PC-BASED OPEN ARCHITECTURE SERVO CONTROLLER FOR CNC MACHINING

Surya Kommareddy, Yamazaki Kazuo, Kagawa Yoshihito
IMS-Mechatronics Laboratory, UC-Davis, Davis CA 95616,
surya@ucdavis.edu.

Abstract

There is an ever-increasing demand from the Industry for a flexible, modular and a cost-effective CNC machine servo controller. The primary aim of this research is to develop a real-time PC-based servomotor control system. Such a machine controller is not only required to perform control functions but also do other functions like database maintenance, tool path planning and optimization, and operator interface among many other useful functions. In realizing this objective, the environment is not only designed keeping in view the Open Architecture Controller (OAC) specifications but also to launch a platform for total factory automation. Good fault-tolerance, factory floor networking and flexibility in terms of software are considered in designing the system.

RT-Linux has been chosen as the suitable real-time platform with Windows NT as the front-end for the system. RT-Linux is very reliable and gives a good performance with a worst case scheduling latency of 12μsec and a worst case interrupt latency of 10μsec on a 650MHz Pentium III processor. The rationale behind choosing various other components of the system that makes it possible to meet the OAC specifications and the functional requirements is elaborated in this paper. In addition, system working environment and some of its good features will be discussed. This open source based CNC servo control platform is a promising technology for the future of factory automation.

Introduction

There is an eve-increasing demand from the industry for an Open Architecture Controller (OAC) which is flexible, modular and cost effective. PC-based control offers such flexibility and openness. Apart from realizing a PC-based servo control system, the PC can be integrated to several other intelligent manufacturing functions so as to realize a platform for total factory automation. In order to implement a controller inside the PC, there is a need for an appropriate operating system that functions in real-time. There are many real-time platforms based on which the controller can be designed, but most of them either fail to meet the functional requirements or the demands of the Manufacturing industry. The primary aim of this research is to develop a Real-time PC-based servomotor control system for CNC machining in conformance with the OAC specifications. The principal functional constraint on this system is on the implementation of the multi-axis feedback loops, with a fastest frequency of 20-50 μsec. In realizing this objective, the environment is not only designed keeping in view the OAC specifications but also to launch a platform for total factory automation. Good fault-tolerance, factory floor networking and flexibility in terms of software are considered in designing the system.

Background

There is a large exodus in the industry from the popular PLC control to PC-Based control. PC-Based control [1], [2] offers great advantages like faster design cycles, lower downtime using diagnostics and simulation tools, increased productivity and decreased maintenance costs. Today's increasing processor speeds and decreasing costs lure us to program most of the conventional hardware controller functions inside a personal computer. Such an approach can not only overcome problems like inflexibility, large physical space, time and costs of up-gradation present in conventional hardware controllers; but also provide us with a better user interface and multi-programming. With the present day high speed PCs loaded with one of the popular desktop Operating Systems like Windows NT would make a good control center for a CNC machine. There are many PC-based machine tool controllers for CNC available in the market today, but they are proprietary and the customer needs to depend on the software vendor for maintenance and support. Again these software solutions are volatile, costly and do not usually conform to the OAC specification [3]. So there is no need for open source platforms that can perform well and offer the user greater flexibility and accessibility. There area many open source real-time platforms available but the choice has to be made carefully considering the design requirements.

The main objective of this research is to develop an OAC platform and implement position, velocity and current feedback loops for multi-axis real-time actuator control, on a PC with Windows NT as the front-end Operating System (OS). Technically speaking, this would mean that the control tasks have to be implemented in a real-time fashion, along with many other usual, less priority tasks. Such a system is expected to service interrupts generated by current, velocity and position feedback loops with periodicity of 20-50 μsec, 100-500 μsec and 500-1000 μsec respectively [4]. The remaining time can be used for actuator path interpolation, database management, system monitoring and many other less priority tasks performed by the NT machine. In addition, the condition that the system so designed should not be affected by any up-gradations or changes in the Windows environment is to be met. Keeping these objectives in view, a platform was designed to implement a real-time servo control system complying with the Open Architecture Controller (OAC) specifications put up by the Industry.

System Concept

In a PC-based control system, the real-time kernel is the essential component and care must be taken in designing the OS. The design constraint imposed in the objective of this research is that Windows NT should be used as primary OS for performing the main machining center operations. In order to qualify as a real-time Operating System (RTOS), an OS needs to have a minimum set of requirements. These requirements are necessary but not sufficient:

- The OS needs to be multi-threaded and pre-emptable.
- The notion of threaded priority had to exist.
- The OS has to support predictable thread synchronization mechanisms.
- A mechanism for priority inheritance must be available.
- The OS behavior should be known and predictable (interrupt latencies, scheduling latencies, etc.)

Windows NT is a popular desktop OS, but there are several reasons what NT cannot be used as an RTOS [5], [6]. Some of the reasons are:

- The number of available priorities in the real-time class is too low for real-time applications.
- The problem of the priority inversion is not solved (for the real-time class process).
- Device drivers can take a lot of time in DPC and no preemption by other is possible.
- Virtual memory implementation involves address translation, swapping which is unacceptable for RT-process.

In view of the above arguments against NT, we can conclude that it is not suitable for ‘hard real-time’ applications. However, it can function as a ‘soft RTOS’ and is reported to be very unpredictable at heavy loads. NT can be made real-time by making some modifications or by some extensions to it.

There are four types of solutions to keep the advantages offered by NT and real-time capabilities to the system:

1. NT can be used as is by enhancing the interrupt handling mechanism by intercepting interrupts. One vendor, LP-Electronik, uses this method and intercepts the non-maskable interrupt (NMI). It is not encouraged to follow this procedure since it is risky as all the intercepts are disabled during the NMI.
2. The second solution would be to implement a Win32 API library on top of a commercial RTOS. The drawbacks of this solution is that standard NT applications cannot be executed any more and the huge set of device drivers can’t be used.
3. The third idea is to make coexist NT and a RTOS on one processor. This is a solution pursued by many vendors like VenturCom Inc. (RTX 4.1) [7], Radisys (INTime 1.20) [8] and Imagination Systems Inc. (Hyperkernel 4.3) [9]. A comparison study of the performances of these products can be found in the report [10] published by Powertrain Group.
4. The final idea is to extend the previous to a multi-processor environment. Here each operating system can be run on a processor independently.

A detailed study on the different solutions is presented in article [5] and [11]. The first and second solutions, does not serve our purpose directly. The fourth solution seems to be simple, but it is what we are trying to get away from; since in multi-processor environment the interrupt behavior of the inter-processor communication channel (VME or PCI or PMC) is not predictable and sometimes needs a customized board design. The third solution is the most popularly pursued by many vendors, i.e. provide real-time extensions to Windows NT. There are two ways to accomplish this:

1. To modify the Hardware Abstraction layer (HAL) by intercepting interrupts and including a small scheduler or RTOS.
2. To run NT as one of the tasks on top of a (supervisor) RTOS.

The HAL has to be modified or at least intercepted. Those HAL modifications such as manipulating clock and interrupt processing mechanism represent an unprecedented and unproved use of HAL. Interception of HAL is also possible via Intel processor tricks. These implementations are therefore only available on Intel based machines. All implementations under this solution need modifications of NT of some kind. Since the required design objective is to have freedom from the software vendor we follow the later method, i.e. running Windows NT as a task under an RTOS.

System Design and Implementation

Using the OAC specifications prescribed by the OMAC advisory group [3], a system engineering analysis with a popular tool called Quality Function Deployment (QFD) analysis, was done to design the system.

The use needs that are obtained from OMAC advisory group, classified into categories like infrastructure, discrete event control, information base management, task coordination, motion control, human interface, sensor interface etc. along with the functional constraints on the control feedback loops is used to derive at the requirements of each of the components of the system.

As required by the design constraints, all the control functions will be implemented in software, i.e. on the PC. Therefore, the component selection of PC for real-time performance is vital for a good OAC design. The essential controller characteristics obtained from the QFD matrix analysis are the real-time programming capability, GUI functionality, networking and non-Real-time programmability. Keeping in view these requirements and the performance requirements of the system RT-Linux is chosen as the Real-time Operating system, since it has short scheduling and interrupt latencies and further more it is an open source platform. Since RT-Linux comes with normal Linux OS, we have all the functionality of a normal desktop operating system. The functionality includes the networking, GUI, programming and several other functions. The other advantage of choosing Linux is that there are methods of communication with Windows machines built in the form of Samba file server. It offers a method of sharing files and other services available over network. For in-depth understanding of the working of RT-Linux, a number of pointers are available at http://www.rtlinux.org. References [12], [13] give a good introductory material on RT-Linux implementation.

RT-Linux offers low-level interface to the basic input/output peripherals. The I/O functions in the RT-

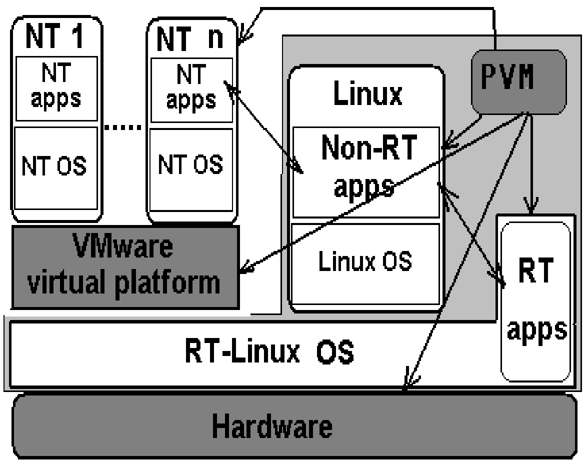

Figure: 1 The proposed system schematic.

Linux are performed as kernel modules, so it is very fast but has the limitation of not being able to perform heavy float point calculations. For a true real-time performance, the data from the I/O ports has to be collected and stored into some buffers, which can be accessed from the non-real-time Linux and processed when wanted. The RT-Linux offers two mechanisms in terms of FIFO and shared memory for data queuing and transfer through the FIFO or the shared-memory, data is accessed by a non-real-time application, processed and put back into another FIFO or shared-memory for the RT-Linux application to read. Some standard has to be established as to be consistent in approach in database management across various platforms. Shared memory can be used for faster exchange of data, since DMA can be used and it is faster. Finally, the data that needs to be stored is done so in files for later use for database management purposes.

Since we need to have Windows NT as the front-end OS, as required by the design constraint, we make it possible using the VMware virtual platform. Using this platform we can have multiple operating systems running on the same processor concurrently. VMware virtual platform can run multiple instances of the same operating systems (with same file system) under some restrictions. This makes it possible to have a fault tolerant system and failure detection of Windows NT, which is prone to crashes very frequently. This implementation can be extended to other versions of Windows OS as VMware platform supports wide range of popular operating systems present in market today. Further information on VMware virtual platform can be obtained in the reference [14].

Parallel Virtual Machines (PVM) [15] has been chosen as a way of implementing diagnostic functions and distributed computing in this design. PVM is a distributed Computing Environment which can be programmed in a variety of languages (Java, C, C++, FORTRAN) and can be run on different types of operating systems (primarily Linux and Windows NT). It also has some good features like failure detection and recovery functions, which are not commonly found in other alternatives like MPI. PVM transparently handles all messages routing, data conversion, and task scheduling across a network of incompatible computer architectures. This distributed computing environment can be used for coordinated factory floor manufacturing operations.

Web based Human Machine Interface (HMI) has been chosen as the method of obtaining HMI-GUI functionality for the controller. Since Windows NT is the front-end and it has to interface with the RT-Linux, which is the base, a network-based application is needed for data communication and other interactive functions. This web-based HMI is implemented in the form of client-server program, using Java and PERL scripts, and can be used in network computing and interfacing as well. The client-server implementation can be programmed with different functionality for access to the common database maintained in the PC, for the controller. This database can be used in conjunction with controllers of other machines to do system level control and management functions.

The control functions are implemented in Linux, which interfaces with the controller database for the system state data. There is always a delay in scheduling a task or in capturing the interrupt and running its service routine. These are considered as delays in the control system, when modeling the digital control system. The delay through the system can be calculated from the knowledge of the continuous time transport delay and the sample interval. The general expression for the time delay τ_d is given by

$$\tau_d = k\ \tau_s - m\ \tau_s$$

Where, τ_s is the sampling interval with k as an integer and m is a fraction. The influence of the partial delay ‘m’ is to introduce an extra zero in the numerator of the z-transform of the process. The position of this extra zero will vary with m, for certain values of m the extra zero will be outside of the unit circle in the z-plane, and thus instability is introduced. The stability of the system with the worst-case delay should be calculated and verified before implementing any control scheme. This is the limitation of this design, though in most of the applications this latency or delay can be ignored. In applications where the delay is critical, a self-tuning control scheme can be employed for strict control. Alternatively, with the knowledge of the time delays that occur on the scheduling or interrupt servicing we can change the command signal to the controller obtained from the path-planning algorithm.

System Performance Tests

All the components in the proposed design were integrated and tested for workability. Failure detection of Windows NT, which is running as a guest OS on VMware virtual platform, was successfully programmed using PVM. A PVM diagnostic application would detect the failure of windows NT as it is or an application under Windows NT and take appropriate corrective action. The property of PVM to dynamically configure its virtual machine by adding machines to it is used to take fault tolerant measures in case of failure.

The RT-Linux machine was tested for real-time performance at various loads. The determinacy was very good, even at heavy loads, but the task switching latency increased as the load increased. The typical task switching latency was found to be around 5-15 μsec. The worst case scheduling latency was found to be 25 μsec on a 300Mhz Pentium-II machine (with 256K cache and 128MB RAM). A similar test on a 650MB dual-processor machine with a worst case latency of 10 μsec is shown in Fig.2. This graph (and the subsequent ones) shows, the periodic cycle time of each sample, so the latency would be the difference between the mean and an individual reading. This test was done without the VMware virtual machine loaded. The same test was repeated with VMware loaded with a Windows NT guest operating system. The scheduling latencies remained the same, as the RT-Linux scheduler would distinguish CPU time requests from a RT-task and a non-RT task, which in this case is the VMware guest operating system.

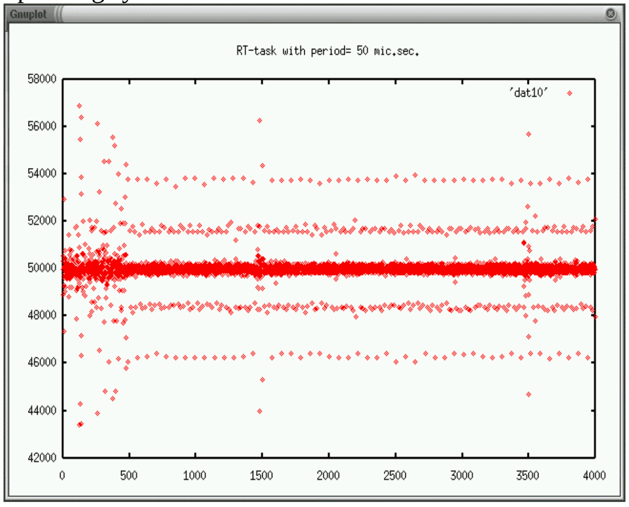

Figure: 2 Scheduling latency test at 80% CPU load on a dual processor machine.

The typical worst-case interrupt latency was found to be 10 μsec for RT-Linux, when the VMware virtual platform was not running as shown in Fig.3 on both single and dual processor machines.

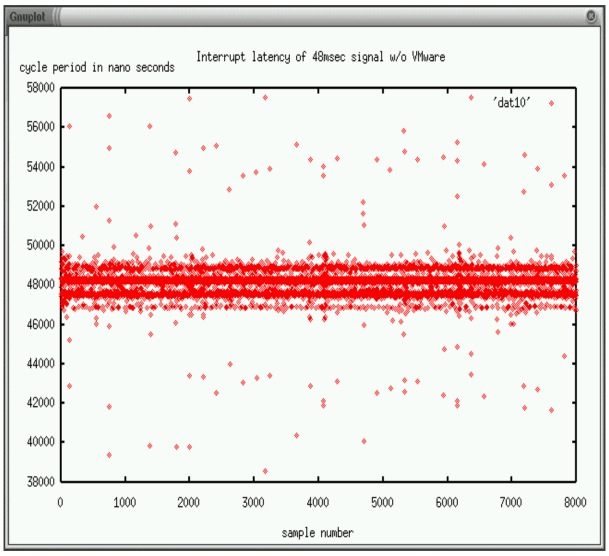

Figure: 3 Interrupt latency test at ~90% CPU load without VMware running on a dual processor machine.

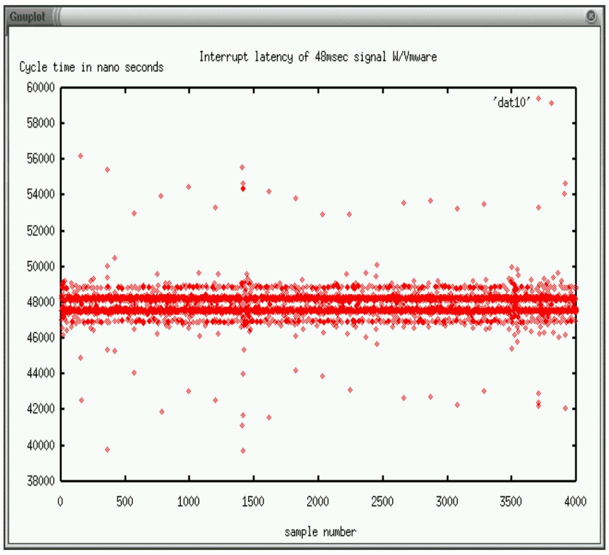

Figure: 4 Interrupt latency test at ~80% CPU load with VMware running on a dual processor machine.

These latencies will theoretically decrease with the processor speed. The worst-case interrupt latency was as high as 70 µsec in the presence of VMware virtual platform, on a single processor machine. The VMware attributes this to the occasional locking of the kernel when it is initialized, which cannot be avoided. The work around for this is to run on a dual processor computer and run VMware confined to a single processor to overcome the kernel lockout period. This strategy paid off a bit and the interrupt latencies reduced to around 10-12 µsec, as shown in Fig.4. There is still a kernel lockout period when the VMware initializes itself into the memory. Further research has to be done to reduce this.

The Windows NT environment was very slow (on a 300 MHz Pentium processor), since it is running as a guest OS on the VMware virtual platform (which is executing on the VMware virtual platform which in turn is executing on the host Linux OS). This problem can be partially solved by stopping all unnecessary tasks both in Linux and Windows NT and by programming important tasks in NT as real-time tasks. The performance in general improves with a faster processor or a second processor.

Concluding Remarks

The system so proposed is still under testing and improvement. This system is modular with greater flexibility with the choice of its components. The choice of the components makes it possible to meet all of the design, functional constraints and cost factor.

With the proposed system, two sets of programs have to be written, one running on the RT-Linux and the other on the Linux/NT. Real-time applications run under RT-Linux with the control applications and other diagnostic or path planning functions running either in Linux or in Windows NT. A good communication mechanism has to be established for data transfer between RT-Linux and NT. This mechanism should be fault proof, such that auto channeling of data is done in case of failure. Proper shutdown of the system should be possible in case of NT failure. Much of this functionality can be implemented using the PVM software.

The interrupt latency problem in presence of VMware virtual platform has to be solved for better performance. There are still problems with the kernel locking while initializing VMware virtual machine. There was some implementation problems with 'CPU TLB stuck' on some motherboards. This problem has been reported to the RT-Linux mailing list. Having a second processor is not a cost-effective solution again though it solves the problem partially.

A thorough testing of the system with a variety of scheduling requirements, task interaction and failure modes on a fully functional system has to be done. Right now only scheduling delays of the real-time tasks has been considered, but inter task communication delays with all sorts of real-time and non-real-time applications running should also be tested.

References

1. *Steeplechase Handbook, practical guide to PC-based control and flow-chart programming,* Steeplechase Software inc. 1999, ISBN: 0-9655406-1-8.
2. *PC-based instrumentation and control,* Tooley Michael, Oxford, 1991, ISBN: 0750620935.
3. *Requirements of OMAC1.1,* a white paper available at http://www.arcweb.com/omac.
4. *A study on flexible and integrated servo controller for actuators in mechatronic systems,* 1989, D. Eng. Thesis presented by Frank de Schepper at the Toyohashi University of Technology, Japan.
5. *Windows NT as a real-time OS,* M. Timmerman & J.C.Monfret, Real-time Magazine, issue 97-2.
6. *Inside Windows NT,* Davis A. Solomon, 1998, Microsoft Press, ISBN: 1572316772.
7. *Venturcom Inc.* homepage at http://www.vci.com.
8. *Radisys* homepage at http://www.radisys.com.
9. *Imagination Systems* at http://www.radisys.com.
10. *Hard real-time extensions of Windows NT evaluation report available at* http://www.vci.com/products/vci_products/rtx/rtx_nt_arc_test_results.html.
11. *Windows NT real-time extensions: an overview,* M. Timmerman & J.C. Monfret, Real-time Magazine, 97-2.
12. *RT-Linux, White paper,* Victor Yodaiken, Department of Computer Science, New Mexico Institute of Technology, available at http://www.rtlinux.org/documents.
13. *RT-Linux as embedded operating systems,* an article by Jerry Epplin in the October 97 issue of Embedded Systems Programming magazine.
14. *Technical White paper on VMware virtual platform,* Feb'99, available in PDF format at http://www.vmware.com/pdf/whitepaper.pdf.
15. *PVM, A User's guide and tutorial for networked parallel computing,* Al Geist, Adam Beguelin, and et.al. The MIT Press, 1994. ISBN: 0-262-57108-0.