Design of an Open Source, Hard Real Time, Controls Implementation Platform for Active Magnetic Bearings

Edgar F. Hilton (efhilton@alum.mit.edu),
Marty A. Humphrey, John A. Stankovic, and Paul E. Allaire
University of Virginia, Charlottesville, VA, USA

Abstract

Suspension of an active magnetic bearing (AMB) system involves significant computational and monitoring effort. Modern AMB systems need 1) logically complex and computationally expensive controller algorithms and I/O; 2) real time data storage of plant information such as states, inputs, and outputs; 3) real time plotting capabilities (such as rotor displacement as a function of time, actuator forces as a function of time, and FFTs); 4) real time controller parameter updates; and 5) real time access to reference signals; and 6) remote monitoring for safety purposes. Control systems based upon embedded Digital Signal Processors (DSP) boards often require specialized programming and development tools, may lack flexibility when computational requirements change, are often financiallly expensive, and may not directly address the aforementioned needs. Even worse, newer and faster DSPs may not be fully pin compatible with their predecessors, thus requiring total redesign of the embedded electronics for a given project.

A novel DSP-less controller implementation system – the Real Time Controls Laboratory (RTiC-Lab) – has been developed explicitly to address these problems. It uses the concept of Real Time Operating Systems to guarantee hard real time constraints. Its design is based on the use of both commodity personal computers and RT-Linux, a free and Open Source modification of Linux intended to support hard real-time computation. RTiC-Lab is also free and Open Source and is intended to serve as a communal effort among the controls engineers in both the controls and AMB community. Discussion is presented on the design of the software architecture, defining timing requirements of the control tasks, and measuring the predictability of RTiC-Lab. An example application is presented which uses RTiC-Lab in the controls implementation on a five degree of freedom AMB.

1 Introduction

AMBs now require multiple computational tasks such as (in order of importance) 1) periodic fixed rate suspension controllers, 2) a key phasor monitoring task, 3) auto-balancing controller, 4) data visualization tasks, 5) network transfer tasks, and 6) miscellaneous additional tasks (e.g. watchdogs and screen refresh tasks). Commonly, each of these is implemented as a sequence of commands, interpreted one command at a time, in a digital computer. All of these tasks may be trivially implemented if enough *independent* computational engines are available. A more cost effective solution is to implement all of these tasks in one single central processing unit (CPU), using some of the many optimal scheduling algorithms that are currently available in the field of Real Time Systems (RTS).

Full embedded control for an AMB system requires significant computational effort – especially for flexible rotors. This is caused by the inherent open loop instability of all AMBs. The success of the AMB is heavily dependent on the design of the controller. In turn, the controller relies heavily on *a priori* knowledge of the plant dynamics. Thus, considerable modeling, characterization, and controller parameter calibration effort is necessary during the early controller implementation stages for a given AMB application. The controls engineer needs to fully evaluate AMB performance via considerable access to plant input/output (I/O), controller states, and controller parameters. Most importantly, and for safety reasons, in high speed AMB applications the controls engineer needs to get access to this data from a safe location which may or may not necessarily be even in the same building.

An important aspect of RTS is the effectiveness of resource allocation strategies so as to satisfy stringent timing-behavior requirements [15]. This is especially true for AMBs. The proper design of an RTS for control requires solutions to many interesting problems - for example, specification and timing behavior; programming language semantics dealing with time, and the use of timing constraints. The correct functioning of the system depends upon an implementation which evaluates the logical power of different forms of timing constraints in solving various coordination problems and determines the least restrictive timing constraints sufficient for the control system. Unlike other combinatorial scheduling problems in operations research which mostly deal with one shot tasks, in real time control systems, the same task may recur very often, either periodically or at irregular intervals, and may have to synchronize or communicate with a number of other tasks [16]. This is the case with AMB control systems such as AMB supported artificial hearts [8, 2, 1] and high speed energy storage flywheels for powering communication satellites [4, 13].

The primary objectives of RTS design for automatic controls include 1) automation of the design and implementation process by exploiting optimizing transforms and scheduling theory, and 2) the synthesis of highly efficient code and customized resource schedulers from timing constraint specifications. Reliance on clever hand coding and difficult to trace timing assumptions are major sources of bugs in real-time programming that can be avoided with recent advances in hard real time structured real time operating systems.

The RTS literature is vast. Fortunately, AMB control applications are best served by basic understanding of scheduling based results. Of these, there are two types, both static (the scheduler has full knowledge of previous, present, and future tasks, as would be the case for a fixed rate controller) and dynamic (the scheduler has full knowledge of previous and current tasks, but does not know the time nor the number of tasks that will arrive in the future) [12, 10, 16]. This paper does not attempt to review all of these algorithms since some of these do not lend themselves too well for AMB applications. Instead, this paper concentrates on one of the so called "static" scheduling algorithms: "Fixed Priority Scheduling" (FPS). This scheduling algorithm is especially useful for periodic tasks, as would be the case in the implementation of most fixed rate controller algorithms such as AMB suspension controllers. This algorithm is used extensively in the development of a hard real time controller implementation platform, the Real Time Controls Laboratory or RTiC-Lab (pronounced Arctic-Lab).

The following sections are set up as follows. First, RTS and their associated basic scheduling results are presented. Second, RTiC-Lab is presented. Finally, implementation results are shown which demonstrate the predictability of RTiC-Lab.

2 Real Time Systems

"Real Time Systems", are systems in which the temporal correctness of the system is at least as important as the logical correctness of those results. For example, in a high speed AMB, it is imperative that our fixed rate suspension controller provide both a strict sampling rate of 8 kHz and the correct control signal – irrespective of the CPU's load . Failure to do so could cause our AMB to catastrophically fall out of suspension.

Unfortunately, there are several misconceptions [15] regarding RTS which have inhibited the AMB and controls communities from both identifying the need for, and correctly implementing RTS technology. Four of the perhaps most commonly cited misconceptions are:

1. *faster hardware implies that all deadlines will be met*: while it may be true that faster hardware will minimize the mean response time of our system, it does not necessarily imply that the system will be predictable [17], that is, that it will execute precisely at the requested sample rate.

2. *RTS are equivalent to control systems programmed in assembly coding, interrupt programming, and complex device drivers*: one of the most important research aspects of real time systems is that researchers concentrate on developing powerful scheduling algorithms and tools that will satisfy all hard timing constraints. Consequently, a controls engineer can now use high level code such as Ada and C instead of a more arcane and platform specific assembly language.

3. *RTS are all developed in an* ad hoc *fashion*: RTS research concentrates on developing powerful, flexible, and structured techniques that formalize the actual development and implementation of RTS. Many structured tools – such as hard real time operating systems – now exist to help develop, validate, and simulate real time systems.

4. *real time is equivalent to fast computing*: "fast" is relative. That is, in the AMB community, a sampling rate of 100 μs is considered "fast". In the robotics community, a sampling rate of $1,000$ μs is already considered "fast". In the geo-sciences community, a sampling rate of $86,400,000$ μs (1 day) is considered "fast". In all three systems, it is imperative that tasks execute at *precisely* the given time or else the results may no longer be valid. Consequently, all three systems are categorized as "real time systems".

The RTS community focuses on many aspects of real time research such as real time hardware, software, validation, and simulation. We, as end users, do not need to understand all of these and must rely on the RTS community to develop most of these technologies. However, it is imperative that we understand, as a minimum, some very powerful scheduling results which greatly aid in the design process of RTS and consequently in the implementation of hard-timing control environments.

Present day AMB applications can best be served by a 1955 job-shop scheduling result [5, 6, 16]: *The maximum lateness and maximum job tardiness are minimized by sequencing the jobs in order of non-decreasing due dates.* This dynamic algorithm, referred to as the *Earliest Deadline First* (EDF) algorithm, is *optimal* for *uniprocessor* systems. Optimality is quantified by: *An optimal scheduling algorithm is one that may fail to meet a deadline only if no other scheduling algorithm can meet it.*

FPS is a scheduling scheme where tasks are assigned a *priority*, or relative importance. Higher priority tasks are given precedence over lower priority tasks, and tasks having the same priority level are assigned into the CPU on a “first come, first served” scheme. Tasks may only *preempt* lower priority tasks and are assumed to be completely independent of each other.

For present day AMB applications, *a priori* assignment of task priorities will best meet the hard deadlines [9] via the *Rate Monotonic Algorithm* (RMA) [11]. This static algorithm assigns priorities to each of the tasks in the following fashion: *The priority of the task is inversely proportional to its period.* In other words, under this assignment policy, tasks having the smallest period (highest frequency) – as would be the case for a suspension controller in an AMB – would have the highest priority. Of most importance is that this algorithm is *optimal* among FPS algorithms for RTS, and that it can be applied for any number n of tasks.

Most importantly, using the RMA algorithm, it is possible to – *a priori* – determine if a set of n tasks will meet all their hard deadlines [11] by the sufficient condition:

$$\sum_{i=1}^{n} \frac{C_i}{P_i} \leq n(2^{1/n} - 1) \qquad (1)$$

where C_i is the worst case execution time of the task, P_i is the period of the task, and the left hand side of (1) is called the total CPU utilization. Note that as the number of tasks becomes infinite, all tasks will meet their deadlines as long as the utilization remains below 69.34%. Thus we now have an effective method of correctly sizing a CPU for a given application.

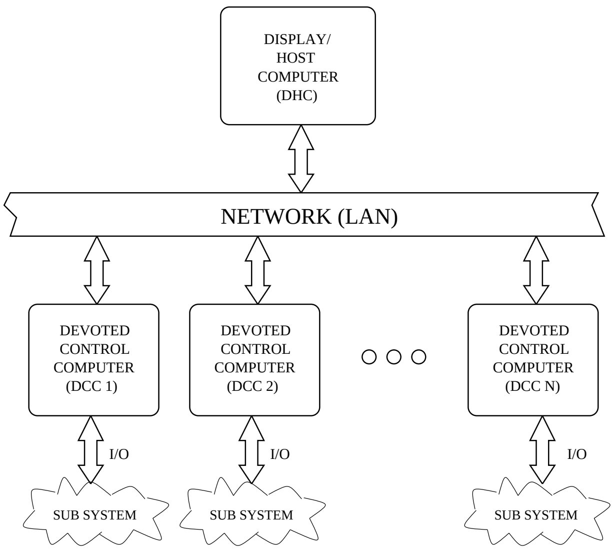

DISPLAY/ HOST COMPUTER (DHC)
NETWORK (LAN)
DEVOTED CONTROL COMPUTER (DCC 1)
DEVOTED CONTROL COMPUTER (DCC 2)
DEVOTED CONTROL COMPUTER (DCC N)
I/O
I/O
I/O
SUB SYSTEM
SUB SYSTEM
SUB SYSTEM

Figure 1: Overview of the Real Time Controls Laboratory network environment

Unfortunately, RMA is optimal only for periodic and *independent* tasks. And, since there is a large need for communication between controller tasks in an AMB (for example, controller states from the suspension task are communicated to a second task which in turn plots the data to the screen), then the RMA algorithm becomes limiting for hard real time applications, sometimes with catastrophic effects [7]. Fortunately solutions have been presented [14] to curve this problem. Interested readers and system developers are *strongly* encouraged to read further on this topic.

In what follows, an AMB control solution has been implemented in a novel controls implementation platform using RTLinux and a set of networked personal computers.

3 The Real Time Controls Laboratory, (RTiC-Lab)

Control of AMBs require an exhaustive tuning and characterization process during the early stages of the AMB life. RTiC-Lab, is explicitly designed to be used not only during these early stages of controller design and plant characterization, but also during subsequent monitoring and control. Designed and tested at the University of Virginia’s Rotating Machinery and Controls Laboratory, it provides an environment in which to implement controller algorithms while providing real time access to controller states, plant outputs, controller actions, controller parameters, and other controller information. All this information can be plotted and filtered – via user defined filters – in

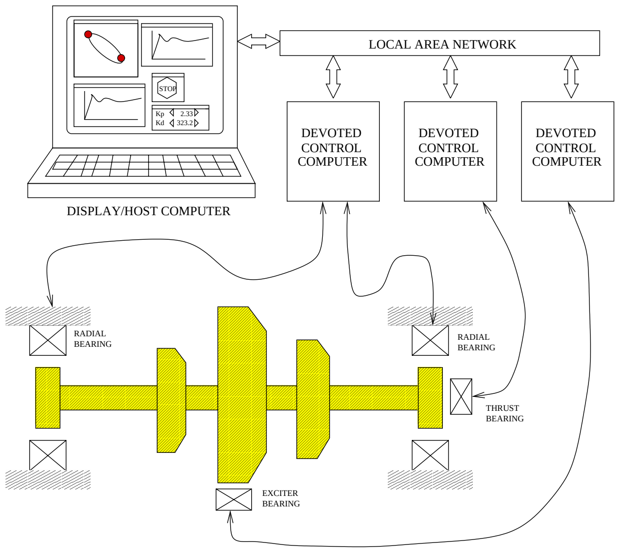


Figure 2: Example of the applicability of RTiC-Lab on an AMB system.

soft real time. The user can further filter the necessary data either in soft real time or *post mortem*. Last and most importantly, the controller parameters can be updated in real time through a user-defined graphical user interface.

RTiC-Lab attempts to incorporate several of the most critical RTS results in order to make RTiC-Lab a powerful controls implementation platform for AMBs and any other system that can use both fixed rate and event driven controllers. Priority assignment has been employed with Liu and Layland's RMA scheduling algorithm. Data is transmitted from the hard real time tasks to the graphical user interface via real time FIFOs.

RTiC-Lab has two very important features not found in any other real time controls implementation platforms. The first and most important one is that RTiC-Lab is and will be – as its underlying Linux and RTLinux platforms – Open Source Software released and protected under the Free Software Foundation's General Public License. That is, users of RTiC-Lab can download the source code, use it, enhance it, and share it with their colleagues. The second feature is that RTiC-Lab is designed to be distributed over a common network of personal computers. That is, RTiC-Lab can be used over a common 10/100 Mb ethernet network.

The general scheme used in the design of RTiC-Lab is shown is Figure 1. A devoted display or host computer (DHC) is networked via 10 or 100 Mb/s TCP/IP network to a set of devoted controls computers (DCCs).

The controls engineer sits at the DHC (which may or may not be at the same room or even building as the DCCs) and coordinates, codes, and synchronizes all DCCs from the DHC. Run time parameters, such as sampling rate, startup delay, and networking parameters can be set for each of the DCCs from the DHC.

Each of the DCCs is a stripped down computer system having no keyboard, mouse, video card, or monitor. These only have both the necessary I/O cards which are used to interface to the plant hardware and the necessary ethernet card to communicate with the DHC.

An AMB example of RTiC-Lab is shown in Figure 2. A single DHC interfaces with three DCCs which in turn interface to the AMB rotor system. The first DCC handles all radial control of the AMB, while a second (and slower) DCC controls the thrust direction of the AMB, and a third DCC is used to add either some excitation or synchronous forces to cancel out rotor imbalances at the midspan. Both controller parameters can be updated through the graphical user interface, and all data is plotted in soft real time at the DHC.

In the event that the controlled plant is both computationally simple and safe enough to be handled exclusively in a single computer, then RTiC-Lab will collapse into one single computer to control the entire plant. Stated differently, the same computer both implements the controller in hard real time and saves, plots, and updates parameters in soft real time.

In accordance with the RT-Linux paradigm [3], RTiC-Lab separates the AMB controller into the hard real time or "embedded" part and the soft real time or "reactive" part. The embedded part of the controller (resident exclusively in the DCCs) includes all tasks having hard timing constraints: 1) the AMB suspension controller(s) (both periodic and event driven), 2) a software watchdog, and 3) a set of interrupt service routines that are used for communication with the reactive task. The reactive task (resident in both DHC and DCCs) is a multi-threaded, user-space application which runs within the Linux kernel. In a standalone system, the reactive task would perform the following functions: 1) communicate with the embedded tasks via RT-FIFOs, 2) display a graphical user interface for the user, 3) perform error checking of the user's controller code, 4) send parameter updates to the embedded tasks as requested by user, and 5) plot data to either screen, save to a file, or print to `stdout`. Alternatively, in a multi-node environment, the DCCs' reactive system is charged with communicating with both the local embedded tasks via RT-FIFOs and with the remote DHC through the LAN. It would also be used to trap some vital errors from the local real time tasks, such as missed deadlines. The DHC – which does not necessarily have

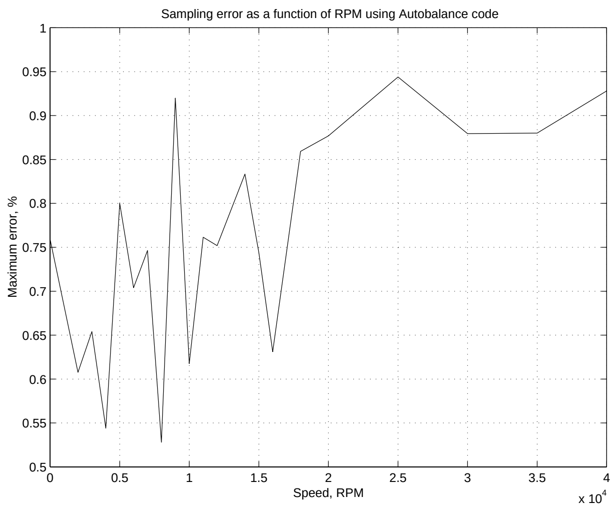


Figure 3: Maximum sample period error for the suspension controller as a function of spin speed

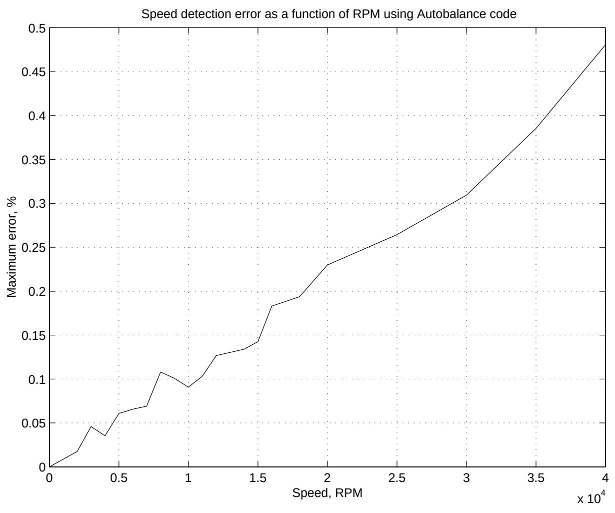


Figure 4: Maximum RPM error of the sampled speed as a function of spin speed

real time support – would then receive the incoming data through the network and would perform all of the necessary graphical duties as described above. In addition, the DHC would be used to coordinate all networked DCCs.

4 Implementation Results

Of most importance to the implementation of an AMB suspension controller is the predictability of RTiC-Lab. Figure 3 shows the sampling error obtained on a Pentium III 600 MHz computer running at 8 kHz the following algorithms: 1) a full μ-synthesis, 44th order, four input, four output, state space, radial magnetic bearing controller, 2) a strictly proper PID axial magnetic bearing controller, 3) a bearing autobalance algorithm, and 4) a parallel port based key phasor monitoring task. Note that items 1, 2, and 3, above, all run in one task, while item 4 runs as a separate task. As can be seen from the plot, the suspension controller sampling error is bounded to well below 1% even at simulated spin speeds approaching $40,000$ RPM.

The next point of interest is determining the maximum error of the sampled spin speed. Figure 4 shows the error of the sampled simulated spin speed for the same controller as above. As can be seen, the maximum error is below 0.5% at simulated speeds approaching $40,000$ RPM.

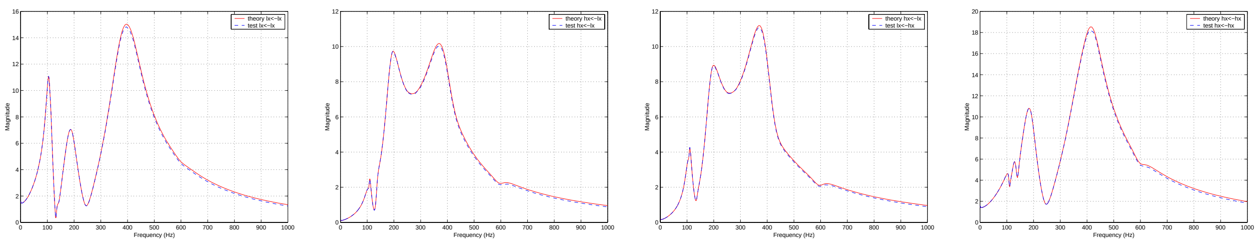
Last, and no less importantly, RTiC-Lab must satisfy the logical correctness of the aforementioned controller. Figures 5 show the theoretical and the measured input/output characteristics of the *mu*-synthesis controller of the low x to low x, low x to high x, high x to low x, and high x to high x input to output ports of the controller, respectively. The measured response denotes the results obtained from a sine sweep as measured at the I/O ports of the computer by use of a Stanford Research Systems two channel dynamic signal analyzer (Model SR785). As can be seen from the plots, both the theoretical and measured values match perfectly.

5 Conclusion

Basic understanding of RTS scheduling results is necessary towards efficient and successful implementation of *predictable* real time controllers for AMB systems. Namely, for periodic tasks, such as fixed rate controllers, it has been shown that priority assignment for a FPS algorithm via the Liu and Layland RMA will lead to predictable control systems for AMBs. Most importantly, via this priority assignment policy, it is possible to implement multiple tasks that are used not only for the actual suspension control but also for real time monitoring, data logging, data display, network communications, and controller parameter updates.

In order to simplify controller implementation by use of these scheduling techniques, the Real Time Controls Laboratory, or RTiC-Lab, was developed at UVA/ROMAC which aids in the controller implementation process. It uses both Linux and Real Time Linux as the implementation platform. And, consistent with the underlying operating system, RTiC-Lab is Open Source.

AMB controller developers who are interested in using this software are encouraged to download the software from `http://www.people.virginia.edu/~efh4v`. They are further encouraged to contribute



(a) Low X to Low X (b) Low X to High X (c) High X to Low X (d) High X to High X

Figure 5: Computer's I/O characteristics. Theoretical and measured amplitudes versus frequency

to its development by sending modifications, drivers, modules, and enhancements to `efhilton@alum.mit.edu`.

References

[1] P. E. Allaire, H. C. Kim, E. H. Maslen, G. B. Bearnson, and D. B. Olsen. Design of a magnetic bearing supported prototype centrifugal artificial heart pump. In *Tribology Transactions*, volume 39-3, pages 663–669, July 1996.

[2] M. Baloh, P. Allaire, E. Hilton, N. Wei, E. Maslen, D. Baun, and R. Flack. Magnetic bearing system for continous flow ventricular assist device. *J. of the ASAIO*, 45(5), 1999.

[3] M. Barbanov and V. Yodaiken. Introducing Real-Time Linux. *Linux Journal*, 34:19–23, February 1997.

[4] R. Bartlett, J. Coyner, S. Djouadi, P. Allaire, E. Hilton, J. Luo, P. Tsiotras, F. Maher, and R. Strunce. A simulation of spacecraft energy momentum wheels using advanced magntetic bearing controllers. In *1999 Invitational NASA/Air Force Flywheel Workshop*, NASA Glenn Research Center, 1999. Invited Paper.

[5] E.G. Coffman and P.J Denning. *Operating Systems Theory*. Prentice-Hall, 1973.

[6] R.W Conway, W.L. Maxwell, and Miller L.W. *Theory of Scheduling*. Addison-Wesley, 1967.

[7] R. Glenn. What really happened in mars? A letter and summary by Dr. Glenn Reeves, Mars Pathfinder Flight Software Engineer to Real Time Community, December 1997.

[8] E. Hilton, P. Allaire, M. Baloh, N. Wei, G. Bearnson, D. Olsen, and P. Khanwilkar. Test controller design, implementation, and performance for a magnetic suspension continous flow ventricular assist device. *Artificial Organs*, 23(8):785–791, 1999.

[9] E. Hilton, M. Humphrey, J. Stankovic, and P.. Allaire. Real time control of magnetic bearing suspension systems with flexible rotors. In *5th Intl. Symp. on Magnetic Suspension Technology*, Santa Barbara, CA, 1999.

[10] P.A. Laplante. *Real-Time Systems Design and Analysis: An Engineer's Handbook*. IEEE Press, IEEE Computer Society, New York, second edition, 1997.

[11] C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *JACM*, 20(1):46–61, 1973.

[12] N. Nissanke. *Realtime Systems*. Prentice Hall, London, 1997.

[13] U. Schönhoff, J. Luo, G. Li, E. Hilton, R. Nordmann, and P. Allaire. Implementation results of μ-synthesis control for an energy storage flywheel test rig. In *7th International Symposium on Magnetic Bearings*, August 23-25 2000.

[14] L. Sha, R. Rajkumar, and J. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transactions on Computers*, 39(9), September 1990.

[15] J. Stankovic. Misconceptions about real-time computing. *IEEE Computer*, 21(10), October 1988.

[16] J. Stankovic and G. Buttazzo. Implications of classical scheduling results for real-time systems. *IEEE Computer*, 28(6), June 1995.

[17] J. Stankovic and K. Ramamritham. What is predictability for real-time systems? *J. Real-Time Systems*, 2:247–254, December 1990.