RELIABLE, HIGH-SPEED DIGITAL CONTROL FOR MAGNETIC BEARINGS

Ronald D. Williams
Paul M. Wayner
Jeffrey A. Ebert
Stephen J. Fedigan
Center for Magnetic Bearings
University of Virginia
Charlottesville, Virginia, U.S.A.

ABSTRACT

Many magnetic bearing applications demand significant processing power, while others require higher reliability than that provided by a single processor system. A digital controller has been developed which strives to meet both of these needs in one powerful, flexible platform. This controller employs four processing modules based on the Texas Instruments TMS320C40 digital signal processor to provide hardware redundancy for fault-tolerance or for multiprocessing. A real-time, multitasking operating system has been designed to support all of these features, while aiding the application designer's implementation of complex control algorithms.

INTRODUCTION

The design of a magnetic bearing controller is a significant challenge for several reasons. High-speed rotors demand high bandwidth control systems, and the problems involved with tuning these systems demand flexible control systems. In addition, to make magnetic bearings practical and cost-effective in industrial settings, the controller must be very reliable for long periods of operation. Traditional analog controllers can meet the bandwidth requirements, but their reliability is often poor. Furthermore, the inflexibility of analog controllers hampers the setup of new rigs and new control schemes. Digital controllers offer tremendous advantages in terms of flexibility and ease of development, but have also traditionally suffered from low reliability. The digital controller under development at the University of Virginia is capable of operating in a fault-tolerant mode that will provide reliable performance without sacrificing bandwidth, flexibility, or ease of use. For applications that require extra computational power, this controller can be configured to operate in a multiprocessing mode.

This work was supported by Virginia's Center for Innovative Technology

BACKGROUND

The Center for Magnetic Bearings at the University of Virginia has been conducting magnetic bearing research in a wide range of areas since 1984. This research has included work in analog controllers, digital controllers and algorithms, power amplifiers, industrial applications, and magnetic bearing/rotor system dynamics [1].

Several digital controllers, tailored for magnetic bearing control, have been designed and built over the past ten years. The first effort was based on an IBM-type personal computer, using a multiplexed analog input/output (I/O) subsystem. This simple system successfully supported a small, flexible rotor, but the controller sampling rate was restricted by the 8086 CPU to approximately 3 kHz [2].

The second controller design attempted to improve performance by creating an architecture customized for magnetic bearing control. This design employed many separate digital control modules (DCM) for executing single-axis control, governed by a supervisory computer. Each DCM was based on the Texas Instruments TMS320C25, a fixed-point digital signal processor (DSP) [1]. This work initiated the use of DSPs for magnetic bearing control at the University of Virginia, a practice that continues in the present design effort.

The third-generation digital controller, the Integrated Controller for Magnetic Bearings (ICMB), was designed using a single CPU, the TMS320C30 floating-point DSP, with modular analog I/O capability [3]. Additional system hardware, such as a Motorola 68HC11 microcontroller, supports the 'C30, so that its

effort may be focused on executing the control algorithm. A real-time multitasking operating system has been developed for this system, easing algorithm design, development, and evaluation [4]. As a result of this work, the ICMB is in near constant use by a variety of projects for development of magnetic bearing control algorithms.

System Requirements

The fault tolerant design goal for the digital controller is to have a reliability of 0.999 over two years of continuous operation. To achieve this goal, modular hardware is used because it facilitates replacement and repair which greatly boosts long-term reliability. Processing modules with identical capabilities will also simplify the design and maintenance of software. Because of the need for high-speed processing in magnetic bearing controls, the fault-tolerant scheme is designed around hardware redundancy rather than time redundancy or information redundancy, both of which can put burdens on the processor and thus reduce processing speed.

Since this platform is targeted at multiple projects with evolving needs, the system must be scalable in regard to processing performance and the number of analog I/O channels. This platform can have multiple processors or just one, a few analog channels or many, plus additional cards for digital I/O can be added, all depending upon the needs of the application. System software should permit a user who has written an application program for a certain number of processors to modify that program easily to execute on either a smaller or larger number of processors. Thus, the system can be configured for very high performance or a less expensive version can be built and then expanded later.

Over the past two years, a suite of applications has been developed on the University of Virginia's 'C30 based hardware platform, including routines that implement high speed digital filters and adaptive on-line balancing. For the most part, these routines have been written in C, with certain time-critical portions written in 'C30 assembly language. The code has been compiled with an Optimizing C Compiler provided by Texas Instruments. Additionally, these routines include service calls to a real-time operating system. Since considerable effort has gone into writing and debugging existing application programs, an important requirement is portability between the 'C30 and 'C40 platforms. Compatibility is ensured by using the 'C40, whose instructions are a super set of the 'C30's, the same suite of development tools, which can generate code for either the 'C30 or the 'C40, and retaining the same application program interface (API) in the advanced version of the Real-Time Operating System (RTOS).

Architecture Specification

The architecture of the system may be divided into three major sections: a processor subsystem consisting of one to four computer modules; a communications and self-testing subsystem, or processor-I/O link (PIOL); and an I/O expansion bus with ten slots. The computer modules are TIM-40s, a Texas Instruments open-standard that is based on their TMS320C40 DSP [5]. The processor subsystem is a fully-interconnected network of these computer modules. The PIOL provides a communication link between the TIM-40s and the I/O cards. The I/O boards include data converters, digital I/O transceivers, and serial devices.

The architecture can be configured to achieve two different system-level requirements: high reliability or multiprocessing. For high reliability the modules are connected in a fault-tolerant configuration as shown in Figure 1.

To provide fault-tolerance for the processing section, we use the redundant TIM-40s to provide Triple Modular Redundancy (TMR) with the additional TIM-40 acting as a hot spare. All the TIM-40s are kept hot, running the same code. The PIOL provides a synchronizing clock signal to all the TIM-40s to ensure that they perform I/O operations at the same time. The PIOL observes the TIM-40 behavior looking for errors. After a TIM-40 has

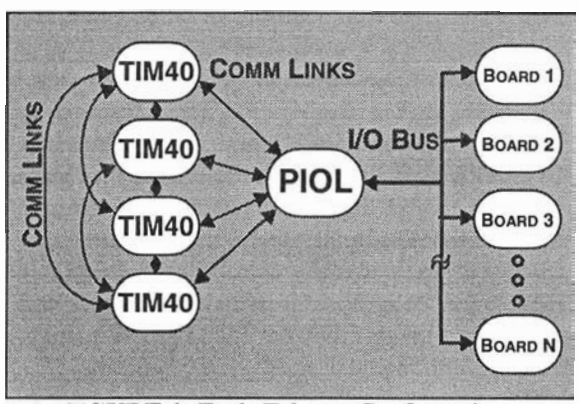

FIGURE 1: Fault-Tolerant Configuration

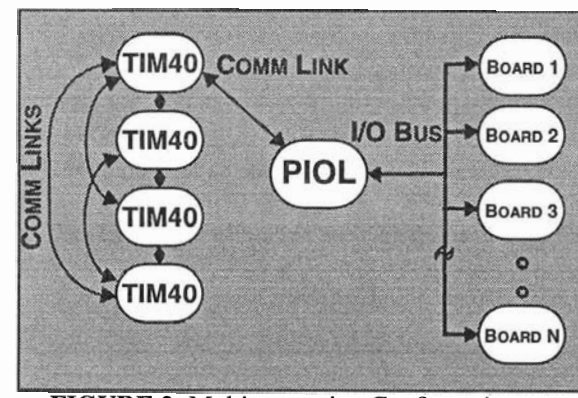

FIGURE 2: Multiprocessing Configuration

made a certain number of errors it is replaced.
For multiprocessing the processors are connected as shown in Figure 2. Only one TIM-40 is connected to the PIOL and it controls all access to the PIOL. The full interconnection of the TIM-40s allows for them to be used in different multiprocessing arrangements such as pipes or an array.
Both configurations are available using the same hardware design. Switching between the two configurations only requires changing the settings of the PIOL and altering the software.

TIM-40 Modules

The computing power of the fourth generation digital controller resides in several computing modules called TIM-40 modules. These modules consist of separate printed circuit boards that are plugged into the motherboard. There are several advantages to this scheme. The TIM-40 modules are based on an commercial open standard, thus it is possible to purchase TIM-40 modules from several commercial vendors and those modules can be used in the controller without modification. The modules are easily replaceable, thus facilitating replacement for upgrading and repair.
The TIM-40 standard provides physical and electrical specifications. The TIM-40 printed circuit board (PCB) measures 4.2" by 2.5" and plugs into the motherboard through two connectors. The processor used is the Texas Instruments TMS320C40 DSP running at 40 MHz. The 'C40 is optimized for DSP applications, and it is capable of a peak performance of 160 million operations per second (MOPS) plus an additional 60 MOPS from the direct memory access (DMA) coprocessor. The module also has 256 Kbytes of zero-wait state read-write memory (SRAM) and 128 Kbytes of read-only memory (EPROM). A simplified block diagram of the module is shown in Figure 3.
The 'C40 Communication Ports (comm ports) are used to pass data bidirectionally to the motherboard at a rate of 20 Mbytes/sec. The global bus connection is not used on our motherboard, however, our TIM-40 module has the connection available for possible future use.

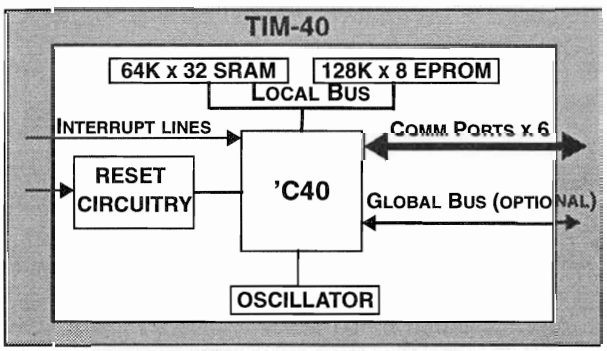

FIGURE 3: TIM-40 Module

Processor-I/O Link

High reliability and fault tolerance were motivating requirements for the digital controller design, but implementations that sacrificed throughput or flexibility were deemed unacceptable. As a result of this decision, we avoided commonly used software error checking techniques and large-scale data encoding methods. A hardware-intensive redundancy and repair scheme was adopted to maintain high throughput while hiding fault tolerance features from the application designer. Later, a separate mode was added to support non-fault-tolerant, multiprocessing operation.
The hardware that provides these features is embodied in the Processor-I/O Link (PIOL), a collection of programmable logic devices that reside on the motherboard of the digital controller. As shown in Figure 4, the PIOL consists of a dedicated communication channel for each of the four TIM-40 processing modules, a central, specialized voting unit, a repair and reconfiguration unit, and a bus controller for the I/O Bus.
As the name suggests, the primary purpose of the PIOL is to link the processor and I/O subsystems. The communication channels emulate the asynchronous comm ports that are found on the 'C40. These communication channels synchronize messages that are transmitted asynchronously by the independent TIM-40s, and the bus controller performs synchronous reads and writes to the various I/O modules that may be connected to the bus.
Application software performs I/O reads and writes by calling the appropriate operating system service routine, passing in the address of a table where the data to be written resides or where the data to be read should be placed. The RTOS initiates a data transfer by writing the appropriate command word to the PIOL. The command word specifies the type of operation, the base location in the I/O address space, and the number of words to be transferred. Block reads and writes as large as 64 words are possible. By using the data transfer facilities of the 'C40 in conjunction with the PIOL, the digital controller is capable of performing I/O accesses in parallel with computations, thus increasing average throughput and sampling frequencies.
The PIOL operates in one of two modes, fault-tolerant or multiprocessing. Thus, the logical configuration of the processor subsystem may be altered by changing the mode of the PIOL when the digital controller is initialized. Figures 1 and 2 illustrate these two logical configurations.
In the fault-tolerant mode, the PIOL provides both passive error correction, by fault-masking, and active repair and reconfiguration, by using accumulated error information. All four of the TIM-40 modules are programmed with the same software, and three of these execute this software simultaneously. The fourth is run-

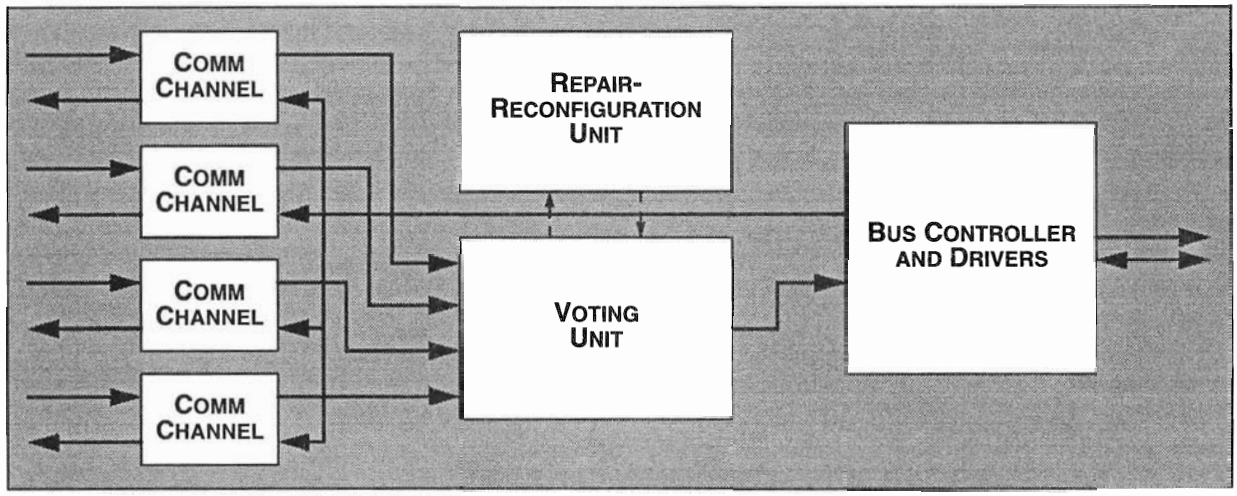

FIGURE 4: Processor-I/O Link Block Diagram

ning, but its outputs are ignored; this is the standby spare which may be used in case of a failure.

Messages that are transmitted by the TIM-40s, whether command or data, are synchronized and passed through the voting unit. If one of the three working processors has experienced a fault, producing an erroneous output, the voting unit will mask this error because the other two processors agree on the correct output. When a read occurs, the same data is transmitted to all three processors so that the inputs to the control algorithm running on each are the same.

After a number of errors from a particular TIM-40 module have been detected and masked by the voting unit, the repair and reconfiguration unit reboots the offending processor in an attempt to correct the problem. The active processors continue to execute the control algorithm, supporting the rotor, while the bad processor restarts and performs a series of self-tests. If the bad processor fails to recover, or if a particular processor must be rebooted frequently, it is marked as failed and replaced by the standby spare. The failed module is electrically isolated from the motherboard, and a technician may then replace it without shutting down the controller.

For applications in which performance is more important than high reliability and long-life, the digital controller may be configured for multiprocessing. In this mode, the PIOL communicates with only one of the four TIM-40 modules, and the voting unit and repair unit are disabled. A web of 'C40 comm ports form a fully interconnected network so that the TIM-40 module that is bound to the I/O may act as a server for the other modules, as shown in Figure 2. Many different parallel processing configurations are possible, and any number of TIM-40 modules, from one to four, may be used without any alteration to the PIOL hardware or the operating system communication routines. Thus, a low-end or developmental system with a single processing module is also possible.

I/O Bus

The I/O Bus must provide a two way communication link between the PIOL and the I/O boards. This I/O Bus must support data transfer rates in the range of the comm links while also providing an appropriate level of fault-tolerance. The I/O bus has a data transfer rate of 10 Mbytes per second which is close to the maximum rate of the comm links. The data and control lines of the I/O bus are encoded with a one-bit correction two-bit detection code to provide fault tolerance for the bus.

The I/O Bus is a 5 MHz synchronous bus controlled by a single controller which is a part of the PIOL. Data is read from and written to I/O boards in the form of 16-bit words. The bus can address 64 boards with 8 channels per board for a total address space of 512 channels. The bus has three operations which are PUT, GET, and IDLE. The PUT operation writes a block of data from the PIOL to sequential channel addresses on the I/O boards. The GET operation reads a block of data from sequential channel addresses on the I/O boards to the PIOL. The IDLE operation causes the I/O controller to neither read or write. Updating board values is done by performing a PUT operation to a reserved address which causes the appropriate board to update its values. It is also possible to update all inputs or all outputs or all inputs and outputs by performing a PUT operation to reserved addresses.

The ability to perform block data transfers makes it easy for the system to read all the input data in one operation and write all the output in another operation. The updating scheme allows simultaneous updating of inputs and outputs. This facilitates state-space control by allowing the controller to capture the system state with one operation.

Real-Time Operating System

To facilitate rapid prototyping of control algorithms, each processing module will be equipped with an advanced version of the Real-Time Operating System developed at the University of Virginia specifically for magnetic bearings applications. The current version of the RTOS provides implementation support for control algorithms executing on a hardware platform centered around a single Texas Instruments TMS320C30 Digital Signal Processor.

For a simple algorithm that employs multiple independent channels of feedback control, the current version of the RTOS directs the sequencing of events in an control cycle and handles all the low level interaction with controller hardware. At the beginning of a control cycle, the RTOS issues a sampling command to the analog input board, and selected input channels are sampled. When the conversion cycle has completed, the RTOS retrieves the data from the A/D board, transfers this data to a specified buffer, and activates the control program. The program can immediately begin transforming the input data into a set of actuator commands which are handed off to the RTOS when the computation is finished. The RTOS suspends the control program when the new commands are received, and waits for the end of the cycle before updating selected channels on the D/A. Despite variations in the control program's run time, the RTOS maintains a uniform sampling rate and a fixed time delay from input to output. This benefits the feedback controller by permitting all discrete-time coefficients to be calculated in advance, thus reducing the computation to a series of multiply and accumulate operations, which the DSP can perform efficiently.

A more sophisticated control scheme can benefit from the pre-emptive, prioritized multitasking environment provided by the operating system. Consider an adaptive balancing system which cancels rotor synchronous vibration at the sensor locations. The algorithm works by injecting a set of synchronous balancing forces, and observing the synchronous response. From this data, it constructs an influence matrix which defines the gain and phase relationship between every force input and sensor output. In turn, this influence matrix is used to determine the set of open loop forces which will minimize shaft vibration at selected sensor locations. Since these open loop forces cancel only the imbalance and do not stabilize the rotor, the automatic balancing scheme must operate in tandem with a feedback controller.

The control scheme just described has program elements which operate on widely different time scales. The portions of the algorithm which calculate feedback control forces, inject the synchronous forces, and calculate the synchronous responses all should be executed a great many times during a rotor revolution and on a periodic basis, while estimating the influence matrix and calculating a new set of balancing forces may take place over several rotor revolutions.

To implement such a control algorithm, the user may place the periodic high frequency and non-periodic low frequency program elements in a separate program loops, or tasks. The high frequency periodic task can be assigned a high priority and the low frequency non-periodic task a low priority. When the high priority task completes its execution for a particular I/O cycle, it voluntarily suspends itself, and the RTOS automatically switches to the low priority task. The low priority task executes until the beginning of the next I/O cycle at which point, the RTOS preempts the low priority task, returning control back to the high priority task.

With this arrangement, the RTOS can allocate a small slice of processing time to the low priority task between successive runs of the high priority task. Since time slicing and context switching is handled automatically, the low priority task can be written as though it executes without interruption. This approach affords an important benefit. Providing that the interaction between the tasks is defined at the outset and remains the same, each control task can be developed, and modified independently. For example, this modularity enables the user to experiment with different estimation or minimization schemes in the low priority task without the need to modify the high priority task.

In fault tolerant mode, each TIM-40 module will execute identical versions of the advanced RTOS and the same set of application tasks. While the new RTOS will direct the sequencing of events in a control cycle, the underlying hardware interactions will be different from the 'C30 platform. For example, when the application issues an I/O request to the operating system, the RTOS will perform a block transfer between local memory and the communication port linked to the PIOL instead of directly reading and writing to memory-mapped I/O.

In multiprocessing mode, only one of the TIM-40 modules can interact directly with the PIOL. The RTOS on this module will act as a server for I/O requests from neighboring modules and from its own application tasks. The RTOS on the other three modules will forward I/O requests from their application tasks to the RTOS acting as I/O server. The I/O requests will be forwarded by sending messages through the communications ports linking the I/O clients to the I/O server. When an I/O request has been fulfilled, a message will be returned to the sender's RTOS, which will in turn pass it to the task that originally made the request. From the application's standpoint, the module appears to have its own local bank of I/O even though only a single bank of I/O exists on the entire processing system. Except for the most I/O intensive tasks, the virtual I/O described here gives the designer freedom to move tasks across processor boundaries. This could prove useful if a sec-

ond module is added to a single module system; tasks with moderate to low I/O bandwidth requirements could be shifted to the second processor to improve system performance without difficulty.

In a multitasking multiprocessing control system, the control algorithms must be broken down into a set of tasks, and those tasks must be mapped onto the existing processor topology. To maximize system performance, it may be necessary to evaluate several different task to processor mappings. If separate operating system service calls are required to link tasks on the same processing module verses tasks in separate modules, the application code would have to be rewritten every time interacting tasks are moved from the same to separate processing modules. Furthermore, if two tasks are on separate TIM-40 modules, an interprocessor communications service may require each task to identify on which processor the other resides. If a task moves across module boundaries, service call parameters would have to be changed.

To allow the control engineer to evaluate different mappings rapidly, the advanced operating system will support seamless intertask communications. This means that two communicating tasks need only know each other's unique identifiers. Messages reach the destination task regardless of where it is actually located. With this scheme, tasks can be moved freely among processors without the need to modify calls to task communications services.

To ensure high performance gains in a parallel processing architecture, the overhead required for intertask communications must be kept to a minimum. The advanced RTOS implements efficient intertask communications by moving data only when absolutely necessary. When two tasks are on the same module, instead of copying data from one task's buffer queue to another's, the two tasks exchange a buffer pointer. By exchanging pointers whenever possible, data is only moved twice during its lifetime on a particular processing module: once when it comes onto the module, and once when it leaves.

When two tasks reside on separate modules, data must be transferred between them. The RTOS virtually eliminates the CPU's participation in the data transfer by taking advantage of the C40's six channel DMA coprocessor [5]. Without the coprocessor, the CPU would have to spend valuable processor cycles transferring data between the communications ports and local memory. With the coprocessor, the CPU can spend more of its time performing computationally intensive tasks, by delegating burdensome transfers to the DMA controller. In fact, the coprocessor even has a special split mode, designed specifically for transfers between the communication ports and local memory.

To simplify setting up a DMA transfer, the RTOS reserves space in intertask communication buffers for DMA initialization data which does not change once the buffer is created. The DMA channel can even autoinitialize, sparing the CPU from loading all the control registers every time a DMA transfer is initialized. To begin a transfer, all the RTOS does is set a pointer to the initialization data and set a start flag in the DMA control register. This reduces the CPU time spent with interprocessor transfers even further.

SUMMARY

The real-time operating system in combination with the hardware architecture will support both the high reliability and high performance missions of the controller. Schemes that carry out fault tolerant coil control can be implemented as background tasks without significantly impacting the performance of the feedback controller. The integrity of the outputs delivered through this mapping scheme are ensured by the processor I/O link, which implements triple modular redundancy with active standby sparing. Sophisticated control schemes, such as one which simultaneously cancels vibration at the running speed and selected harmonics demand the high performance offered by the multiprocessing mode. The flexible processor topology permits the designer to distribute the processing load in a way that will provide the highest possible throughput. The real-time operating system in turn enables the designer to determine rapidly the optimum task to processor mapping by providing both seamless intertask communications and virtual I/O.

REFERENCES

1. D'Addio, J.K., R.D. Williams, F.J. Keith, S.J. Fedigan. "Advanced Digital Controller Design for Magnetic Bearings." In Proceedings of the Conference on Recent Advances in Active Control of Sound and Vibration, p. 408-419, Blacksburg, Virginia.April, 1991.

2. Keith, F.J., "Digital Control System Design for Active Magnetic Bearings," M.S. Thesis, University of Virginia, May, 1988.

3. D'Addio, J.K. "An Integrated Magnetic Bearing Controller," M.S. Thesis, University of Virginia, May, 1992.

4. Fedigan, S.J., "A Real-Time Operating System for a Magnetic Bearings Digital Controller," M.S. Thesis, University of Virginia, May 1993.

5. *TMS320C4X User's Guide*, Texas Instruments, 1993.