastering capability was performed, and the results were compared with that of the SCSI controller.

3.2 Interrupt Handling on Windows NT

In Windows NT and any other Windows systems, device drivers handle the interactions with peripheral devices. The objective of this chapter is to develop a special device driver to be involved in the "Kernel" of the Windows NT operating system so that a group of I/O devices necessary for motion control can be accessed and run in real time. Figure 3.2.1 shows the procedure for handling interrupts under the Windows NT environment. When an interrupt request comes in, the CPU jumps to special routines, called Interrupt Service Routine (ISR) and Deferred Procedure Call (DPC). In ISR, all interrupt requests coming from other devices having lower priority levels are masked off, whereas in DPC no interrupt is masked off. Namely, any other interrupt can be accepted during the DPC execution, even though the interrupt requested has a lower priority level than that of the one currently processed. The preemptive multitasking policy of Windows NT requests that only the time-critical tasks must be performed in ISR so that a particular device does not occupy the CPU for a long time. After leaving the ISR soon, most of the tasks that are not time-critical are performed in DPC.

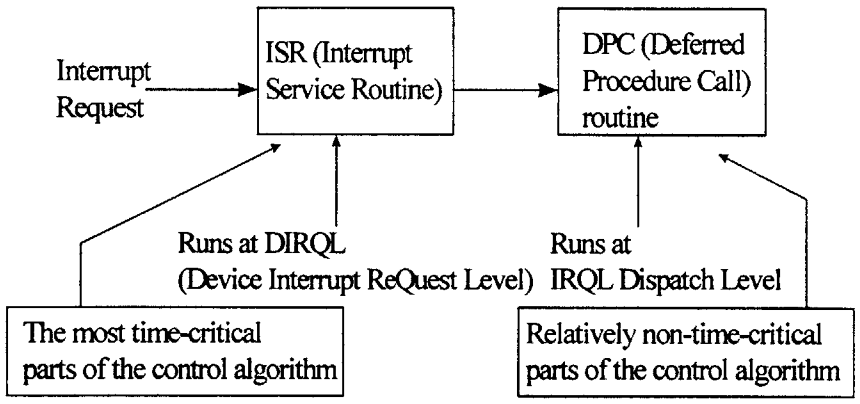

Interrupt Request
ISR (Interrupt Service Routine)
DPC (Deferred Procedure Call) routine
Runs at DIRQL (Device Interrupt ReQuest Level)
Runs at IRQL Dispatch Level
The most time-critical parts of the control algorithm
Relatively non-time-critical parts of the control algorithm

Figure 3.2.1: Basic process of interrupt handling on Windows NT

Interrupt Priority Level	Device Name
0 (highest)	System Timer
1	Keyboard
2	cascaded from slave PIC
3	COM 2 *
4	COM 1 *
5	LPT 2 *
6	Floppy Disk Controller
7	LPT 1 *
8	Real Time Clock
9	Redirection to IRQ 2
10	Reserved *
11	Reserved *
12	PS/2 Mouse
13	Reserved (Co-processor) *
14	Hard Disk Controller
15 (lowest)	Hard Disk Controller

Table 3.2.1: Interrupt priority levels of Windows NT

*: IRQ lines that are used by non-critical devices for basic operation of PC. Thus, these lines are usually open to the users.

Table 3.2.1 shows the interrupt priority levels assigned to each device in the Windows NT system. Levels 3 and 4 are assigned to a user's devices. Note that a keyboard and a certain type of mouse have higher priority levels than the user devices.

To guarantee the exact sampling interval for the real time control, a counter/timer board is used to request interrupts to the CPU. We generate two square waves with different intervals for two separate interrupt procedures. The ISRs in the device driver are programmed to carry out the feedback loops according to the specified control algorithm.

3.3 Real Time Performance with Different Types of Buses

A computer is composed of many components. These include a hard disk, memory, CD-ROM driver and other peripherals. Data exchanges between these components are done through a bus system. In other words, the bus plays a similar role to that of a bridge between islands on the sea. If the bridge is narrow, the overall traffic between islands will be crowded and slowed down. In the same way, the better the performance of the bus system becomes, the faster the overall computer speed will be. When the CPU reads information from the hard disk or CD-ROM driver, the data travels through the bus and sometimes it may be the bottleneck of the whole PC performance if the bus does not catch up with the speed of the CPU.

As mentioned before, the real time control system uses the interrupt of the CPU. Therefore, during the time a heavy data access occurs, there is a possibility that the performance of the real time control system might be deteriorated because the CPU could be totally occupied by the data transfer job. It is certain that the interrupt priority used for the real time control is higher than that of the hard disk and CD-ROM data transfer. So at first glance, the real time job should not be influenced or disturbed by the mass storage access and seems to work perfectly without any problems. However, the experiment shows that there are certain cases when the interrupt response of the CPU just disappears (which means that the interrupt request from the external timer is not accepted by the CPU) during heavy data transfer such as when reading or writing a huge file. This problem is shown in Figure 3.3.1.

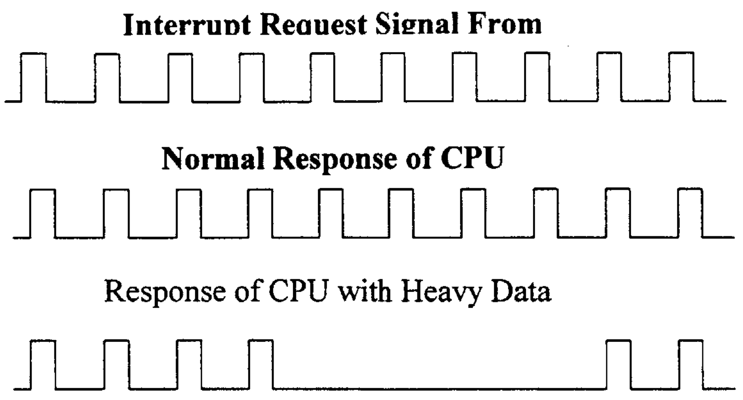

Interrupt Request Signal From
Normal Response of CPU
Response of CPU with Heavy Data

Figure 3.3.1: Disappearance of Interrupt

This problem was observed when the PC was equipped with the EIDE (Enhanced IDE) interface card and EIDE mass storage devices. The intensive experiment was done and the interrupt disappearance time sometimes lasted up to 5 ~ 10 milliseconds. If the control system relied on the this kind of operating system, the machines would go unstable. The interrupt disappearance phenomena are not to be neglected in the precise motion control system. This problem was solved by adopting a SCSI interface card with bus-mastering DMA and mass storage devices based on SCSI. The characteristics of SCSI and also the EIDE interface controller cards are presented in the following sections.

3.3.1 EIDE (Enhanced Integrated Disk Electronics) and SCSI (Small Computer Systems Interface) Drivers

The EDIE interface and the SCSI bus were the two most popular interfaces for computer peripherals recently. The EIDE interface can be found almost only in the PC industry. In contrast, the SCSI bus was designed not only for PCs, but for use in a wide variety of computers ranging from PCs to workstations and even mainframes. As the real time operation of our system shows different behaviors with heavy data access depending on the bus type, it is important to know the basic differences of these two.

EIDE Interface

The EIDE interface is an extension of IDE (Integrated Disk Electronics) interface. This interface was originally developed for the connection between the hard disk and the CPU. As it was expanded to EIDE, access of removable mass storage including the CD-ROM driver was made possible. The IDE Controller is physically embedded in the peripheral unit itself. The only components left on the PC side is an IDE bus adapter, which is composed of simple buffers and a few decoders. As there is no complicated structure on the bus adapter side, the IDE architecture-based system is generally cheaper than that of the SCSI architecture.

It is possible to connect two hard disk drives to one adapter with the IDE interface. If the host bus is an ISA bus, the bus adapter is the IDE adapter, which is composed of simple components. On the contrary, if the host bus is EISA or PCI, the bus adapter must have a relatively complicated architecture. In this case, the main advantage of the IDE interface, which is low cost, is lost. For the same cost, an equally effective SCSI host adapter can be used, which makes it possible to connect a variety of mass storage devices.

The current PCs are composed of a CPU and many integrated chipsets. A chipset consists of several customized VLSIs that integrate many small chips that used to exist in the early age of PCs. For example, the 8259A interrupt controller and 8253 counter chips are all embedded in a customized VLSI. Many of the recent chipsets include the EIDE adapter by default, so that no extra EIDE adapter is needed for the hard disk or the CD-ROM driver. Also, many of the popular chipsets are based on PCI architecture these days. This means that a PCI-EIDE adapter is generally included in the chipsets and no extra cost is needed for it. Although PCI-EIDE architecture is not as powerful as PCI-SCSI architecture, it still provides us with enough capability of hard disk and CD-ROM driver for general purpose use.
Although the embedded PCI-EIDE architecture is quite enough for office environment use, we still need SCSI architecture for stable and robust operation of Windows NT real time control system. There are five classes of commands for the IDE interface. These commands are listed below.

The commands used for data transfer are 1, 2, and 4. In PIO read and write commands, data transfer is managed by the CPU itself. An interrupt occurs whenever a sector of hard disk (usually 512 bytes in the case of a typical PC) is read or written. This means that 2000 interrupts must occur for the data transfer of a 1Mbyte file. This way of data transfer puts heavy load to the CPU during hard disk access. On the contrary, the CPU can be completely free if we use DMA commands for data transfer. Interrupts occur only twice (at the beginning and the end of data transfer) no matter how large the amount of data is. But unfortunately, most of today's IDE or EIDE controllers use PIO read and write commands for data transfer because the overall transfer rate is still faster than using the slow DMA chip (commands 4 of above table) which is embedded in the motherboard. This is due to the high speed clock of the recent CPUs. (For example, the Pentium Pro that was used marks 200 MHz.) But from the standpoint of efficiency, the PIO transfer method is far behind the DMA method.

SCSI Bus

The SCSI bus is not dedicated for the PC, rather, it is an universal definition of a bus for all kinds of computers. It is also different from IDE or EIDE in that it can be used for any kind of peripherals such as a CD-ROM driver or a tape driver, i.e., it is a device independent I/O bus. The original SCSI bus can address up to eight devices. The devices are discriminated by the numbers called SCSI ID. In PC architecture, the host adapter connected to the PC itself is treated as a SCSI device. With this independent architecture, SCSI provides much more flexibility and functionality than IDE. For example, data can be transferred between SCSI devices without any help from the CPU. In this sense, file copy operation between a SCSI hard disk and CD-ROM can be done without any help from the CPU. This is extremely useful in multitasking environments such as Windows NT or OS/2.

If a PC had a SCSI bridge controller and assigned it a SCSI ID, each of the bridge controllers can have eight logical units. In this case each LUN (Logical Unit Number) can represent a separate peripheral device.

3.3.2 Bus Mastering DMA

The problem with the interrupt disappearance occurs when heavy data is accessed from the EIDE hard disk or CD-ROM driver. This is basically due to the fact that the EIDE mass storage device controller uses PIO mode for data transfer, which takes up significant CPU time in the interrupt service level. This problem could be solved by using the peripherals based on the bus mastering concept.

The hard disk drive or the CD-ROM drive must move data between the physical data storage and the computer's RAM. There are several methods for moving data, and the overall performance of the system is significantly affected by which method is adopted for moving data. (This is especially true in the case that other tasks use interrupt capability as its main core resource, as is the case in the full software control system.) .

The most general and least complex method is called Programmed Input / Output (PIO). This has been used since the early generation of the IBM PC. In the PIO method, the CPU itself moves data between the mass storage devices and system memory using interrupt capability. The primary drawback to PIO data transfer is that the CPU must be utilized for each sector of data to be moved. I/O operation is relatively slower than other CPU operations such as computation or data transfer between memory blocks. In this protocol, the CPU transfers a 512-byte sector when the disk controller generates an interrupt request. As a result, to move a data of 1 MB, 20 interrupts are needed to be generated, and the CPU has to repeat going through the same interrupt service routine 20 times. This gives the CPU a significant load so that the CPU sometimes skips the real time control task during heavy data transfer.

One way to overcome this kind of drawback is using the Direct Memory Access (DMA) function. The basic concept is that data transfer is not done by the CPU, but by a dedicated chip that is designed only for data transfer. In this case, what the CPU has to do is to give out a command to the DMA chip to start data transfer between certain areas. After that, the DMA chip takes care of all the data transfer and the CPU does not have to be involved in the transfer job at all, which otherwise lowers the overhead of the CPU.

DMA is not a new concept at all in the computer industry. Rather, it has been used in floppy disk data transfer since the first appearance of IBM-PC. In floppy disk data transfer, a DMA chip, which is typically mounted in the motherboard of the computer is used. (This built-in chip is called "Third Party DMA.") Even the hard disk data transfer was done by this DMA in the very early days when IBM-XT was popular. But from the IBM-AT days the hard disk data transfer began to be done by the CPU itself because the built-in DMA chip was too slow relative to the speed of the CPU. The speed of the built-in DMA chip (originally named 8237A from

Intel) was only 4.77 MHz (7.16 MHz at most) and the data was transferred by 8 bits. This is significantly slower than the speed of Pentium or Pentium Pro today, which goes up to 200 or 300 MHz and has 32 bit of data transfer width. This is why the PIO method has been used for hard disk data transfer from the IBM-AT. The DMA in the PC was almost "forgotten" as a relic of ancient times except that it is used for floppy drive data transfer, and it became a standard to use the CPU itself for data transfer for the hard disk or the CD-ROM driver.

But as the PC world entered into the multitasking environment, the situation changed dramatically. Multiple application programs ran at the same time in a PC, and more and more CPU bandwidth was necessary to satisfy the many programs running simultaneously in one machine. Naturally, programmers and PC manufacturers desperately looked for the ways to off-load the CPU, and the DMA concept became spotlighted again. However, the traditional DMA chip is too slow to be used. Thus, came the concept of "Bus Mastering DMA" or "First Party DMA."

The disk drive controllers that support bus mastering DMA have the ability to move data to and from the system RAM without the help of the CPU or a third party DMA chip. In bus mastering DMA, the data transfer is taken care of mostly by the first party DMA chip which is generally embedded in the peripheral controller itself. The CPU does only the triggering of the data transfer; it does not have to perform any part of the data transfer itself. The bus mastering DMA chip will do everything concerned with data transfer. This is also the same in the case of third party DMA which has existed as far back as the time when the floppy controllers were used. But the DMA chip for the bus mastering operation embedded in the host controllers are generally much faster than the old-fashioned third party DMA chip, because it is developed relatively new. Another reason, which makes first party DMA more favorable for the real time control system, is the relation with the bus cycle in data transfer. In PIO protocol-based data access and first party DMA-based data transfer, two bus cycles are needed for moving a word between peripheral device and the system memory - one for reading and the other for writing. But in the case of the bus mastering DMA, reading or writing associated with peripheral device side is done without any relation to the bus cycle, rather, it is done by the host adapter itself. Consequently the bus cycle needed for moving a word is only one for accessing the system memory in the case of bus mastering DMA. In addition, the system RAM can be accessed using high speed methods like page mode access.

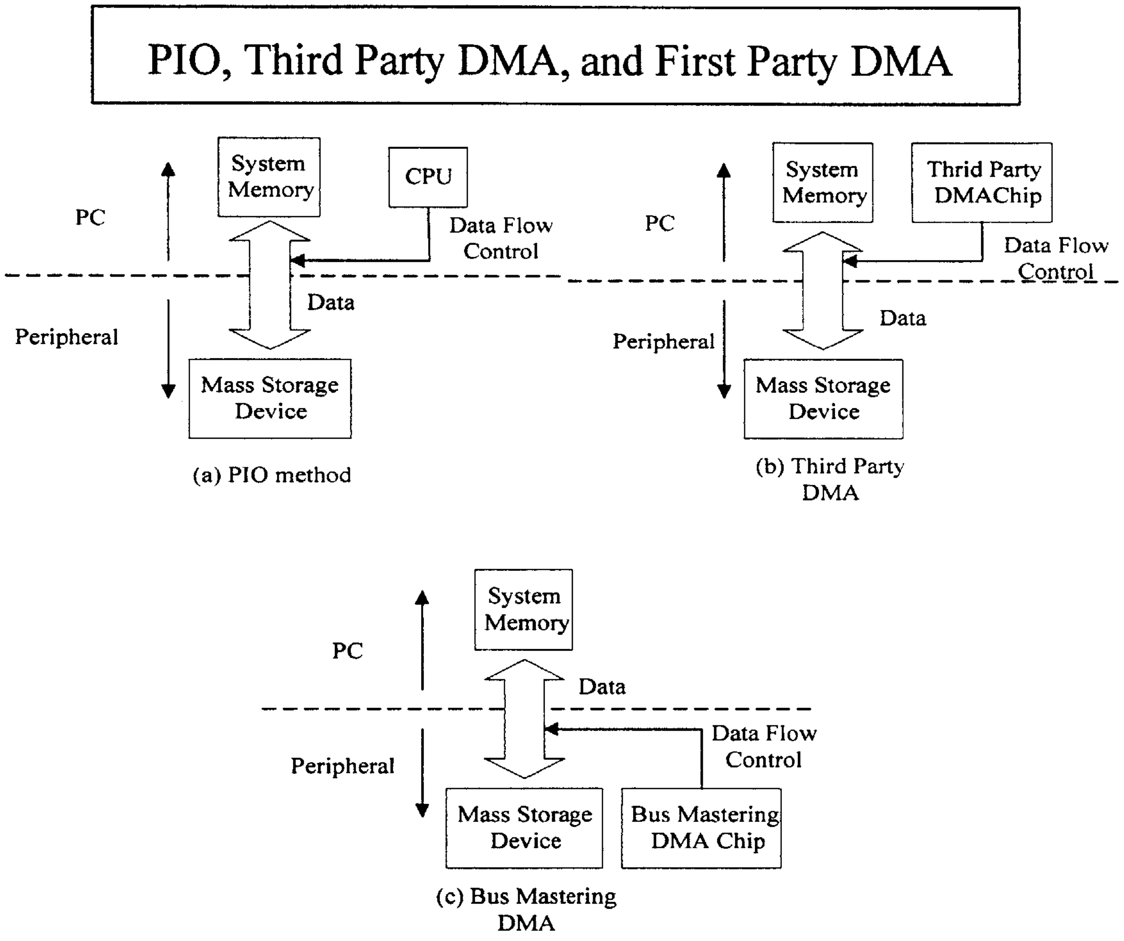


Figure 3.3.2: PIO and DMA

Because of this, mass storage devices with bus mastering DMA can move data much faster than those with PIO or third party DMA. The bus mastering DMA is especially better compared to the PIO method in that the data transfer business is done almost without spending the precious CPU interrupt time. As our real time control system uses interrupt capability significantly, we may encounter a situation in which the control performance may be deteriorated or interfered with during heavy data transfer if we use the PIO based hard disk or CD-ROM driver. By adopting bus mastering DMA, we can efficiently avoid this kind of potential problem. The comparison of three data transfer methods – PIO, third party DMA, and first party DMA – is shown on Figure 3.3.2.

The bus mastering concept can be implemented with both EIDE and SCSI devices. The problem is that current commercial PCs mainly use the PIO mode for data transfer with EIDE devices. Of course, there are a few newly developed main board chipsets that support the bus mastering mass storage device controller (such as Intel's 430FX, 430HX, 430VX, and 440FX chipsets). But main stream chipsets do not support the bus mastering EIDE yet. So even though the chipsets support it, it is of no use because major multitasking operating systems such as Windows NT do not currently support bus mastering for the compatibility with non-bus mastering chipsets. (Though the device drivers for mass storage devices with bus mastering DMA-based chipsets are available for Windows 95 these days, it is hard to say that they are reliable enough yet.) So even though we use a computer with the above chipsets, the data transfer is still done by the PIO protocol under Windows NT.

On the contrary, many of the recent SCSI adapters (for example, Adaptec AHA-2940UW) are equipped with the bus mastering DMA chip in itself. This chip lets the CPU be free during data transfer between a SCSI device and the memory of the PC. This means that the CPU does not have to be tied up with a data transfer job and can do something else. With this kind of host adapter, the interrupt occurs only at the start and at the end of data transfer. (This is similar to the DMA command mode of the IDE interface, which is almost forgotten in the current industry.) At the start of data transfer, the CPU sends the SCSI adapter the commands of the data transfer, the beginning address and the amount of data to transfer. After that, everything is done by the host adapter itself and the CPU is completely free during that time. At the end of data transfer the host adapter generates another interrupt and lets the CPU know that it is finished.

For this real time control system, we certainly need bus mastering capability for the reason stated above. As current EIDE devices seldom support bus mastering, adopting SCSI devices with bus mastering is the best solution for our real time control system. In summary, the SCSI bus adapter, equipped with bus mastering DMA functionality, could be used in the real time software AC servo system. In the next section, this point is proved experimentally.

3.3.3 Counting the Number of Interrupts during Heavy Data Access

To show that using the SCSI bus solves the problem of the interrupt response disappearance problem, the following experiments were carried out. The number of interrupts was counted along with the system clock timing to determine whether there are any interrupts missing. To count the number of interrupt responses of the CPU in a specific time, a program for counting the interrupt responses was developed. This program was run on two PCs with different bus architecture, one with EIDE and the other with SCSI. The first experiment shows the result with the EIDE interface when accessing a file of 40 MB from hard disk. It took about 61 seconds to access the 40 MB data from the EIDE hard disk, and 538444 of first interrupts and 56551 of second interrupts occurred. If no interrupt is to be missing, the first interrupts should be counted to be around 610000 and the second interrupts around 61000. This experiment shows that more than 10 % of the first interrupts are missing and about 8 % of second interrupts are also missing with the EIDE hard disk. The second experiment shows the result with the EIDE CD-ROM driver. This shows that it took about 72.6 seconds for the 40 MB data access and the first interrupts were counted 643637 and the second 70288. These numbers should be 726000 and 72600 respectively if no interrupt was missing, which also shows that around 10 % of the first interrupts and around 4 % of second interrupts were missing during data transfer from the EIDE CD-ROM driver.

The third experiment shows the result of the access of the same amount of data from the SCSI hard disk with bus mastering DMA functionality. It took about 13 seconds for data access and the first interrupts were counted 129253 showing that almost no interrupt request was skipped. The second interrupts were counted 12994 which also shows similar result as the first interrupts. The SCSI CD-ROM driver had a similarly good result.

The performance monitor of the Windows NT explains the advantage of the bus master DMA of the SCSI bus. It is shown that both the interrupt time and the processor time suddenly increases when data access begins in the case of EIDE devices. For the SCSI devices there is very little change (or almost none!) of interrupt time and processor time during data access. This is due to the bus mastering DMA functionality used in the SCSI host adapter. In EIDE devices, the interrupt time jumps up because it uses the PIO reading and writing mode for data access. Consequently, a stable digital control by interrupt operation of Windows NT can be guaranteed even during heavy data access as long as the SCSI adapter with bus mastering DMA is used. This means that our real time control system works with good stability as long as the SCSI architecture is used.

3.3.4 Feasibility Test of Using the Super-DMA Hard Drive

In previous sections of this chapter, it was concluded that a SCSI (Small Computer Systems Interface) hard disk drive is needed for the software AC motor servo system to prevent the interrupts from disappearing. The SCSI drives have a Bus Mastering DMA (Direct Memory Allocation) feature embedded in the system. DMA is a method of allowing data to be moved from one location to another in a computer without intervention from the central processor (CPU). Without the Bus Mastering DMA feature, an interrupt is called while the hard disk drive is being accessed. Therefore, other interrupts do not get processed while the hard disk drive is being accessed. However, the Bus Mastering DMA has the ability to transfer data from the hard disk to the RAM and from the RAM to the hard disk. The CPU only triggers the data transfer and the Bus Mastering DMA chip takes care of the actual data transfer. This is the reason the interrupts do not disappear in the computer with the SCSI drive controller even during a huge file transfer from the hard disk to the RAM or from the RAM to the hard disk.

Bus Mastering DMA is available for another disk controller besides the SCSI. The new chipset from Intel supports Bus Mastering DMA even for EIDE as long as the hard disk drive is compatible with DMA. It is called Super-DMA. Most recently manufactured EIDE hard drives are compatible with Super-DMA. The Super-DMA chipset enables the software servo system to not be restricted to SCSI. The system can have an EIDE hard drive with DMA compatibility at a lower cost.

System Clock (100us)	Current Interrupt Count (100us)	Velocity Interrupt Count (250us)	% Interrupt Loss (100us)	% Interrupt Loss (250us)
202391	174021	71093	14.0%	12.2%
109681	168051	68511	14.3%	12.6%
122175	106928	43551	12.5%	10.9%
110759	95872	38997	13.4%	12.0%

Table 3.3.1: Data Transfer from EIDE Hard Drive without DMA Compatibility

System Clock Duration	Current Interrupt Count (100us)	Velocity Interrupt Count (250us)	% Interrupt Loss (100us)	% Interrupt Loss (250us)
182963	182946	73181	0.0%	0.0%
120172	120130	48055	0.0%	0.0%
101445	101450	40586	0.0%	0.0%
91631	91182	36483	0.0%	0.0%

Table 3.3.2: Data Transfer from EIDE Hard Drive with DMA Compatibility

System Clock Duration	Current Interrupt Count (100us)	Velocity Interrupt Count (250us)	% Interrupt Loss (100us)	% Interrupt Loss (250us)
257169	257108	102843	0.0%	0.0%
163134	163073	65229	0.0%	0.0%
128284	128290	51316	0.0%	0.0%
146210	146268	58507	0.0%	0.0%

Table 3.3.3: Data Transfer from SCSI Hard Drive

To test the feasibility of the Super-DMA, a series of tests were done. The computer used for the experiment was the Dell Dimensions XPS P133MHz with a 32Mb RAM. The exact same setup as the current software servo system was implemented in that computer. An ordinary device driver with an interrupt counting feature was compiled and executed. The device driver program counted both interrupts for the current (100us) and velocity (250us) feedback. The interrupts were counted with respect to the time intervals calculated from the system clock. For example, if the test was executed for 10 seconds, there should be 100,000 current feedback interrupt counts and 25,000 velocity feedback interrupt counts. The actual measurements from the feasibility test are shown in Tables 3.3.1 to 3.3.3.

The interrupt counts were measured while large files were being copied from one directory to another to detect possible interrupt conflicts during the test. The exact same files were used for all the tests to maintain consistency. The test was taken for a long enough period of time, about 10 seconds, that the measurement error is minimized. The tables show that there is up to a 14% interrupt loss in the regular EIDE controller without the DMA compatibility. However, there is virtually no interrupt loss for the EIDE controller with the DMA compatibility (Super-DMA) and the SCSI controller. Therefore, both the EIDE controller with the DMA compatibility and the SCSI controller are adequate for the software AC motor servo system.

3.4 Evaluation of Windows NT as a real time operating system

There are five important points that should be observed in any real time operating system. These points are briefly explained in the following sections and are used to evaluate the full software AC servo system.

3.4.1 Determinism

An operating system is deterministic to the extent that it performs operations at fixed, predetermined times or within predetermined time intervals. When multiple processes are competing for resources and processor time, no system will be fully deterministic. In a real-time operating system, process requests for service are dictated by external events and timings. The extent to which an operating system can deterministically satisfy requests depends, first, on the speed with which it can respond to interrupts and, second, on whether the system has sufficient capacity to handle all requests within the required time. One useful measurement of the ability of an operating system to function deterministically is the maximum delay from the point of the arrival of a high-priority device interrupt request to when servicing begins. In non-real time operating systems, this delay may be in the range of tens to hundreds of milliseconds, whereas in real time operating systems that delay may be a few microseconds or milliseconds.

The Windows NT is basically not a real time operating system. Naturally it is not so deterministic under the preemptive multitasking environment on which most of the user level application programs run. Therefore it is obvious that application programs cannot be used for real time purpose. But as we are running real time tasks with the help of the interrupt, the deterministic character of Windows NT is not that bad, though it cannot be said to be excellent. This is demonstrated in the experiment described in the following section, as it shows good responsiveness of the Windows NT operating system.

3.4.2 Responsiveness (Interrupt latency measurement)

Type of CPU \ OS	DOS	Windows NT
386DX – 33 MHz	30 μs ± 15 μs	N/A
Pentium 133 MHz	6 μs ± 1 μs	9 μs
Pentium Pro 200 MHz	5 μs	6 μs
Pentium II 400 MHz	N/A	3.5 μs

Table 3.4.1: Interrupt Latency Measurement Results

Interrupt latency can be one of the most reliable criteria in determining the responsiveness of an operating system. (As was stated above, it is also a good criteria to see whether an operating system is deterministic or not.) The evaluation of interrupt latency of Windows NT gives us satisfactorily good result as shown in Table 3.4.1.

In the DOS environment, there are quite many fluctuations on interrupt latency, which doesn't seem to be good for periodical sampling. But in Windows NT, fluctuation almost disappears. It seems that it is because Windows NT is scheduling all the threads already, so that the CPU is always ready to accept interrupt requests without any confusion.

Interrupt latency in Windows NT is longer than that in DOS, as expected. But it is not as high as 10 or 20 times which we have been worried about. As the fastest sampling rate which will be used in current feedback is expected to be around 10 μs, interrupt latency of 6 μs in Windows NT with Pentium Pro 200 MHz will not be a significant problem. So, it is verified that using Windows NT as an operating system for real time control seems to be adequate in terms of the interrupt latency problem.

3.4.3 User Control

User control is generally much broader in a real time operating system than in ordinary operating systems. In a typical non-real time operating system, the user either has no control over the scheduling function of the operating system or can only provide broad guidance such as grouping users into more than one priority class. In a real time operating system, however, it is essential to allow the user fine-grained control over task priority. The user should be able to distinguish between hard and soft tasks and to specify relative priorities within each class. A real time system will also allow the user to specify such characteristics as the use of paging or process swapping, what processes must always be resident in main memory, what disk transfer algorithms are to be used, what rights the processes in various priority bands have, and so on.

Windows NT basically does not allow the user to specify the priorities of individual user level application tasks. Even if there are several tricky ways for the programmers to do this Windows NT strongly defends itself from any efforts users make to get into its scheduling jobs. So if we only think about the user level applications, the user controllability of Windows NT is far behind the need for real time performance. But as our system carries out most of its time-critical work in the interrupt service routines, we can specify the priorities of the tasks according to the predefined interrupt request priority levels of the CPU. In addition, Windows NT supports multitasking with inter-process communication tools such as semaphores and events. But it is still impossible for the user to specify use of paging, process swapping, and so on. As a consequence, the user controllability of Windows NT cannot be said to be satisfactory, but still enough for general real time tasks if we use the interrupt capacity.

3.4.4 Reliability

As real time operating system usually control heavy and dangerous machines, reliability is typically far more important for real time operating systems than non-real time operating systems. The Windows NT was designed to run each application in its own processes and cannot read or write outside of its own address space. The operating system data is isolated from applications. Applications interact with the kernel indirectly using well-defined user-mode APIs. Thus it is almost impossible for Windows NT to stop due to the errors caused by user level applications, showing the reliability of this operating system. But still, kernel mode drivers might cause Windows NT to stop during critical operations. So any machine that is controlled by the Windows NT operating system should be equipped with some emergency shutdown devices to prevent any disastrous accidents. In addotopm, Windows NT has a good functionality for failure analysis, which will eventually reduce the unexpected shutdown of the operating system.

3.4.5 Fail-Soft Operation

Fail-soft operation is a characteristic that refers to the ability of a system to fail in such a way as to preserve as much capability and data as possible. For example, a typical UNIX system, when it detects a corruption of data within a kernel, issues a failure message on the system console, dumps the memory contents to the disk for later failure analysis, and terminates execution of the system. In contrast, a real time system will attempt to either correct the problem or minimize its effects while continuing to run. Typically, the system will notify a user or user process that it should attempt corrective action and then continue operation perhaps at a reduced level of service. In the event that shutdown is necessary, an attempt is made to maintain file and data consistency.

Windows NT is basically a kind of UNIX in its architecture. As a result, it behaves similar to UNIX when it detects any problems or corruption - dumping the memory into a file in hard disk, and terminating the system or restarting the whole system. From the viewpoint of later failure analysis, this feature is good for system maintenance, even if it doesn't have such functionality as continuing the process in spite of error detection of kernel, which is one of the necessary functions of a real time operating system. (Windows NT never stops with any failure of user level application. Only kernel level failure can stop Windows NT.) For recovery options, Windows NT performs four jobs before shutdown in emergency. Writing an event to the system log, sending an administrative alert, writing a debugging information to a file, and automatic rebooting. The user can enable or disable these functions selectively.

Chapter 4
Time Budget and I/O Speed Issues

4.1 Overview

In the previous chapter, the Windows NT-based software AC servo control system was presented. The main feature of the system is that all the computations and operations including multi-layered feedback controls, the three phase commutation, d-q axis control algorithm with decoupling control, and digital PWM generation are performed only by the PC's host CPU. One critical question is how much CPU time is occupied by the real-time controller, embedded in the kernel of the Windows NT operating system, and whether the CPU still has enough time for other operations such as disk drive access and GUI operations.

Therefore, one of the most important issues in the full software AC servo controller is the time budget. The full software controller uses the CPU of a PC for the logic and the calculation instead of using some DSP chips or electronic circuits. In order to guarantee the real time control characteristic, hard interrupts are called in every sampling period. The hard interrupts take away the computing power of the CPU entirely. The PC may crash when the CPU load is too great for it to handle. Therefore, determining the right maximum load for a specific CPU is essential in the full software controller. Knowing the time budget could help in determining the current sampling rates or the number of axes a specific PC could control.

Another reason for determining the time budget of a system is to obtain the required CPU load for each process such as I/O reading, logic, and calculations. From the time budget, the bottleneck of the system could be identified. The system uses ISA bus for I/O, which is shown to be the bottleneck in the system.

In this chapter, the time budget of the system with a commercial board is determined. The time budget shows that the old system is not capable of controlling multiple axes at the high current sampling rates, which is often required in the robotics and NC machine industries. In the later sections, the development of a new interface board is presented with its time budget to control multiple axes at the high sampling rates.

4.2 Time Budget Formulation

In the previous section, we proposed the Windows NT-based software AC servo control system. The main feature of the system is that all the computations and operations including multi-layered feedback controls, the three phase commutation and digital PWM generation are performed only by the PC's host CPU. One critical question is how much CPU time is occupied by the real-time controller, embedded in the kernel of the Windows NT operating system, and whether the CPU still has enough time for other operations such as disk drive access and GUI operations. The objective of this section is to show that the Windows NT-based software AC servo can be implemented on a Pentium PC for multi-axis control applications such as robot control. Since the current feedback and PWM computations are the most time consuming, we will focus on the time budget for these computation at the highest sampling rate.

Let us first define the following parameters:

t_w : Time required for one writing to I/O address
t_r : Time required for one reading from I/O address
t_m : Time required for one multiplication
t_d : Time required for one division
t_{other} : Time required for other algebraic computation in Control Computation (addition, subtraction, jumping, etc)
R : Reading and Writing speed ratio between USHORT type and UCHAR type
N : Number of Axes

One complete cycle of current feedback and PWM computations includes many operations; time budget for each operation is described as follows.

<u>Current Feedback</u>

[1] Total Interrupt Latency : T_i

Latency for getting into interrupt routine + ending the interrupt routine

[2] A/D Conversion : T_{ADC}

(i) Initialization (N axes) : N*(2 writes + 2 reads)

(ii) Conversion of current feedback (N axes) : N*(2 reads)

$T_{ADC} = 2*R*N*t_r + N*(2t_w + 2t_r) = 2*R*N*(t_w + 2t_r)$

[3] PWM signal generation : T_p

4 writes for each phase for each axis (3 phases and N axes)

$T_p = 4t_w*3*N = 12N*t_w$

[4] Control Computation (PI Control for N axes) : T_c

$T_c = N*(8t_d + 9t_m + 26t_{other})$

[5] Time required for other commands (General assignments, jumping, etc.) : T_{etc}

$T_{etc} = N*(5t_d + t_m + 40t_{other}) + t_d + t_m + 7t_{other}$

Total time required for one current feedback for N axes : $T_{current}$

$T_{current} = T_i + T_{ADC} + T_p + T_c + T_{etc}$

$T_{current} = (12t_w + 2R*t_w + 4R*t_r + 13t_d + 10t_m + 66t_{other})*N + t_d + t_m + 7t_{other} + T_i$

Position and Velocity Feedback

[1] Total Interrupt Latency : T_i

Latency for getting into interrupt routine + ending the interrupt routine

[2] Encoder Reading : $T_{encoder}$

(i) Initialization (N axes) : N*(1 writes)

(ii) Encoder Reading (N axes) : N*(3 reads)

$T_{encoder} = R*N*(t_w + 3t_r)$

[3] Commutation (N axes) : T_c

$$T_c = N*(13t_d + 17t_m + 54t_{other})$$

[4] Time required for other commands (General assignments, jumping, etc.) : T_{etc}

$$T_{etc} = N*t_{other} + 2t_{other}$$

Total time required for one current feedback for N axes : $T_{pos/vel}$

$$T_{pos/vel} = T_i + T_{encoder} + T_c + T_{etc}$$

$$T_{pos/vel} = (R*t_w + 3*R*t_r + 13t_d + 17t_m + 55t_{other})*N + 2t_{other} + T_i$$

In the case of the Pentium Pro 200 MHz CPU, with 64MB RAM, 2GB Hard Disk, 256K Cache, the corresponding values are as follows:

- $T_I = 12.0$ μs
- $t_w = 1.2$ μs
- $t_r = 0.5$ μs
- $t_m = 0.09$ μs
- $t_d = 0.1$ μs
- $t_{other} =$ 0.01μs
- $R = 1.7$

Note that these values are approximate values. The total amount of time for each of the current control and position/velocity feedback control is

$$T_{current} = (12t_w + 2R*t_w + 4R*t_r + 13t_d + 10t_m + 66t_{other})*N + t_d + t_m + 7t_{other} + T_I$$
$$= 24.75N + 12.26\ (\mu sec)$$

$$T_{pos/vel} = (R*t_w + 3*R*t_r + 13t_d + 17t_m + 55t_{other})*N + 2t_{other} + T_I$$
$$= 7.97N + 12.02\ (\mu sec)$$

The equations indicate that the interrupt time increases linearly with the number of axis for both the current feedback and the position/velocity feedback.

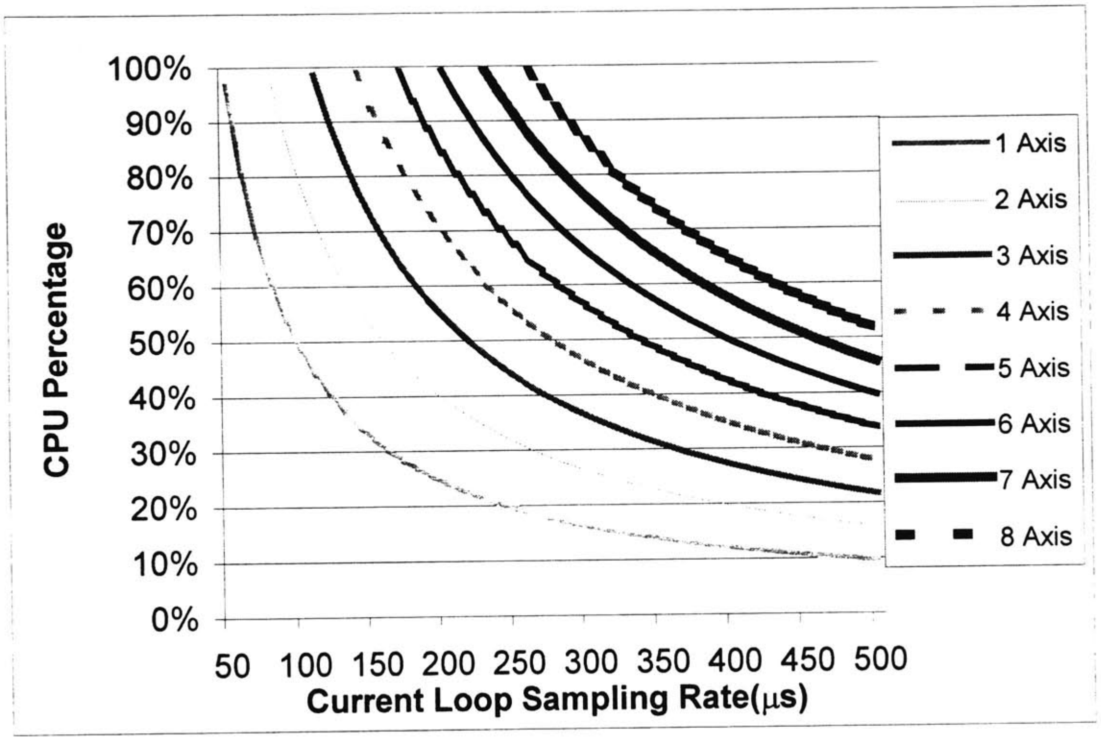


Figure 4.2.1: CPU Load for Pentium Pro 200MHz

There are a number of variables that can be adjusted to meet the goal depending on the desired performance, configuration of the computer, and number of axes. In this time budget, the configuration of the computer was fixed to that of the Pentium Pro 200 MHz. The interrupt times for the current and the position/velocity feedback are also fixed. The actual values are shown in the above equations. However, the sampling frequency and the number of axes being controlled can be varied.

Let us consider the relationship between the number of axes to control and the required sampling period. Since the industrial convention is that the current feedback requires ten times more frequent feedback than the position/velocity feedback, the total equivalent interrupt operation time T_{total} is formulated as:

$$T_{total} = T_{current} + T_{pos/vel}/10$$

For example, if the number of axes is six, the total time required to operate the multi-layered AC servo control will be

$$T_{current}\,(N=6) = 191\mu s$$

Therefore, the current sampling time must be at least 250 microsecond or slower. Figure 4.2.1 shows the CPU load in terms of percentage. Each curve represents the number of axes given the current sampling rates. In order to control multiple axes, the current feedback sampling rates must be low. If only one axis is to be controlled, then the current feedback could be set to a high frequency.

This result shows that the six axis control application could not be implemented with a PC with the Windows NT-based AC servo control system if the current sampling rate is set to 100 μsec. In order to control six axes, the current sampling rate must be set to 250 microseconds or higher. However, the current sampling rate of 250 microseconds is too slow for a high performance controller. The sampling should be about 100 microseconds or faster. For this system, two axes could be controlled simultaneously at 100 microseconds current sampling rate. However, most robots or NC machines have more than two axes. Therefore, the controller must be able to control more axes simultaneously without giving too much load to the CPU. In the next section, possible improvement methods are discussed.

4.3 Time Budget Improvement

In the previous section, the time budget showed that the current controller is able to control only up to two axes simultaneously. The performance in the time budget must improve much more in order to control the NC machines or the robots, which have at least four axes. In this section, the possible improvement method is discussed and the new time budget is presented with the improvement.

There are two possible ways to improve the time budget. The easier and the general way is to use a faster CPU available. The calculation and the logic would be handled much faster when the CPU is upgraded. Although it is an easier way to improve the time budget, a more fundamental issue must be addressed. The computing power of the current CPUs are not the bottleneck in the time budget. The computing power of the CPU is so much faster than the conventional microprocessors used in the industry. The bottleneck is in the I/O access. The current system has an ISA bus for the interface and each access in the ISA bus takes more than one microsecond. Therefore, the time budget could be improved significantly when there are less ISA bus accesses.

There are a number of readings and writings in the AC servo motor controls. The encoder and the current feedback signals and the duration on the PWM are needed to be written and read for each axis. Many readings and writings for the channel selections and the triggering are also needed. In order to minimize the I/O readings and writings, any I/O access other than actual reading and writing of data must be eliminated.

Although minimum I/O access is desired for the improvement in the time budget, the commercial boards require the channel selection and the triggering because they have to be general. Therefore, a new board that could bypass all the channel selection and the triggering is developed. The architecture of the new board is shown in Figure 4.3.1.

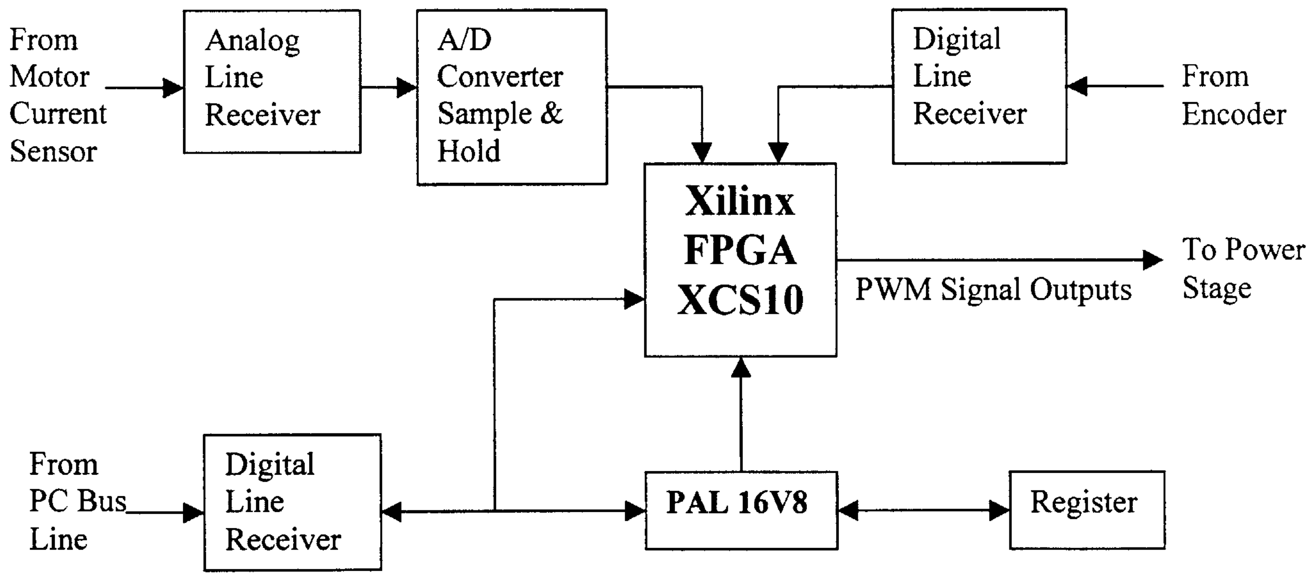


Figure 4.3.1: Block diagram of FPGA-based interface board

The board has a reconfigurable interface architecture by using a field programmable gate array (FPGA) chip. This board is closely linked with the full software servo controller. It is dynamically programmed by the software in the controller and provides much flexibility along with the full software servo controller. The board is also programmed with the auto selection logic to minimize the I/O readings and writings. The software controller knows the order of its readings and writings, therefore, it programs the board accordingly to auto-select and trigger at the right time.

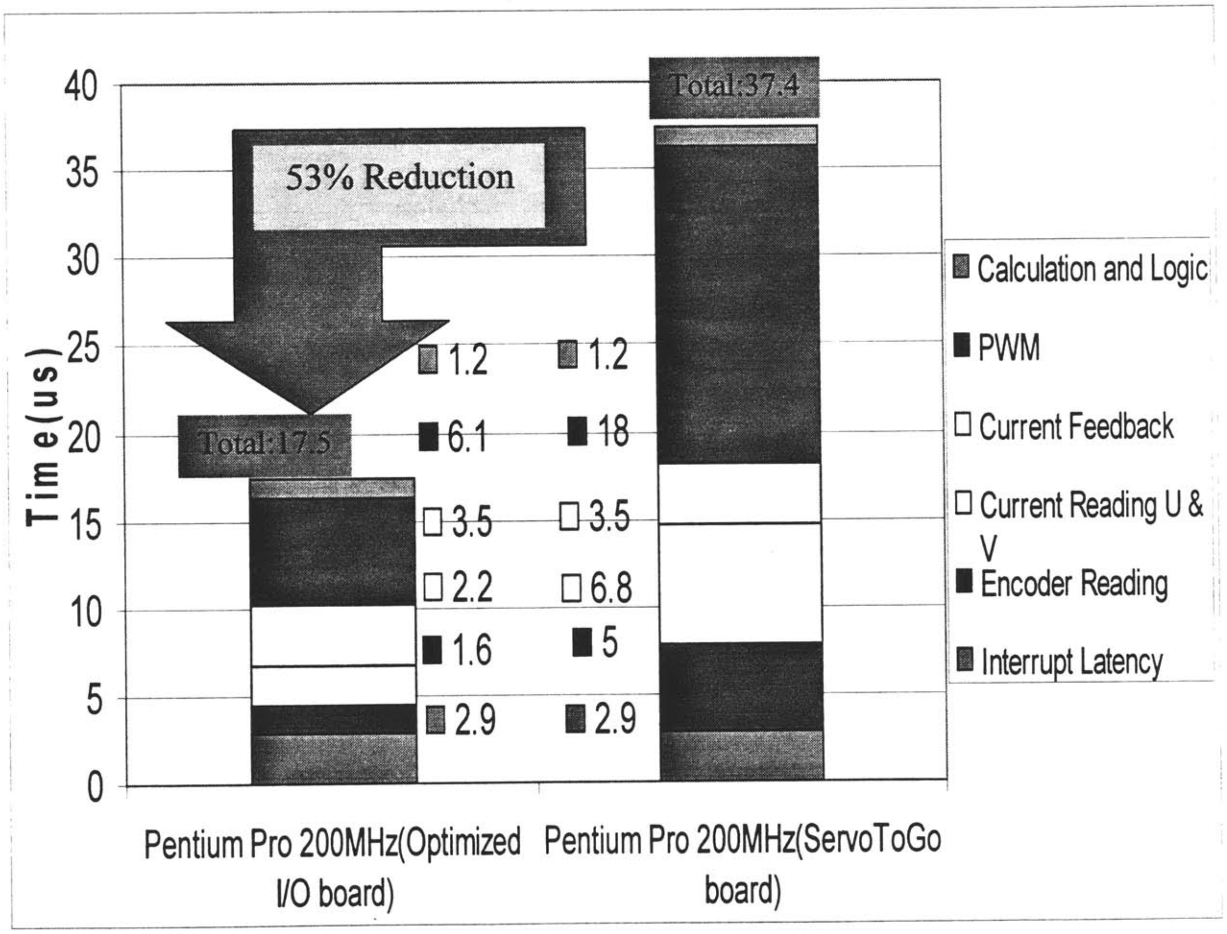


Figure 4.3.2: Time Budget Improvement with the FPGA board in Pentium Pro 200MHz

The FPGA board's auto-selection logic eliminates all the unnecessary readings and writings, thus, improving the time budget of the system. Figure 4.3.2 shows the time budget improvement with the FPGA board in the Pentium Pro 200MHz. The time budget is broken into different parts in terms of the functionality in the controller. The time it takes to control one axis at a 100 microsecond current sampling rate was reduced to 17.5 microseconds from 37.4 microsecond, which is about a 53% reduction. It clearly shows that the I/O access in ISA bus is the true bottleneck in the system and optimizing the board saves much time in the time budget.

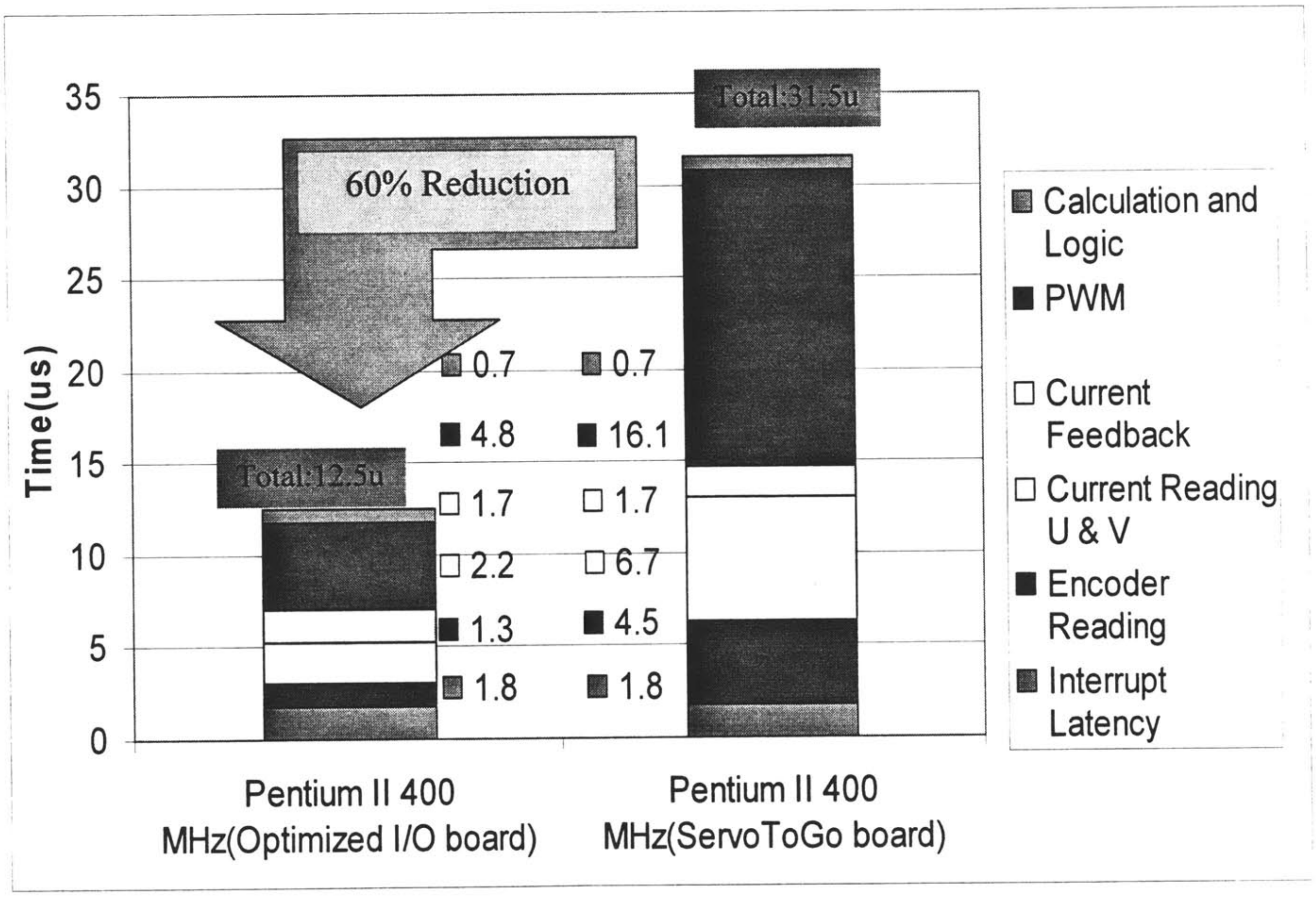


Figure 4.3.3: Time Budget Improvement with the FPGA board in Pentium II 400MHz

The system with a faster CPU has even greater improvement when the FPGA board is used instead of the ordinary commercial board. Figure 4.3.3 shows the time budget improvement with the FPGA board in Pentium II 400MHz. The time it takes to control one axis was reduced from 31.5 microseconds to 12.5 microseconds, which is about a 60% reduction in the time budget. It is larger than the reduction rate of the Pentium Pro 200MHz by 7%. The I/O access is greater bottleneck for the fast CPU than the slower CPU, because the slower CPU has a bigger portion of the time budget for the logic and the calculations. Therefore, reducing number of I/O access would give a higher improvement rate in the faster CPU.

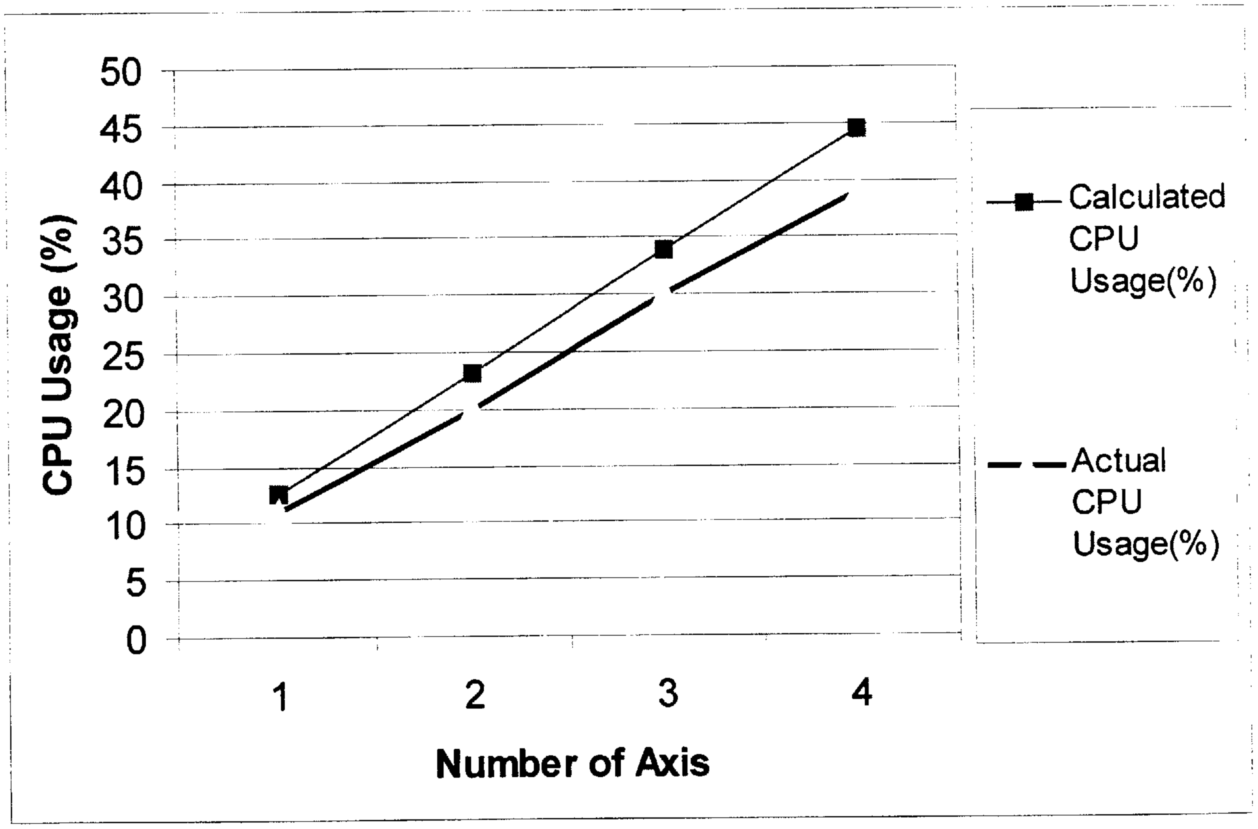


Figure 4.3.4: Comparison between the actual and the calculated CPU usage at 100 microsecond current sampling with a Pentium II 400 MHz computer

The time budget and its improvement are the calculated value. The conservative estimations were used for the calculations in order to any crashing of the PC. In Figure 4.3.4, the actual and the calculated CPU usage at 100 microsecond current sampling rate with Pentium II 400 MHz were compared. The conservative estimation calculation were lower than the actual CPU usage than the calculations.

Figure 4.3.4 shows that four axes could be controlled simultaneously in less than 40 microseconds allowing plenty of CPU time for other use. Each axis takes less than 10 microseconds except the first axis due to the interrupt latency in Windows NT. At a 100 microsecond sampling rate, four axis NC machines or six axis robots could be controlled without too much load on the CPU. In section 4.4, a six axis robot is controlled at a 100 microsecond sampling rate and the monitored CPU load was discussed.

4.4 Six Axis Robot Control

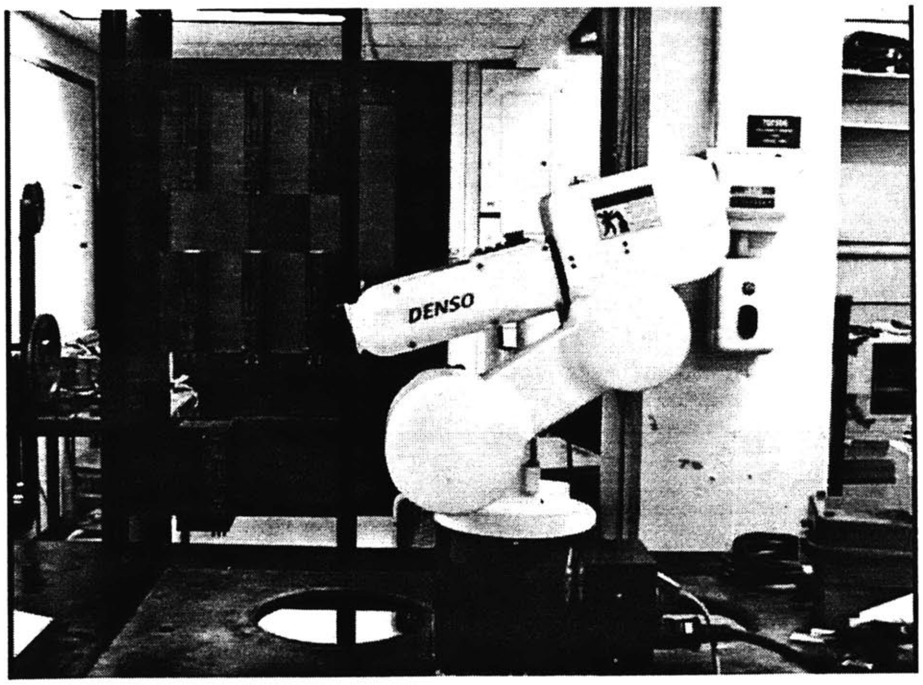
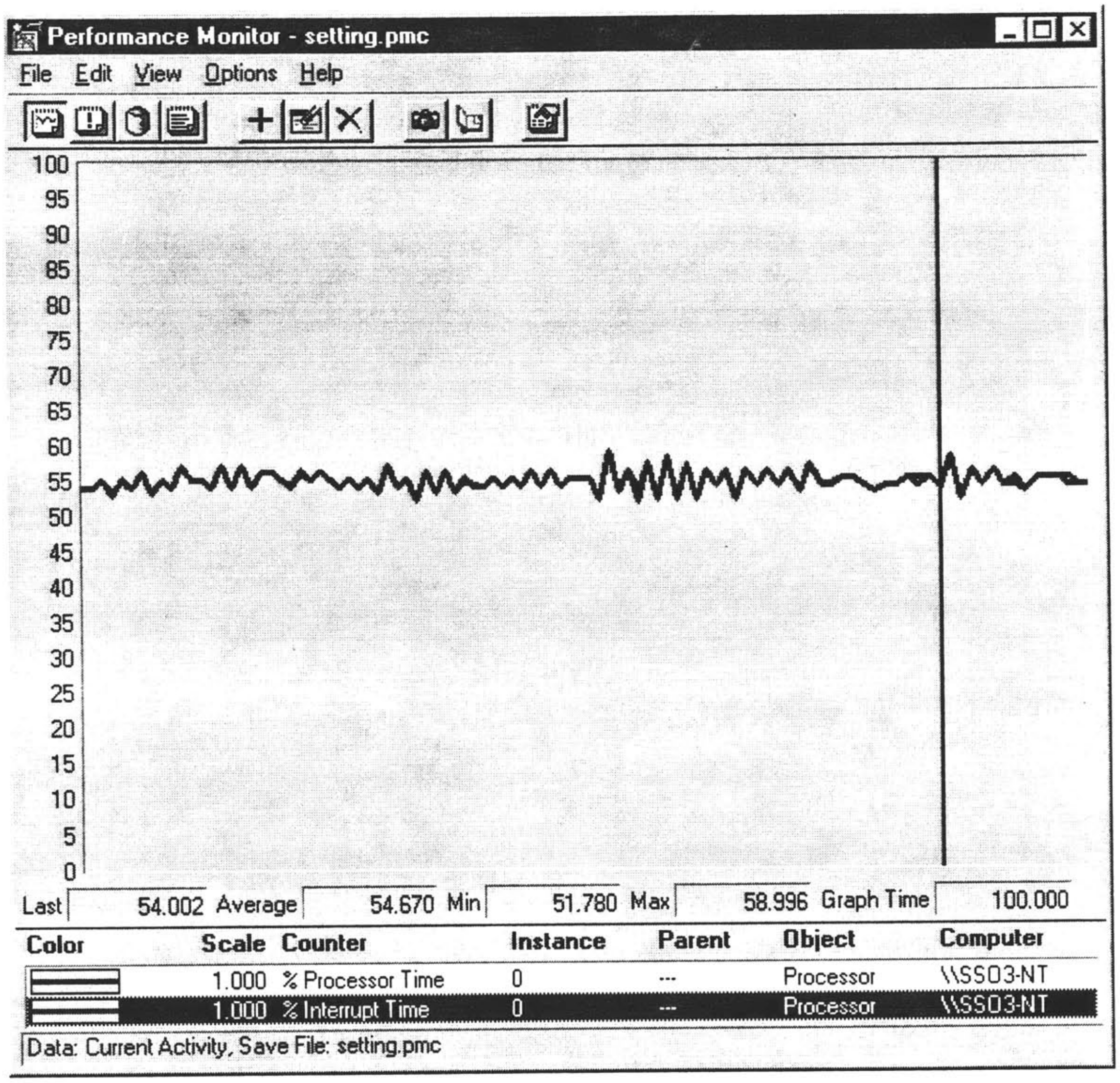


Figure 4.4.1 : Denso Six Axis Robot


Performance Monitor - setting.pmc
File Edit View Options Help
100
95
90
85
80
75
70
65
60
55
50
45
40
35
30
25
20
15
10
5
0

Last 54.002 Average 54.670 Min 51.780 Max 58.996 Graph Time 100.000

Color	Scale	Counter	Instance	Parent	Object	Computer
▭	1.000	% Processor Time	0	---	Processor	\\SS03-NT
▭	1.000	% Interrupt Time	0	---	Processor	\\SS03-NT

Data: Current Activity, Save File: setting.pmc

Figure 4.4.2 : CPU Load Monitoring

Six axis robot control is feasible at fast sampling rates due to the improvement in the interface board design, thus, resulting in a more efficient time budget. The new full software controller was implemented on a Denso six axis robot, Figure 4.4.1, which has an AC servo motor in each joint. The robot was controlled in joint space to reduce the control effort because the goal of the implementation was to show the feasibility of using the CPU for a six axis control. Two FPGA boards were used because each FPGA board interfaces only up to four axes. The base clocks were synchronized. Kollmorgen power supply and power amplifiers were used.

The CPU load was monitored and plotted in Figure 4.4.2, while all six axes were controlled. It used about 55% with a 100 microsecond Current sampling rate. For five axis control, the CPU load was 46%.

4.5 Conclusions and Recommendations

The time budget and its improvement are presented in this chapter. The time budget is one of the most critical issues in the full software AC servo controllers. The host CPU must be able to handle all the logic and calculations as well as other application programs with the Windows graphic user interface. The old system with the ordinary commercial ISA bus board is not suitable for the high performance robot or NC machine controllers. The time budget shows that the current sampling frequency must be low in order to control multiple axes despite the need for high current sampling frequency in the high performance controllers.

The problem with the time budget is analyzed and the solutions are presented. The bottleneck of the time budget in the system is the I/O access in the ISA bus. The ISA bus has a very slow data transfer rate compare to the speed of the recent CPUs. A new board with a field programmable gate array (FPGA) chip is developed with the intention of optimizing the time budget. The new board minimizes the number of I/O readings and writings by eliminating all the unnecessary channel selections and triggering. The new system with the FPGA board showed a 53% CPU load reduction in Pentium Pro 200MHz and a 60% reduction in Pentium II 400MHz. The actual time budget is also presented in this chapter. The time budget shows that the four axis control at a 100 microsecond current sampling rate used less than 40% of the host CPU processing time. Therefore, a four axis NC machine could be controlled with a single PC at the desired current sampling rate. A six axis robot controller, at a 100 microsecond current sampling rate, took about 55% of the processing time of the host CPU, which leaves plenty of time for other applications and the graphic user interface.

The time budget could be optimized even further if a PCI board is used. The current system has minimized the number of I/O readings and writings. However, the I/O readings and writings still take up the majority of the time budget. If the PCI bus is used instead of the ISA bus the I/O would be much faster and the time budget could be improved up to 300% compared to the currently optimized ISA bus system. In that case, slower PCs could be used or more axes could be controlled by a single PC. The advantage in the time budget could also be used to increase the sampling frequency and implementation of the advanced control algorithms utilizing the flexibility of the full software AC servo system.

Chapter 5
AC Servo Motor Control

5.1 Overview

This chapter begins with the basic structure of the AC servomotor and the local feedback algorithm which is most widely used in the AC servo industry. Local feedback is the easiest feedback algorithm, especially for the regular analog AC servo amplifier. Therefore, local feedback is used, although there are many drawbacks in using the local feedback control algorithm.

This chapter develops mathematical models for other control algorithms such as d-q transformation and decoupling control, utilizing the flexibility and the openness of the software servo controller developed by the author. First, the Clarke transformation and the Park transformation of the equation of the AC motor were derived. Using the transformed equation of the motor, the direct and quadrature axis control (d-q control) algorithm was developed. The d-q control directly regulates the torque and the iron losses in the AC servo motor rather than regulating each phase current in local phase feedback control. Therefore, d-q control provides more accurate torque for the higher level control algorithms, such as velocity and position control.

In addition to d-q control, the decoupling control is derived and implemented. The decoupling control compensates for the non-linearity of the AC servo motor caused by mutual inductance of the motor. It also compensates for the back EMF of the AC servo motor allowing a faster bandwidth for the current controller. The data from all the different algorithms are presented in this chapter and the performance of each control is compared and discussed.

5.2 Review of Control Algorithms

5.2.1 Basic structure of AC servo motor

Most servo motor industries use the three phase local feedback control for the AC servo controller because they use electronic circuits or digital signal processing chips as a controller. The conventional controllers do not provide flexibility. However, the software AC servo controller can change the entire control algorithm just by compiling different source codes. Therefore, many control algorithms can be implemented easily.

In this chapter, three different AC servo motor control algorithms will be introduced. The first one is the three phase control algorithm. The second one is the direct-quadrature axis control algorithm and the *d-q* transformation. The last one is the Back EMF compensation control. In section 5.3, the experimental data from each control algorithm will be presented and discussed.

5.2.2 Three Phase Current Control Algorithm

The three phase current control is the most widely used algorithm in the servo motor industry. The corresponding model and the relationship between voltage and current of a brushless servomotor is as follows:

$$\begin{bmatrix} Uu \\ Uv \\ Uw \end{bmatrix} = \begin{bmatrix} R+pLa & -\frac{1}{2}pMa & -\frac{1}{2}pMa \\ -\frac{1}{2}pMa & R+pLa & -\frac{1}{2}pMa \\ -\frac{1}{2}pMa & -\frac{1}{2}pMa & R+pLa \end{bmatrix} \begin{bmatrix} iu \\ iv \\ iw \end{bmatrix} + \begin{bmatrix} eu \\ ev \\ ew \end{bmatrix}$$

$$p = derivative = \frac{d}{dt}$$

In this equation, U_u,U_v, and U_w are the voltages of each phase coil, i_u, i_v, and i_w are the currents flowing through each phase coil, and e_u, e_v, and e_w are the Back EMF voltages induced by the rotation of the rotor. R, L_a, and M_a are the resistance, inductance, and mutual inductance of the coils respectively. The diagram of the model is shown in Figure 5.2.1.

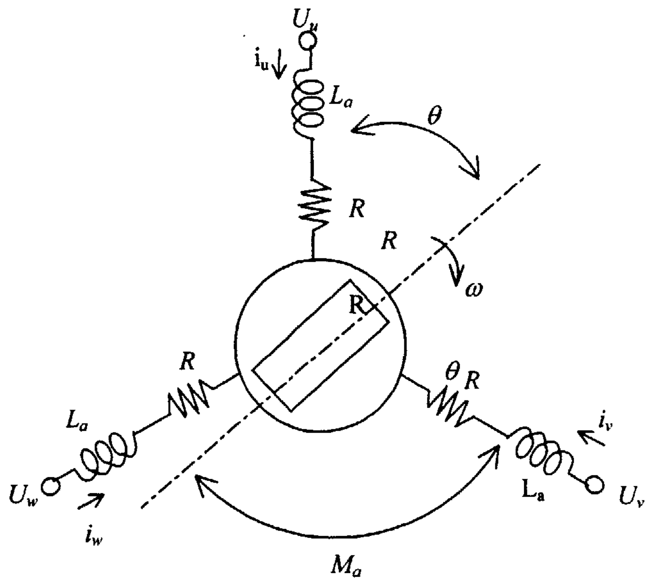


Figure 5.2.1: Three phase model of AC servomotor

Figure 5.2.2 shows the block diagram of the direct three-phase control method. In this control algorithm the current in the *U* phase and *V* phase are directly controlled. Phase *W* is usually not directly controlled because $i_u + i_v + i_w = 0$, therefore, the *W* phase can be indirectly controlled when the other two phases are controlled. The reference current i_{ud} and i_{vd} are inverse *d-q* transformed from i_{qd}, which is the torque command. The formulation of the *d-q* transformation will be discussed in the next section. The

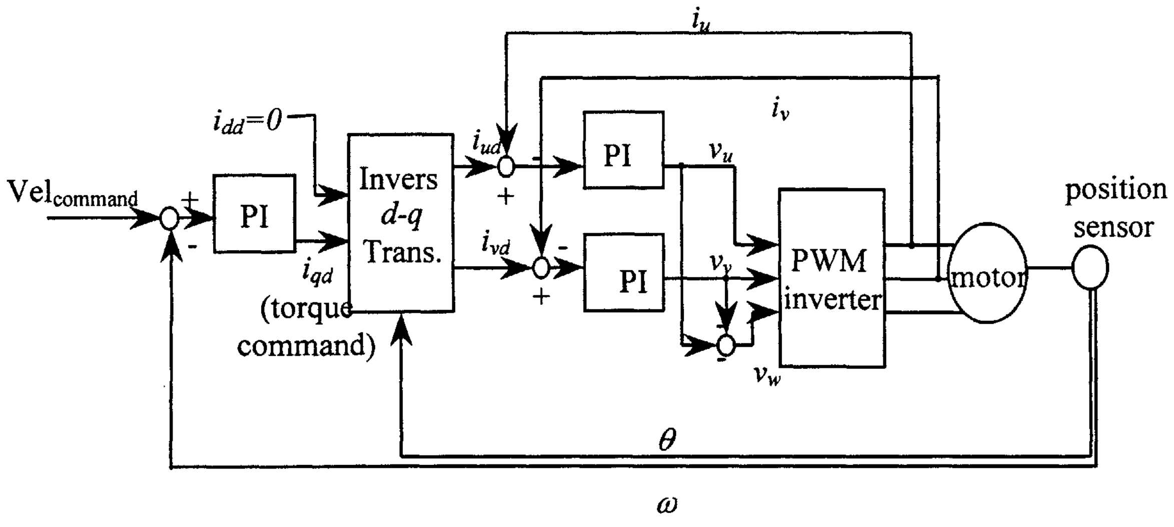


Figure 5.2.2: Three phase local feedback current control of AC servomotor

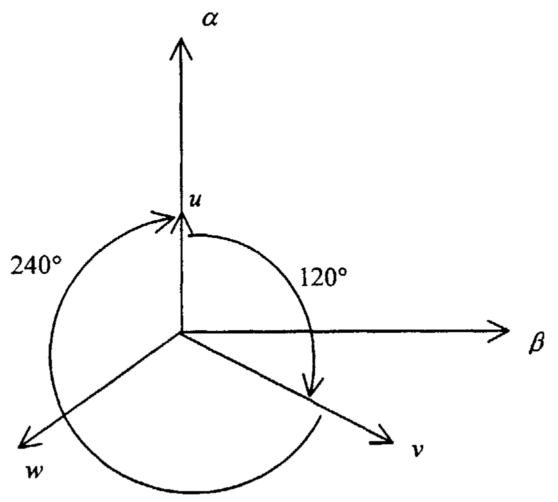
algorithm for PI control is as follows:

$$V_u = K_p * (i_{ud} - i_u) + K_i * \sum (i_{ud} - i_u) * T_s$$

In the equation, T_s is the sampling time and K_p and K_i are the proportional and integral gains. PI control is chosen here to eliminate the steady state error of the system. However, the most significant drawback of the direct three phase control is phase error. Phase error is inevitable even with the PI control due to the nature of the alternating current feedback control.

5.2.3 Direct – Quadrature (*d-q*) Axis Control

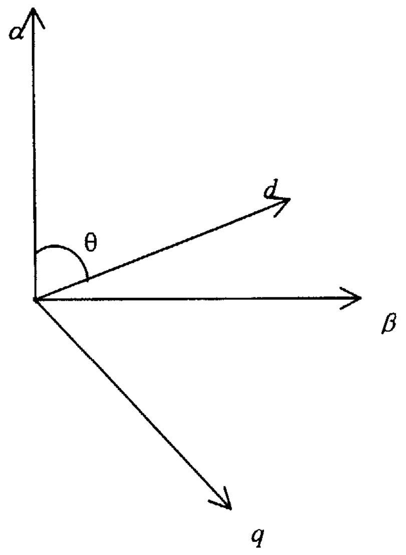
Alternating current flows inside a motor. The alternating current (sine wave), however, can be regarded as direct current by having axes (*d* and *q*) that rotate synchronously with the alternating current. The relative velocity then becomes zero, making the mathematical model simpler. This is called *d-q* transformation.



$$\begin{bmatrix} i_\alpha \\ i_\beta \\ i_z \end{bmatrix} = \sqrt{\frac{2}{3}} \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} i_u \\ i_v \\ i_w \end{bmatrix}$$

Figure 5.2.3: Transformation between α,β and U,V,W Axes

Figure 5.2.3 shows the relationship between the α,β and the U,V,W axes. The α and β axes are used as the intermediate step to the *d-q* axis transformation. Figure 5.2.4 shows the relationship between the α, β and d-q axes. These two relationships are combined to make the d-q transformation. The formulation is derived below. From the d,q to U,V,W relationship, the d,q to U,V relationship can be obtained because $i_u + i_v + i_w = 0$.



$$\begin{bmatrix} id \\ iq \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} i\alpha \\ i\beta \end{bmatrix}$$

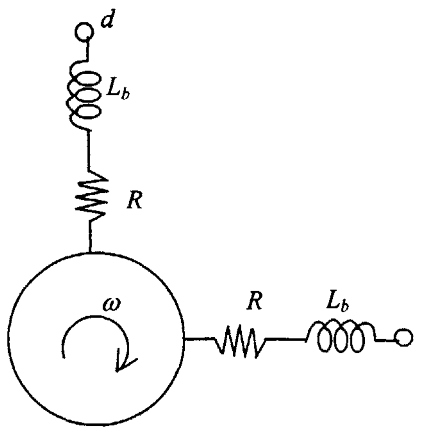
Figure 5.2.4: Transformation between α,β and d-q Axes

$$\begin{bmatrix} id \\ iq \end{bmatrix} = \sqrt{\frac{2}{3}} \begin{bmatrix} \cos\theta & \cos(\theta+120) & \cos(\theta+240) \\ \sin\theta & \sin(\theta+120) & \sin(\theta+240) \end{bmatrix} \begin{bmatrix} iu \\ iv \\ iw \end{bmatrix}$$

Since $i_u+i_v+i_w=0$

$$\begin{bmatrix} id \\ iq \end{bmatrix} = \sqrt{2} \begin{bmatrix} -\sin(\theta+60) & -\sin\theta \\ -\cos(\theta+60) & \cos\theta \end{bmatrix} \begin{bmatrix} iu \\ iv \end{bmatrix}$$

Using this transformation, the d-q axis control algorithm was implemented to improve the control performance. Unlike in the three phase control, the current in the d and q axes is not alternating because the d-q axis rotates relative to the rotor position. Therefore, PI control with proper gains can eliminate the steady state errors. Also, a Back EMF compensator can be designed to generate the desired torque at a high angular velocity. Figure 5.2.5 shows the model of the brushless servo motor in the *d-q* axis. The dynamic equation of the brushless servo motor is also described.



$$\begin{bmatrix} V_d \\ V_q \end{bmatrix} = \begin{bmatrix} R+pL_b & -\omega L_b \\ \omega L_b & R+pL_b \end{bmatrix} \begin{bmatrix} i_d \\ i_q \end{bmatrix} + \begin{bmatrix} 0 \\ \omega \Phi_b \end{bmatrix}$$

$$p = derivative = \frac{d}{dt}$$

Figure 5.2.5: *D-Q* axis model of brushless servo motor

In the dynamic equation, V_d and V_q are the voltages in the *d* and *q* axes, and i_d and i_q are the corresponding current in the *d* and *q* axes. L_b is the phase inductance, ω is the rotor angular velocity, Φ_b is the magnetic flux constant, and *R* is the armature resistance. In this equation, V_q has a Back EMF term, which would prevent V_q from reaching the desired voltage. Back EMF is a disturbance. However, it can be compensated if the voltage drop by the Back EMF interference is estimated. The control block diagram is shown in Figure 5.2.6. The control algorithm for PI control is shown below.

$$V_q = K_p * [(i_{qd} - i_q) + \frac{1}{Ti} * \sum (i_{qd} - i_q) * T_s]$$

In this equation, K_p, T_i, and T_s represent the proportional gain, the integral time constant, and the sampling rate respectively.

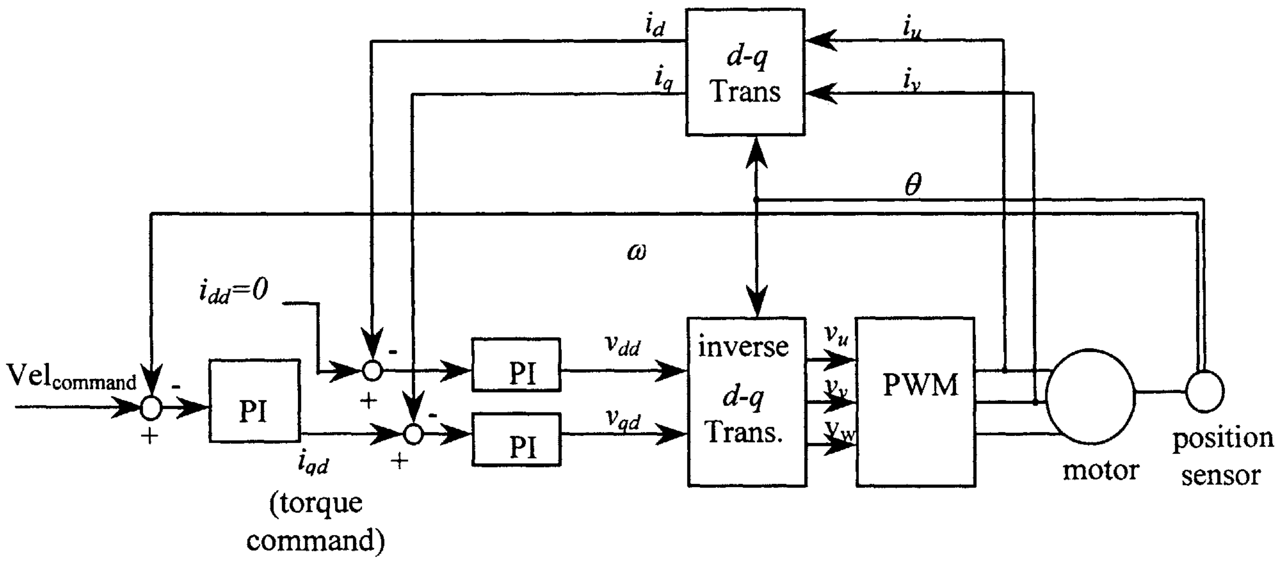


Figure 5.2.6: *D-Q* axis feedback control

5.2.4 Back EMF Compensation

As discussed in Section 5.2.3, Back EMF causes a voltage drop as the rotor angular velocity increases. The voltage drop eventually causes difficulty generating the desired torque. However, the current feedback must provide the desired torque at any angular velocity. Superior performance of the velocity response can be expected when the current feedback provides a good torque source. Therefore, the disturbance must be compensated for by a better brushless servo motor control performance. In the dynamic equation of the brushless AC servo motor shown below, each term is cross-coupled with each other.

$$p\begin{bmatrix} i_d \\ i_q \end{bmatrix} = \begin{bmatrix} -\frac{R}{L} & \omega \\ -\omega & -\frac{R}{L} \end{bmatrix} \begin{bmatrix} i_d \\ i_q \end{bmatrix} + \frac{1}{L}\begin{bmatrix} v_d \\ v_q \end{bmatrix} - \frac{1}{L}\begin{bmatrix} 0 \\ \omega\phi \end{bmatrix}$$

Therefore, it is favorable to decouple each term by adding the decoupling terms to eliminate the coupling terms. The decoupling control algorithms are shown below.

$$v_d = v'_d - \omega L i_q$$

$$v_q = v'_q + \omega(\phi + L i_d)$$

Figure 5.2.7 shows the block diagram of the control algorithm of the Back EMF. This control algorithm with the decoupling control algorithm should provide the desired torque and any angular velocity.

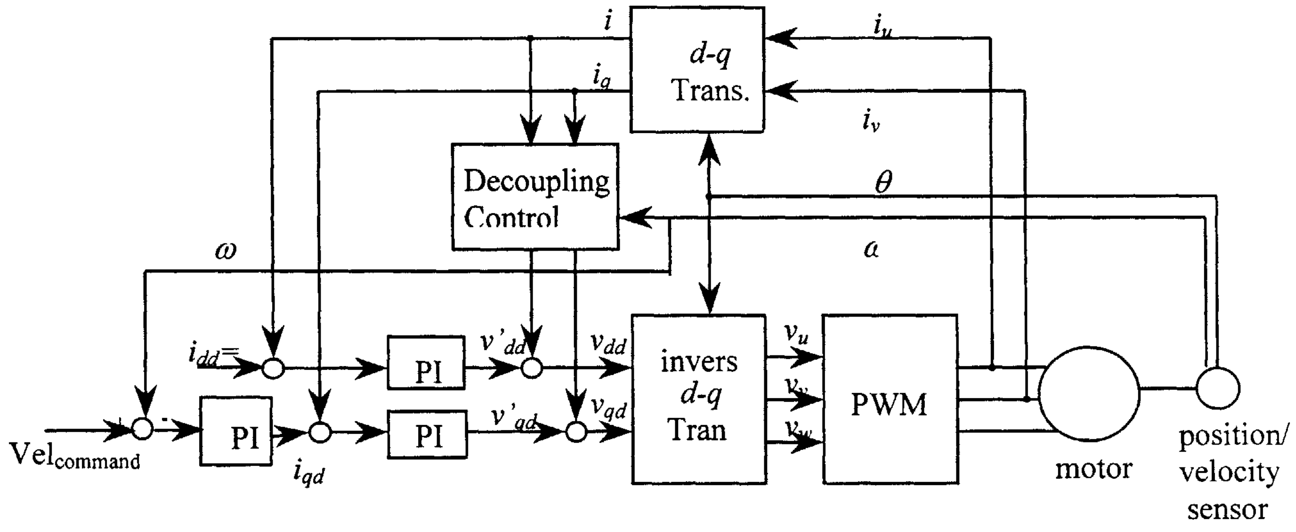


Figure 5.2.7: *D-Q* axis feedback control with Back EMF compensator

5.3 Implementation of AC Motor Control Algorithms by Full Software Servo

5.3.1 The Experimental Setup

Three control algorithms were implemented and the experimental data was obtained. In this section, the results will be presented and discussed. The experiment was done using Dell Dimension XPS Pro 200MHz with 64Mb of RAM. The sampling rate for current and velocity were $100\mu s$ and $250\mu s$ respectively. Two general I/O boards were used and one timer board was used. The inverter and buffer were used to amplify the signals from the computer. The power block was used to run the motor. The encoder signal and the current feedback signal were passed to the PC. The setup diagram is shown in Figure 5.3.1.

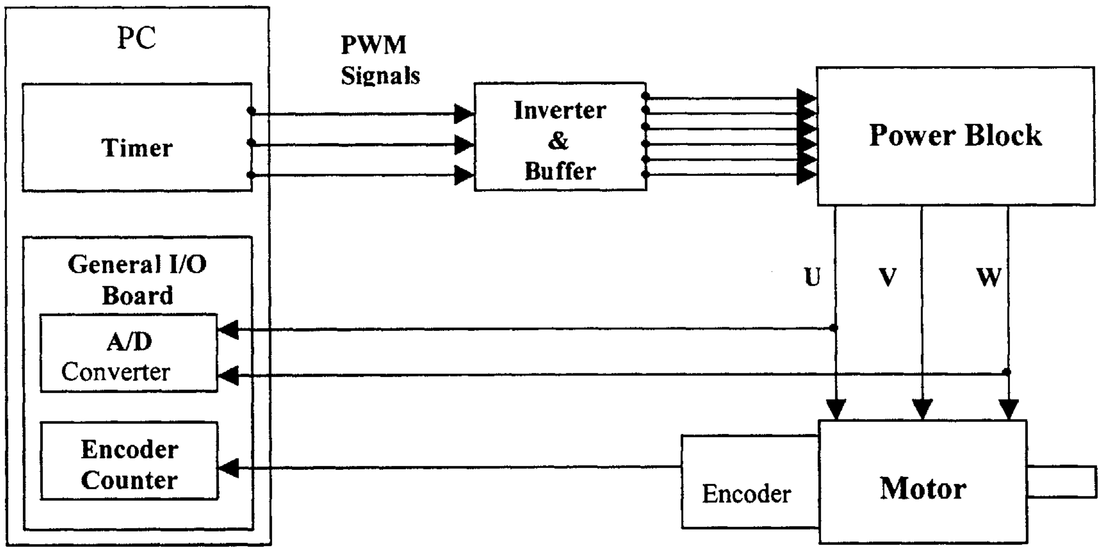


Figure 5.3.1: Structure diagram of full digital AC servo control setup

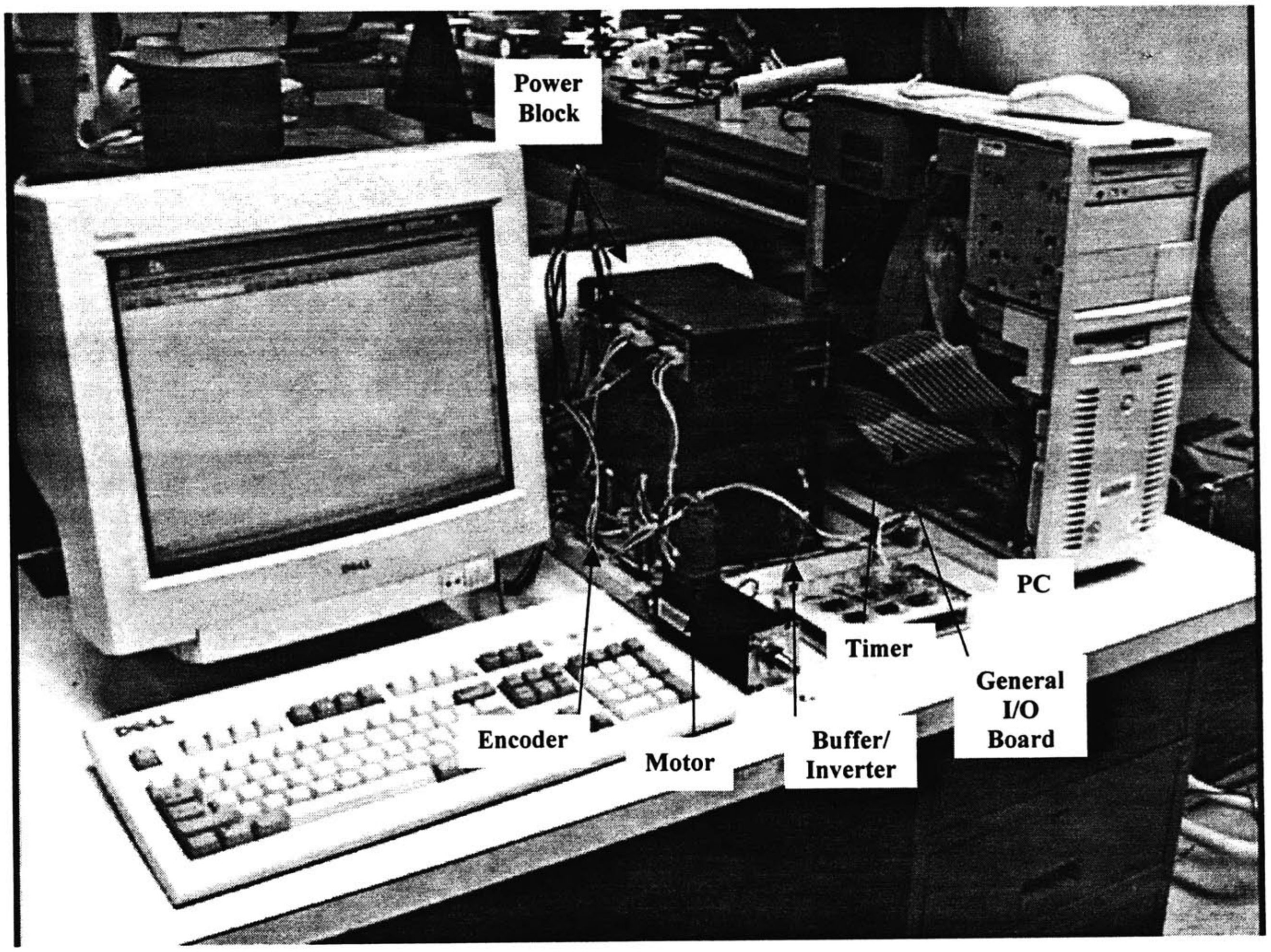


Figure 5.3.2: Picture of the Windows NT-based AC servo system setup

Figure 5.3.2 shows a picture of the experimental setup. The brushless servomotor used for the experiment is the model MPM662FRM-AM made by Custom Servo Motors Inc. The specifications of the motor are in the table below.

Power	Max Speed	Peak Current	Continuous Current	Torque Sensitivity	Back EMF Constant
160 W	5000 RPM	7.2 A	2.4 A	0.23 Nm/A	18.2 Vrms/Krpm

Static Friction	Inductance	Rotor Inertia	D.C. Resistance	Motor Weight
0.06 Nm	10.4 mH	10 Kgmm^2	4.0 ohms	1.3 Kg

The power block is manufactured by Kollmorgen Motion Technologies Group. The power block includes a power supply and an amplifier for this experiment. The model numbers are PS28 for the power supply and RP03 for the power amplifier. The specifications of the power supply are shown below.

Main AC Input	Phase	Frequency	Current RMS	Current RMS (2sec)	Current RMS Peak (50ms)
190 ~ 260 VAC	3 Phase	47 ~ 63 Hz	28 A	56 A	100 A

Output Power	Volts	Control AC Line Input	Phase	Frequency	Current RMS
8.4 KW	310 V	190 ~ 260 V	1 phase	47 ~ 63 Hz	1.5 A

The specifications of the power amplifier are shown in the table below.

Main DC Bus Max	Main DC Bus Min	Output Current Continuous RMS	Output Current Peak (8 sec)	Internal Heat Dissipation
110 VDC	365 VDC	3.0 A	6.0 A	50 W

PWM Max Frequency	Current Feedback Output Scaling	PWM Input Minimum	PWM Input Maximum
15 KHz	1.06 Amp/Volt	2.5 Volt @9mA	5.3V @ 36mA 1.25V @2.5mA

The I/O boards used for digital to analog converting and analog to digital converting are manufactured by Servo To Go, Inc. The board also has the interval timer for the interrupt service routine and encoder input. The board has a number of functionalities. There are eight channels of encoder input, each with 24 bit counters. There are eight channels for DAC and also for ADC. The ADC channels have a +10 volts to –10 volts span with 13 bit resolution and the DAC channels have +/-5 or +/-10 volts spans, which is configurable, with 13 bit resolution. The interval timer is capable of interrupting the PC and is programmable to 10 minutes in 25 microsecond increments. The board also has the digital input and output function.

The model name of the counter board is CIO-CTR20HD, which is manufactured by Computer Boards, Inc. The CIO-CTR20HD has 20 channels of counter and contains four AM9513 counter timer chips. Each AM9513 handles 5 channels of the counter. The 9513 chip is fully programmable and takes two addresses per chip, one of which is a data path to the counter's load and hold registers. Each counter has an input source, a counter register, a load register, a hold register, an output, and a gate. The CIO-CTR20HD occupies eight I/O addresses. The base address is determined by setting a bank of switches on the board. The source of the pulses supplied to the 9513 for timing operations is programmable. The 9513 chip has its internal clock source at 1MHz and 5MHz. It also can be programmed to choose an external clock source. The current setup uses an external clock source at 3MHz. The frequency of the PWM signal can be calculated by the following:

$$Frequency\_PWM = \left[\frac{PWM\_Range}{Clock\_Source\_Frequency}\right]^{-1}$$

The *PWM_Range* is the size of counter range in each channel. For the experiments, the range was 256, which is 8 bits of information. The PWM signal must have a high enough frequency to avoid the nonlinear effect but must also be low enough for the transistors in the power amplifier to be able to handle. The reason the external clock source of 3MHz is chosen is to meet the frequency qualification of the PWM signals. The frequency of the PWM signal is about 12KHz using the external clock source. The internal clock source at 1MHz generates a 4KHz PWM frequency, which is too low to avoid the nonlinear dynamics in the electrical domain. The internal clock source at 5MHz generates about a 20KHz PWM frequency, which is out of the range of the switching frequency of the transistors in the power amplifier.

5.3.2 Step Responses of the Three Phase Local Feedback and the *d-q* Axis Feedback Control

The three phase local feedback control takes each phase, which is alternating current for a rotating motor. If there is no disturbance and delay, the three phase feedback will give the desired torque and zero internal power loss. However, there is significant disturbance and delay and each phase cannot be perfectly controlled. Therefore, the *d-q* axis controller was implemented. Both the *d* axis and *q* axis are constant as long as the torque command is constant when the alternating phase currents are transformed into the *d* and *q* axes. The *d* axis represents the internal power loss in the system and the *q* axis represents the desired torque current.

The most significant difference between the three phase control and the *d-q* axis control is the form of the current being controlled. For the current control, a PI controller is used to eliminate the steady state error. However, there will always be steady state error when the three phase control is implemented, even if the PI controller is used because the current is alternating. The integral control is not fast enough for the changing phase current and yields steady state error.

Figure 5.3.3 shows the step response of the d and q axis current for the three phase control and the *d-q* axis control. Both responses were implemented with a sampling rate, Ts, of $100\mu s$. The values for the gains K_P and K_i were 42.4volt/amp and 18.2volt/(amp*ms) for the three phase control.

$$V_u = K_P * (i_{ud} - i_u) + K_i * \sum (i_{ud} - i_u) * T_s$$

$$K_i = \frac{K_P}{T_i}$$

The values for the gains K_p and the integral time constant T_i were 40volt/amp and 2.6ms for the *d-q* axis control. If the integral gain is represented in terms of K_i then the K_i is K_p divided by T_i, which is 15.4volt/(amp*ms). Therefore, the gains of the d-q axis control are similar to those of the three phase control.

$$V_q = K_p * [(i_{qd} - i_q) + \frac{1}{Ti} * \sum (i_{qd} - i_q) * T_s]$$

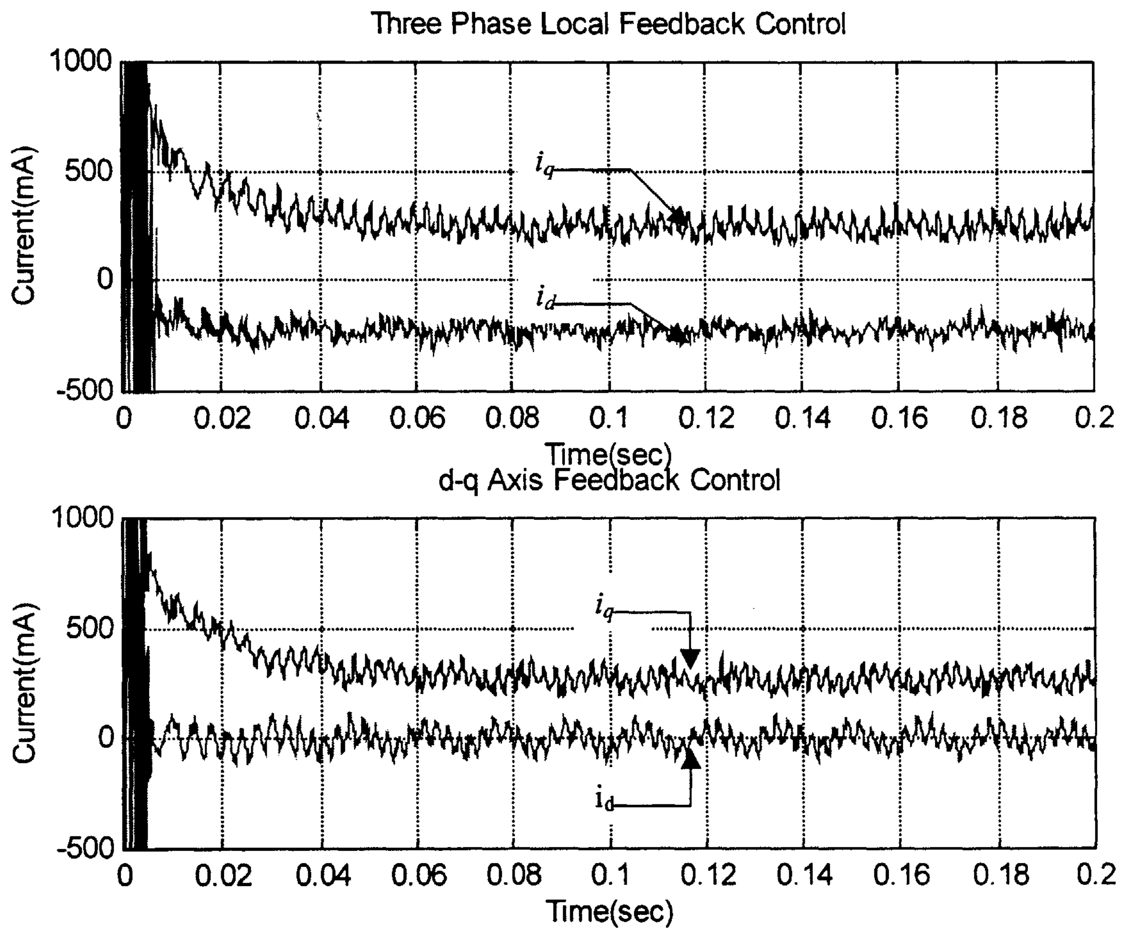


Figure 5.3.3: Current step response of the three phase local feedback and the *d-q* axis feedback control - I_{qd} (torque command) and I_{dd} (internal power loss) are 1000mA

The response of the phase control shows that both i_d and i_q do not stay at the desired value. However, the response of the *d-q* axis control shows that only i_q does not stay at the desired value. The steady state error of i_d is eliminated when the *d-q* controller is implemented. The steady state error of i_d and i_q is due to the Back EMF of the motor. As the rotor turns faster, the Back EMF increases and drops the actual voltage applied to the motor. More experimental data regarding Back EMF will be presented in the next chapter.

5.3.3 Frequency Responses of the Three Phase Local Feedback and the *d-q* Axis Feedback Control

The frequency responses of the three phase controller and the *d-q* axis controller are shown in Figure 5.3.4 and Figure 5.3.5. For each frequency response, a sinusoidal reference generated by the software is fed into the system. In this experiment, i_d was set to zero. The black lines represent the reference and the gray lines represent the actual current measurements in the *q* axis. The frequency responses were measured at 28Hz, 139Hz, 278Hz, and 417Hz. The gains of the responses show that the *d-q* control has a higher break frequency. The two responses can be compared qualitatively. The gain is one at the low frequency. When the frequency is higher the gain of the phase control frequency response decreases. However, the response of the *d-q* axis shows that the gain is approximately one even at 417Hz. The phase differences of the frequency response are difficult to measure quantitatively because of discretization of the sampled data. Nevertheless, the quantitative analysis will be available when the entire range of frequency response is done. However, there was not any significant difference in phase between the three phase control and the *d-q* control.

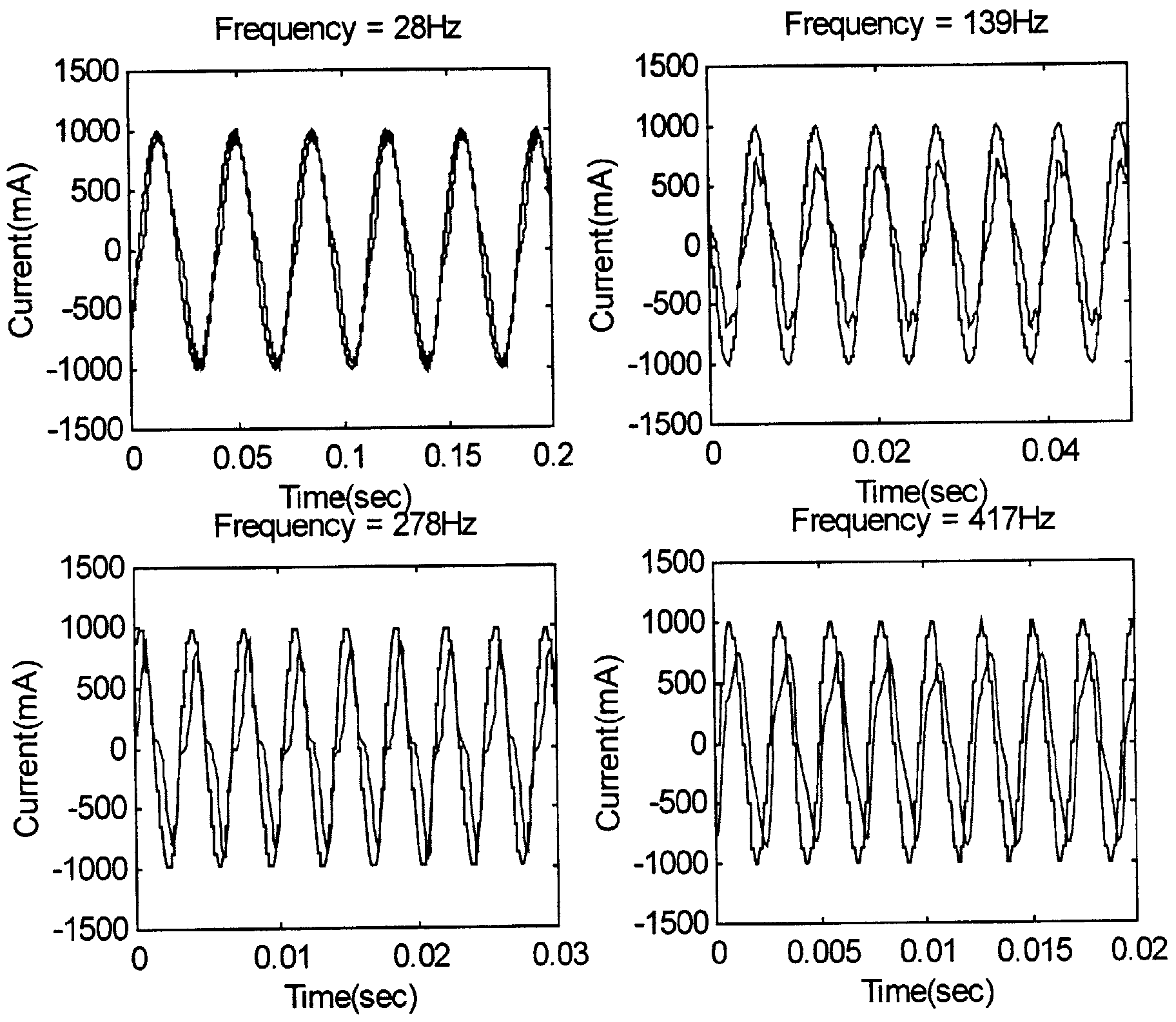


Figure 5.3.4: Current frequency response with the three phase local feedback control

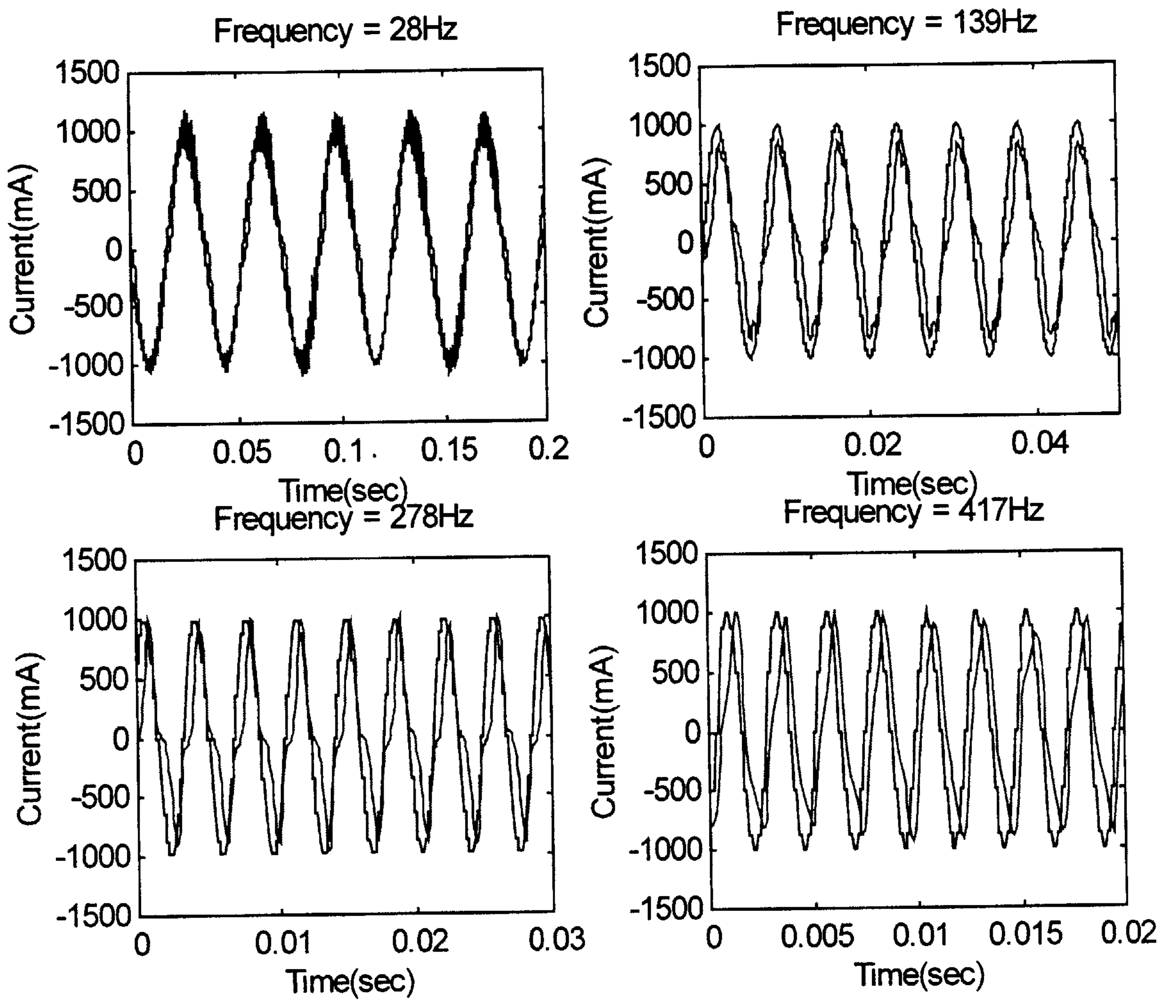


Figure 5.3.5: Current frequency response with the *d-q* axis feedback control

5.3.4 Velocity Step and Frequency Responses of the *d-q* Axis Control

Sections 5.3.1 and 5.3.2 show that the *d-q* axis control performs better than the three phase feedback control in regulating the torque command. For the upper level control, such as velocity or position, the purpose of the current control is to regulate the torque to the desired level. Therefore, the *d-q* axis current control should be used as a lower level feedback control. In this section, the velocity response using the *d-q* axis current control is discussed. For the velocity feedback control, PI control has been used with proportional gain of 900Hz and an integral gain of 40. The PI control algorithm is as follows:

$$i_{qd} = K_p * (\omega_d - \omega) + K_i * \sum (\omega_d - \omega)$$

Figure 5.3.6 shows the velocity step response. The reference velocity was 780 rpm, the velocity sampling rate was 250μs, and the current sampling rate was 100μs. The rotor was not attached to any external setup. The calculation of the parameters is shown below:

$$Overshoot\ (\%) = 100 \times \frac{MaxPeak - DesiredValue}{DesiredValue} = 36\%$$

$$= 100\, e^{-\frac{\xi\pi}{\sqrt{1-\xi^2}}}$$

$$\xi = 0.31$$

Due to a small damping ratio, there is a large overshoot of 36%. The motor is freely rotating without any external damping. The only damping is the damping of the rotor bearing and electrical damping from the Back EMF

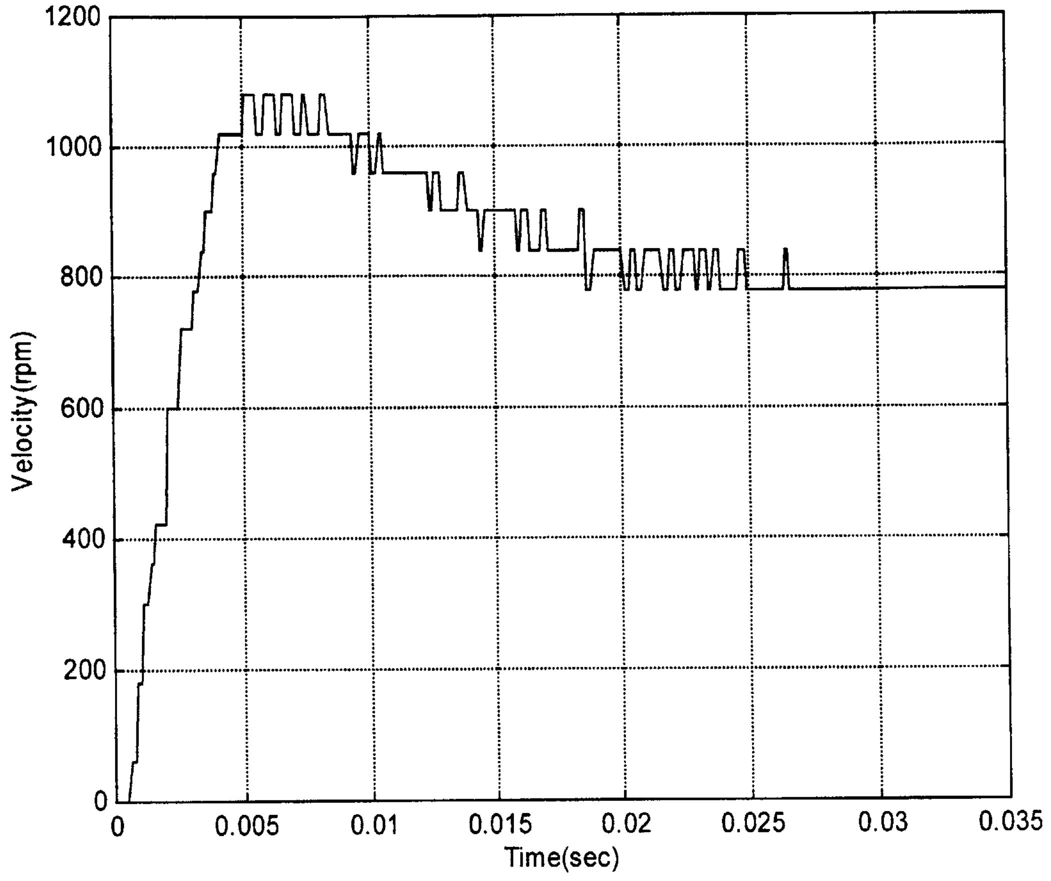


Figure 5.3.6: Velocity step response using the *d-q* axis control.
Reference velocity = 780 RPM

Tr(Rising Time) = Duration Between 10% and 90% of the Desired Value)

= 0.0025sec = 2.5ms

$$= \frac{2.16\xi + 0.6}{\omega_n}$$

$$\omega_n = 81Hz$$

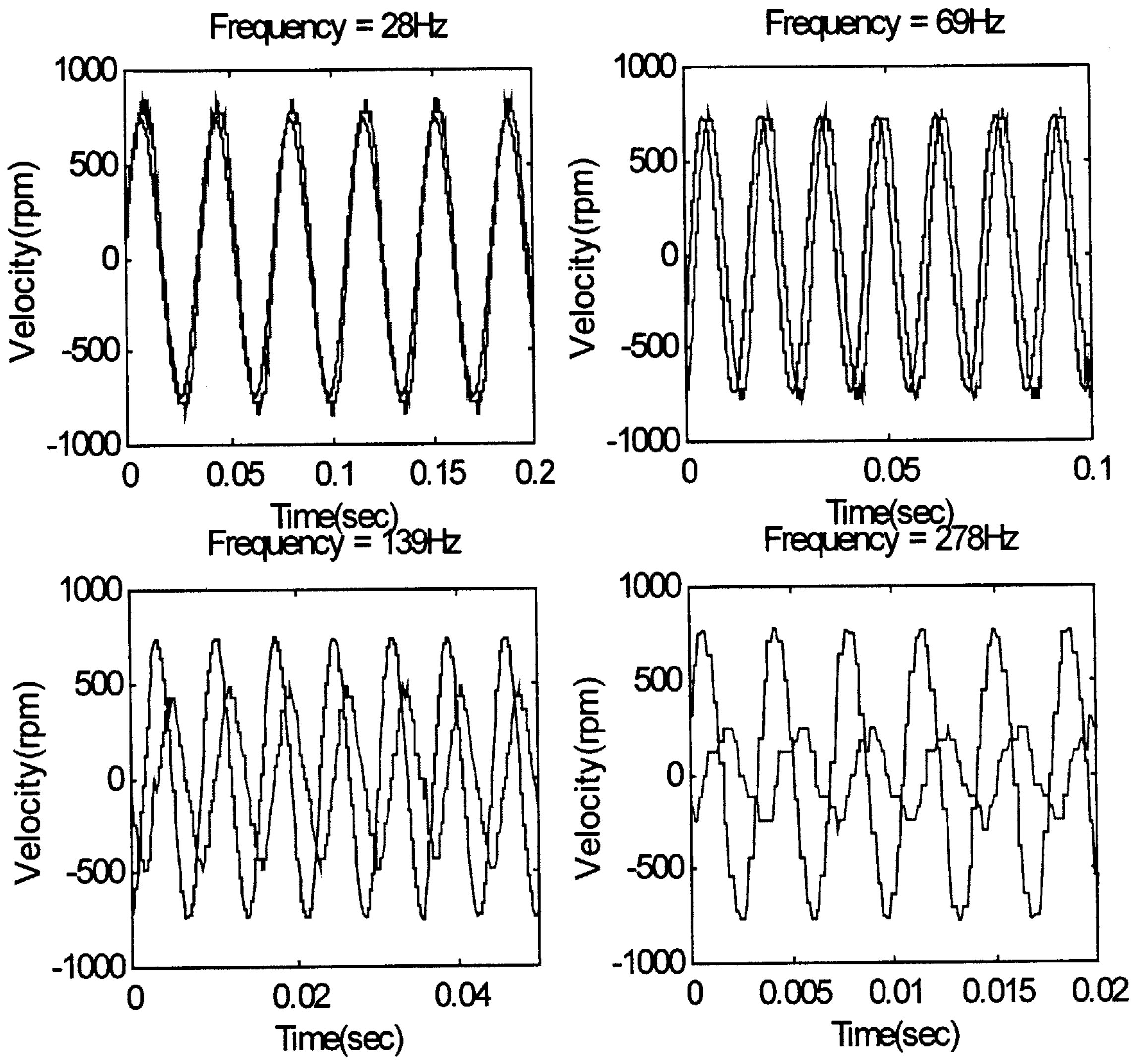


Figure 5.3.7: velocity frequency response using the *d-q* axis control

The rising time was 2.5*ms*. The ω_n was found to be 81Hz. Using ζ and ω_n, the settling

$$T_s(SettingTime) = \frac{4}{\xi\omega_n} = 0.025s = 25ms$$

time was estimated to be 25*ms*, which could be verified from Figure 5.3.7.

The frequency responses are shown in Figure 5.3.7. The setup was the same as that of the step response. The gain is approximately one for 28Hz and 69Hz. The gain starts decreasing after 69Hz. In the step response, ω_n was found to be 81Hz, which is probably where the gain begins to decrease.

5.4 Experimental Results and Discussions on the Effect of Back EMF Compensation

The steady state error of i_q due to Back EMF has been mentioned earlier in this chapter. Back EMF is a predictable disturbance that affects the output value, which is i_q in this system. Figure 5.4.1 shows the error of the i_q increasing with time in the current step response. This error has an effect in the upper level control because the torque does not reach the desired value. The lower level control should provide the desired torque. In order to obtain the desired torque, Back EMF must be compensated.

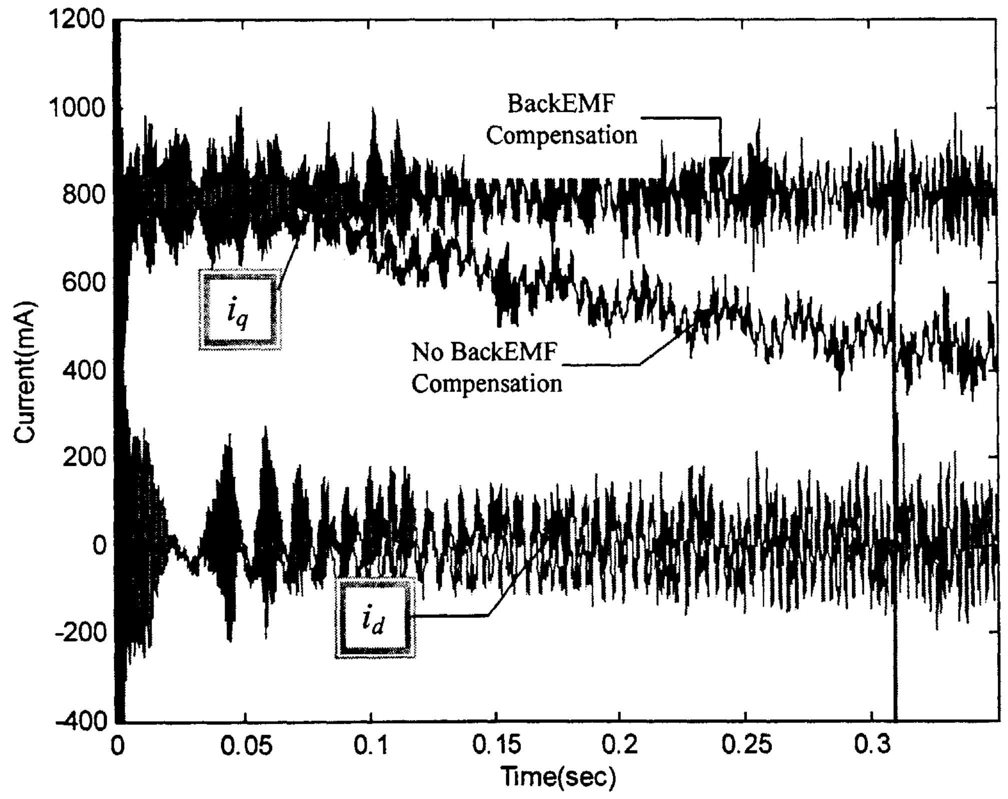


Figure 5.4.1: Current step response with and without BackEMF compensation
Reference current = 800mA

One method of determining the Back EMF constant is turning the rotor with an external motor. When the rotor turns, Back EMF will be created and the value can be measured with respect to the angular velocity. The Back EMF compensator can be designed using the constant.

Figure 5.4.1 shows the step response of the currents, i_q and i_d. The reference i_q was 800mA and the reference i_d was 0mA. Two of the data sets represent the i_d and the i_q with the Back EMF compensation, and the other two represent the data sets without it. The values for the i_d with and without the Back EMF compensation stay around zero. However, the values for i_q are different between the one with the compensation and the one without it. The values of the i_q data with compensation follow the line of the reference i_q, which is 800mA. However, the value of the i_q without compensation decreases with time due to Back EMF. The controller with Back EMF compensation is clearly better torque source maintaining the desired torque when the angular velocity varies.

The lower level control affects the performance of the upper level control such as velocity control. In Figure 5.4.2, velocity step responses with no Back EMF compensation and with compensation are plotted. Two different Back EMF constants are used for the current responses, one higher than the other. The corresponding current responses are plotted in Figure 5.4.3. In Figure 5.4.2, the response with the Back EMF compensation is faster than the one without the compensation. When there is no compensation, the Back EMF drops the actual voltage applied to the motor and prevents the actual current from reaching the desired value. Therefore, the angular velocity of the motor takes a longer time to get to the desired level as well. Figure 5.4.3 shows the current without compensation dropping as the velocity increases, while the currents with the compensation are maintained at the desired current level regardless of the angular velocity.

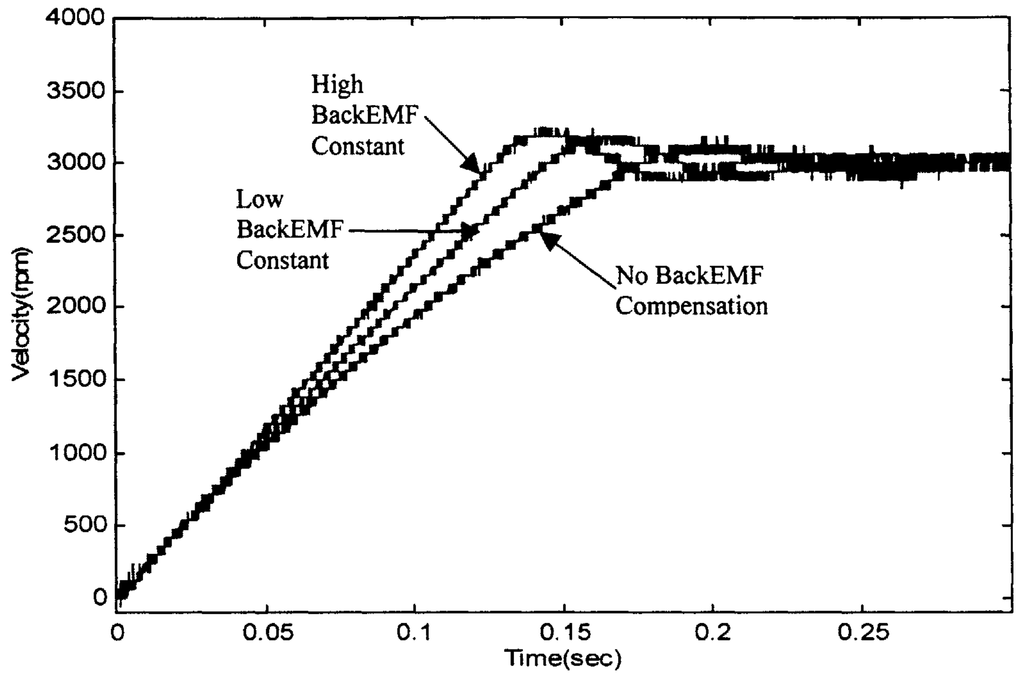


Figure 5.4.2: Velocity step response with the effect of BackEMF compensation with load --- Reference velocity = 3000 RPM

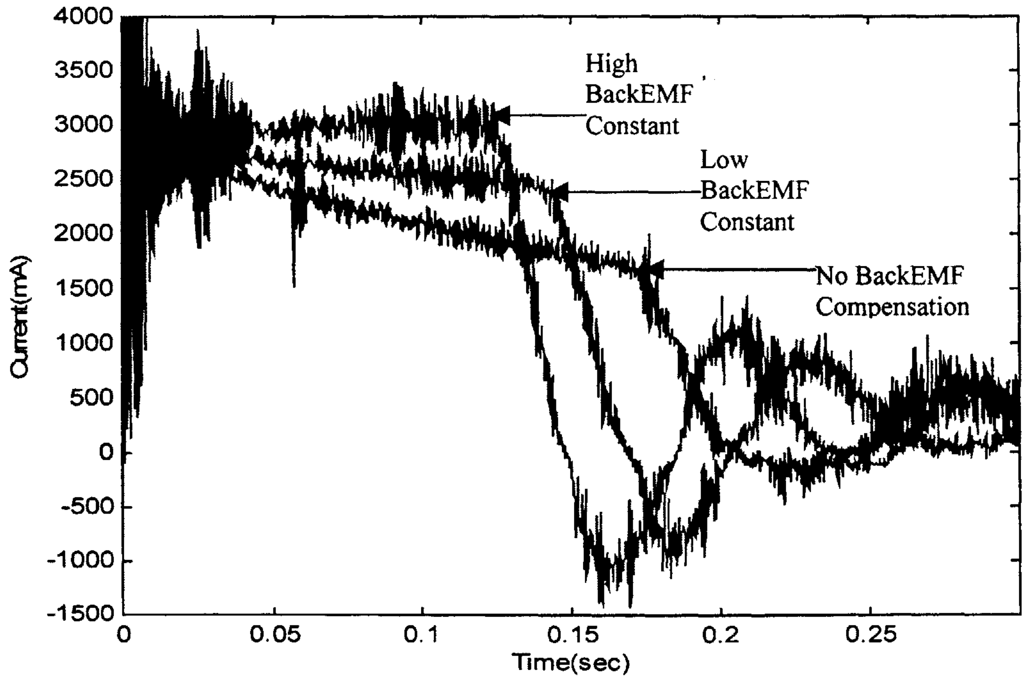


Figure 5.4.3: Corresponding current for velocity step response of Figure 5.4.2

One issue of concern with the Back EMF compensation is the loss of damping in the system. The Back EMF serves the role as a natural damper giving stability to the system. Therefore, eliminating the Back EMF might cause less damping in the system. However, according to Figure 5.4.2, the overshoot of the velocity response with the compensation is not much higher than the one without the compensation. The calculated overshoot and damping ratio for the response without the compensation was 8% and 0.67 respectively. The calculated overshoot and damping ratio for the response with the compensation was 6% and 0.63 respectively. The data shows that the loss of damping in the system due to the lack of Back EMF compensation is not a significant amount.

5.5 Conclusions

In this chapter, three AC servo motor control algorithms were compared. The mathematical models of the AC servomotors and the d-q transform were derived. The *d-q* axis feedback control algorithm performed better than the three phase feedback control algorithm. Although both controllers used PI control, the three phase feedback control still had steady state error because it had to try to control alternating current. The performance of the *d-q* control was also better in the frequency response, showing a higher bandwidth than the local three phase control.

Both the *d-q* axis controller and the three phase controller had a huge drop of i_q due to Back EMF. Therefore, the Back EMF compensator was designed and implemented. The decoupling control was also implement with the Back EMF compensation in order to eliminate the nonlinear terms in the equation of motion. . The results showed that this control algorithm maintains i_q at the desired value. As a result, the velocity feedback loop had a faster bandwidth because the current feedback was able to provide the desired i_q for the upper level control.

Chapter 6

Dynamic Pulse Width Modulation (PWM)

6.1 Overview

Pulse Width Modulation (PWM) is a method of supplying variable voltage by varying the width of each pulse. Figure 6.1 shows one sample center aligned PWM period. The center aligned PWM consists of two different components-two low duration parts and one high duration part. The high duration part occurs when the power transistor turns on. The lower duration parts occur when it is off. The high duration part defines the voltage that the PWM is representing. The percentage of the high duration indicates the percentage of the high voltage flowing through the circuit, which powers the motor. For example, if the high voltage were 300 VDC, then 60% high duration would mean 180V are supplied. If one PWM period were 100μs, which is 10kHz in terms of PWM frequency, 60% high duration means that the transistor is turned on for 60μs.

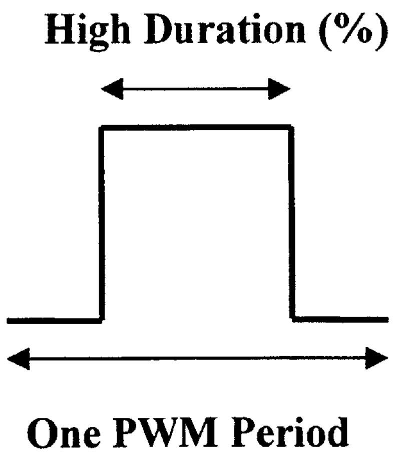


Figure 6.1: Sample PWM Period

PWM is the most widely used method to supply variable voltages to AC servo motors. However, the current practice using PWM still has room for improvement. In this chapter, the current practice and its problems are discussed, and a solution will be provided. The solution uses intelligent PWM, which changes dynamically knowing the desired voltage and utilizing the information from the previous PWM. A new algorithm for this intelligent PWM will be discussed later in this chapter. Variables will be defined for mathematical analysis and different algorithms will be presented for different cases. Matlab simulation and data will also be presented to show the effectiveness of the new PWM algorithm. The physical constraints of the dynamic PWM are also discussed and the solutions to those problems are addressed. Finally, the future implementation effort using the full software servo will be discussed.

6.2 Conventional Techniques and Issues

6.2.1 Current Practice

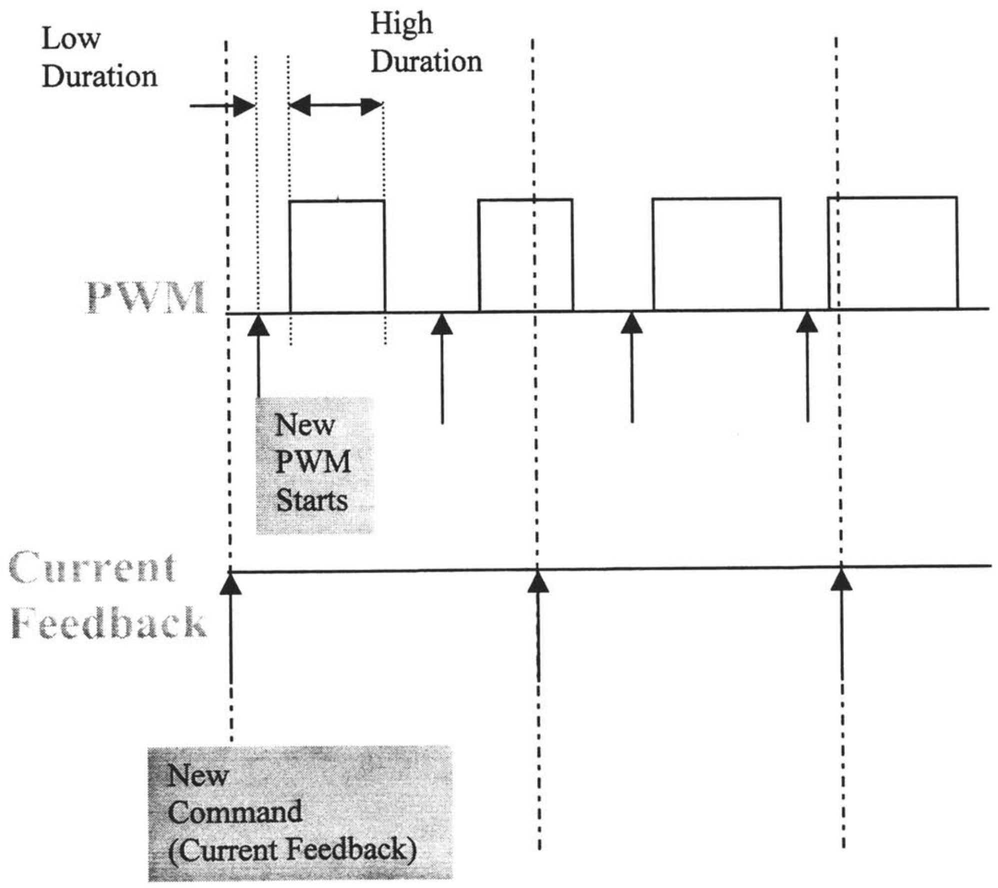


Figure 6.2.1: Timing Diagram of PWM and Current Sampling

PWM frequency and the current feedback sampling are usually independent of each other. Figure 6.2.1 shows the timing diagram of PWM and the current feedback sampling. The vertical arrows on the PWM time line indicate when the new PWM starts. The vertical arrows on the current feedback line indicate when the new commands are generated from commutation and the current feedback loop. The asynchronous nature of each time line can be seen from the figure. The first new command, the first arrow on the current feedback line, is generated just prior to the new PWM. Therefore, the information from the current feedback could be implemented in the PWM right away. However, the last command, the last arrow on the current feedback line, is generated right after the PWM started. Therefore, the new information from the current feedback needs to wait for almost one entire PWM period to be implemented. As shown in Figure 6.2.1, the new information could be implemented right away or with some delay depending on the mutual timing of each process. PWM and the current feedback have different frequencies; the frequency of the current feedback is usually lower than that of the PWM. As discussed above, the maximum delay for the implementation of the new information from the current feedback is one PWM period. Therefore, if the frequency of the PWM is increased, the delay could be shortened. These problems are discussed in detail in the next section.

6.2.2 Delay in PWM Generation

The current practice of the PWM method supplies varying voltage with high frequency pulses. The period of the PWM is usually much smaller than that of the electrical constant of AC servo motors. The high frequency pulses are filtered by the motor, with a much slower response, and generate smooth current. If the frequency of the PWM is not fast enough, the current will not be smooth. The PWM voltage is defined to be the area of the PWM in one period, assuming the time constant is much faster than that of AC servo motors.

However, it is not clear when the PWM voltage actually gives effective voltage to the motor because the effective voltage is determined by averaging one PWM period. In Figure 6.2.2, one PWM period is shown. The new information, or command, is generated from the current feedback and commutation loops just as the PWM starts. Therefore, the PWM is executed right away with the new information. However, it is not clear where the effective voltage is. It is intuitive though, that the effective voltage would not occur in the beginning of the PWM. If the effective voltage does not occur in the beginning of the PWM, there is a delay. Although the PWM started with the new information, the effective voltage would not be in the beginning of the PWM, perhaps more towards the center. Therefore, there is a natural delay embedded in the PWM method due to the effective voltage, equal to approximately one half of the PWM period.

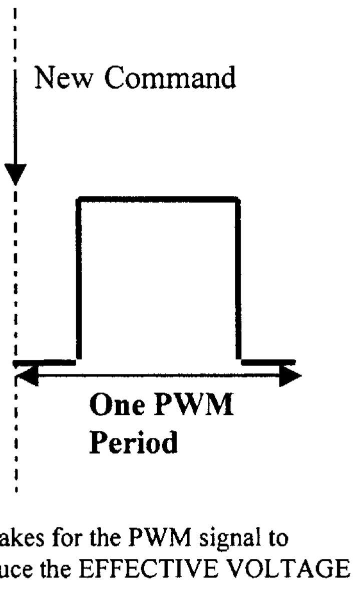
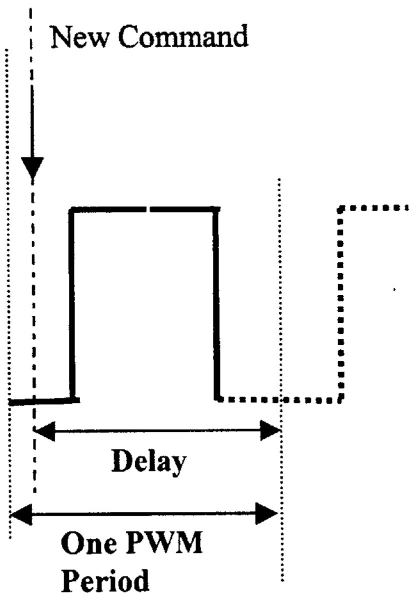

New Command
One PWM Period
Delay 1 : Time it takes for the PWM signal to integrate and produce the EFFECTIVE VOLTAGE

Figure 6.2.2: Delay Due to Effective Voltage

There is another source of delay in the PWM method. The new command from the current feedback and the commutation loops does not always arrive just prior to the start of the PWM. The new command stays in the buffer until the generation of the next PWM. As shown in figure 6.2.3, the delay could be up to one PWM period if the new command arrives right after the PWM has already started. The average delay due to the timing difference is one half the PWM period. One solution is to synchronize the PWM and the current feedback. However, the frequencies are different and it is hard to synchronize them without any mistiming.

There are mainly two sources of delay in the current practice of PWM, as discussed above. The first one is natural to the PWM method and the second comes from the limitation of the current practice of PWM and the current feedback timing. The combination of these two types of delay will be called "Effective Delay" from this point on. The average effective delay is one PWM period and the maximum effective delay is one and a half PWM periods. The new algorithm discussed in a later section could solve the second delay issue completely and much or all of the first issues as well.



Delay1 : Time it takes for the new command to be implemented
Maximum delay = one PWM Period

Figure 6.2.3: Delay Due to Asynchronous Current Feedback and PWM

6.3 Dynamic PWM

6.3.1 Introduction

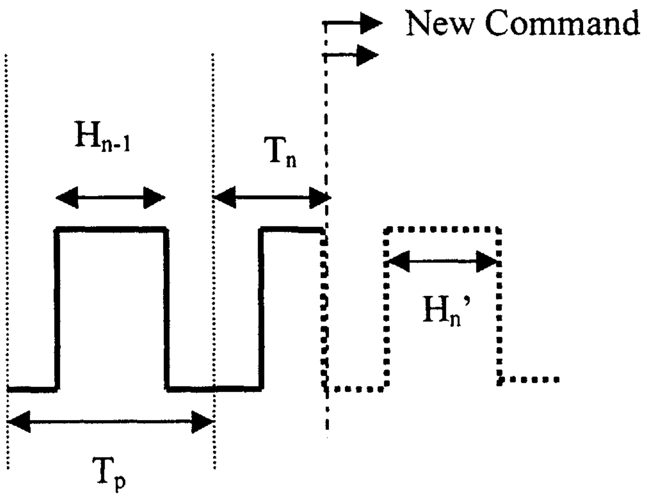
A new method is developed and discussed in this section in order to solve those delay problems mentioned above. The basic concept is to use the previous and current PWM information to generate a compensated PWM to have a faster response without a minimum of delay. The algorithms for different cases are discussed in section 6.3.4. These algorithms are derived using simple algebraic manipulation. The procedure is as follows:

1) Determine the desired high duration, as usual.
2) Acquire the status of the current PWM implemented in the system.
3) Calculate the compensated high duration command to achieve the desired high duration using the currently implemented PWM.
4) Implement the new high duration command and generate a new PWM right away by interrupting the existing PWM. It is already compensated for in the previous step.

This new technique dynamically changes the existing PWM to compensate for the new command from the current feedback and it is named "Dynamic PWM." The dynamic PWM technique heavily depends on the averaging property of the PWM. The goal of the dynamic PWM is to have the effective voltage, which represents the desired voltage, to be as early as possible. The dynamic PWM method not only eliminates the delay due to the nature of PWM but also generate the PWM as if the desired future values were known. This should become clearer in the next section with different cases and examples.

6.3.2 Definition of Variables

There are some variables that need to be defined in order to make the explanation of the algorithm a little simpler. In Figure 6.3.1, T_p is defined to be one PWM period; the inverse is the PWM frequency. T_n is defined to be the time elapsed from the start of the current PWM. H_n is the high duration period command generated from the normal procedure; H_n represents the desired voltage. H_{n-1} is the previous high duration period command. The subscript n represents the current status and n-1 represents the previous one. H_n' is the compensated high duration period command that is actually executed to generate the desired effective voltage represented by H_n. L_n represents the low duration period of the current PWM, which is automatically determined to be half the difference between the PWM period and the high duration period. It is half the difference because the low duration period is broken into two parts, before the high duration period and after the high duration period, for the case of the center aligned PWM.



$\mathbf{T_p}$: One Period for One PWM=PWM $_{\text{Carrier freq}}{}^{-1}$. *(us)*
$\mathbf{T_n}$: Time Elapsed for the Current PWM *(us)*
$\mathbf{H_n}$: Actual High Duration of Current PWM *(us)*
$\mathbf{H_{n-1}}$: Actual High Duration of Previous PWM *(us)*
$\mathbf{H_n'}$: Compensated High Duration Command of Current PWM *(us)*
$\mathbf{L_n}$: Low Duration of Current PWM
$\mathbf{L_n = (T_p - H_n) / 2}$

Figure 6.3.1: Definition of Variables for Dynamic PWM Method

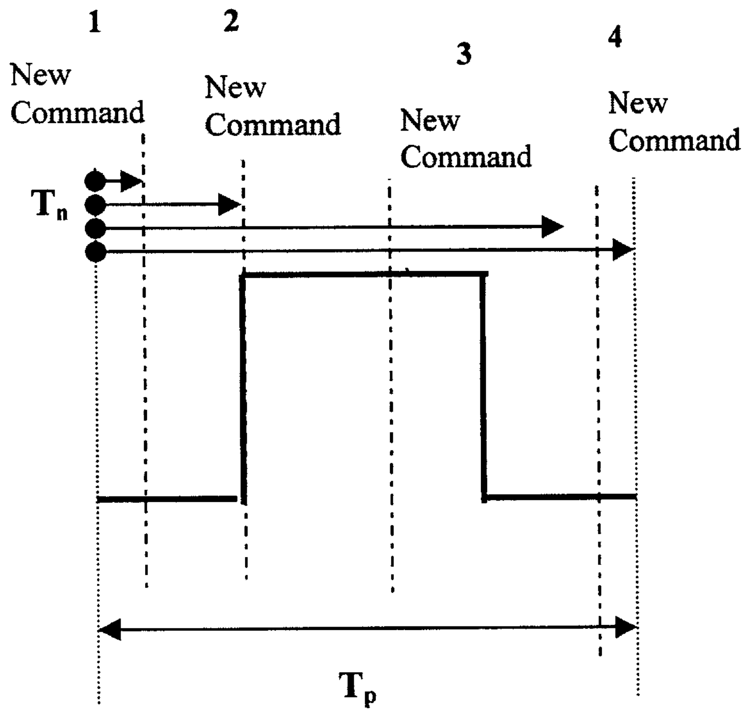
6.3.3 Introduction of Different Cases

The dynamic PMW method has four different algorithms for the different cases, as shown in Figure 6.3.2, depending on the status of the current PWM when the new high duration command is generated. The Cases are as follows:

1. The new command is at the first low duration. The current PWM just started and it is on the first low duration; therefore, the compensated PWM could continue.
2. The new command is at the high duration. The new command needs to interrupt the high duration and start a new compensated PWM.
3. The new command is at the second low duration. In this case, the new PWM needs to compensate for almost the entire previous PWM.
4. The new command is at the transition from the first low duration to the high duration. When it is too close to the transition period, special attention needs to be drawn. If

the high duration was just starting, then the new PWM must wait before its execution because it could give too much load to the transistors in the power amplifier. Therefore, a minimum waiting period is required. This case is not considered in the Matlab simulation.

In the next section, the algorithms for the different cases are presented.



Case 1 : New command at the first low duration ($T_n < L_n$)

Case 2 : New command at the high duration ($T_n > L_n$)

Case 3 : New command at the second low duration ($T_n => (H_n+L_n)$)

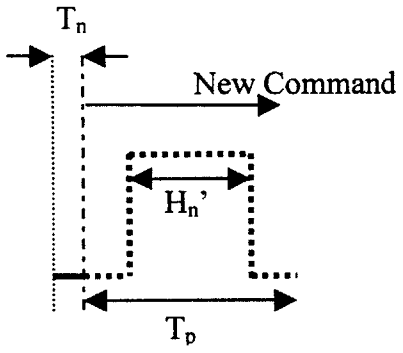
Case 4 : New command at the transition from first low duration to high duration ($T_n = L_n$)

Figure 6.3.2: Different Cases for the Dynamic PWM Method

6.3.4 Dynamic PWM Algorithms for the Different Cases

Figure 6.3.3 shows the different algorithms for the different cases in the dynamic PWM method. All the algorithms are derived from simple algebraic manipulation. There are two equations for each case in Figure 6.3.3. The first one for H_n and the second one for H_n' are the same equations. The first equation is more intuitive than the second equation. Therefore, the second equation, the actual equation for the implementation, is obtained from the first equation. The first equation is the product of the PWM period and the ratio of the high duration and the time consumed. The ratio is more specifically the sum of the high duration that would be implemented divided by the total time from the beginning of the current PWM to the end of the compensated PWM.

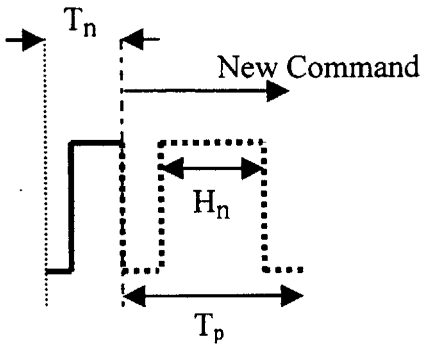
Case 1 : $T_n < L_n$



$$H_n = \frac{(H_n')}{(T_n + T_p)} * T_p$$

$$H_n' = \frac{H_n(T_n + T_p)}{T_p}$$

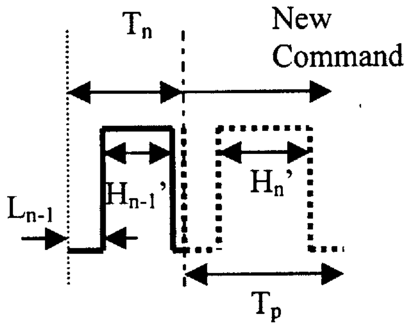
Case 2 : $(L_n+H_n) > Tn > L_n$



$$H_n = \frac{H_n'+(T_n - L_n)}{(T_n + T_p)} * T_p$$

$$H_n' = \frac{H_n(T_n + T_p)}{T_p} - (T_n - L_n)$$

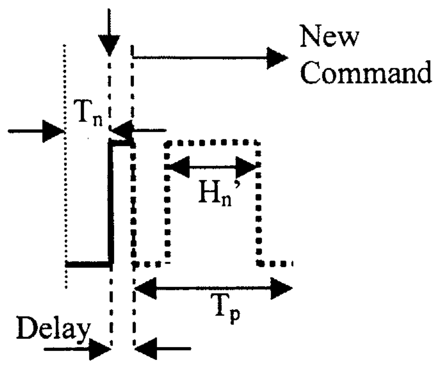
Case 3 : $(T_n >(L_n+ H_n))$



$$H_n = \frac{H_n'+H_{n-1}}{(T_n + T_p)} * T_p$$

$$H_n' = \frac{H_n(T_n + T_p)}{T_p} - H_{n-1}$$

Case 4 : $(T_n = L_n)$



$$H_n = \frac{H_n'+Delay}{(Delay+ T_n + T_p)} * T_p$$

$$H_n' = \frac{H_n(Delay+ T_n + T_p)}{T_p} - Delay$$

Figure 6.3.3: Algorithms for the Different Cases

Case 1, where $T_n < L_n$, occurs when the time elapsed from the start of the current PWM cycle (T_n) is shorter than the current low duration, meaning that it is at the first low duration. Case 2, $(L_n+H_n) > T_n > L_n$, occurs when T_n is longer than the current low duration but shorter than the first low and high duration combined, meaning that it is at the high duration. The third case, $T_n > (L_n+H_n)$, occurs when T_n is longer than the low and high duration combined, meaning that it is at the second low duration. The last case, $T_n = L_n$, occurs when T_n is about as long as the low duration, meaning that it is at the first transition period between the low duration and high duration and special attention is required. In the fourth case, some delay is applied. The value of the delay should be minimized to the minimum time required for the transistors to turn on and off.

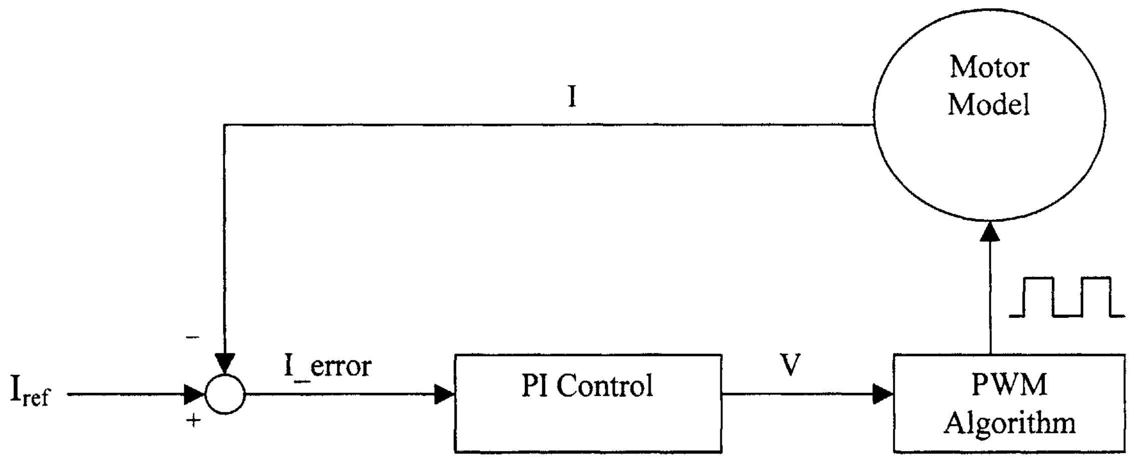
6.4 Simulation Experiments

6.4.1 Objective

The issues concerning the delay in the PWM method was discussed in section 6.2. There are basically two types of delay. One from the averaging nature of the PWM and the other from the timing difference between the PWM generation and the current feedback command update. The dynamic PWM should eliminate all the delay from the timing difference. It should also eliminate some of the delay from the averaging nature of the PWM. It seems intuitive from the presentation of the dynamic PWM algorithms that the delay should be minimized. In this section, the actual simulation results are presented to show the effectiveness of the dynamic PWM. Two sets of data were taken: the high frequency response and the instability analysis. The high frequency response shows the difference in the phase lag and the difference in the gain due to the phase delay. The difference in the phase lag should show clearly how much of the delay in the PWM is eliminated by the dynamic PWM method. The instability analysis was also done. Although the motor is usually not an unstable system, the delay in the PWM and also the digital control sampling cause the system to go unstable at very high gains. Therefore, the system with less delay would go unstable at higher gains than the system with more delay. If the dynamic PWM truly eliminates the delay problems in PWM, the data should be more favorable to the dynamic PWM.

6.4.2 Simulation Setup

Matlab software is used to simulate the AC servo systems with different PWM generators. The following were some issues concerning the simulations. First, the AC servomotor is a complicated system. Each phase of the motor is coupled to one another. In order to precisely model the motor, the angular velocity of the motors must be predicted. In this simulation, the AC servomotor model was greatly simplified because the point of the simulation was to show the relative difference in the two PWM methods, not to predict any quantitative data. The Clarke and Park transforms were performed to simplify the model to torque and iron loss axes. Then, the angular velocity was assumed to be zero to eliminate the nonlinear terms in the equations. Figure 6.4.1 shows the simulation model and the simplified equation of the AC servomotor.



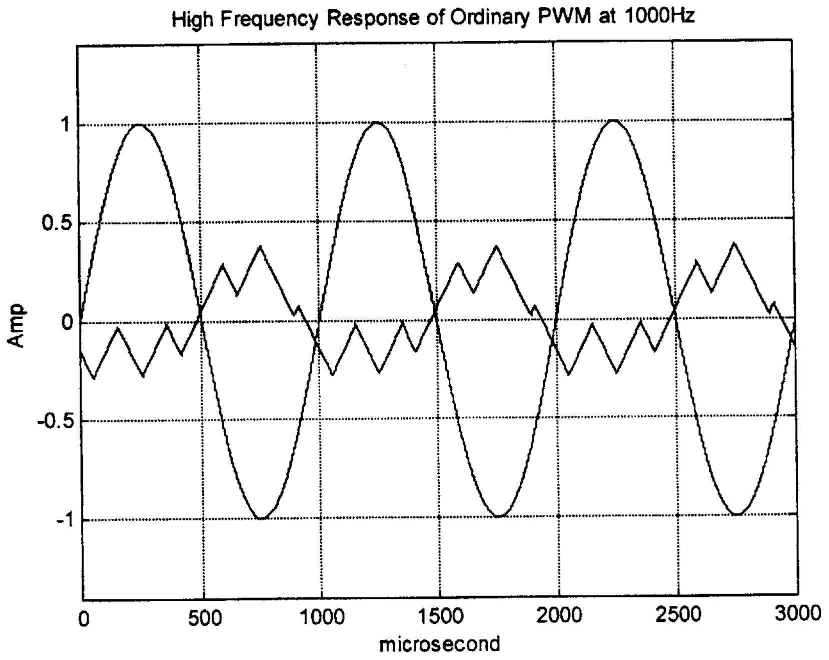
$$V = Ri + L_b \frac{di}{dt} \qquad\qquad i_{n+1} = \frac{\Delta t}{L_b}(V - Ri) + i_n$$

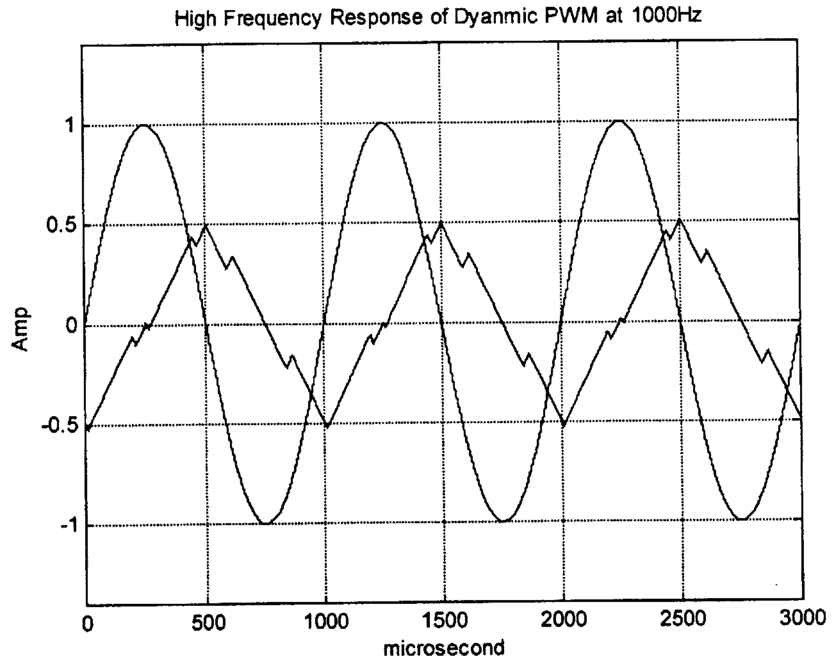
Figure 6.4.1: Simulation Model and Simplified Equation of the AC Servo Motor

The motor equation was numerically solved for increments of 100 nanoseconds using Euler's explicit integration method. The parameter values of the custom AC servomotor were used for this experiment. The resistance, R, was 4 Ohms and the inductance, L_b, was 13 mH. For the both PWM methods, the PI control gains and the motor models were identical. The PWM frequency was 5 kHz, equivalent to a 200 microsecond PWM period, which is the standard PWM frequency for the most respected company in the industry. The current feedback was set to 4kHz to ensure completion of at least one PWM in the current feedback cycle.

6.4.3 Results and Discussions

In section 6.2, the effective delay was analyzed to be about one PWM period. Therefore, the response of the dynamic PWM should have about one PWM period less phase lag at high frequency. Figure 6.4.2 shows the high frequency response of both systems. The top one is the response of the ordinary PWM and the bottom one is the response of the dynamic PWM. The vertical axis represents current input and output and the horizontal axis represents the time in microseconds. Phase and gain differences are shown clearly in this data. The phase lags of the ordinary PWM system and the dynamic PWM system are about 160 degrees and 90 degrees respectively. Therefore, the dynamic PWM has about 70 degrees less phase lag than the ordinary PWM. The phase lag of 70 degrees is about a 195 microsecond delay in a 1000 Hz frequency response. The simulation result matches the prediction from the analysis, a delay of about 200 microseconds, which is the period of the PWM cycle. The gain of the dynamic PWM is also better than that of the ordinary PWM.





6.4.2 High Frequency Response of Ordinary PWM vs. Dynamic PWM at 1000Hz

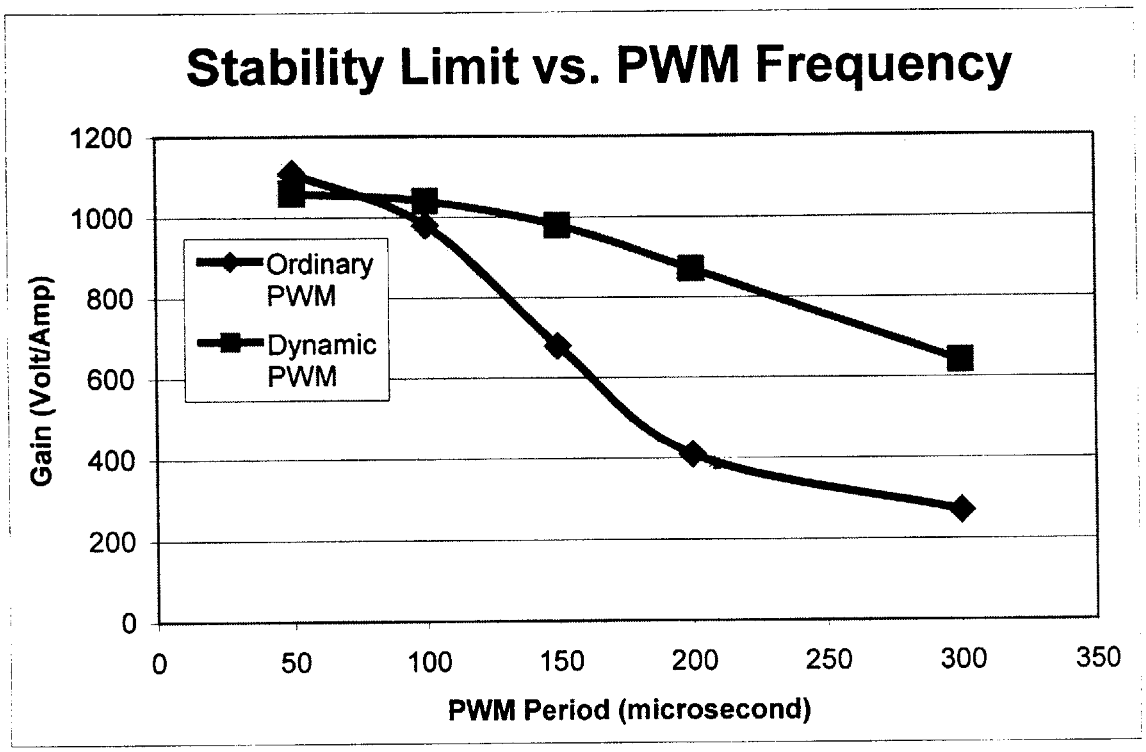


Figure 6.4.3: Instability Analysis

Another analysis from the simulation is the instability analysis. Since the system goes unstable due to the delay embedded in the PWM methods and the digital control sampling, the maximum gains could be compared between the ordinary and the dynamic PWM. The maximum gains are obtained by increasing the gain until the system goes unstable. There is some gray area when the systems are marginally stable. However, there is such a distinctive difference between the two systems that this data is adequate for comparing the two systems qualitatively. Figure 6.4.3 shows a graph of the maximum gains for both systems. The vertical axis represents the maximum gain in Volt/Amp and the horizontal axis represents the PWM period in microseconds. The dynamic PWM has a longer stable region than the ordinary PWM. For example, the maximum gain was about 640 Volt/Amp at a 150 microsecond PWM period in the ordinary PWM. However, the same gain could be obtained at a 270 microsecond PWM period in the dynamic PWM. The dynamic PWM is able to increase the gain more than the ordinary PWM because it minimizes the delay caused by PWM.

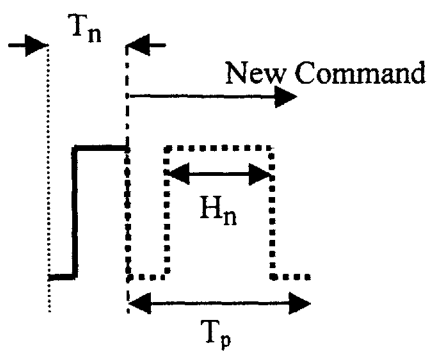
It is shown clearly from the simulation that the dynamic PWM provides a way to power the motor with less delay. This algorithm could be implemented in the full software AC servo controller utilizing the flexibility of the controller. In the next chapter, the plan for future implementation is discussed.

6.5 Physical Constraint and Solution

The previous section shows the effectiveness of the dynamic PWM compared to the ordinary PWM method. The Matlab simulation shows that much of the effective delay is eliminated as estimated in the analysis in section 6.2. However, the simulation and the actual experimental result could differ due to the physical constraints. In this section, the possible problems for the implementation in the real system are considered. One major physical constraint is the switching frequency of the power transistors. It is already compensated in the dynamic PWM algorithm as "Case 4" shown in Figure 6.5.1. However, this issue needs special attention.

It is widely known that faster switching frequency usually results in better performance. Power transistor technology has improved much as well. However, there is a limit to the switching frequency because these transistors need to handle high power. The dynamic PWM gives an advantage in terms of the PWM frequency as discussed in section 6.4.3. Therefore, the same performance could be expected with lower PWM career frequency when the dynamic PWM algorithm is implemented compared to the performance of the high PWM career frequency in the ordinary PWM algorithm. However, the maximum switching frequency or the minimum switching time must be considered carefully for the dynamic PWM due to "Case 2" and "Case 4" shown in Figure 6.5.1. In these cases, the high duration is interrupted and the transistors are turned off abruptly. These two cases may require transistors with very small minimum switching capability. Although it is not the career frequency of the transistors, the minimum switching frequency could be a major physical constraint for the implementation. Abrupt switching of the transistors should be avoided in order to use the dynamic PWM without much change or concern in the hardware.

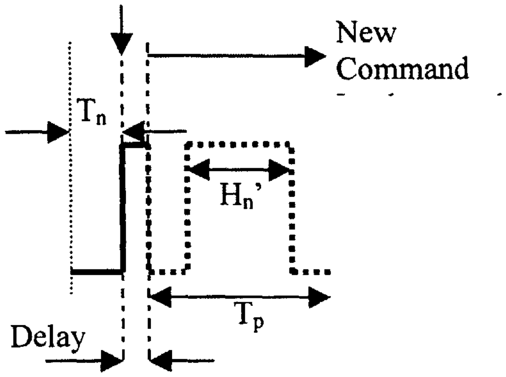
Case 2 : $(L_n+H_n) > Tn > L_n$



$$H_n = \frac{H_n'+(T_n-L_n)}{(T_n+T_p)}*T_p$$

$$H_n' = \frac{H_n(T_n+T_p)}{T_p}-(T_n-L_n)$$

Case 4 : $(T_n = L_n)$

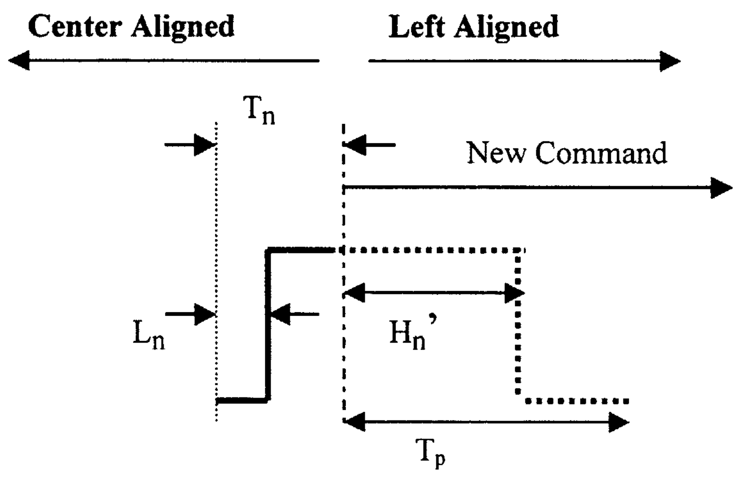


$$H_n = \frac{H_n'+Delay}{(Delay+T_n+T_p)}*T_p$$

$$H_n' = \frac{H_n(Delay+T_n+T_p)}{T_p}-Delay$$

Figure 6.5.1: Cases with possible physical constraint problem

The interruption of the high duration occurs when the new command starts at the time of high duration in the dynamic PWM. The center aligned PWM always starts with the low duration first, then the high duration causing the current high duration to be interrupted. If the compensated PWM started with the high duration of this specific case the existing high duration could just continue without any interruption. Therefore, using the left aligned PWM for this case could be proposed. Figure 6.5.2 graphically shows the compensation by the left aligned PWM. This one case can be substituted for both cases 2 and 4 discussed in section 6.3. Case 4 is not needed anymore when the left aligned PWM is used because the transistors do not have to turn off and on quickly. In the case of the left aligned PWM, the high duration starts first, then the rest of the PWM period is completed during the low duration. This algorithm could be implemented utilizing the full software AC servo controller, especially with the dynamic link with the FPGA interface board. This method should give similar results to the regular algorithms discussed in section 6.3 because the average effective voltages of both algorithms are the same. It should eliminate the physical constraint of the power transistor maximum switching frequency.



$$H_n = \frac{H_n' + (T_n - L_n)}{(T_n + T_p)} * T_p$$

$$H_n' = \frac{H_n(T_n + T_p)}{T_p} - (T_n - L_n)$$

Figure 6.5.2: Dynamic PWM compensation by left aligned PWM

Another issue of the physical constraint is the data transfer speed of the ISA bus. The slow bus speed is a problem not only for the time budget but also for the flexibility in the system. The problem occurs because the dynamic PWM requires exact status of the existing PWM. The current status needs to be watched at the board level and the information needs to be transferred to the software level because it is hard for the software to keep track of time with very high accuracy. If the bus is slow then, the status information sent by the board could be obsolete when it arrives to the algorithm in the software.

One solution is to compensate for the bus delay. In order to compensate for the bus delay, the consistency of the bus delay must be carefully evaluated. In the case of the significant fluctuation of the bus delay, the dynamic PWM algorithm could be entirely performed by the FPGA board and it could be programmed dynamically by the software.

6.6 Future Implementation

One of the most significant advantages of the full software AC servo controllers is its flexibility. Some advanced control algorithms have already been implemented utilizing the controllers' flexibility. The dynamic PWM could be implemented as well.

Figure 6.6 shows the diagram for the implementation flowchart of the dynamic PWM in the full software AC servo controller. The chart is composed of three types of components, indicated in the key on the top left corner. The software and the FPGA need to have some intelligence in order to implement the algorithm. The software needs to calculate the compensated command given the time elapsed since the beginning of the current PWM. The FPGA board needs to distinguish whether to terminate the existing PWM and start a new one when the command from the PC arrives, or to store it in the buffer for the next PWM. The only concern is the transferring of information on the time elapsed since the beginning of the current PWM. The information may take about one microsecond to be transferred from the FPGA board to the software as discussed in section 6.5. The delay could be compensated in the software. However, if the delay is not consistent, the width of the PWM may be distorted from the desired width. This would require a careful assessment of the current system to determine whether there is significant inconsistency in the transferring of the information. If that is the case, then the FPGA board must contain all the logic to calculate the compensated high duration to implement the dynamic PWM.

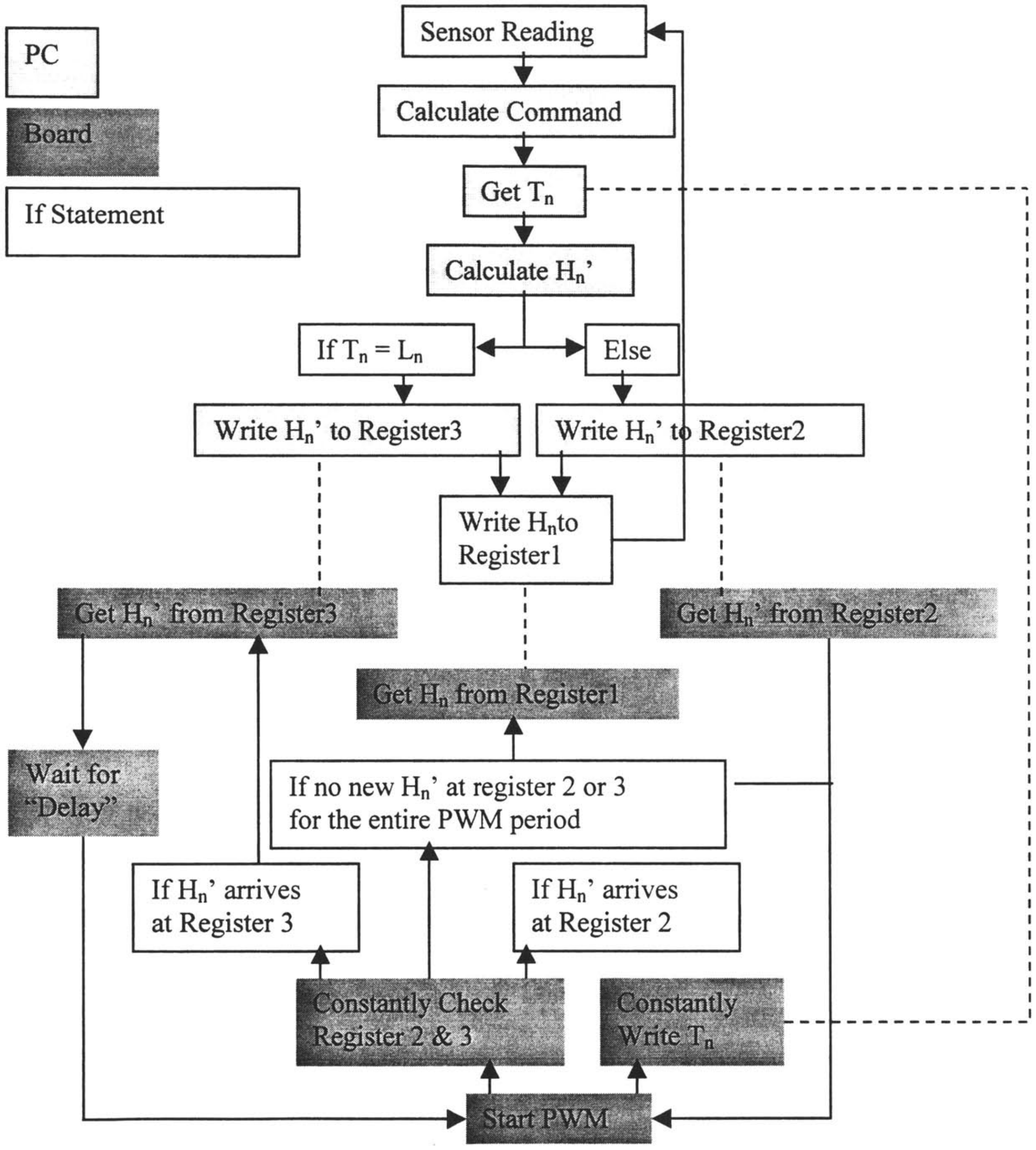

PC
Board
If Statement
Sensor Reading
Calculate Command
Get T_n
Calculate H_n'
If $T_n = L_n$
Else
Write H_n' to Register3
Write H_n' to Register2
Write H_n to Register1
Get H_n' from Register3
Get H_n' from Register2
Get H_n from Register1
Wait for "Delay"
If no new H_n' at register 2 or 3 for the entire PWM period
If H_n' arrives at Register 3
If H_n' arrives at Register 2
Constantly Check Register 2 & 3
Constantly Write T_n
Start PWM

Figure 6.6: Implementation Flowchart for Full Software Servo System

Chapter 7
Conclusions and Recommendations

The full software AC servo controller is developed for multi-axis AC servo control applications such as robotics and NC machines. The full software AC servo controller uses the Windows NT operating system. The Windows NT operating system provides a user-friendly interface and networking capability. In order to make Windows NT a real time controller, a special kernel level program was developed. It consistently handles external interrupts for the time critical real time operation. The reliable interrupt handling architecture as well as the characteristics of the real time control system is discussed. Issues concerning the disk controller are addressed and it was concluded that the disk controllers with bus mastering DMA compatibility could be used to insure the reliability of the interrupt generation in the controller. The experimental data showed that the full software AC servo controller is capable of controlling multi-axis AC servomotors simultaneously with reliable periodic interrupts. The overall description of the full software AC servo controller was presented and compared with the traditional AC servo controller. The networking capability of Windows NT was explored to present a concept for a centralized intelligent factory automation.

The time budget of the full software AC servo system was carefully analyzed in this thesis. The time budget of the old system with the ordinary commercial board showed that multiple axes could be controlled only at slow current sampling frequency. The bottleneck of the time budget was determined to be the ISA bus data transfer speed and the system was optimized with a new FPGA ISA bus board. The FPGA board minimizes the number of I/O readings and writings by eliminating unnecessary channel selection and triggering. In addition, the CPU loads were reduced by more than half. The experimental time budget of the new system showed that the multiple axis machines could be controlled with a single CPU. In order to improve the time budget even further, development of the PCI bus is suggested because the ISA bus I/O access is still the bottleneck of the new system.

The robotics and NC machine industries require fast bandwidth motion controllers. There are advanced algorithms that could improve the bandwidth of the system. In this thesis, d-q axis control and decoupling controls, other than the ordinary local three phase control, were implemented. The implementation is a rather simple task with the full software AC servo controller because of flexible architecture. The d-q axis control performed better than the local three phase feedback control. The local feedback controller had increasing internal power loss in the motor with increasing angular velocity. However, the d-q axis control kept the internal power loss at zero. The bandwidth of the system was faster with d-q axis control. However, the torque current was not maintained due to Back EMF for both controllers. Therefore, the decoupling control with Back EMF compensation was implemented. The results showed that the torque current was maintained at the desired torque, achieving the objective of an ideal controller. The velocity response with the Back EMF compensation had a faster bandwidth than that of other control algorithms.

A new PWM algorithm was designed and formulated by the author utilizing the flexibility of the full software AC servo system. The ordinary PWM has an effective delay which could result in up to one and half PWM periods. The Matlab simulation of the dynamic PWM showed that it could eliminate all or most of the delay caused by the PWM algorithm. It is also shown that the dynamic PWM could handle higher gains than the ordinary PWM in the current feedback at the same sampling frequency. The possible physical constraints of the dynamic PWM are also discussed and the solutions are presented. The dynamic PWM solves the fundamental delay problem in the PWM method. As a result, it could be a better foundation to build high performance robotics and NC machine controllers.

The flexibility of the full software AC servo system, especially with the dynamic link to the new FPGA interface board, provides the environment for the creation of new intelligent algorithms. It also opens up the possible implementation of the existing advanced control algorithms. It is cost effective and always readily available. The user interface is greatly improved. The networking and other features of Windows NT provide vast opportunities for the development of creative and intelligent motion controllers. The advantage of the full software AC servo controller is tremendous. It brings in the highly developed PC industry into the motion control industry. It enables the motion control industry to take advantage of the rapid development of PCs and its peripherals. The full software AC servo controllers will open a new era in the motion controller industry.

Bibliography

[1] R. H. Bishop and R. C. Dorf, "Modern Control Systems," Addison-Wesley Publishing Company, 7th edition, 1994.

[2] D. C. Hanselman, "Brushless Permanent-Magnet Motor Design," McGraw-Hill, Inc., 1994.

[3] B. Adkins and R. G. Harley, "The General Theory of Alternating Current Machines: Application to Practical Problems," Chapman and Hall Ltd., 1975.

[4] Y. Dote and S. Kinoshita, "Brushless Servomotors Fundamentals and Applications," Clarendon Press, 1990.

[5] G. F. Franklin and J. D. Powell, "Digital Control of Dynamic Systems," Addison Wesley Longman, Inc., 3rd edition, 1998.

[6] K. G. Shin and H. Kim, "Derivation and Application of Hard Deadlines for Real-Time Control Systems," IEEE Trans. Of Systems, Man, and Cybernetics, vol. 22, no. 6, Nov., 1992

[7] P. Puschner and C. Koza, "Calculating the Maximum Execution Time of Real-Time Programs," Journal of Real-Time Systems, vol. 1, no. 2, Sept. 1989

[8] M. Harmon, T. P. Baker, and D. B. Whalley, "A Retargetable Technique for Predicting Execution Time," Proc. of IEEE Real-Time Systems Symp., 1992

[9] A. Mok, "Evaluating Tight Execution Time Bounds of Programs by Annotations," Proc. of IEEE Workshop on Real-Time Systems and Software, 1989

[10] T. S. Craig, "Queuing Spin Lock Algorithms to Support Timing Predictability," Proc. of IEEE Real-time Systems Symp., Dec. 1993

[11] H. Takada and K. Sakamura, "Predictable Spin Lock Algorithms with Preemption," Proc. of 11th IEEE Workshop on Real-Time Operating Systems and Software, Seattle, May, 1994

[12] K. Jeffay, "On Latency Management in Time-Shared Operating Systems," Proc. of 11th IEEE Workshop on Real-Time Operating Systems and Software, Seattle, May, 1994

[13] Real-Time Systems and Microsoft Windows NT, MSDN Library, Microsoft Corporation, June, 1995