WestVirginiaUniversity
THE RESEARCH REPOSITORY @ WVU

Graduate Theses, Dissertations, and Problem Reports

2005

Software tools for real-time simulation and control

Raghu Sankarayogi
West Virginia University

Follow this and additional works at: https://researchrepository.wvu.edu/etd

Recommended Citation
Sankarayogi, Raghu, "Software tools for real-time simulation and control" (2005). *Graduate Theses, Dissertations, and Problem Reports*. 1682.
https://researchrepository.wvu.edu/etd/1682

This Thesis is protected by copyright and/or related rights. It has been brought to you by the The Research Repository @ WVU with permission from the rights-holder(s). You are free to use this Thesis in any way that is permitted by the copyright and related rights legislation that applies to your use. For other uses you must obtain permission from the rights-holder(s) directly, unless additional rights are indicated by a Creative Commons license in the record and/ or on the work itself. This Thesis has been accepted for inclusion in WVU Graduate Theses, Dissertations, and Problem Reports collection by an authorized administrator of The Research Repository @ WVU. For more information, please contact researchrepository@mail.wvu.edu.

Software Tools for Real-Time Simulation and Control

by

Raghu Sankarayogi

Thesis submitted to the
College of Engineering and Mineral Resources
at West Virginia University
in partial fulfillment of the requirements
for the degree of

Master of Science
in
Electrical Engineering

Research Assistant Professor Karl Schoder, Ph.D.
Professor Powsiri Klinkhachorn, Ph.D.
Professor Ali Feliachi, Ph.D., Chair

Lane Department of Computer Science and Electrical Engineering

Morgantown, West Virginia
2005

Keywords: Real-Time Simulation, Hardware-In-the-Loop, Real-Time Linux, Matlab/Simulink

Copyright 2005 Raghu Sankarayogi

Abstract

Software Tools for Real-Time Simulation and Control

by

Raghu Sankarayogi
Master of Science in Electrical Engineering

West Virginia University

Professor Ali Feliachi, Ph.D., Chair

The objective of this thesis is to design and simulate a multi-agent based energy management system for a shipboard power system in hard real-time environment. The automatic reconfiguration of shipboard power systems is essential to improve survivability. Multi-agent technology is used in designing the reconfigurable energy management system using a self-stabilizing maximum flow algorithm. The agent based energy management system is designed in a Matlab/Simulink environment. Reconfiguration is performed for several situations including start-up, loss of an agent, limited available power, and distribution to priority ranked loads. The number of steps taken to reach the global solution and the time taken are very promising. With the growing importance of timing accuracy in simulating control systems during design and development, there is an increased need for these simulations to run in a real-time environment. This research further focuses on software tools that support hard real-time environment to run real-time simulations. A detailed survey has been conducted on freely available real-time operating systems and other software tools to setup a desktop PC supporting real-time environment. Matlab/Simulink/RTW-RTAI was selected as real-time computer aided control design software for demonstrating real-time simulation of agent based energy management system. The timing accuracy of these simulations has been verified successfully.

Acknowledgments

This research was sponsored in part by a grant from US DEPSCoR and ONR (DOD/ONR N000 14-031-0660).

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview and Motivation

Desktop PC based simulation, which includes designing a mathematical model for a physical system and executing it on a digital computer, has been the primary method for analyzing and studying the behavior of dynamic systems. Software tools such as computer aided control system design software (CACSDS) and programming tools are essential in the field of control systems design, modeling, and simulation. Simulating events in a time frame in which they would naturally occur is known as real-time simulation. The importance of timing accuracy of these simulations strongly depends on the application but may be crucial in ensuring intended system performance.

The main objective of this research is to design an agent based reconfigurable energy management system for shipboard power systems [33]. Real-time simulations should run both in software and Hardware-In-the-Loop configuration by building reduced scale hardware prototype of the shipboard power system. Software tools supporting real-time environments are required to achieve the above mentioned objective and to analyze its real world performance.

Matlab/Simulink[1] is one such tool that provides mathematical programming and graphical modeling platform for designing and simulating systems. Real-Time Workshop[2], a MATLAB/Simulink add-on for automatic code generation, can be used for general Microsoft Windows [79] based real-time applications with the help of Real-Time Windows Target [48]. This CACSDS provides its own real-time kernel, which runs in Windows' "Ring 0" as high-priority task [49], to ensure the best possible performance. Nevertheless, to achieve true real-time with guaranteed timing and low latencies with support for a more general

[1]MATLAB/Simulink is a registered trademark by The MathWorks, Inc. [47]
[2]Real-Time Workshop is a registered trademark by The MathWorks, Inc. [50]

environment, the appropriate step is to switch to a real-time operating systems (RTOS).

Hardware-In-the-Loop simulation requires the CACSDS to interface external hardware. DC-DC buck converter [52], which is an integral part of the shipboard power system, is selected to test the HIL capability of the real-time CACSDS.

To implement agent based energy management system in hardware, a communication network such as Controller Area Network (CAN) [14] is required for communication among agents. Hence, the real-time CACSDS should support the interface with this communication infrastructure.

1.2 Problem

The gain in popularity of electric commercial ships resulted in the development of all electrical Naval warships. The demand for reducing manning and costs while improving survivability lead to gradual replacement of mechanical-hydraulic systems and combustion engines by electric solutions.

The control system for the shipboard power system should ensure the reliable operation in both normal and emergency situations. The power systems of warships face extreme situations and several incidents [82] have shown the necessity for dependable automation strategies for reconfiguration while coping with emergency situations. The key problem faced by the shipboard power systems in an emergency situation is supplying power to at least the loads of more significance. Hence, the desired energy management system should address this problem and ensure the power flow to the loads according to their priorities.

Though the capability of the desired energy management system can be verified by running the simulations using general purpose tools, there is a need for understanding timing issues of these simulations. To analyze real-world performance of the system, real-time simulation is essential.

1.3 Objective and Approach

The objective of this thesis is to design a multi-agent based energy management system with automatic reconfiguration and demonstrate real-time simulation of the system with the available software tools.

The above mentioned problem, which can also be termed as power flow problem, was earlier dealt with techniques like Newton-Raphson Algorithm and Linear Programming. These approaches are discussed in detail in [11] and [12].

A different approach in solving the power flow problem for a decentralized energy management system using spatially distributed agents is discussed in [33]. In this thesis, a similar approach is followed in designing the energy management system but with modifications to solve the problem in a real-time environment.

1.4 Thesis Outline

In Chapter 2 a detailed literature review is given on real-time operating systems, currently available Linux variants with real-time support such as RTLinux and RTAI, programming tools that support real-time programming such as Ada 95 and Real-Time Java, CACSDS used for modeling and simulation such as MATLAB/Simulink and Scilab/Scicos, open source projects supplying free device drivers for data acquisition cards such as Comedi, and communication networks such as the Controller Area Network (CAN).

In Chapter 3 interface support of CACSDS integrated with RTOS such as MATLAB/Simulink/RTAI-LAB and Scilab/Scicos/RTAICodeGen for communication devices (CAN) is explored.

In Chapter 4 a real-time experiment in form of a buck converter control using Scilab/Scicos/R-TAICodeGen in real-time is presented.

In Chapter 5 real-time simulation of multi-agent based energy management system for shipboard power systems is demonstrated as a real-time application.

In Chapter 6 conclusions are given with recommendations for future work.

Chapter 2

Literature Review

2.1 Software Platforms

2.1.1 Introduction to Real-Time

In general, the term "real-time" is used in reference to a system where timing needs to be predictable. In the context of computing technology, a process is said to be running in real-time if it follows certain timing constraints and completes the process within a specified time without failure. A real-time system is comprised of processes that are associated with timing these constraints. A deadline is a good example for a timing constraint real-time systems are expected to meet. Basically, real-time systems are classified into two categories: hard real-time and soft real-time [25]. The main criterion behind this classification is the degree of tolerance of missing those deadlines. In a hard real-time system, deadlines cannot be missed. Whereas in the soft real-time systems, there is a certain degree of tolerance for missing deadlines. The importance of the deadlines depends on the application of interest. For example [], if a video player is considered, missing one or two frames once in a while is acceptable but there is a certain limit on the number of missing frames and the frequency of missing. This type of system is called a soft real-time system and deadlines can be missed but not too often. Hard real-time systems strictly follow the assigned deadlines and the success depends on the task execution within deadlines. Control systems used in a nuclear reactor or in a plane are good examples of hard real-time systems where missing a deadline could prove fatal.

2.1.2 Significance and Requirements

The above mentioned example explains the significance of deadlines in hard real-time systems. The rapid increase in the cost of failure, re-engineering, and accuracy of hard real-time systems in various fields including electrical power engineering shows the importance of hard real-time performance. In order to achieve this kind of hard real-time performance, dedicated central processing units such as digital signal processors are used to meet the deadlines within the scheduled time without failure.

A general purpose central processing unit (GPCPU), which is found in daily used desktop personal computers, is in general not suitable for hard real-time applications as several uncertainty factors such as virtual memory and memory management hamper deterministic latencies [45]. But the high end, well programmed GPCPU with high speed computing features (number of floating point operations per second in the millions) such as single instruction multiple data and symmetric multi processors can implement fairly complex control systems [23]. RTOS should meet the basic requirements including predictability, pre-emptability, support for multi-threaded scheduling, concept of priorities, and resource sharing mechanisms avoiding priority inversion. In selecting RTOS for real-time applications, latency values play an important role. Latency is defined as the response time of a system to any external event such as hardware interrupts or any application within the system [2]. So, the lower the latency the quicker the response of the system to events. This feature is important in real-time systems. A real-time operating system, which posses all these characteristics can be used to achieve hard real-time performance with latencies of only a few micro seconds [23]. Today, there are many RTOS available on various platforms including Microsoft Windows, Linux, Solaris, and SunOS. The RTOS based on Linux are gaining importance due to open source code and licensing (GNU Public License) [18]. Researchers, developers, and programmers prefer Linux as the open platform gives flexibility in evaluating and developing applications.

2.1.3 Why Linux based RTOS?

Before further exploring freely available and commercial implementations, a discussion on the real-time capabilities of the standard Linux kernel is on order. Linux supports supports multi-threading and several Portable Operating System Interface (POSIX, [36]) compliant libraries are available, e.g., the FSU Pthread library [4]. POSIX defines how the operating system and programs interface with each other. The POSIX 1003.13 standard defines a “Multi-User Real-Time System Profile” allowing real-time processes to be executed in a predictable order by using a special scheduler and by locking the process itself into the memory to avoid paging onto the harddisk. Here, the term “predictable” means the timing behavior is always within an acceptable range. The standard Linux kernel uses system calls to provide the improved deterministic performance. The resulting performance can be described as soft real-time but not hard real-time as there is no possibility of preempting kernel tasks [45]. Also, the nature of Linux's scheduling algorithm is more suitable for a general purpose operating system than to a real-time purpose and intended to maximize the average throughput. The worst-case deviation from deadlines may prove to be very high when the system is not idle.

2.1.4 Available RTOS based on Linux

Several implementations of both commercial and open-source distributions of Linux based real-time operating systems are available. A list of commercial variants includes RTLinuxPro by FSMLabs [32], uLinux by Lineo Solutions [41], LynxOS by LynuxWorks [44], QNX by QNX Software Systems [64], and VXWorks by Wind River [84]. A more detailed list can be found in [46]. At the same time many open source versions of real-time implementations of Linux are available [43]. Some of these open source versions are discussed below:

1. ART Linux [42]: A real-time extension to Linux that was developed by Japanese engineer Youichi Ishiwata at ETL (Electrotechnical Laboratory at Japan). Though it is inspired by RTLinux (RTLinux will be discussed in detail later), it offers certain advantages such as compatibility of device drivers at source level and binary level compatibility of the user programs. But, ART Linux has been tested successfully only on the kernel versions 2.2.xx, which are outdated. Also, ART Linux has not been formally submitted to GPL and, therefore, there is some uncertainty concerning license issues.

2. KURT-Linux [60]: KURT-Linux stands for Kansas University Real-Time Linux. The list of features includes user defined schedulability and capability of executing the real-time processes independent of non real-time processes. The main drawback is the support of only x86 platforms and older Linux versions such as Kernel 2.4.xx.

3. QLinux [71]: QLinux is an implementation by the University of Massachusetts and IBM Research and provides Quality of Service (QoS) that can only guarantee soft real-time performance. The latest available release is based on Linux kernel 2.2.0.

4. RTLinux [29]: RTLinux is a small hard real-time operating system that runs Linux as a thread in its idle moments. It was a result of a small research project at New Mexico Institute of Mining and Technology led by Victor Yodaiken. RTLinux decouples the real-time and non-real-time parts of the operating system and strictly separates corresponding applications to improve reliability, simplicity, and speed. The latest version of RTLinux released by FSMLabs supports kernel 2.6.

5. RTAI [20]: The Real-Time Application Interface was developed by members of the Department of Aerospace and Engineering, Politecnico di Milano, Italy, and provides guaranteed hard real-time scheduling and supports uni-processors and symmetric multi-processors. The working principle behind RTAI is the Hardware Abstraction Layer (HAL). This layer is added to the standard Linux kernel and is comprised of pointers to the interrupt vectors and the interrupt enable/disable functions. RTAI has been successfully implemented on kernel 2.6.5.

When compared to other versions, RTLinux and RTAI stand out as the more versatile, compatible, and reliable open source implementations of real-time Linux. A detailed description of RTLinux and RTAI and a comparison is given below.

2.1.5 RTLinux

RTLinux (Real-Time Linux) [35, 29] is a small hard real-time kernel which assigns the lowest priority to the standard Linux kernel. To achieve this, few lines of code patch the standard Linux kernel and recompilation yields RTLinux. In the process of patching, the macros clearing and setting interrupts, which are key in enabling and disabling interrupts, are replaced [65]. Thus, RTLinux takes control over the interrupts. The actual working of RTLinux can be best understood by visualizing the RTLinux kernel sitting in between the standard Linux kernel and the system hardware. This is actually done by loading core RTLinux modules. RTLinux also replaces the standard Linux scheduler. The real-time kernel of RTLinux is seen as actual hardware by the standard Linux kernel and thus intercepts all hardware interrupts for achieving real-time performance.

In general, any attempt to modify an operating system in order to support both real-time and non real-time aspects would complicate the operating system further and may result in a large, unreliable, and inefficient operating system. To avoid this, RTLinux decouples the real-time and non real-time parts of the operating system. Thus, minimum latency for response to an interrupt can be achieved irrespective of the standard Linux kernel activities. RTLinux comes in two versions: RTLinuxFree [31] and RTLinuxPro [32]. The first one is an open source version and is licensed under GPL [30]. This community supported free version of RTLinux has all basic features but some problems with licensing of modules. RTLinuxPro, which comes with a commercial license, has professional support, rapid prototyping graphics user interface, and a debugging environment pre-configured kernel. The user space real-time capability is much improved when compared to its open source version. Further, RTLinuxPro supports various additional features such as real-time networking, protected memory real-time, and controls kit software by FSMLabs [28].

2.1.6 RTAI

Another open-source implementation of real-time Linux is the Real Time Application Interface (RTAI, [20]). The motivation for the development of RTAI came with the recognition of RTLinux's deficiencies. The RTAI project began at the Dipartimento di Ingegneria Aerospaziale - Politecnico di Milano in 1996/97. Initially, RTAI developers faced some problems with then available kernel version 2.0.xx. The new idea was to have a Real-Time Linux implementation by having Real-Time Hardware Abstraction Layer (RTHAL) on which RTAI can be developed. The release of kernel 2.2.xx, which had a clear interface to the hardware, solved most of the problems faced by RTAI developers. It resulted in further developments to support systems with both single and multi processors. RTAI is a kernel patch applied to the standard Linux kernel.

RTAI architecture is similar to its counterpart RTLinux in running the standard Linux kernel as a lowest priority process. RTAI also makes use of the loadable modules feature and is module oriented. The hardware abstraction layer supports various core modules and helps in achieving desired on-demand real-time capability. RTHAL modifies all the pointers of standard Linux kernel such that RTAI can take their place whenever hard realtime is required. By doing this, RTAI ensures that all the hard real-time activities have full authority over hardware and interrupts triggered by the Linux kernel and applications. But, at the same time RTHAL makes sure that all the Linux interrupts, which are marked as pending during hard real-time activities, are dispatched afterwards without failure [24].

RTAI's architecture is shown in Fig. 2.1. The RTAI block consists of interrupt dispatcher and scheduler. When loaded, Linux scheduler is replaced by the RTAI scheduler and thus running Linux as a low priority task. The interrupt dispatcher takes control of software interrupts allowing tasks to run in real-time.

An overview of RTAI with RTHAL is shown in Fig. 2.2. RTHAL is a structure of function pointers, which are related to interrupts. When RTAI modules are loaded, the function pointers in the RTHAL structure are changed to point to the equivalent functions in the real-time kernel. Thus, RTAI intercepts the interrupts. The Linux applications running in user space communicate with real-time applications, which run in kernel space, by using first in–first out (FIFO) pipes for data buffering between the two processes.

There are no major differences between RTLinux and RTAI as they both use the concept of RTHAL. Table 2.1 gives a summary of RTLinuxFree and RTAI features [66].

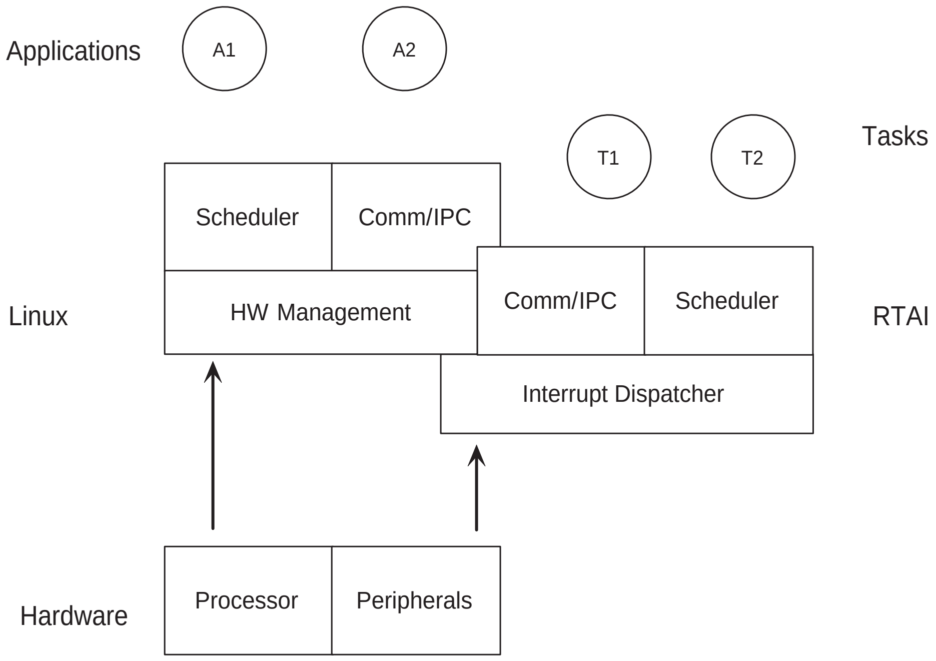


Figure 2.1: RTAI architecture [53]

Table 2.1: Comparison of RTLinuxFree and RTAI

RTOS	**RTLinuxFree**	**RTAI**
License	General Public License and Commercial	Lesser General Public License
Supported Architecture	i386, PPC, ARM.	i386, MIPS, PPC, ARM
Latency	Less than 10 micro seconds	Less than 10 micro seconds
Stable Kernel	Kernel 2.6	Kernel 2.6.11
API	POSIX	RTAI-Custom API
Memory	Shared	Dynamic and Shared

2.1.7 Ada 95

Ada is a general purpose object-oriented programming language designed to meet the needs of both large and small scale programming [10]. It is used in applications ranging from real-time embedded to

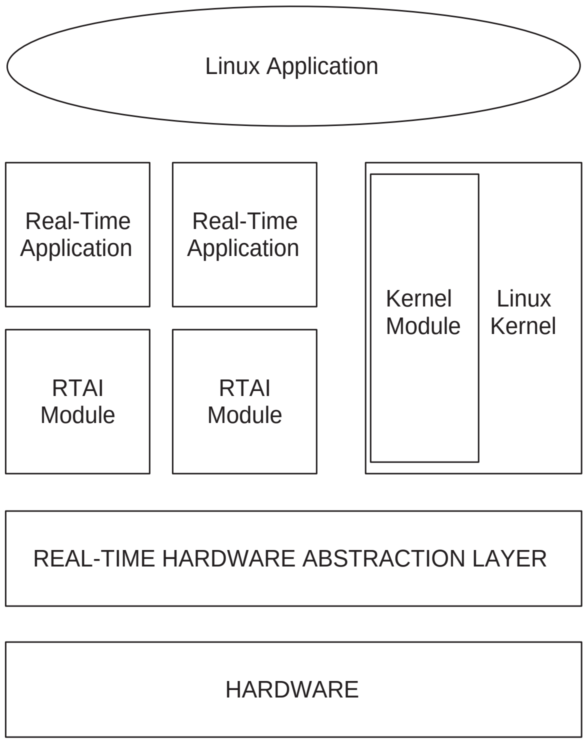


Figure 2.2: RTAI overview [69]

large scale systems. Ada's strengths lie in the support of concurrent programming, strong syntax checking, high portability, high datatype compatibility, and flexible data structures.

Ada was developed during the 1970s [75] in an attempt to solve the "software crisis" faced by US Department of Defense for military and mission critical systems. It was named after an English mathematician who is considered as the world's first programmer, Ada Lovelace. It achieved its first ISO standardization in 1987. Department of Defense sponsored the research for further refinement and improvement in performance of Ada. Later, Ada achieved both ISO and ANSI standardization in 1995, which is also referred to as Ada 95 (see for example [1]).

Ada supports structured control statements, procedures and functions, and standard libraries. With the help of packages and subprograms, Ada also supports programming in the large. Ada's support for concurrent programming is considered as one of the major strengths and root cause for development of Ada [10]. An Ada task is a program that performs a sequence of actions. In a single task program, only one task will run the entire program by running the sequence of actions one by one. In a multi-task program, several tasks comprise the overall program and perform various sets of sequences of actions and these tasks could run at the same time. In the case of a single processor system, a multi-task program is extended by running small parts of a task at a time and get back to an unfinished task where it was left previously. This approach is called interleaved concurrency [19]. Whereas in the case of a multi-processor system, tasks can be run by several processors and this phenomenon is called overlapped concurrency.

Synchronization among tasks is required for efficient concurrency. Ada 95 provides three mechanisms that help in synchronization of and communication between tasks: Shared Variables, Protected Objects, and Rendezvous [7].

Another major area of Ada applications is real-time programming. It involves producing real-time applications in the form of loadable kernel modules. By loading these modules the application directly runs in kernel space enjoying high priority and full pre-emption possibilities over interrupts of all levels. As these applications run in kernel space, one has to make sure that the real-time program is flawless otherwise the application can take down the entire system along with it. In most of the cases, C is used as a primary language for developing real-time applications. But using C increases the vulnerability of the produced real-time application to crash due to the use of pointers, which can result in memory reference errors. Ada, which has stronger type and syntax checking, could be a better option for developing real-time applications.

With the help of the Real-Time Annex [78], Ada tasking supports the concept of periodic tasks, event-driven tasks, and unbounded priority inversions. The support for system level programming makes it a much better tool for real-time programming as it can access the system hardware for developing time critical code sequences and interrupt handlers.

A number of Ada compilers are available today. GNAT [27] compiler is considered as a standard general purpose compiler for Ada as it is supported on various platforms. Though Ada itself can be used for real-time programming applications, there is a definite need to have Ada integrated with a RTOS so that it can deliver hard real-time performance. GNAT compiler for Ada has been modified and integrated with RTLinuxFree and this modified version of the compiler is called RTLGNAT [67]. Using RTLGNAT, programs written in Ada are compiled to produce RTLinux loadable kernel modules. Thus, the programs can be run in kernel space with higher priority than any other running Linux application. The latest RTLGNAT-1.0 is supported on kernel 2.4.22 with RTLinuxFree-3.2-pre3.

Ada is also used in systems level application programming [21]. These applications require access to low level device drivers to control hardware. In general, device drivers for any hardware are written in C/C++ due to its flexibility and popularity among programmers. Thus, an interface between Ada and C/C++ to access low level device drivers is required. Ada uses Pragmas for interfacing with C/C++. A

Pragma is a compiler directive used by Ada to follow the appropriate conventions for a function's data exchange. Pragma Import, Pragma Export, and Pragma Convention are the main Pragmas used by Ada. Pragma Import is used to import functions and data types defined in foreign languages and is the key in interfacing Ada with C/C++ based device drivers. Pragma Export is used to export the subprograms from Ada to C. Pragma Convention denotes that Ada object should use the convention of the foreign language to be interfaced. The Ada program with all the required Pragmas is compiled and linked with object files (file to be interfaced) with the help of GNAT commands to produce an Ada executable with embedded C program calls. The availability of the C interface resulted in usage of Ada in many embedded applications [7].

When it comes to hard real-time, as discussed earlier, the only compiler available for this purpose is RTLGNAT. The latest available version supports older versions of Linux such as kernel 2.4.xx. The version of RTLinuxFree on which RTLGNAT is built has some license issues with FIFO modules. Hence, some changes had to be made in the source files of RTLGNAT and additionally required files to get it running on the latest stable version of RTLinuxfree (3.2-pre3). Further drawbacks of Ada concern gradually decreasing community support and slowing development of the language itself.

2.1.8 Real-Time Java

Java is an object-oriented programming language and widely used in web applications [40]. Java has a syntax similar to that of C++ with the exception of not using pointers. Java technology was introduced in 1991 for developing software for a hand-held home entertainment device. Later with the rapid growth in popularity of the World-Wide-Web, Sun Systems further developed Java and strengthened its position as general purpose language, and many standard Java libraries are available today for use by programmers. Programs written in Java are not compiled into machine code as in the case of other programming languages. Instead, the program is compiled into a byte code that can be interpreted by the Java Virtual Machine (JVM). The JVM has been developed for all the architectures available today and hence Java is termed as processor independent language. JVM takes care of memory management problems such as memory leaks with the help of garbage collector that runs as a process and automatically frees the memory whenever the created object reaches its lifetime.

Due to Java's advantages and features, developers were interested in applying Java technology in real-time applications. Nevertheless, when it comes to real-time, Java's advantages turn into limitations. First, Java Virtual Machine specifications leave a lot room for ambiguity in timing and thread priorities [40]. The garbage collector, which is supposed to be Java's strength, creates problems for real-time programming as the process of garbage collection, once started, cannot be interrupted by any process. Other shortcomings of standard Java concern priority inversion, asynchronous event handling, absence of a priority based scheduler, lack of support for interrupt handlers, and physical memory access. Research concerning elimination of these issues is still in its early stages [57]. The National Institute for Standards and Technology (NIST) developed a set of rules and standard requirements: "Requirements for Real-Time Extensions for the Java Platform [59]" for a real-time Java, which state that the specification should not restrict the use of Java to a particular environment, should not lose the compatibility with normal programs, should follow the rule "Write Once Run Anywhere," and be ready to implement advanced features in the future. One RT-Java specifications developed based on these recommendations is the Java Specification Request (JSR-1) by Sun and IBM[1]. Further development of JSR-1 resulted in the Real-Time Specification for Java, RTSJ [5, 68].

Based on the standardized rules for RT-Java, various versions of RT-Java for embedded applications were developed. JBED [54], PICO Java [77], and PERC [56] are some of the implementations that target small embedded applications. Any implementation of RT-Java complying with RTSJ can be used for real-time and embedded applications. A pre-emptive scheduler with at least 28 priorities is included to introduce the concept of priorities for threads. Priority inheritance protocol is used to avoid the problem of priority inversion. The usage of No Heap Real-Time (NHRT) threads, which cannot be interrupted by the garbage collector, fixes the problem of inevitable delays caused by garbage collection. Further, the concept of different memory areas is introduced in which the memory is divided into three areas: physical, immortal, and scoped memory. Objects created in these areas cannot be traditionally garbage collected.

Several implementations of RT-Java comply with RTSJ: KVM (Kilo Virtual Machine) [76], J2ME (Java Micro Edition) [70], and aJile (Single chip Java micro controller) [72] are some of the examples.

Complying totally with RTSJ allows implementations to achieve soft real-time performance but the current available technology in RT-Java cannot be used for hard real-time tasks in mission-critical applications. Nevertheless, Java experts ensure that it has got the capability of supporting hard real-time and distributed real-time systems – if not today, may be in the near future [40].

2.2 CACSDS

Computer Aided Control System Design Software (CACSDS) provides an environment for programming, modeling, and simulating control systems and allows analyze performance. A CACSDS provides

[1]See `https://rtsj.dev.java.net/` for more information

a powerful and flexible programming environment and graphical user interface (GUI) to help the user in building a system model.

Two of the CACSDS available today are Matlab/Simulink [47] and Scilab/Scicos [38]. Matlab/Simulink is a commercial product and Scilab/Scicos is open-source. Both of theses CACSDS support real-time simulations. Matlab/Simulink through its add-on software Real-Time Workshop and Scilab/Scicos through the open-source project RTAICodeGen as discussed in the following sections.

2.2.1 Matlab/Simulink/Real-Time Workshop

Matlab, which stands for Matrix Laboratory, is an easy-to-use and powerful tool for technical computing. The core Matlab provides the Matlab language, mathematical function library, development environment, graphics, and application program interface to programming languages such as C and Fortran.

Simulink is a software package integrated with Matlab and used for modeling, simulating, and analyzing continuous, discrete, and hybrid systems. Simulink comes with a GUI that allows build new models through a drag-and-drop feature of library blocks. Simulink also supports user defined blocks with its system-functions (S-functions, [51]) written in Matlab, C/C++, Fortran, or Ada.

The Real-Time Workshop [50] provides a C code generator environment for rapid prototyping and development. It generates source code from Simulink models to create real-time software applications, and is applicable for any model irrespective of its time domain (continuous or discrete).

For real-time control and Hardware-In-the-Loop simulations, Matlab/Simulink combined with Real-Time Windows Target [48] provide input-output blocks to connect external hardware. The generated code can be run in real-time using Simulink's external mode. With the help of Simulink GUI, a user can observe signals during simulations.

As introduced earlier, the combination of Matlab/Simulink/Real-Time Windows Target provides real-time extensions for Windows through a kernel that runs the compiled Simulink model in the operating system's "Ring 0" at highest priority [49] to achieve high performance. To take advantage of a general hard real-time environment, the CACSDS should be used in combination with a RTOS, where the developed application is run in kernel space with preemption capabilities and control over interrupts.

To integrate Matlab/Simulink with a RTOS such as RTAI, RTAI-LIB, which is part of RTAI [20, 8] and a library consisting of S-functions written in C, is copied under the Matlab root directory. This library is then compiled to generate building blocks based on RTAI. The created library provides Comedi interface blocks [17] for connecting external hardware and scope blocks for monitoring signals using the provided RTAI-LAB graphics user interface. A detailed installation guide is presented in Appendix A.4.

After installing RTAI-LIB, the system to be simulated in real-time can be modeled using blocks available from Simulink. The code is generated in a directory designated directory in Matlab's path and compiled and linked using RTAI makefiles to generate the executable file that runs in hard real-time [22]. Once the executable has been started, the RTAI-LAB GUI is used for monitoring the signals during the simulations.

2.2.2 Scilab/Scicos/RTAICodeGen

Scilab is a scientific software package developed by the "Institut National De Recherche En Informatique Et Et Automatique (INRIA)" at "Ecole Nationale Des Ponts Et Chaussees (ENPC)" since 1990. Though not as advanced as Matlab, Scilab can match Matlab in large number of aspects [38]. This free and open source software package comes with a "connected object simulator" known as Scicos, which is similar to Simulink and is used for modeling and simulating dynamic systems. Scilab/Scicos is supported on most platforms such as Windows and Linux. In this thesis, Linux version of Scilab/Scicos is used as CACSDS.

Unlike Matlab/Simulink, which can be extended by Real-Time Workshop and Real-Time Windows Target, INRIA does not provide such tool directly. Roberto Bucher, a researcher at the University of Applied Sciences at Southern Switzerland (SUPSI), has made a valuable contribution to RTAI by developing a code generator that provides a real-time toolbox for both Scicos and Simulink called RTAI-LIB [8]. This code generator integrates RTAI libraries with Scilab macros. RTAI-LIB has blocks that can be used for signal generation, signal recording and display, and real-time data acquisition [9]. Table 2.2 shows a list of available blocks and their purpose.

Once Scilab/Scicos is installed on a system, it can be integrated with RTAI and the required code generation files installed as Scilab macros. After this setup process, users get access to RTAI code generation option in the Scicos GUI. Once the model of the considered system has been built, the executable real-time module can be generated. This is done by running the RTAI code generator on the Scicos superblock that contains the dynamic system model. Execution of the generated file starts the real-time tasks. Afterwards, the RTAI-LAB is launched to connect to the tasks and display the signals through its digital instruments such as scopes and meters. An installation guide is included in Appendix A.2.

As previously mentioned, connecting hardware to a PC instead of its software model and performing the simulation is known as Hardware-In-the-Loop simulation. A simple HIL scenario consists of a PC (with CACSDS installed on a real-time platform) connected to the actual hardware through a data acquisition (DAQ) card. The DAQ card forms the means of communication between the control code generated

Table 2.2: List of available Scicos blocks from RTAI-LIB

Name of Block	Purpose
RTAI_Sinus	Sine wave generator
RTAI_Step	Step function
RTAI_Square	Square wave generator
RTAI_Extdata	Source block using input file
RTAI_ComediDATAIn	Comedi input block for reading analog input
RTAI_ComediDIOIn	Comedi input block for reading digital input
PCAN_In	PCAN input block for reading messages
RTAI Mbx_Rcv	Input block for interprocess communication using mailboxes
RTAI_Scope	Scope block for reading signals
RTAI_Meter	Digital representation of analog meter
RTAI_Led	Led display
RTAI_Fifo	Output block used for writing data to a fifo
RTAI_ComediDATAout	Comedi output block for sending analog output
RTAI_ComediDIOIn	Comedi output block for sending digital output
RTAI Mbx_Rcv	Output block for interprocess communication using mailboxes
PCAN_Out	PCAN output block for writing messages to CAN device

on the PC and the external hardware. It provides a two way path for input and output signals from the PC and the hardware. In order to have this DAQ process functioning, system has to recognize the DAQ card used. The device drivers provided by the manufacturers of the DAQ card are usually supporting Windows only. The Linux based device drivers can be obtained by using Comedi drivers [17].

Comedi is a project started by David Schleef and others and aims at developing open source device drivers, tools, and libraries for data acquisition. The device drivers developed support hardware from several manufacturers of data acquisition boards [16]. The Comedi team has developed two packages for the purpose of data acquisition: Comedi and Comedilib. Comedi is a collection of device drivers for the various brands of DAQ boards, and provides the drivers in form of loadable Linux kernel modules. Comedilib is a library that provides an interface to the Comedi drivers. This library has some useful tools for the user such as calibration and demo programs. Once the Comedi package is installed, the user can install the Comedilib package and test/calibrate the card using the calibration utility in order to verify whether the device driver for the corresponding hardware is functional. By using some of the demo programs that come with the Comedilib package, complete information of the corresponding DAQ device can be obtained.

Comedi and Comedilib should be installed after installing RTAI and SCILAB. Once Comedi and Comedilib are installed, RTAI has to be re-configured with the Comedi option enabled so that Comedi and Comedilib are integrated. The detailed procedure of installing Comedi and Comedilib is given in Appendix A.3.

2.3 Communication Networks

2.3.1 Controller Area Network

Controller Area Network (CAN, [14]) is a serial bus communication network. The increase in interest in distributed control systems in automobiles, cost of wiring, need of high availability, high speed, and reliable data communication resulted in development of CAN. Initially, CAN was used only in the automobile industry. Later on CAN found its applications in various process industries and robotics. This two-wire, half-duplex, high speed network system is known for its robustness, reliability, and connection of up to 2032 devices (theoretically) supporting communication rates of up to 1 Mb/s.

CAN was first developed by the engineers at BOSCH [73], a German based company in 1986. It began with the search for a network protocol that would meet the automotive industry's requirements. Uwe Kiencke started working on the development of a serial bus network in 1983. Later, he and other engineers at BOSCH invented the "Automotive Serial Controller Area Network," which was based on a non-destructive arbitration mechanism and would grant bus access to the messages according to their priorities.

The latest BOSCH CAN specification was published in 1991 and consists of two parts [6]: Standard CAN (Version 2.0 Part A) defines an 11-bit message identifier and Extended CAN (Version 2.0 Part B) defines a 29-bit identifier. Two different ISO standards for CAN exist [39], and the difference is in the physical layer: ISO 11898 is meant for high speed applications of up to 1Mb/s and ISO 11519 has an upper limit of 125kb/s.

A Finland based elevator company and some Swedish based textile industries were the first companies that used CAN bus for non-automotive purposes. Today, CANOpen [13] and DeviceNet [15] are standardized application layers extensively used in the process industry for machine controls. The latest development of CAN was the time triggered communication protocol (TTCAN) by CAN in Automation (CIA) [14].

In 2000 CAN was also introduced as a communication platform by the National Marine Electronics Association in the NMEA 2000 interface standard [55]. The standard is meant to fulfill the requirements of a serial data communications network to inter-connect marine electronic equipment on vessels.

CAN is a multi-master network based on Carrier Sense Multiple Access/Collision Detection + Arbitration through Message Priority (CSMA/CD + AMP). Messages sent do not contain any address and are not destined to any particular node. Instead, the message is sent to all the nodes and the corresponding node, which is supposed to receive the message, identifies the message with the help of the information it carries in the identifier bits. The identifier is sent at the beginning of every message and allows both defining different messages and different message priorities. As low bits are dominant, the lower the message identifier the higher the message's priority.

Figures 2.3 and 2.4 [58] show the CAN message structures for both protocol version 2.0A and 2.0B. CAN uses message frames for data transfer. There are two message frame formats: data frame and remote frame. Data frames are used to send information according to either CAN 2.0A (Standard messages with 11-bit identifier) and 2.0B (Extended messages with 29-bit identifier). In general, a standard message frame of a CAN system consists of various fields as follows:

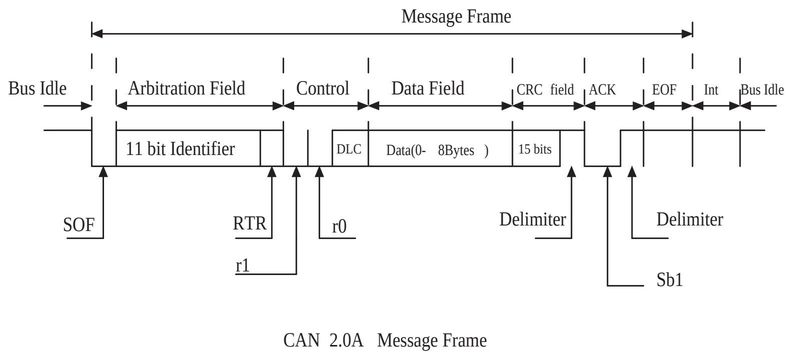


Figure 2.3: Message structure CAN 2.0A

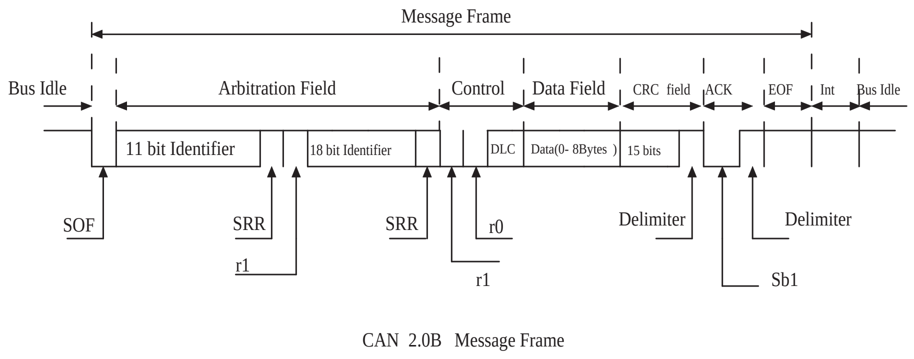


Figure 2.4: Message structure CAN 2.0B

- SOF: Start of Frame indicates the beginning of a frame.
- Arbitration Field: Contains a messages identifier and the Remote Transmission Request (RTR) bit which tells whether the message carries any data or itself is requesting some.
- Control: This field consists of a total of six bits with two reserved bits (r0 and r1) and 4 bits of data length code (DLC) which gives the number of bytes in the data field that follows.
- Data Field: Data field contains 0-8 bytes.
- CRC: Contains a fifteen bit cyclic redundancy check for validating the received message and a recessive delimiter bit.
- ACK: This field has 2-bits, first bit is the slot bit transmitted as recessive and the other one is recessive delimiter bit.
- EOF: End of frame consisting of seven recessive bits
- INT: Intermission has 3 reserved bits.

Extended message frame format has almost similar message structure standard format except of the extended format with 29-bit identifier and also has a Substitute Remote Request (SRR) in the arbitration field, which makes sure that standard message gets the higher priority when both 2.0A and 2.0B identifier messages are used at the same bus.

Remote frames are used to request data rather than sending. A message with the RTR bit set requests data according to its identifier.

CAN protocol supports a high number of distinguished messages per system. It is possible to have 2048 different messages in the case of an 11-bit message identifier, and 512 million in the case of a 29-bit message identifier. CAN system supports event oriented message transmission system due to its multi-master architecture.

CAN bus system is a highly reliable communication network due to its outstanding capabilities of error detection and fault confinement. Network-wide data consistency is achieved with the use of error frames which destroy the faulty message. Each CAN controller is provided with a counter which keeps track of the number of errors and when a certain limit is reached the erroneous node is automatically disconnected from the bus.

Errors occurring in a CAN bus are of two types: bit error and message error. A bit error occurs whenever a transmitting node inserts more than five consecutive low or high bits or if there is a different bit value returned when compared with the sent message. Message error could occur if there is any checksum error, acknowledge error, or inconsistency in the format of a message.

2.3.2 GPIB/IEEE 488

General Purpose Interface Bus (GPIB) [85] is one of the popular communication networks developed by Hewlett Packard in 1965 and, hence, it is also referred to as Hewlett Packard Interface Bus (HPIB). GPIB was mainly used for connecting different computers and programmable instruments within a short distance. In 1987, GPIB was defined in IEEE standard 488 (1-1987 Standard Digital Interface for Programmable Instrumentation, [37]), since then it is also referred to as IEEE 488. The latest standard for GPIB is IEEE 488.2.

To use GPIB, the GPIB adapter and the GPIB lead are essential. The bus system consists of 16 signal lines and eight ground lines. The 16 signal lines are further divided into eight data lines, three hand-shake lines, and five interface management lines. The GPIB standard allows up to 15 devices connected to one bus. The maximum speed for data transfer supported is 200 KB/s. With the eight data lines, GPIB can be classified as parallel networking bus. As the bus interface is function independent of instruments, almost any instrument can be interfaced with it.

The 15 devices that can be connected to a single bus are categorized into controller, talker, and listener. It is required that at an instant there is at least one controller and one talker or listener. Though there could be many controllers, at any single time only one controller is active. The active controller performs the bus control options for all the connected instruments thus acting like a master. Talker and listener transmit and receive the data according to the instructions from the controller.

The data lines are used for transfer of addresses, control information, and data. ASCII or binary are the general formats for data. The control of the data transfer is done by the hand-shake lines NRFD (Not Ready for Data), NDAC (No Data Accepted), and DAV (Data Valid). Interface Management Lines are used for managing the data transfer across the interface [81].

GPIB comes with some physical restrictions regarding the maximum distance between two devices on a bus (4 meters). This restriction adds the requirement of extenders and expanders. GPIB is relatively expensive and not meant for real-time communication.

2.3.3 Modbus

Modbus is a serial bus communication protocol developed by Modicon in 1978 [80] for exchanging information between products on the factory floor. It then became a standard for communication between programmable logic controllers. Modbus's physical layer is based on a pair of shielded and twisted wires.

RS-232C compatible serial interface is used by the Modicon controllers on a standard Modbus. The controllers, which are usually networked directly using modems, communicate using master-slave technique. The master device initiates all the queries and the slave devices respond to these queries and supply the requested data. In the case of Modbus, host processors and programming panels could be masters and programmable logic controllers act as slaves.

ASCII and RTU (Remote Terminal Unit) are the two used transmission modes. In ASCII, two 8-bit bytes are sent as 2 ASCII characters, whereas in RTU data is sent sent a two 4-bit hexadecimal characters. Of the two types, RTU is more efficient and has a higher throughput. The maximum distance supported on the network is 350 m.

Modbus protocol is simple in architecture. Enhancements for Modbus protocol are ModbusPlus and Modbus/TCP by supporting peer-to-peer communication through encapsulation of information on the bus into a networked structure. These enhancement protocols also support a maximum distance of 1500 m for a bus with the help of repeaters.

Modbus protocol is widely used in Programmable Logic Control process systems but the master-slave architecture does not allow it to be used for real-time purposes. For real-time data transfer, each and every device should be able to initiate queries.

2.3.4 Conclusions

Comparing the above discussed bus communication networks, Controller Area Network proves to be a desireable choice due to the following features:

- Support of high data transfer rate (1 Mb/s).
- Suitable architecture (multi-master) for real-time distributed applications.
- Reliable with strong error detection and fault confinement mechanisms.

Chapter 3

Application: CACSDS/Controller Area Network Integration

3.1 Overview

The Controller Area Network as discussed earlier is a popular choice in real-time control and communication applications. Besides advantages such as reliability and low cost, the support of high baudrate of up to 1 Mb/s is one of the main reasons for its increasing significance in the real-time world.

A CAN device interface when integrated with RTOS enables the user to transmit and receive real-time CAN messages. The real-time CAN messaging is useful especially in communication aspect of mission critical applications. Further, if a CAN device interface is integrated with RT-CACSDS, it will help the user to conduct real-time CAN messaging along with real-time Hardware-In-the-Loop simulations.

3.2 Hardware

Out of various CAN-PC interface devices available today, USB-CAN device from Peak-Systems [62] was chosen as it is portable and, more importantly, Linux based device drivers were freely available for integration with RTOS and CACSDS such as RTAI/RT-Linux and Scicos/Simulink.

The device is based on a Phillips SJA1000 CAN Controller with 16MHz clock frequency, 82C251 CAN transceiver, and has a 9-PIN connector. Figure 3.1 shows a picture of the PCAN-USB device and connector.

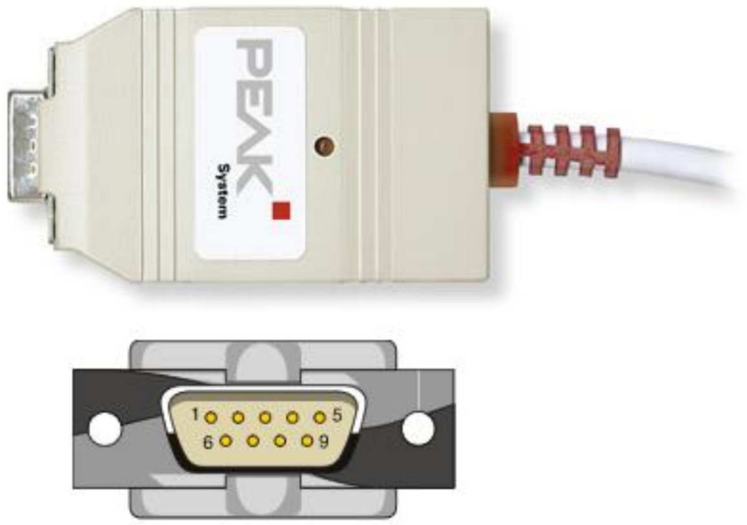


Figure 3.1: PCAN-USB device with 9-PIN connector [61]

3.3 Integration with RT-CACSDS

After installation of Linux based device drivers for this PCAN-USB as supplied by Peak-Systems, loadable kernel module, library, and some scripts for testing the device are available. Connecting the device to the PC triggers creation of a device file at /dev/pcan0. All these device driver files are written in C and C++ and can be easily integrated to RTOS and CACSDS. RTAI-LIB developed by Roberto Bucher [9] has a Scicos block interfacing PCAN-Dongle device for parallel port. These blocks are made of source files written in C and call the basic CAN device library functions such as opening, reading, writing, and status checking, respectively. Figure 3.2 shows a flowcharts explaining the sequence of functions called during the operation of a CAN device.

To have the PCAN-USB device recognized by Scicos, the initialization function CAN_Open() is replaced by Linux_CAN_Open(), which uses the actual device file /pcan/dev0 as one of the parameters. The message structure of the CAN device has three main parts: message ID, message length, and the message's data bytes. The PCAN Scicos block did not have any parameter that allows the user to specify custom baudrate. So, a parameter is added to select standard baudrate values such as 1000 kb/s, 500 kb/s, etc. These numbers are assigned the standard baudrate hexadecimal values in the source file of the block. For sending CAN messages, a source block in Scicos under RTAI-LIB is required that takes in the values supplied in a file or through Scicos GUI and sends them to PCAN-OUT block, which ultimately transmits the messages. But the Scicos scheme does not allow a user to send external messages in to CAN bus. Hence, the messages to be sent are defined in the initialization file peak.c. A parameter is included which takes in an integer value and selects the predefined message from the peak.c file. Thus, by using PCAN-OUT block, one can send predefined custom CAN messages with custom baudrate in real-time. The disadvantage is that the user has to recompile the library (RTAI-LIB) to send messages other than the predefined ones.

RTAI-LIB for Simulink, which is part of the RTAI distribution, does not have any blocks interfacing CAN devices. Hence, a C based S-function [51] of already existing Simulink S-functions is used as a template and a S-function of PCAN device is created linking the basic library files from the device drivers. In Simulink, to create such a block with access to devices, the source files including header files and library files are to be copied into the Matlabroot/rtw/c/libsrc directory and the S-function compiled under the Matlabroot/rtw/c/rtai/devices/ directory. Static library for the PCAN device is to be created under /usr/lib/ directory to link the CAN block.

Simulink blocks are developed to interface the PCAN device to receive and transmit CAN messages. The block shown in Fig. 3.3 is used for receiving CAN messages. The block has five output ports providing CAN bus status, message identifier, message type, message length, and data. The Simulink block shown in Fig. 3.4 has three input ports for message identifier, message length, and data. This block can be used for transmitting messages to CAN bus. Both blocks have two parameters: baudrate and sampling time. The C S-functions used in the input and output blocks are given in Appendix B.4.

Once the PCAN block is compiled and the executable is run, the messages are written to the actual CAN device hardware. To check whether the PCAN block is transmitting the correct messages, a simple circuitry involving a eZdsp 2812 [74] is used. Figure 3.5 shows the schematic connecting the PCAN-USB device to eZdsp 2812 through a CAN bus and the CAN driver SN65HVD230DR. The eZdsp board is connected to another PC running software for receiving and transmitting data.

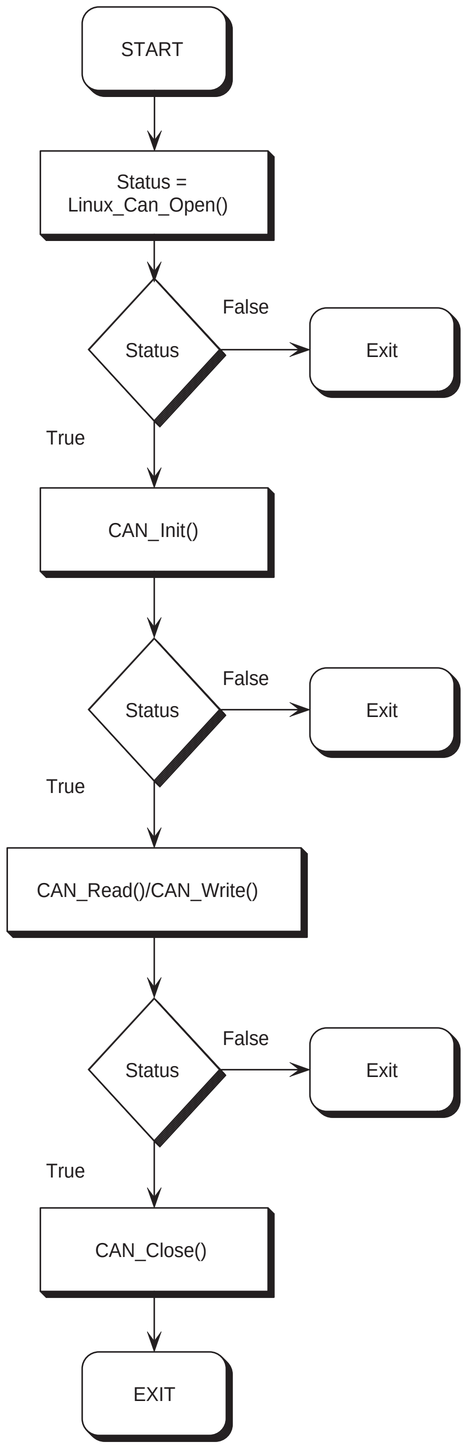


Figure 3.2: Flowchart showing PCAN block functionality

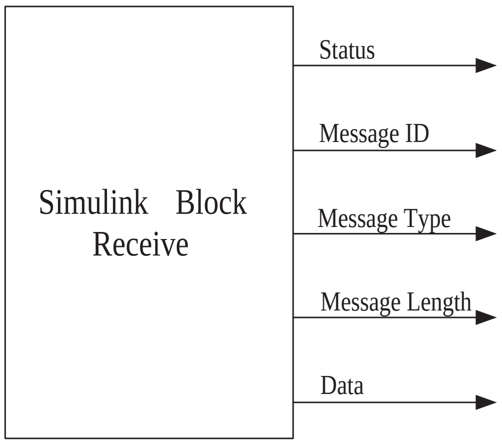


Figure 3.3: Simulink block for receiving CAN messages

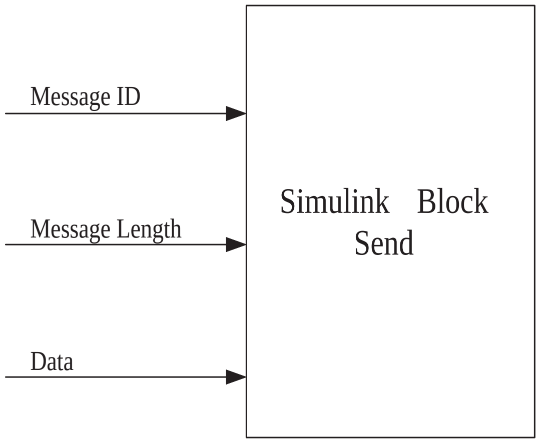


Figure 3.4: Simulink block for CAN message transmission

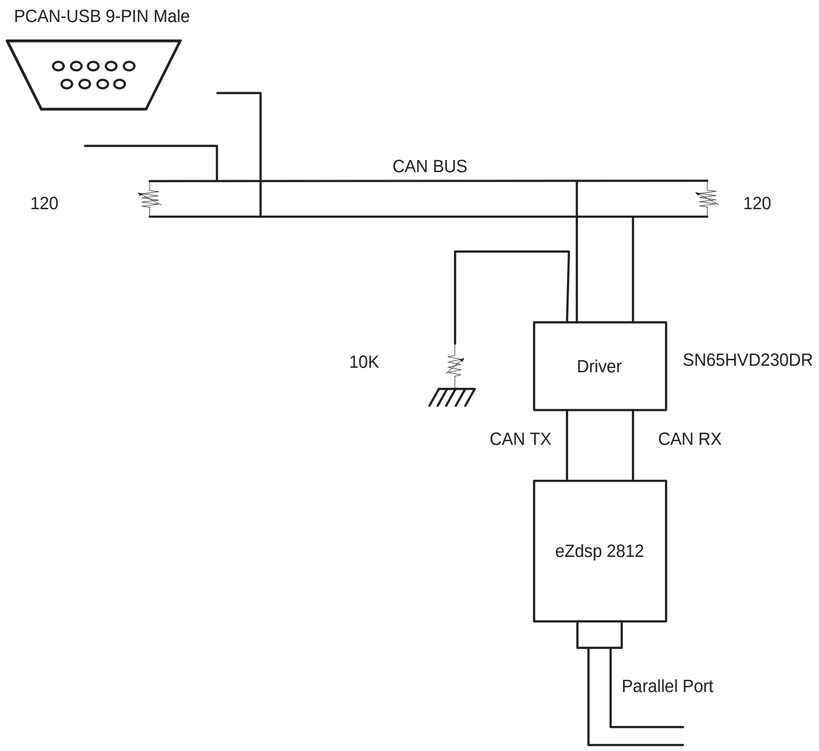


Figure 3.5: Schematic PCAN-eZdsp

Chapter 4

Application: HIL – Control Experiment

4.1 Introduction

This chapter tests RTAI's capabilities of Hardware-In-the-Loop simulation and control. The output voltage of a DC-DC Buck converter is controlled by the software part as implemented using RT-CACSDS Scilab/Scicos/RTAICodeGen.

4.2 Buck Converter

The DC voltage obtained by rectifying the line voltage only is not regulated. To convert this unregulated voltage to a regulated DC voltage, switching mode DC-DC converters are used. If the switch mode DC-DC converter is designed to lower the voltage then it known as a step down Buck converter.

The DC-DC converters are in general used for DC-power supplies and DC motor applications. Since the load in the application could change and the converter's components are not ideal, control of the output voltage of DC-DC converter is required. In a switch mode DC-DC converter, the value of output voltage is adjusted by varying the on- and off-periods of the switch. In one of the methods to control the output DC voltage, the duty ratio, which is defined as the ratio of on-period to off-period is varied. This method is known as Pulse Width Modulation (PWM). In this method, the control signal that controls the state of switch is generated by comparing a modulating control signal with a repetitive high-frequency carrier waveform. The control signal is obtained by amplifying the error or the difference between the actual output voltage and the desired voltage. When the amplified error signal, which varies slowly in time relative to the switching frequency, is greater than the carrier waveform, the logical switch control signal becomes high causing the switch to turn on [52]. Figure 4.1 shows the schematic of the implementation where the control of the Buck converter is achieved through the control signal generated by the software controls and interfaced via the data acquisition card.

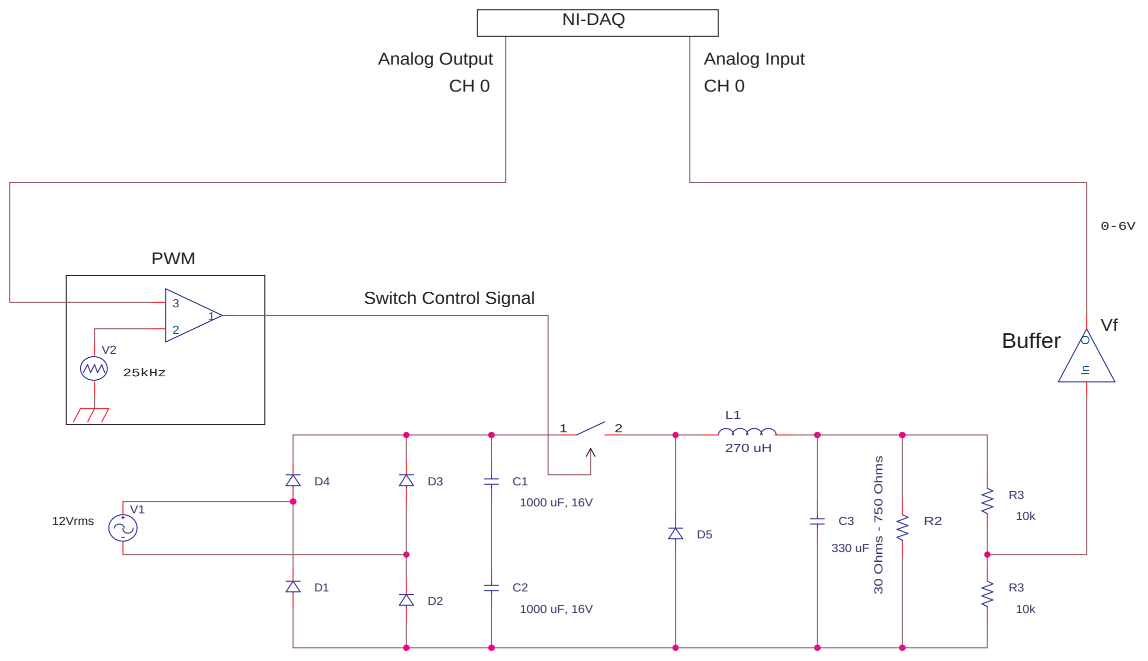


Figure 4.1: Schematic of buck converter

4.3 Design

An ideal DC-DC Buck converter has an ideal switch and constant input voltage. In practice, the switch will cause a small voltage drop and input voltage changes need to be compensated. The static characteristics of the converter are shown in Fig. 4.2. The presence of inductive components, which aid in filtering and reducing ripple currents, require a reverse diode. During the on-period the diode becomes reverse biased and the input provides energy to the load as well as inductor. During the off-period the inductor current flows through the diode transferring some of the stored energy to the load. The problem of fluctuating output voltage is reduced by using capacities at the input and output ports.

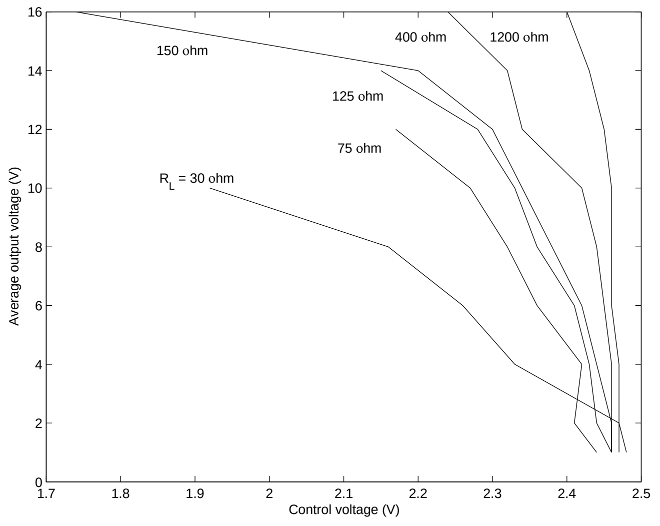


Figure 4.2: Buck converter characteristics

4.4 Control

The DC-DC converter is controlled in real-time using a controller designed in Scilab/Scicos/RTAICode-Gen (see Fig. 4.3). The AC-DC power supply using single-phase bridge rectifier produces unregulated DC voltage. The Buck converter, which includes a MOSFET switch driven by Pulse Width Modulation controlled switch, diode, and filter, produces a regulated DC voltage. A resistor bank serves as variable load.

The output voltage varies with the loading and line conditions. The measured output voltage signal is sent to the software based controller through the input channel of DAQ card as a feedback signal.

The Scicos blocks RTAI-Step, RTAI-Scope, Comedidataread, and Comedidatawrite are from the RTAI-LIB. A fixed reference signal is obtained from the RTAI-Step block. Comedi based blocks provide the interface to the DAQ card. A filter is used to reduce measurement noise. Usage of RTAI-Scope blocks in the Scicos scheme enables the user to monitor input and output signals while the system is running. The control signal from the discrete proportional-integral controller is sent to the PWM circuit through comedidatawrite block.

The Scicos scheme is compiled under RTAICodeGen that generates the executable running in real-time. A sampling time of 4 ms was chosen.

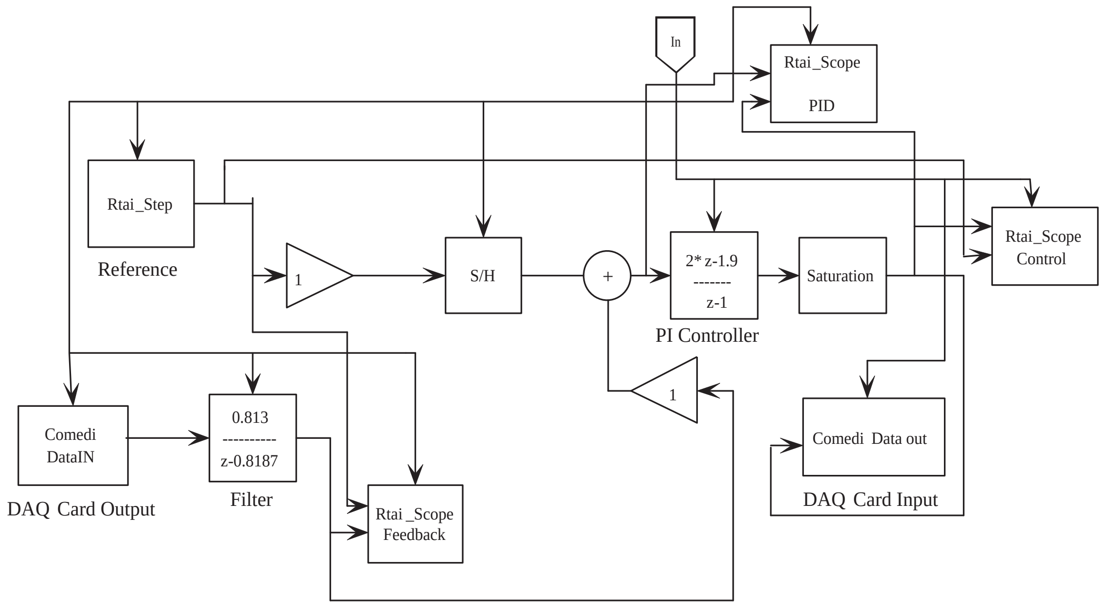


Figure 4.3: DC Voltage control scheme in Scicos

4.5 Results

The HIL-arrangement was tested for different loading conditions. The three loading conditions used represented a light load of 400 Ω, a medium load of 125 Ω, and heavy load of 30 Ω. The recorded traces of the converter's output voltage are shown in Fig. 4.4. In each case the output voltage recovers from both loading and unloading the converter and returns to the chosen reference voltage of 12.8 V.

4.6 Conclusions

This experiment was chosen to proof the concept of HIL capability of RT-CACSDS. The setup allows testing necessary components for achieving real-time control of hardware connected to a general purpose modeling and control design tool. The Buck converter is an important component of the envisioned all electric shipboard power system, and its incorporation into real-time modeling and simulation is an important step toward improved design capabilities.

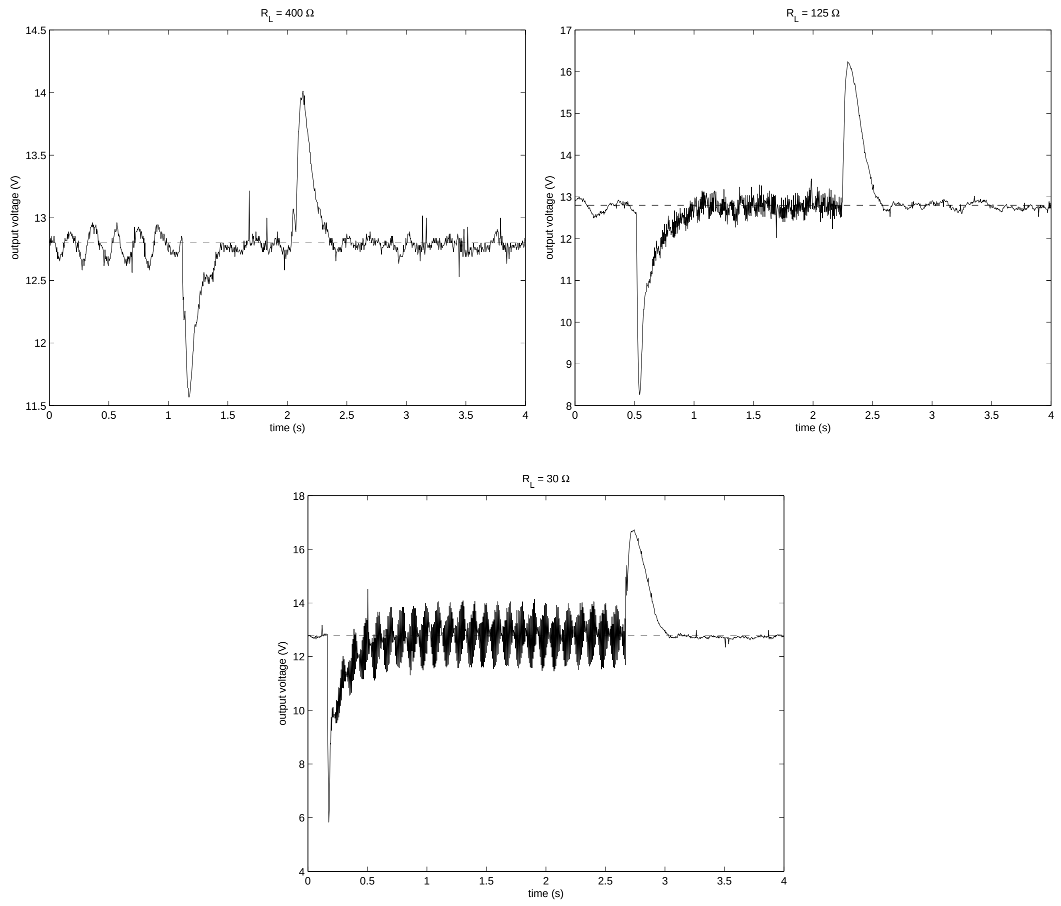


Figure 4.4: Output voltage for different loading conditions

Chapter 5

Application: Agent based Automatic Reconfiguration

5.1 Introduction

In this chapter, an automatic reconfiguration scheme for an energy management system with multi-agent technology is presented. Real-time simulation of the energy management system is demonstrated. The system under consideration is based on the concept of "Automatic Reconfiguration for Energy Management Systems of Electrical Shipboard Power System" using multi-agent systems as documented in [33].

5.2 Shipboard Power System

The system is a simplified version of the reduced scale integrated power system by the US Navy for demonstrating control design, survivability strategies, and stability analysis techniques [63]. The overall system is comprised of two parts: the Generation and Propulsion Testbed and DC Distribution Testbed (DCDT). The generation part has two AC sources: Port AC system and Starboard AC system consisting of turbine, exciter, AC generator, and propulsion motor. The two AC sources supply the propulsion systems and feed other loads via the DC distribution system. For reliability and robustness, AC is converted to DC for distribution of electric power. The DCDT has two DC buses: Port Bus and Starboard Bus. The loads are grouped into zones with each zone being supplied by additional DC-DC converters [33].

The Simulink model of SPS is shown in Fig. 5.1. The blocks, AC (AC generator), gas turbine, and power supply for initial startup represent the AC generation part of the SPS. The output of AC generator is connected to propulsion system and AC-DC supply system (DC PS). The DC power is then supplied through DC buses to three load zones where each zone has two loads. All the components of the system are modeled using first and second order transfer functions representing the relationship between active power and voltage values.

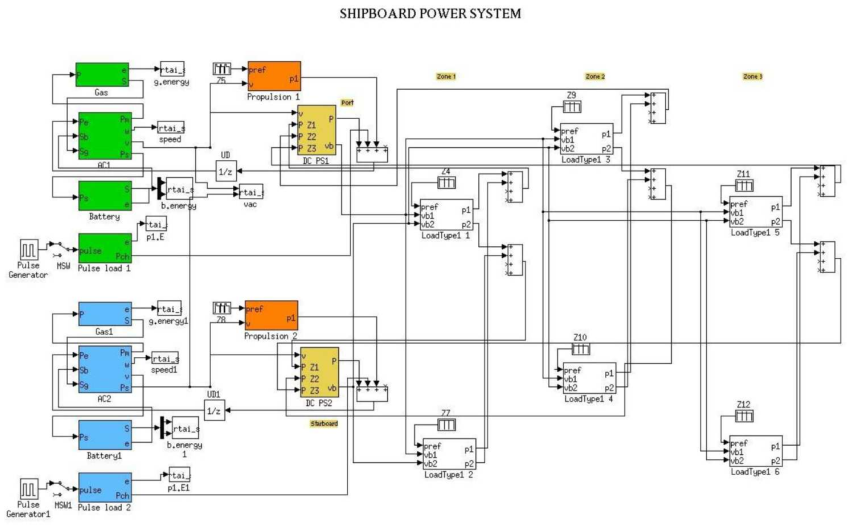


Figure 5.1: SPS Simulink model

5.3 Multi-Agent System

The concept of software agents used in building distributed systems that co-operate to reach a common goal was introduced in [83]. Similar philosophy is followed in implementing the multi-agent based control framework and applied to build the energy management system.

An agent can be defined as a system that tries to reach its design objectives by taking autonomous action in the environment it is placed in [83]. If a system has many such agents that interact with each other to reach common objectives while simultaneously each agent pursues its individual objectives, the system can be termed as multi-agent system [26].

The agents form a network of problem solvers and work together to solve problems that are beyond their individual capabilities. The ability to reconfigure in a decentralized approach makes this a fault tolerant scheme in building self-configuring energy management system.

5.4 Maximum Flow Problem and Graph Theory

Graph theory provides an appropriate framework for implementing the above discussed concept of distributed agents. The fundamental problem in graph theory is the maximum flow problem. Various sequential and parallel algorithms have been developed [3] to solve this problem. The algorithm developed in [34] is used to solve the power flow problem as it is simple, passive, and self-stabilizing.

Figure 5.2 shows a directed graph. Points s, i, j, k and t are nodes. The arc between the nodes i and j is called $edge(i, j)$ with a non-negative real-valued capacity c_{ij} and a real-valued flow f_{ij}. s is the source node with only outgoing edges and t is the sink node with only incoming edges. Residue of an edge can be defined as c_{ij}-f_{ij}=r_{ij}. Figure 5.3 shows the residual graph. The capacities of all the edges are 4. Directed path from source node to sink node is called augmenting path. In the figure below the augment path is $s - i - j - t$.

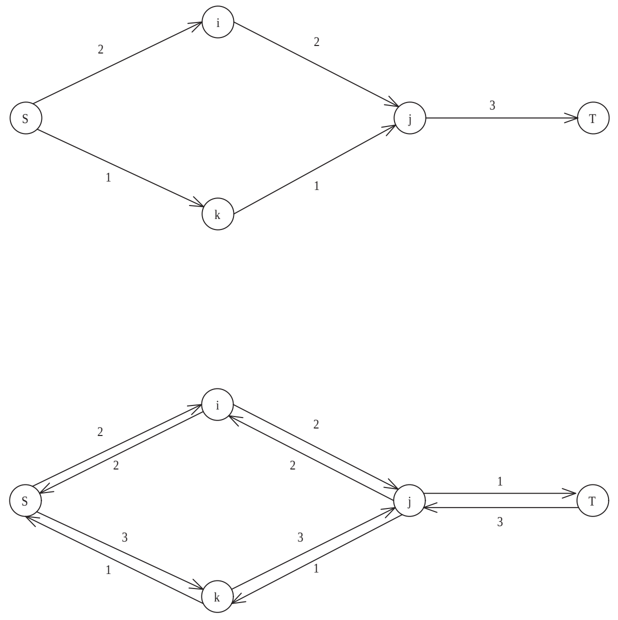


Figure 5.2: Directed digraph

The maximum flow problem is to maximize the flow from the source to the sink node. This is solved by increasing the flow on the augmenting path by the minimal residual value of the path. Figure 5.3 shows the solution reached by following the above procedure. The minimal residual capacity of augmenting path $s - i - j - t$ is $min(2, 2, 1) = 1$. Therefore, increasing the flow by 1 yields the maximum flow solution.

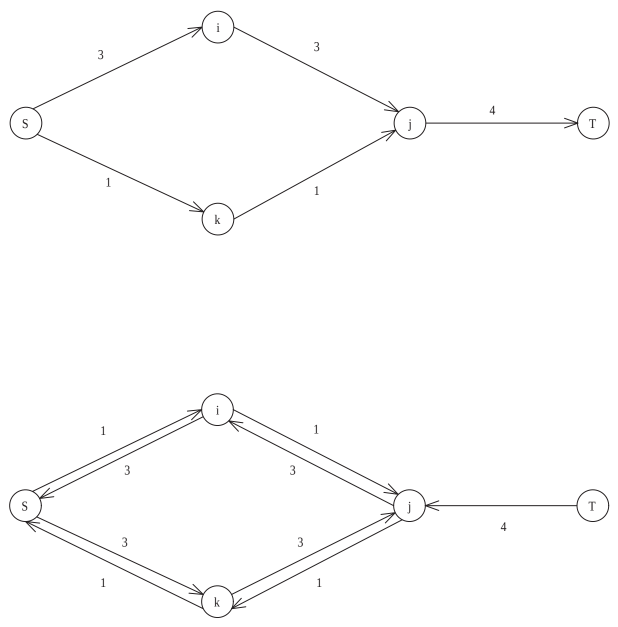


Figure 5.3: Digraph with solution

5.5 Maximum Flow Algorithm - Guarded Statements

A similar approach is followed in solving power flow problem as part of an energy management system for shipboard power systems. The first step is to transfer the entire system model into a directed graph (digraph) with each node representing a component. All AC generators are accommodated by links to the source node and all the load nodes by links to the sink node. Each node acts as an agent with common objective of ensuring power flow to all the loads according to their priorities.

Each agent runs an algorithm containing guards and corresponding actions changing the local variables associated with the agent to reach its design objective. In the case of the computational model for the power systems's digraph, the local variables changed are the flow f_{ij} according to the d-value of the particular node to make the demand at that node zero and the d-value itself. The d-value of a node can be defined as the believed shortest distance to the source node in the residual graph.

The following discusses the algorithm's guarded statements as run by each agent. In a digraph, let n be the number of nodes, i is the node under consideration, k is a predecessor node for i, and j is an adjacent node. The demand of node i, $demand(i)$, can be defined as outflow minus inflow. The d-value is represented as $d(i)$. The $edge(k, i)$ is an incoming edge for the node i with incoming flow of f_{ki} and a capacity of c_{ki}. All the nodes run the following rules except for the source node, which is idle. The d-value of the source node is permanently set to zero.

GS 1: If $c_{ki} > f_{ki}$ and $d(k) < n$ then $d(i) = d(k) + 1$ else $d(i) = n$.

Node i checks if the d-value of its predecessor nodes is less than n and if there is any possible direct path to the source node. If the conditions are satisfied then $d(i)$ is calculated as $d(i) = d(k) + 1$, where $d(k)$ is the d-value of the predecessor node. Otherwise $d(i)$ is assigned the maximum value n stating that there is no possible direct path form the source node available.

GS 2: If $demand(i) < 0$ then $f_{ki} = f_{ki} - \min(demand(i), f_{ki})$

If the demand at node i is less than zero then the inflow is greater than the outflow. The action to be taken in order to make the demand zero is to decrease the incoming flow by the minimum of the demand at node i and flow f_{ki}.

GS 3: If $demand(i) > 0$ and $d(i) < n$ then f_{ki}= $f_{ki} + \min(demand(i), (c_{ki} - f_{ki}))$

If demand at node i is greater than zero, i.e., outflow is greater than inflow, and the d-value is less than n, the corrective action is taken on the incoming edge to make demand zero by increasing the flow f_{ki} by minimum of $demand(i)$ and the residual capacity of the edge $k - i$, c_{ki}-f_{ki}.

GS 4: If $demand(i) > 0$ and $d(i) = n$ then f_{ij} = $f_{ij} - \min(demand(i), f_{ij})$

If the node's d-value has reached n and the demand is positive then the outgoing flow is reduced on the outgoing edges according to the priorities propagated from the load nodes.

GS 5: If there is a capacity violation then the flow on the corresponding edge is reduced by the amount $f_{ij} - c_{ij}$.

GS 6: If a node is directly connected to a load node then it is called a special node.

For a node i, if $demand(i) > 0$ and $d(i) = n$ and i is not special node and i connected to special node j and flow on the $edge(i, j) > 0$ then: if $sp\_nodepriority = 1$ then $f_{min} = find\_min(demand(i)$, flow from special node to low priority load) and $f(i, spl\_node) = f(i, spl\_node) - f_{min}$; if node $i = spl\_node$, then reduce flow towards low priority load;

If $demand(i) = 0$ and $d(i) = n$, and $i <> specialnode$, if i connected to special node and if special node is connected to any unsupplied high priority loads then reduce supply to low priority loads by minimum of $demand(spl\_node)$ and flow to priority loads until the high priority load is supplied. Furthermore, increase the flow to high priority load accordingly.

GS 7: Publish the status.

This guarded statement informs the shipboard power system about the convergence of the solution.

The algorithm is implemented in Simulink. Each statement in the algorithm is represented by if-else and action subsystem blocks from the ports and subsystems blockset. The if-else and action blocks are transformed into a subsystem representing an agent. Figure 5.4 below shows the outline of an agent as a Simulink block and its contents. Twenty two such blocks are used to represent the shipboard power system agents including 8 load nodes (2-propulsion loads, 6-regular loads), 6-converter modules, two AC generators, and two power supplies.

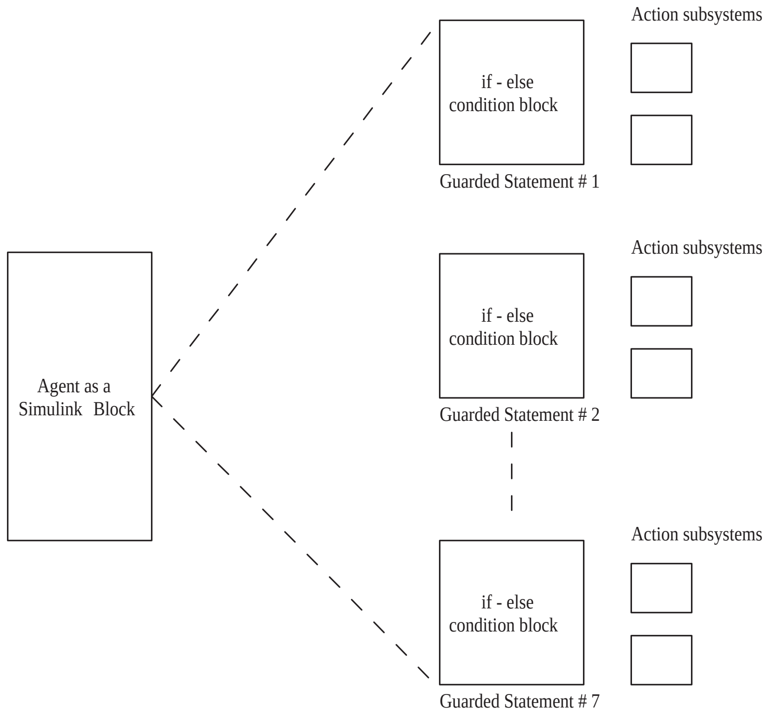


Figure 5.4: An agent as Simulink block

Capacities, flows, and d-values are initialized with the help of a Matlab script as given in Appendix B.3. All the agents are connected to their neighbors and exchange information. Each agent has a memory block associated with it to store the latest information regarding the flow and d-values. The twenty two node system is then run in Simulink. If the demands of all nodes are equal to zero and the load nodes are supplied power flow according to their demands and priorities then the algorithm has converged.

Interfacing the agent based energy management system with the shipboard power system physical model is taken care by the implementation layer. The implementation layer checks if the solution for the power flow problem has converged. If the solution has converged, a value is passed on to the connected loads of the shipboard power system. This value, which ranges from 0 to 1, is a ratio of the supplied flow and load demand where zero denotes that the load is completely cut off and 1 denotes that the load is supplied in full.

5.6 Real-Time Simulation

The Simulink model of the agent-based energy management system and the physical model of shipboard power system is simulated in real-time using Matlab/Simulink/RTW-RTAI. The entire system is run with a sampling rate of 1 ms using fixed step discrete solver.

Appendix A.4 shows the procedure to integrate Matlab/Simulink with RTAI. When integrated with Matlab, RTAI provides a Simulink library with scopes and log blocks. These blocks are used in the Simulink model instead of regular Simulink scope blocks.

From the Real-Time Workshop menu of Simulink, ‘rtai.tlc’ is selected as target language compiler and the code is generated by using “generate code only” option. With rtai-config added to the path variable, this code is compiled by issuing a ‘make’ command to create an executable for the Simulink model.

RTAI modules are loaded before launching the executable. With usage of ‘-w’ option, the task waits for the external initialization, which can be done using RTAI-LAB GUI.

Figure 5.5 shows a snapshot of RTAI-LAB GUI that allows the user to initiate the tasks, monitor the signals, log the data, and change the parameters of the model on the fly during the simulation. Thus, the GUI serves as a Command and Control Center as it further allows the user to change the parameters of the model such as capacities of the edges and priorities of the loads.

5.7 Real-Time Component

The executable for the Simulink model is a result of compiling code with RTAI library files located in Matlabhomedir/rtw/c/rtai and Matlabhomedir/rtw/lib/libsrc. The file rtmain.c, one of the main files is compiled into an object file rtmain.o, which is directly linked with the final executable tasks [8].

- rt_Main is the main task which initiates the other two tasks.
- rt_HostInterface is the task responsible for inter-task communication.
- rt_BaseRate is the task that actually runs in real-time. It is a Hard Real-Time Interrupt Service Routine.

The function “WaitTimingEvent” is a part of the outer-most loop of the real-time code and waits until the next sampling time is reached before calling the Simulink model. The model is said to run in real-time if Simulink manages to complete all the computations and is ready for the next time stamp

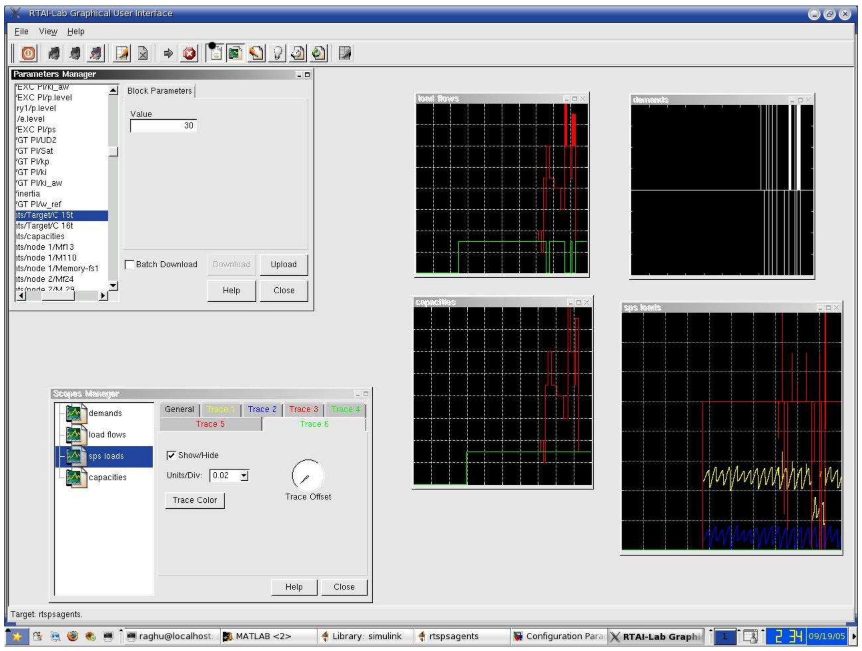


Figure 5.5: RTAI-LAB GUI serving as Command and Control Center

on a regular basis without failure. To profile the running model and check the behavior of the tasks at every time stamp, Linux based command ‘outb’ is used for low level parallel port output. The syntax for the command is outb(‘databyte’,‘portnumber’). The commands outb(0x00,0x378) and outb(0xFF,0x378) were inserted into the main loop to signal computation and idling time. The output of the parallel port is connected to an oscilloscope and the spikes at every time stamp during which the Simulink model is executed can be recorded. The following gives the plots obtained from the logged data and describes significant observation.

Figure 5.6 shows a plot for data logged during a single time stamp (1 ms). The x-axis represents the duration of one period and each time stamp represents one micro second. Y-axis is the voltage of the output of the parallel port. The pulse width gives the execution time for the model (40 μs).

Figure 5.7 shows the histogram representing the behavior of the execution time. The data was logged over a period of 20 ms, during which various values of execution time occurred. X-axis denotes the execution time in micro seconds and Y-axis represents the percentage of occurrence of these execution times. The plot shows that Simulink computation time was around 28 μs for 50% of the time and for 20% of the time the execution time increased up to 48 μs (this is when the Simulink model is in reconfiguration process).

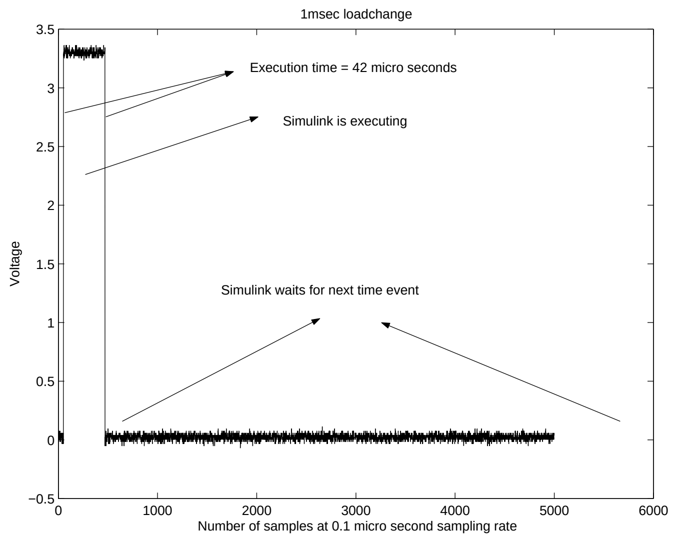


Figure 5.6: Oscilloscope data for single time stamp, 1 ms

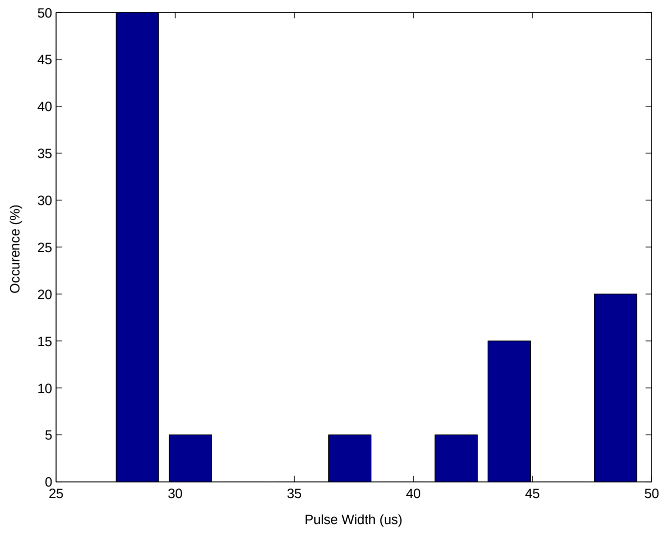


Figure 5.7: Histogram showing the behavior of the task

Further, to check the real-time capability of RTAI, the consistency of the desired sampling time is checked. The data is logged at a sampling rate of 100 ns. It used to calculate the time between two computation instances. This value should be close to the selected sample time of 1 ms. A histogram is shown in Fig. 5.8 and gives the variation of the sample time for a period of 100 ms. It can be seen that the sample time was 0.9992 ms for more than 90% of the time. The reason for the value being less than 1 ms is due to the inaccuracies caused by the available recording capabilities.

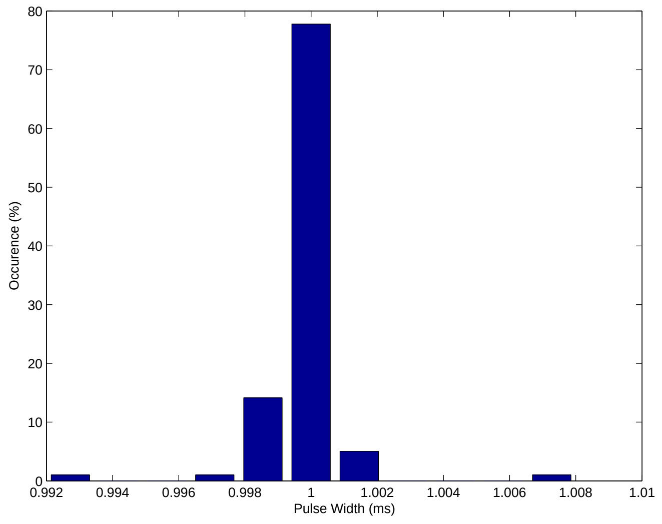


Figure 5.8: Histogram showing the behavior of the sampling time

Therefore, from this histogram can be concluded that the energy management system is running in real-time without missing deadlines, i.e, not violating the 1 ms sample time for computations, and consistently follows the target rate with only minor jitter.

5.8 Energy Management Case Studies

Agent based energy management system is tested for various scenarios including startup scenario, loss of edge, and load changes. In startup scenario, system starts with no power supplied to any load. Five different modes were chosen: Full power, Travelling, Harbor, Cruising, and Fighting. Table 5.3 shows the values of priorities and the percentage of loads chosen for these modes.

Data for flow values on all edges, demand values at all nodes, and d-values of all nodes was logged using RTAI_SCOPE blocks at 1 ms sampling rate. This raw data is then used to calculate total number of moves by all agents for each scenario. Appendix B.1 shows the Matlab script used to calculate number of moves. Figure 5.9 shows the total number of moves for sampling rates of 1 ms, which settles to a constant value denoting that the solution for maximum flow problem has converged.

Figure 5.10 shows plots giving the status of all agents during the simulation for sampling rate of 1ms. A parameter called agent status factor is defined as the ratio of the sum of status of all agents to the total

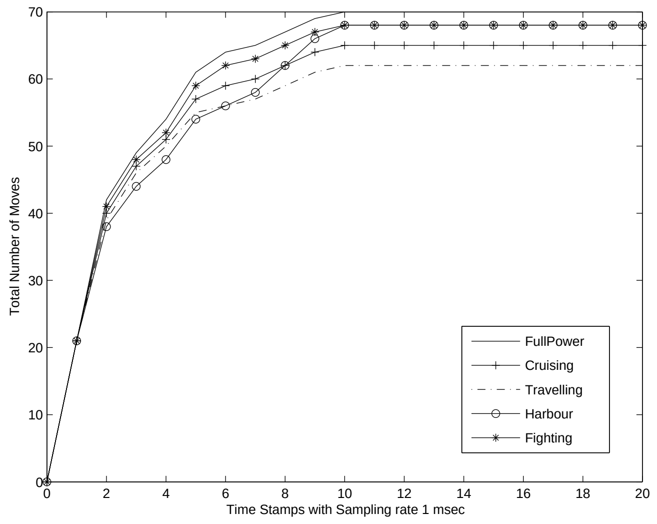


Figure 5.9: Total number of moves for various modes

number of agents. This value, which ranges from 0 to 1, is calculated from the raw data, the Matlab script is given in Appendix B.2, and plotted against the simulation time. The plot depicts that during the initial stages of simulation 85% of the agents were actively negotiating and their number gradually decreases with time and becomes zero after 10 time stamps denoting that the solution has converged. Table 5.1 gives the maximum, average, and minimum number of moves observed for several test runs.

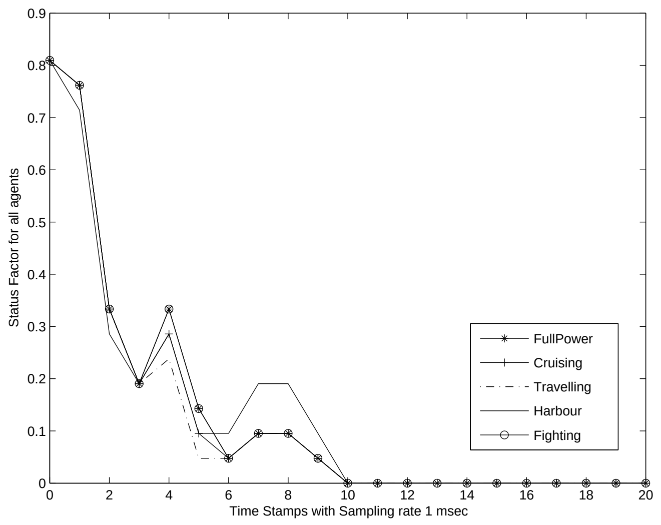


Figure 5.10: Agent Status Factor for 1 ms sampling rate

Table 5.1: Maximum, minimum and average moves for the ‘startup’ scenario

		Full-power	Cruising	Travelling	Harbor	Fighting
Startup	Max	88	72	77	73	75
	Avg	77	65	70	64	68
	Min	68	59	61	59	62

A similar procedure is followed in the load change scenario. Load priorities were chosen arbitrarily and the edge capacity of the load nodes to the target were changed randomly to ensure that the high priority loads were supplied in full all the time.

The figures below show the digraph models for 22 node energy management system for various cases of load changes. Figure 5.11 shows the initial setup. Figure 5.12 shows that when demand on node 15 is increased to maximum demand that can be supplied, as the priority of the node 15 is high (1) and its adjacent load node is of low priority (0), the lower priority load is completely cut off and the high priority load is supplied in full. Table 5.2 gives the maximum, average, and minimum number of moves made by agents during the process of random load changes.

Table 5.2: Maximum, minimum, and average moves for the ‘load change’ scenario

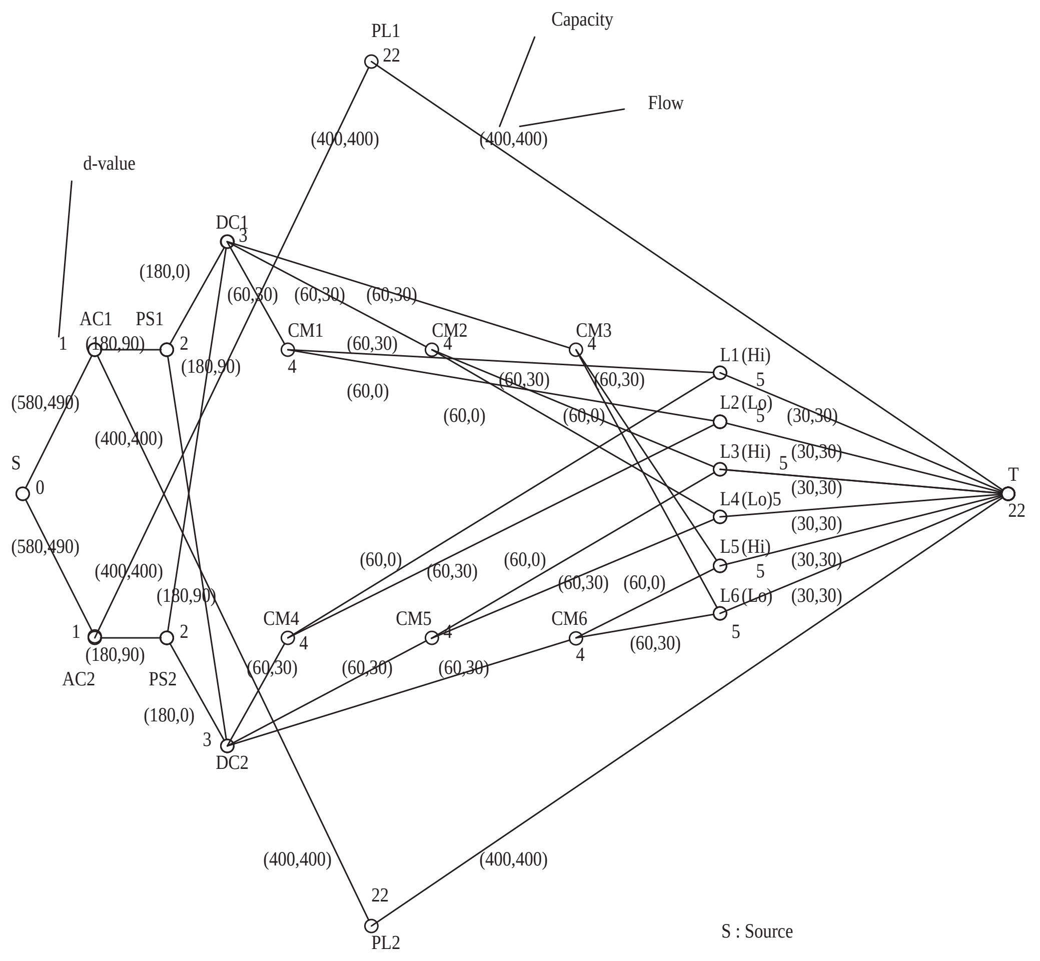
		Moves
Loadchange	Max	45
	Avg	23
	Min	7

Figure 5.13 shows the loss of edge scenario. The capacity value of the edge connecting node 15 and the sink is made zero. The adjacent low priority load node (which was cut off earlier to supply the high priority load) is again supplied according to its demand.

Figures 5.14 and 5.15 show the number of moves made by the agents to reconfigure the system for the above made changes and the status of the agents during the reconfiguration.

5.9 Conclusions

Multi-agent technology is implemented in building automatic reconfigurable energy management system for a shipboard power system. Simulink was chosen as the tool for modeling the energy management system. To check the reconfiguration capability of the energy management system, various case studies were run using real-time capabilities of Linux-RTAI as a real-time operating system with a sampling rate of 1 ms and real-time capability of the simulation was verified using an external monitoring device.



S : Source

AC : AC Generator

PS : Power Supply

DC : DC Bus

CM : Converter Module

PL : Pulse Load

Hi : High Priority (1)

Lo : Low Priority (0)

L : Load

T : Target - Sink Node

Figure 5.11: Initial setup

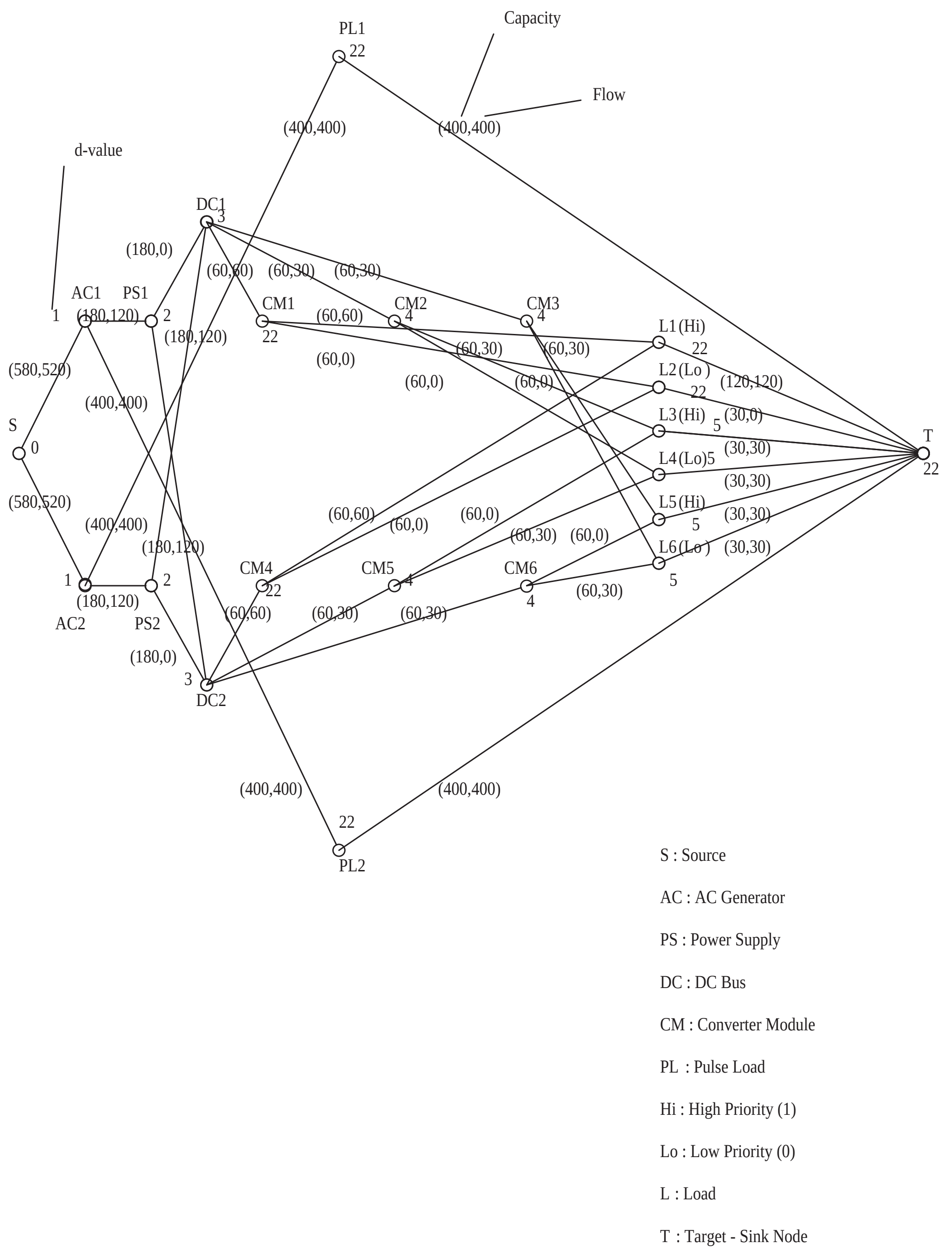
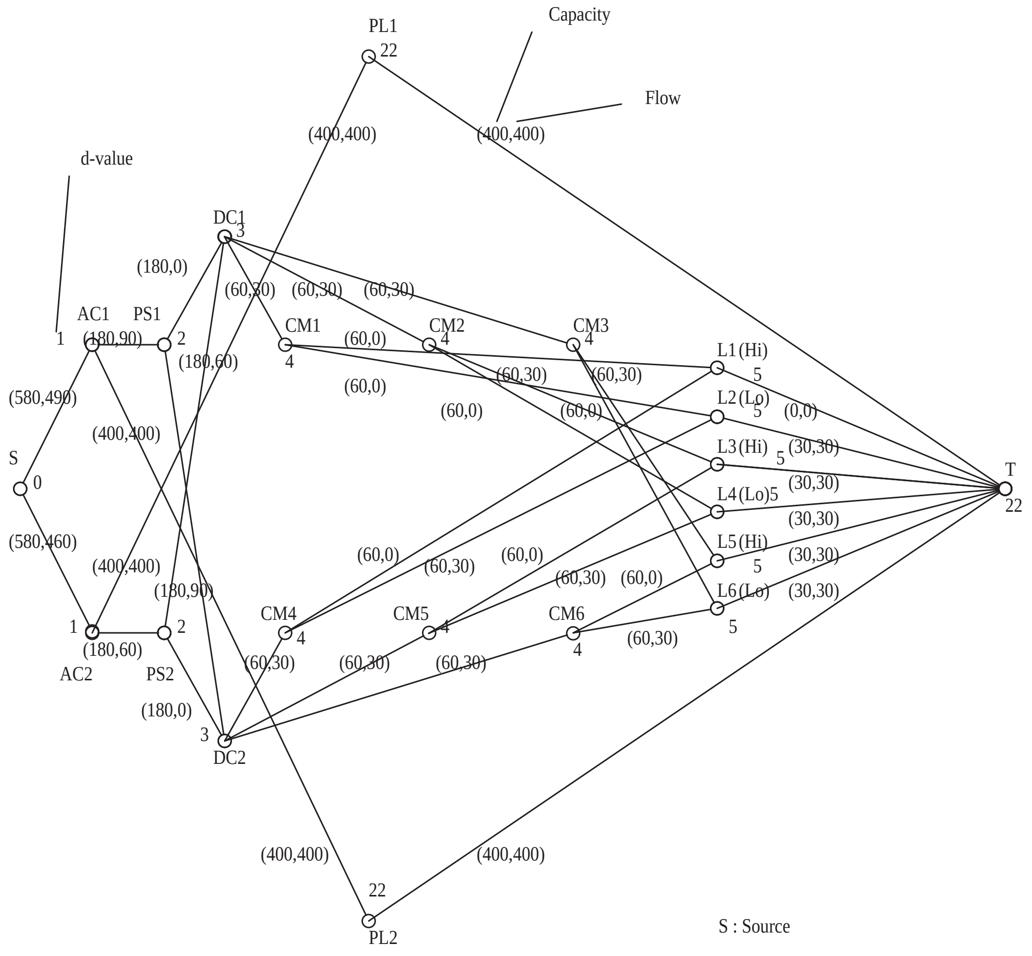


Figure 5.12: Change in demand



S : Source

AC : AC Generator

PS : Power Supply

DC : DC Bus

CM : Converter Module

PL : Pulse Load

Hi : High Priority (1)

Lo : Low Priority (0)

L : Load

T : Target - Sink Node

Figure 5.13: Loss of edge

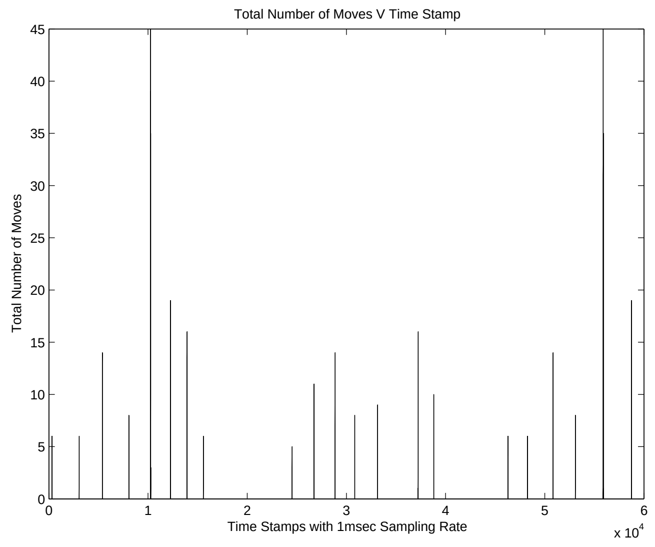


Figure 5.14: Number of Moves for 1 ms sampling rate during random load change

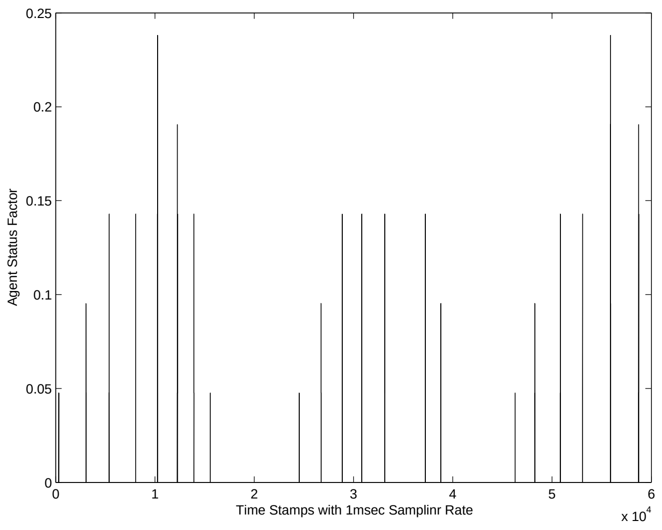


Figure 5.15: Agent Status Factor for 1 ms sampling rate during random load change

Table 5.3: Values of the loads and priorities in various modes

Mode	Load	Priority	Capacity in %
Full-power	Star Propulsion Load	High	100
	Port bus Propulsion Load	High	100
	Zone1 Load1	High	100
	Zone1 Load2	High	100
	Zone2 Load1	High	100
	Zone2 Load2	High	100
	Zone3 Load1	High	100
	Zone3 Load2	High	100
Cruising	Star Propulsion Load	High	40
	Port bus Propulsion Load	High	40
	Zone1 Load1	High	40
	Zone1 Load2	Low	40
	Zone2 Load1	High	70
	Zone2 Load2	Low	70
	Zone3 Load1	High	70
	Zone3 Load2	Low	70
Travelling	Star Propulsion Load	High	70
	Port bus Propulsion Load	High	70
	Zone1 Load1	Low	70
	Zone1 Load2	High	70
	Zone2 Load1	Low	40
	Zone2 Load2	High	40
	Zone3 Load1	Low	70
	Zone3 Load2	High	70
Harbor	Star Propulsion Load	Low	0
	Port bus Propulsion Load	Low	0
	Zone1 Load1	High	30
	Zone1 Load2	Low	30
	Zone2 Load1	High	30
	Zone2 Load2	Low	30
	Zone3 Load1	Low	30
	Zone1 Load2	High	30
Fighting	Star Propulsion Load	High	50
	Port bus Propulsion Load	High	50
	Zone1 Load1	High	90
	Zone1 Load2	High	90
	Zone2 Load1	High	90
	Zone2 Load2	High	90
	Zone3 Load1	High	50
	Zone3 Load2	High	50

Chapter 6

Conclusions

6.1 Summary

The objective of this thesis was to evaluate and develop real-time platforms for a multi-agent based shipboard energy management system. The automatic reconfiguration of shipboard power systems is a crucial step towards improving survivability. Multi-agent technology was applied to implement the reconfigurable energy management scheme using a self-stabilizing maximum flow algorithm. The agent based energy management system is designed in a Matlab/Simulink environment. Reconfiguration is performed for several situations including start-up, loss of an agent, limited available power, and distribution to priority ranked loads. The number of steps taken to reach the global solution and the time taken are very promising. The timing accuracy of these simulations has been verified successfully. A survey concerning freely available real-time operating systems and software tools to setup a desktop PC supporting real-time environment was conducted. Matlab/Simulink/RTW-RTAI was selected as real-time computer aided control design software for demonstrating real-time simulation of agent based energy management system, HIL applications, and communication.

Table 6.1 gives a list of software tools available and their support of various aspects with respect to real-time modeling and simulation. The following conclusions are drawn from this work:

1. Linux based RTOS is an ideal choice for a platform to work with hard real-time applications.
2. When compared to RT-Linux, RTAI stands out due to the following reasons:
    (a) High stability
    (b) Large community support
    (c) Continuous development keeping pace with the latest stable kernel releases

   (d) Completely open source (unlike its counterpart RT-Linux in which case the advanced version comes with a commercial license).

3. Another major advantage in choosing RTAI is that it could be easily integrated to the completely open source Scilab/Scicos and the commercial Matlab/Simulink, which is currently the more powerful and flexible CACSDS.
4. Though Scilab/Scicos, when integrated with RTAI, forms a powerful tool for modeling and simulation of real-time control systems, it has its own limitations such as:
   (a) Lack of support for modeling large-scale systems such as power systems that include complex variables.
   (b) As the entire model is compiled into an executable that runs in the kernel space, debugging of the resulting application becomes very complex.
   (c) Interfacing external software such as Ada 95 and C is not that flexible, though Scicos comes with a building block supporting C interface, it is limited to simple and small applications.
   (d) Scilab/Scicos lags behind its commercial counterpart Matlab/Simulink in terms of support and documentation.
5. Matlab/Simulink when integrated with RTAI proves to be a better choice for CACSDS supporting hard real-time when compared to Scilab/Scicos in terms of support for modeling large-scale systems and complex variables.
6. Integration of external programming languages such as Ada 95 and C is simpler in the case of Matlab/Simulink with the help of S-functions.
7. Integration of CAN device interface is more flexible in the case of Matlab/Simulink when compared to Scilab/Scicos.

The applications presented demonstrate the platforms real-time capabilities with respect to modeling and simulating, communications, and Hardware-in-the-Loop arrangements. These applications were chosen to evaluate important aspects in modeling, simulating, and controlling of an ongoing ONR/DoE research project that concerns multi-agent based reconfiguration of shipboard electric power systems. As this thesis provides the proof-of-concept, further work concerning a small-scale hardware prototype can build on these tools and benefit from early evaluation of hard real-time requirements.

Table 6.1: List of software tools

Software Tools	**Matlab/ Simulink/ RTW on Windows**	**Ada 95 on RTLinux**	**Real-Time Java**	**Scilab/Scicos on RTAI**	**Matlab/ Simulink on RTAI**
Real-Time Support	Soft	Hard	Soft	Hard	Hard
Graphics	Yes	No	No	Yes	Yes
Real-Time Communication	Yes	No	No	No	Yes
Control Design	Yes	Yes	No	No	Yes
C-Interface	Yes	Yes	Yes	No	Yes
General Purpose Code	Yes	Yes	Yes	Yes	Yes
Support	Technical	No	Community	Community	Technical and Community
Math-FFT	Yes	No	No	No	Yes
Multi-Tasking	No	Yes	Yes	No	Yes
Documentation	Yes	No	Yes	No	Yes

6.2 Future Work

The agent-based reconfiguration concept still lacks consideration of communication delays between agents. Clearly, design of the energy management system would benefit from simulating these delays and allow further adjustments in expected real-world performance. Also, support for networked control simulations under RTAI could be added and verified with the help of already available third part add-on libraries for Simulink.

Real-time simulation of multi-agent system in a distributed environment (on more than one computer) would further explore the real-time capability of software tools such as RTAI-LAB.

Until now, the systems used for real-time simulation were discrete models. New release of RTAI-Magma supports real-time simulation of continuous systems as well and will be one more important addition to the software tools available.

References

[1] AdaPower. *Ada Reference Manual*. http://www.adapower.com/rm95/, 2005.

[2] Aeolian Inc. *Introduction to Linux for Real-Time Control*. available at: http://aeolean.com/html/RealTimeLinux/RealTimeLinuxReport-2.0.0.pdf, 2002.

[3] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows - Theory, Algorithms and Applications*. Prentice Hall, 1993.

[4] T. Baker. *FSU Pthreads*. http://moss.csc.ncsu.edu/ mueller/pthreads/, 2005.

[5] G. Bollella, B. Brosgol, P. Dibble, S. Furr, J. Gosling, D. Hardin, and M. Turnbull. *The real-time specification for Java*. Addison-Wesley, 2000.

[6] Robert Bosch GmbH. *CAN Specification*. available at: http://www.algonet.se/ staffann/developer/can2spec.pdf, 1991.

[7] B. J. Brosgol. *Ada and Java: real-time advantages*. available at: http://www.embedded.com/showArticle.jhtml?articleID=16100316, 2003.

[8] R. Bucher. *Roberto Bucher's homepage*. http://a.die.supsi.ch/ bucher/, 2005.

[9] R. Bucher. *Scilab Howto*. available at: http://a.die.supsi.ch/ bucher/scilab-howto.pdf, 2005.

[10] A. Burns and A. Wellings. *Concurrency in Ada*. Cambridge University Press, 1998.

[11] K. L. Butler, N. D. R. Sarma, and V. R. Prasad. Network reconfiguration for service restoration in shipboard power distribution systems. *IEEE Transactions on Power Systems*, 16(4):653–661, November 2001.

[12] K. L. Butler-Purry and N. D. R. Sarma. Self-healing reconfiguration for restoration of naval shipboard power systems. *IEEE Transactions on Power Systems*, 19(2):754–762, May 2004.

[13] CAN In Automation. *CANopen*. available at: http://www.can-cia.org/canopen/, 2005.

[14] CAN In Automation. *Controller Area Network*. available at: http://www.can-cia.org/can/, 2005.

[15] CAN In Automation. *DeviceNet*. http://www.can-cia.org/devicenet/, 2005.

[16] Comedi. *Comedi Hardware Support List*. http://www.comedi.org/hardware.html, 2005.

[17] Comedi. Comedi homepage. Website, 2005.

[18] Ty Coon. *GPL License*. available at: http://www.gnu.org/copyleft/gpl.html, 1991.

[19] Department of Defense, Ada Joint Program Office. *Ada 95 Quality and Style Guide: Guidelines for Professional Programmers*. http://www.adaic.com/docs/95style/html/sec_6/, 2005.

[20] Dipartimento di Ingegneria Aerospaziale Politecnico di Milano. *RTAI Homepage*. http://www.rtai.org/, 2005.

[21] S. Doran. Interfacing low-level C device drivers with Ada 95. In *Annual International Conference on Ada*. ACM Press, 1999.

[22] L. Dozio. RTAI readme. Website, 2005.

[23] L. Dozio and P. Mantegazza. Linux real time application interface RTAI in low cost high performance motion control. In *Motion Control*. National Italian Association for Automation, 2003.

[24] L. Dozio and P. Mantegazza. Real-time distributed control systems using rtai. *Sixth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC)*, May 2003.

[25] Gensen. D. E. Hard and soft real-time. Website, 2004.

[26] J. Ferber. *An Introduction to Distributed Artificial Intelligence*. Addison-Wesley, 1999.

[27] Free Software Foundation. Gnat compiler. Website, 1998.

[28] FSMLabs, Inc. *Controls Kit Homepage*. http://www.fsmlabs.com/controls-kit.html, 2005.

[29] FSMLabs, Inc. *FSMLabs*. http://www.fsmlabs.com/, 2005.

[30] FSMLabs, Inc. *RTLinux GPL*. available at: http://www.rtlinuxfree.com/, 2005.

[31] FSMLabs, Inc. *RTLinuxFree Homepage*. http://www.rtlinuxfree.com/, 2005.

[32] FSMLabs, Inc. *RTLinuxPro Homepage*. http://www.fsmlabs.com/rtlinuxpro.html, 2005.

[33] S. B. Ganesh, K. Schoder, H.-J. Lai, A. Al-Hinai, and A. Feliachi. Energy management system with automatic reconfiguration for electric shipboard power system. *Proc. of Reconfiguration and Survivability Symposium (RSS) 2005*, February 2005.

[34] S. Gosh, A. Gupta, and S. V. Pemmaraju. A self stabilizing algorithm for the maximum flow problem. *Distributed Computing*, 10(4):167–180, 1997.

[35] E. R. Hilton and V. Yodaiken. Real-time applications with RTLinux. *Linux Journal*, 2001.

[36] IEEE and The Open Group. *IEEE POSIX*. http://posixcertified.ieee.org/, 2005.

[37] IEEE Standards Association. *Home page*. available at: http://standards.ieee.org/, 2005.

[38] Institut National de Recherche en Informatique et Automatique. *Scilab Homepage*. http://www.scilab.org/, 2005.

[39] ISO. *International Organization for Standardization*. http://www.iso.org/, 2005.

[40] M. Johnson. *Real-Time Java*. available at: http://www.abo.fi/∼mjohnson/studier/rtj.pdf, 2002.

[41] Inc. Lineo Solutions. *Lineo Solutions Homepage*. http://www.lineo.co.jp/eng/index.html, 2005.

[42] LinuxDevices. *ART Linux*. available at: http://www.linuxdevices.com/links/LK6839947367.html, 2000.

[43] LinuxDevices. *The Real-time Linux Software Quick Reference Guide*. available at http://www.linuxdevices.com/articles/AT8073314981.html, 2003.

[44] Inc. LynuxWorks. *LynuxWorks Homepage*. http://www.lynuxworks.com/, 2005.

[45] P. Mantegazza, E. Bianchi, L. Dozio, S. Papacharalambous, S. Hughes, and D. Beal. RTAI: Real-time application interface. *Linux Journal*, April 2000.

[46] P. N. Mathre. *Real-Time Operating Systems*. available at:
http://www.onesmartclick.com/rtos/rtos.html, 2004.

[47] Mathworks Inc. Mathworks homepage. Website, 2005.

[48] Mathworks Inc. *Real-Time Windows Target*. http://www.mathworks.com/products/rtwt/, 2005.

[49] Mathworks Inc. *Real-Time Windows Target Documentation, Real-Time Kernel*.
http://www.mathworks.com/access/helpdesk/help/toolbox/rtwin/ug/, 2005.

[50] Mathworks Inc. *Real-Time Workshop*. http://www.mathworks.com/products/rtw/, 2005.

[51] Mathworks Inc. *S-Function*.
http://www.mathworks.com/access/helpdesk/help/toolbox/simulink/sfg/index.html, 2005.

[52] N. Mohan, T. M. Undeland, and W. P. Robbins. *Power Electronics: Converters, Applications, and Design*. Johne Wyle, 1995.

[53] P. Mourot. *RTAI Architecture*.
http://www.aero.polimi.it/ rtai/documentation/articles/patric_mourot-rtai_internal_presentation.html, 2003.

[54] A. Muir. *JBed - a Java-based Real-Time Operating System*. available at:
http://www.microjava.com/jvm/software/jit/jbed2?content_id=695, 2001.

[55] National Marine Electronics Association. *NMEA 2000 A Digital Interface for 21st Century Download*. available at: http://www.nmea.org/Standards/Publications/NMEA2000ADigitalInterface.pdf, 2002.

[56] K. Nilsen. *PERC*. http://mindprod.com/jgloss/perc.html, 1996.

[57] A. Nilsson. *Compiling Java for Real-Time Systems*.
http://curry.ludat.lth.se/cs/events/Entries/20040430112010833320417/lucasView_html, 2004.

[58] S. Nilsson. *CAN Introduction*. http://www.algonet.se/ staffann/developer/CAN.htm, 2005.

[59] NIST. *Requirements for Real-time Extensions for the Java Platform*. http://www.itl.nist.gov/div897/ctg/real-time/intro.html, 2004.

[60] University of Kansas. *KURT Linux*. http://www.ittc.ku.edu/kurt, 2005.

[61] Peak-System Technik GmbH. *PCAN-USB Documentation*. available at: http://www.peak-system.com/pdfs/gb/PEAK-SysE.pdf, 2005.

[62] Peak-System technik GmbH. Peak-system homepage. Website, 2005.

[63] S. D. Pekarek, J. Tichenor, S. D. Sudhoff, J. D. Sauer, D. E. Delisle, and E. J. Zivi. Overview of naval combat survivability program. *Proceedings of 13th International Ship Control Systems Symposium*, 2003.

[64] QNX Software, Inc. *QNX*. http://www.qnx.com/products/rtos/, 2005.

[65] R. U. Rehman. Open source real-time operating systems. *Sys Admin Magazine*, 2001.

[66] I. Ripoli. *RTLinux versus RTAI*. http://bernia.disca.upv.es/rtportal/comparative/rtl_vs_rtai.html, 2002.

[67] I. Ripoll. *RTL Gnat*. http://rtportal.upv.es/apps/rtl-gnat/, 2005.

[68] RTSJ. *Real-Time Specification for Java*. http://www.rtsj.org/, 2005.

[69] R. Schwebel. *RTAI Overview*. http://www.elektroniknet.de/topics/embeddedsystems/fachthemen/artikel/02008c.htm, 2002.

[70] M. E. Sharon. *Java Micro Edition*. http://uberthings.com/mobile/, 2005.

[71] P. Shenoy. *QLinux*. http://lass.cs.umass.edu/software/qlinux/, 2002.

[72] M. I. Skitz. *Automated JavaScript Imports Language Extension*. http://ajile.sourceforge.net/, 2005.

[73] Soundlabs Group. *CAN History*. http://www.soundlabsgroup.com.au/canbus/can_history.htm, 2005.

[74] Spectrum Digital. *eZdsp Manual*. available at: http://www.spectrumdigital.com, 2005.

[75] Stage Harbor Software. *Ada 95 History*. http://www.learnada.com/history.htm, 2005.

[76] Sun Microsystems. *Kilo Virtual Machine: White Paper*. available at: http://java.sun.com/products/cldc/wp/, 2000.

[77] Sun Microsystems. *PICO Java*. http://www.sun.com/microelectronics/picoJava/, 2005.

[78] T. S. Taft and R. A. Duff. *Ada 95 Reference Manual: Language and Standard Libraries, International Standard ISO/IEC 8652:1995(E)*. Springer-Verlag, 1997.

[79] M. Timmerman. *Windows NT*. available at: http://www.omimo.be/magazine/97q2/winntasrtos.htm, 2005.

[80] TopWorx. *Modbus History*. available at: http://www.topworx.com/fnmb.html, 2005.

[81] TransEra. *GPIB Information*. available at: http://www.transera.com/htbasic/tutgpib.html, 2005.

[82] A. J. Tucker. Opportunities and challenges in ship systems and control at ONR. *IEEE Conference on Decision and Control*, December 2001.

[83] G. Weiss. *Multi Agent Systems - A Modern Approach to Distributed Artificial Intelligence*. MIT Press, 1999.

[84] Wind River. *VxWorks*. http://www.windriver.com/, 2005.

[85] P. Zsolt. *GPIB Introduction*. available at: http://www.hit.bme.hu/people/papay/edu/GPIB/tutor.htm, 2005.

Appendix A

Installation Guides

A.1 Mandrake 10 with 2.6.7-Adeos

1. INSTALLING MDK 10
   Select the following while installing Mandrake
   - Standard security
   - Use existing partitions or partition accordingly (suggested: one partition for the root directory ("/"), one for /home, and a swap-partition)
   - Select mount-points for partitions

   Selecting your x-environment – the work hear was performed using KDE. Install the bootloader on (First sector of drive (MBR)) and then, REBOOT.
2. Install all the utilities required for installing RTAI and Scilab, i.e., g77 fortran compiler, etc.
3. Configuring the kernel 2.6.7 downloaded from www.kernel.org
   - #cd /usr/src
   - #tar -jxvf linux2.6.7.tar.bz2
   - #tar jxvf rtai3.1.tar.bz2 (http://www.aero.polimi.it/RTAI/rtai3.1.tar.bz2)
   - #ln -s rtai3.1 rtai
   - #cd linux
   - #patch p1¡/usr/src/rtai/rtaicore/arch/i386/patches/hal72.6.7.patch.patch

   In order to solve the supermount problem due to which one may not be able to access their cdroms and zip-drives, a supermount patch should be applied to the kernel before compiling. FOR THIS TO WORK, SUPERMOUNT OPTION SHOULD BE SELECTED WHILE CONFIGURING THE NEW KERNEL IN THE FILE SYSTEMS SECTION.
   Get the supermount patch for 2.6 kernel from the url http://umn.dl.sourceforge.net/sourceforge/supermount-ng/supermount-2.0.4-2.6.2.patch.gz and apply the patch.
   - #cp supermount2.0.42.6.6.patch /usr/src/linux/
   - #cd /usr/src/linux
   - #patch p1 ¡supermount2.0.42.6.6.patch

After successful patching, edit the /etc/fstab file, for example the line corresponding to the cdrom device should look like the following:
none /mnt/cdrom supermount dev=/dev/cdrom,fs=auto:iso9660,ro,,iocharset=iso88591 0 0
or you can try this line in the fstab file
/dev/hdc /mnt/cdrom auto umask=0,user,iocharset=iso88591,codepage=850,noauto,ro,exec 0 0
Configuring the Kernel

- #make menuconfig
  Do not select
  - "set version information on all module symbols" in loadable modules support
  - APM BIOS support in General setup
  - Kernel Hacking support.
  - Under the Console drivers section, do not select the frame buffer option rivafb

  Select
  - Adeos support
  - supermount support
  - Reisfrs
  - ext3 Journalising support
  - /dev in file systems.
  - under the "character devices", select all the options under subsection "direct rendering support"

  Issue the following commands to compile the kernel:
- #make
- #make modules_install
- #make install

Note: make bzImage step is not required for 2.6.7. The last step above copies the kernel image into /boot directory and will make an entry in /etc/lilo.conf. Just check lilo by running /sbin/lilo at the command line. 267adeos should appear. If not, make an entry in lilo.conf file manually. Following are the lines to be added:
image=/boot/vmlinuz2.6.7adeos label="rtai3.1" root=/dev/XYZ append="video=rvafb:off devfs=mount acpi=ht" readonly

4. After booting to the new kernel, RTAI should be installed. Following are the utilities required to have RTAI/RTAI-LAB installed properly.
   Get the file MesaLib5.0.2.tar.bz2 from web link http://sourceforge.net/project/showfiles.php?group_id=3
   Copy this file in to /usr/local/ directory and do the following:

   - #tar zxvf MesaLib5.0.2.tar.gz
   - #cd MesaLib5.0.2
   - #./configure prefix=/usr/local enablestatic
   - #make
   - #make install

   Get the cvs version efltk by doing the steps given below:

- #cd /usr/local/
- #cvs d:pserver:anonymous@cvs.sourceforge.net:/cvsroot/ede login press ENTER for password
- #cvs z3 d:pserver:anonymous@cvs.sourceforge.net:/cvsroot/ede co efltk
- #cd efltk
- #./build.gcc prefix=/usr/local enablexdbe enableopengl enable threads
- #make install
- #/sbin/ldconfig

Make a symbolic link of /tmp/efltk under the directory /usr/local/ after installing efltk, make sure that the file efltkconfig is there in both the directories /usr/local/bin and /usr/local/efltk/bin. If not, just make a copy from /usr/local/efltk.
#cp /usr/local/lib/efltk* /usr/lib/

A.2 Scilab-3.1 + RTAI-3.1 + RTAICodeGen

Download the latest stable version of Scilab3.1 from www.scilab.org, follow the procedure given below:

To have man pages working, utilities such as expat1.95.64mdk.i586.rpm, sablotron0.982mdk.i586.rpm and libsablotron00.982mdk.i586.rpm should be installed. These files are available at http://mirrors.usc.edu/pub/yumrepository/mandrake/9.2/i586/RPMS/ To have Scilab installed with 3D athena widgets enabled, Xaw3d utility should be installed first. This utility comes with the distribution CD of mandrake, just search for the Xaw3d in the install software utility.

1. Installing Scilab
   - # ./configure
   - # make all
     IF unable to setup xaw3d – SIMPLY AVOID xaw3d
   - # ./configure withoutxaw3d
   - # make all
2. INSTALLING RTAI
   - #make xconfig
     - Select number of cpus as 1.
     - Do not select SMP
     - Do not select the option “Build RTAI doc (PDF/HTML)
     - Do not select LTT
     - Select rtailab
   - #make
   - #make install
3. Setting up the RTAICodGen for Scilab/Scicos.
   Follow these steps to properly install all the Scilab/Scicos addons for RTAI-Lab:
   - Become superuser

- Go in the “macros” directory (/usr/src/rtai/rtailab/scilab/macros)
- modify in the file “Makefile” the line “SCILAB_DIR = /usr/scilab3.0” to fit your SCILAB installation.
- run “make install”
- # make user (do this again as a normal user) Before testing the code generated, add the rtai to your PATH,
- #export PATH=$PATH:/usr/realtime/bin

To have the code generated properly, one must have glibcstaticdevel2.3.214mdk.i586 utility.

4. INSTALLING RTAILAB

To have the rtailab GUI works properly, for the graphic cards from NVIDIA company, one must install the latest NVIDIA drivers from their website, http://download.nvidia.com/XFree86/Linuxx86/1.06106/NVIDIALinuxx861.06106pkg1.run.
Make sure that your kernel doesn’t have any frame buffering support, rivafb. Following are the steps for installation of drivers.

- #vi /etc/inittab/
  Change the run level from 5 to 3, save and exit. Get the file from the above nvidia web link NVIDIALinuxx861.06106pkg1.run
- #reboot
  On the console,
- #cd /usr/
- #sh NVIDIALinuxx861.06106pkg1.run
- #vi /etc/inittab/
  Change run level from 3 to 5, save and exit.
- #vi /etc/lilo.conf
  Add the video_rivafb:off option in your lilo file it should look like this. image=/vmlinuz label=“rtai” root=/dev/hda6 append=“video=rivafb:off”
- #lilo
- #vi /etc/X11/XF86Config4
  This opens the config file for X, do the following:
  - In the ‘Device’ section rename Driver “nv”(or Driver “vesa”) with Driver “nvidia” In the Module section, make sure you have:
  - Load “glx” You should also remove the following lines: Load “dri”
    Load “GLcore”
  - #reboot

You will be booted back to Xserver. Then, do the following in order to avoid any library clashes.
#cp /usr/lib/libGL.so.1.0.6106* /usr/lib/libGL.so.1.2*
WHILE RECONFIGURING RTAI FOR SOME REASON, IF ENCOUNTERED WITH THE ERROR SAYING “libtool: link: cannot find the library ‘/usr/X11R6/lib/libGL.la” DO THE FOLLOWING.
#ln -s /usr/lib/libGL.la /usr/X11R6/lib/libGL.la
Now, check the RTAILAB scope display.

A.3 Comedi/Comedilib

Get the CVS version of Comedi and Comedilib into the /usr/local/ from CVS server. Following are the commands to get it.

- #cvs d :pserver:anonymous@cvs.comedi.org:/var/cvs login
- #cvs d :pserver:anonymous@cvs.comedi.org:/var/cvs co comedi
- #cvs d :pserver:anonymous@cvs.comedi.org:/var/cvs co comedilib

One must have autoconf2.58 and automake1.7.8 libtools1.5,docbooksgml (Name:docbookdtd31sgmlVersion: 1.08mdkSize: 291 KB) utilities installed in their /usr directory. For the above utilities, use ./configure prefix=/usr option. Before installing COMEDI, make a directory named rtai under / lib/modules/2.4.24adeos/ and copy all the modules into this directory from /usr/realtime/modules then do the following:

- #cd /usr/local/comedi
- #./autogen.sh
- #./configure
- #make
- #make install
  Installing comedilib.
- #cd /usr/local/comedilib/
- #./autogen.sh
- #./configure
- #make
- #make install
  Go back to the rtai directory
- #cd /usr/src/rtai
- #make xconfig
  Select the comedi support and the serial support in the rtaiaddons section, save the configuration and issue the following commands.
- #make
- #make install
  Again copy all the modules in the /usr/realtime/modules/ into the /lib/modules/2.4.24adoes/ rtai/ directory. For loading all the modules required, one can use the following scripts:

```
cd /usr/realtime/modules insmod rtai\_hal.
insmod rtai\_lxrt.
insmod rtai\_sem.
insmod rtai\_mbx.
insmod rtai\_fifos.
insmod rtai\_math.
insmod rtai\_msg.
insmod rtai\_netrpc.
lsmod
```

After installing the comedi/comedilib, in order to calibrate your card, following are the modules to be loaded along with the above specified rtai modules.

Run this as the shell script, it will load the devices, required comedi/comedilib modules, loads required modules for configuring the card which is now being used, NI PCI 6040E. Furthermore, it prints the card info followed by the calibration result.

```
cd /usr/local/comedi/
make dev
cd /lib/modules/2.4.24adeos/comedi/
insmod comedi.ko
insmod kcomedilib.ko sleep 1s
insmod
rtai\_comedi.ko
insmod mite.ko

cd /usr/local/comedi/comedi/drivers/ insmod comedi\_fc.ko insmod
8255.ko insmod ni\_pcimio.

comedi\_config /dev/comedi0 ni\_pcimio
/usr/local/comedilib/demo/./info
/usr/local/bin/./comedi\_calibrate
```

A.4 Matlab/Simulink + RTAI

After installing the Linux version of Matlab 7.0, to integrate with RTAI following are the steps.

- Download the files “rtmain.c”,“rtai.tlc” and “rtai.tmf” from Roberto Buchers website http://a.die.supsi.ch/ bucher/.
- Create a directory named rtai under /Matlabhomedir/rtw/c/
- Copy the downloaded files into the now created rtai directory.
- Copy the setup.m file into rtai directory from /usr/src/rtai/rtai-lab/matlab/
- #cp /usr/src/rtai/rtai-lab/matlab/setup.m /Matlabhomedir/rtw/c/rtai/

From the Matlab session run “setup”, with your current working directory as /Matlabhomedir/rtw/c/rtai/. For testing the integration of Matlab with RTAI, following are the steps. Create a *.mdl example in Simulink (test.mdl supplied with RTAI distribution). Generate the code from the RTW menu using generate code only option with selecting rtai.tlc as the target language compiler.

A directory test_rtai is created in the working directory with all the required .c and .h files for rtai. Compile the code from the directory “test_rtai” by typing “make -f test.mk” at the console. Load the required rtai modules, using the script “loadrtai” in the RTAI-Lab tree. Run the real time code “./test [options]” (to see the list of options type “./test –usage”)

Run the RTAI-Lab (rtailab or xrtailab) to communicate with the real time code locally or remotely.

Appendix B

Scripts

B.1 Calculating Number of Moves

```
clear all; clc; i = 1; j = 0; md = 0; mf = 0; cd = 0; cf = 0;

flow =load('load flows '); dval = load('d-values');

k = zeros(400,37); z = zeros(400,23);
for j = 1:399
 for i = 2:37
      if flow(j,i) ~= flow(j+1,i)
         cf = cf+1;
      end
     k(j+1,i) = cf;
 end
end for y = 1:399
 for x = 2:23
      if dval(y,x) ~= dval(y+1,x)
         cd = cd+1;
      end
     z(y+1,x) = cd;
 end
end

mf = k(1:21,37); md = z(1:21,23);
fullm = mf + md;
t = 0:50:1000;
plot(t,fullm); save moves_fullpower fullm;
```

B.2 Calculating Agent Status Factor

```
clear all;
clc;
i = 1; j = 0; cd = 0; cf = 0;

a = load('d-values'); s.d = a(:,2:end); s.d.source = a(:,2);
s.d.node1 = a(:,3); s.d.node2 = a(:,4); s.d.node3 = a(:,5);
s.d.node4 = a(:,6); s.d.node5 = a(:,7); s.d.node6 = a(:,8);
s.d.node7 = a(:,9); s.d.node8 = a(:,10); s.d.node9 = a(:,11);
s.d.node10 = a(:,12); s.d.node11 = a(:,13); s.d.node12 = a(:,14);
s.d.node13 = a(:,15); s.d.node14 = a(:,16); s.d.node15 = a(:,17);
s.d.node16 = a(:,18); s.d.node17 = a(:,19); s.d.node18 = a(:,20);
s.d.node19 = a(:,21); s.d.node20 = a(:,22); s.d.target = a(:,23);
b = load('load flows '); s.flow = b(:,2:end); s.flow.fs1 = b(:,2);
s.flow.fs2 = b(:,3); s.flow.f13 = b(:,4); s.flow.f110 = b(:,5);
s.flow.f24 = b(:,6); s.flow.f29 = b(:,7); s.flow.f35 = b(:,8);
s.flow.f36 = b(:,9); s.flow.f45 = b(:,10); s.flow.f46 = b(:,11);
s.flow.f57 = b(:,12); s.flow.f511 = b(:,13);

s.flow.f513 = b(:,14); s.flow.f68 = b(:,15);
s.flow.f612 = b(:,16); s.flow.f614 = b(:,17);
s.flow.f715 = b(:,18); s.flow.f716 = b(:,19);
s.flow.f815 = b(:,20); s.flow.f816 = b(:,21);
s.flow.f1117 = b(:,22); s.flow.f1118 = b(:,23);
s.flow.f1217 = b(:,24); s.flow.f1218 = b(:,25);
s.flow.f1319 = b(:,26); s.flow.f1320 = b(:,27);
s.flow.f1419 = b(:,28); s.flow.f1420 = b(:,29);
s.flow.f9t = b(:,30); s.flow.f10t = b(:,31);
s.flow.f15t = b(:,32); s.flow.f16t = b(:,33);
s.flow.f17t = b(:,34); s.flow.f18t = b(:,35);
s.flow.f19t = b(:,36); s.flow.f20t = b(:,37);

%node 1
status = zeros(400,21); i1 = 0;

for i1 = 1:399
    if s.d.node1(i1) ~= s.d.node1(i1+1) | s.flow.fs1(i1) ~=
     s.flow.fs1(i1+1) & s.d.node1(i1)<22 | s.flow.f13(i1) ~=
     s.flow.f13(i1+1) & s.d.node1(i1)== 22 | s.flow.f110(i1) ~=
     s.flow.f110(i1+1) & s.d.node1(i1)==22
     status(i1,1) = 1;
    else status(i1,1) = 0;
    end
```

```
if s.d.node2(i1) ~= s.d.node2(i1+1) | s.flow.fs2(i1) ~=
 s.flow.fs2(i1+1) & s.d.node2(i1)<22 | s.flow.f24(i1) ~=
 s.flow.f24(i1+1) & s.d.node2(i1)==22 | s.flow.f29(i1) ~=
 s.flow.f29(i1+1)  & s.d.node2(i1)==22
  status(i1,2) = 1;
else status(i1,2) = 0;
end
if s.d.node3(i1) ~= s.d.node3(i1+1) | s.flow.f13(i1) ~=
 s.flow.f13(i1+1) & s.d.node3(i1)<22 | s.flow.f35(i1) ~=
 s.flow.f35(i1+1) & s.d.node3(i1)==22 | s.flow.f36(i1) ~=
 s.flow.f36(i1+1)  & s.d.node3(i1)==22
  status(i1,3) = 1;
else status(i1,3) = 0;
end
if s.d.node4(i1) ~= s.d.node4(i1+1) | s.flow.f24(i1) ~=
 s.flow.f24(i1+1) & s.d.node4(i1)<22 | s.flow.f45(i1) ~=
 s.flow.f45(i1+1) & s.d.node4(i1)==22 | s.flow.f46(i1) ~=
 s.flow.f46(i1+1)  & s.d.node4(i1)==22
  status(i1,4) = 1;
else status(i1,4) = 0;
end
if s.d.node5(i1) ~= s.d.node5(i1+1) | s.flow.f35(i1) ~=
 s.flow.f35(i1+1) & s.d.node5(i1)<22 | s.flow.f45(i1) ~=
 s.flow.f45(i1+1) & s.d.node5(i1)<22 | s.flow.f57(i1) ~=
 s.flow.f57(i1+1) & s.d.node5(i1)==22 | s.flow.f511(i1) ~=
 s.flow.f511(i1+1) & s.d.node5(i1)==22 | s.flow.f513(i1) ~=
 s.flow.f513(i1+1) & s.d.node5(i1)==22
  status(i1,5) = 1;
else status(i1,5) = 0;
end
if s.d.node6(i1) ~= s.d.node6(i1+1) | s.flow.f36(i1) ~=
 s.flow.f36(i1+1) & s.d.node6(i1)<22 | s.flow.f46(i1) ~=
 s.flow.f46(i1+1) & s.d.node6(i1)<22 | s.flow.f68(i1) ~=
 s.flow.f68(i1+1) & s.d.node6(i1)==22 | s.flow.f612(i1) ~=
 s.flow.f612(i1+1) & s.d.node6(i1)==22 | s.flow.f614(i1) ~=
 s.flow.f614(i1+1) & s.d.node6(i1)==22
  status(i1,6) = 1;
else status(i1,6) = 0;
end
if s.d.node7(i1) ~= s.d.node7(i1+1) | s.flow.f57(i1) ~=
 s.flow.f57(i1+1) & s.d.node7(i1)<22 | s.flow.f715(i1) ~=
 s.flow.f715(i1+1) & s.d.node7(i1)==22 | s.flow.f716(i1) ~=
 s.flow.f716(i1+1)  & s.d.node7(i1)==22
  status(i1,7) = 1;
```

```
else status(i1,7) = 0;
end
if s.d.node8(i1) ~= s.d.node8(i1+1) | s.flow.f68(i1) ~=
 s.flow.f68(i1+1) & s.d.node8(i1)<22 | s.flow.f815(i1) ~=
 s.flow.f815(i1+1) & s.d.node8(i1)==22 | s.flow.f816(i1) ~=
 s.flow.f816(i1+1)  & s.d.node8(i1)==22
 status(i1,8) = 1;
else status(i1,8) = 0;
end
if s.d.node11(i1) ~= s.d.node11(i1+1) | s.flow.f511(i1) ~=
 s.flow.f511(i1+1) & s.d.node11(i1)<22 | s.flow.f1117(i1) ~=
 s.flow.f1117(i1+1) & s.d.node11(i1)==22 | s.flow.f1118(i1) ~=
 s.flow.f1118(i1+1)  & s.d.node11(i1)==22
 status(i1,11) = 1;
else status(i1,11) = 0;
end
if s.d.node12(i1) ~= s.d.node12(i1+1) | s.flow.f612(i1) ~=
 s.flow.f612(i1+1) & s.d.node12(i1)<22 | s.flow.f1217(i1) ~=
 s.flow.f1217(i1+1) & s.d.node12(i1)==22 | s.flow.f1218(i1) ~=
 s.flow.f1218(i1+1)  & s.d.node12(i1)==22
 status(i1,12) = 1;
else status(i1,12) = 0;
end
if s.d.node13(i1) ~= s.d.node13(i1+1) | s.flow.f513(i1) ~=
 s.flow.f513(i1+1) & s.d.node13(i1)<22 | s.flow.f1319(i1) ~=
 s.flow.f1319(i1+1) & s.d.node13(i1)==22 | s.flow.f1320(i1) ~=
 s.flow.f1320(i1+1)  & s.d.node13(i1)==22
 status(i1,13) = 1;
else status(i1,13) = 0;
end
if s.d.node14(i1) ~= s.d.node14(i1+1) | s.flow.f614(i1) ~=
 s.flow.f614(i1+1) & s.d.node14(i1)<22 | s.flow.f1419(i1) ~=
 s.flow.f1419(i1+1) & s.d.node14(i1)==22 | s.flow.f1420(i1) ~=
 s.flow.f1420(i1+1)  & s.d.node14(i1)==22
 status(i1,14) = 1;
else status(i1,14) = 0;
end
if s.d.node9(i1) ~= s.d.node9(i1+1) | s.flow.f29(i1) ~=
 s.flow.f29(i1+1) & s.d.node9(i1)<22 | s.flow.f9t(i1) ~=
 s.flow.f9t(i1+1) & s.d.node9(i1)==22
 status(i1,9) = 1;
else status(i1,9) = 0;
end
if s.d.node10(i1) ~= s.d.node10(i1+1) | s.flow.f110(i1) ~=
     s.flow.f110(i1+1) & s.d.node10(i1)<22 | s.flow.f10t(i1) ~=
     s.flow.f10t(i1+1) & s.d.node10(i1)==22
     status(i1,10) = 1;
   else status(i1,10) = 0;
   end
   if s.d.node15(i1) ~= s.d.node15(i1+1) | s.flow.f715(i1) ~=
     s.flow.f715(i1+1) & s.d.node15(i1)<22 | s.flow.f815(i1) ~=
     s.flow.f815(i1+1) & s.d.node15(i1)<22 | s.flow.f15t(i1) ~=
     s.flow.f15t(i1+1) & s.d.node15(i1)==22
     status(i1,15) = 1;
   else status(i1,15) = 0;
   end
   if s.d.node16(i1) ~= s.d.node16(i1+1) | s.flow.f716(i1) ~=
     s.flow.f716(i1+1) & s.d.node16(i1)<22 | s.flow.f816(i1) ~=
     s.flow.f816(i1+1) & s.d.node16(i1)<22 | s.flow.f16t(i1) ~=
     s.flow.f16t(i1+1) & s.d.node16(i1)==22
     status(i1,16) = 1;
   else status(i1,16) = 0;
   end
   if s.d.node17(i1) ~= s.d.node17(i1+1) | s.flow.f1117(i1) ~=
     s.flow.f1117(i1+1) & s.d.node17(i1)<22 | s.flow.f1217(i1) ~=
     s.flow.f1217(i1+1) & s.d.node17(i1)<22 | s.flow.f17t(i1) ~=
     s.flow.f17t(i1+1) & s.d.node17(i1)==22
     status(i1,17) = 1;
   else status(i1,17) = 0;
   end
   if s.d.node18(i1) ~= s.d.node18(i1+1) | s.flow.f1118(i1) ~=
     s.flow.f1118(i1+1) & s.d.node18(i1)<22 | s.flow.f1218(i1) ~=
     s.flow.f1218(i1+1) & s.d.node18(i1)<22 | s.flow.f18t(i1) ~=
     s.flow.f18t(i1+1) & s.d.node18(i1)==22
     status(i1,18) = 1;
   else status(i1,18) = 0;
   end
   if s.d.node19(i1) ~= s.d.node19(i1+1) | s.flow.f1319(i1) ~=
     s.flow.f1319(i1+1) & s.d.node19(i1)<22 | s.flow.f1419(i1) ~=
     s.flow.f1419(i1+1) & s.d.node19(i1)<22 | s.flow.f19t(i1) ~=
     s.flow.f19t(i1+1) & s.d.node19(i1)==22
     status(i1,19) = 1;
   else status(i1,19) = 0;
   end
   if s.d.node20(i1) ~= s.d.node20(i1+1) | s.flow.f1320(i1) ~=
     s.flow.f1320(i1+1) & s.d.node20(i1)<22 | s.flow.f1420(i1) ~=
     s.flow.f1420(i1+1) & s.d.node20(i1)<22 | s.flow.f20t(i1) ~=
     s.flow.f20t(i1+1) & s.d.node20(i1)==22
```

```
         status(i1,20) = 1;
        else status(i1,20) = 0;
        end
        if s.d.target(i1) ~= s.d.target(i1+1) | s.flow.f9t(i1) ~=
         s.flow.f9t(i1+1) & s.d.target(i1)<22 | s.flow.f15t(i1) ~=
         s.flow.f15t(i1+1) & s.d.target(i1)<22 | s.flow.f16t(i1) ~=
         s.flow.f16t(i1+1) & s.d.target(i1)<22 | s.flow.f17t(i1) ~=
         s.flow.f17t(i1+1) & s.d.target(i1)<22 | s.flow.f18t(i1) ~=
         s.flow.f18t(i1+1) & s.d.target(i1)<22 | s.flow.f19t(i1) ~=
         s.flow.f19t(i1+1) & s.d.target(i1)<22 | s.flow.f20t(i1) ~=
         s.flow.f20t(i1+1) & s.d.target(i1)<22 | s.flow.f10t(i1) ~=
         s.flow.f10t(i1+1) & s.d.target(i1)<22
         status(i1,21) = 1;
        else status(i1,21) = 0;
        end
end
    statusa = sum(status');
    stat = statusa';
    factorfull = 1/21 * stat(1:21,1);
    t = 0:50:1000;
    plot(t,factorfull);
    save factor_fullpower factorfull;
```

B.3 Matlab Initialization Script for Simulink Model

```
% Initializing flows , capacities and d-values
clear; clc; f1 = 0;
%Capacities
cs1  = 580; % capacity of the edge from source to node1
cs2  = 580; % capacity of the edge from source to node2
c13  = 180; % capacity of the edge from node1 to node3
c110 = 400; % capacity of the edge from node1 to node10
c24  = 180; % capacity of the edge from node1 to node4
c29  = 400; % capacity of the edge from node3 to node9
c35  = 180; % capacity of the edge from node3 to node5
c36  = 180; % capacity of the edge from node3 to node6
c45  = 180; % capacity of the edge from node4 to node5
c46  = 180; % capacity of the edge from node4 to node6
c57  =  60; % capacity of the edge from node5 to node7
c511 =  60; % capacity of the edge from node5 to node11
c513 =  60; % capacity of the edge from node5 to node13
c68  =  60; % capacity of the edge from node6 to node8
c612 =  60; % capacity of the edge from node6 to node12
```

```
c614 =  60; % capacity of the edge from node6 to node14
c715 =  60; % capacity of the edge from node7 to node15
c716 =  60; % capacity of the edge from node7 to node16
c815 =  60; % capacity of the edge from node8 to node15
c816 =  60; % capacity of the edge from node8 to node16
c1117 = 60; % capacity of the edge from node11 to node17
c1118 = 60; % capacity of the edge from node11 to node18
c1217 = 60; % capacity of the edge from node12 to node17
c1218 = 60; % capacity of the edge from node12 to node18
c1319 = 60; % capacity of the edge from node13 to node19
c1320 = 60; % capacity of the edge from node13 to node20
c1419 = 60; % capacity of the edge from node14 to node19
c1420 = 60; % capacity of the edge from node14 to node20
c9t  = 400; % capacity of the edge from node9 to target
c10t = 400; % capacity of the edge from node10 to target
c15t =  30; % capacity of the edge from node15 to target
c16t =  30; % capacity of the edge from node16 to target
c17t =  30; % capacity of the edge from node17 to target
c18t =  30; % capacity of the edge from node18 to target
c19t =  30; % capacity of the edge from node19 to target
c20t =  30; % capacity of the edge from node20 to target
ctt = 100000; % capacity of the outgoing edge of the target

%Flows
fs1  = f1*40; % flow of the edge from sourfe to node1
fs2  = f1*40; % flow of the edge from sourfe to node2
f13  = f1*10; % flow of the edge from node1 to node3
f110 = f1*10; % flow of the edge from node1 to node10
f24  = f1*10; % flow of the edge from node1 to node4
f29  = f1*10; % flow of the edge from node3 to node9
f35  = f1*10; % flow of the edge from node3 to node5
f36  = f1*10; % flow of the edge from node3 to node6
f45  = f1*10; % flow of the edge from node4 to node5
f46  = f1*10; % flow of the edge from node4 to node6
f57  = f1*10; % flow of the edge from node5 to node7
f511 = f1*10; % flow of the edge from node5 to node11
f513 = f1*10; % flow of the edge from node5 to node13
f68  = f1*10; % flow of the edge from node6 to node8
f612 = f1*10; % flow of the edge from node6 to node12
f614 = f1*10; % flow of the edge from node6 to node14
f715 = f1*10; % flow of the edge from node7 to node15
f716 = f1*10; % flow of the edge from node7 to node16
f815 = f1*10; % flow of the edge from node8 to node15
f816 = f1*10; % flow of the edge from node8 to node16
```

```
f1117 = f1*10; % flow of the edge from node11 to node17
f1118 = f1*10; % flow of the edge from node11 to node18
f1217 = f1*10; % flow of the edge from node12 to node17
f1218 = f1*10; % flow of the edge from node12 to node18
f1319 = f1*10; % flow of the edge from node13 to node19
f1320 = f1*10; % flow of the edge from node13 to node20
f1419 = f1*10; % flow of the edge from node14 to node19
f1420 = f1*10; % flow of the edge from node14 to node20
f9t   = f1*10; % flow of the edge from node9 to target
f10t  = f1*10; % flow of the edge from node10 to target
f15t =  f1*10; % flow of the edge from node15 to target
f16t =  f1*10; % flow of the edge from node16 to target
f17t =  f1*10; % flow of the edge from node17 to target
f18t =  f1*10; % flow of the edge from node18 to target
f19t =  f1*10; % flow of the edge from node19 to target
f20t =  f1*10; % flow of the edge from node20 to target

%D-values
ds = 0; % d-value of source
d1 = 0; % d-value of node1
d2 = 0; % d-value of node2
d3 = 0; % d-value of node3
d4 = 0; % d-value of node4
d5 = 0; % d-value of node5
d6 = 0; % d-value of node6
d7 = 0; % d-value of node7
d8 = 0; % d-value of node8
d9 = 0; % d-value of node9
d10 = 0; % d-value of node10
d11 = 0; % d-value of node11
d12 = 0; % d-value of node12
d13 = 0; % d-value of node13
d14 = 0; % d-value of node14
d15 = 0; % d-value of node15
d16 = 0; % d-value of node16
d17 = 0; % d-value of node17
d18 = 0; % d-value of node18
d19 = 0; % d-value of node19
d20 = 0; % d-value of node20
dt  = 0; % d-value of target

%Priorities
p9  = 1; p10 = 1; p15 = 1; p16 = 0; p17 = 1; p18 = 0; p19 = 1; p20
= 0; p7 = max(p15,p16); p8 = max(p15,p16); p11 = max(p17,p18); p12
= max(p17,p18); p13 = max(p19,p20); p14 = max(p19,p20);

%N - number of nodes
n = 22;

% lab-model parameters
% represents a simplified ESPS by generic transfer function blocks

% discrete simulation time step
dT = 1e-3;

% gas turbine
gtd.energy = gas.te*[1;1]; % available energy
gtd.denergy = [-1;-1]; gtd.pi.kp = [2;1]; gtd.pi.ki = [1;1];
gtd.pi.limit = [1.02;,1.02]; gtd.pt2.kp = [.98;.98]; gtd.pt2.d =
[0.9;0.9]; gtd.pt2.w0 = [4;4]; for ii=1:length(gtd.pi.kp),
    gtd.pt2.num{ii} = [gtd.pt2.kp(ii)];
    gtd.pt2.den{ii} = [1/gtd.pt2.w0(ii)^2 ..
      2*gtd.pt2.d(ii)/gtd.pt2.w0(ii) 1];
    [numd,dend] = ctf2dtf(gtd.pt2.num{ii},gtd.pt2.den{ii},dT);
    gtd.pt2.numd{ii,:} = {numd};
    gtd.pt2.dend{ii,:} = {dend};
end

% exciter
exc.pi.kp = [4;1]; exc.pi.ki = [2;1]; exc.pi.limit = [1.02;,1.02];
exc.pt1.kp = [1;1]; exc.pt1.t1 = [.1;.1]; exc.p.level = [.1;.1];
exc.ps = [.05;.05];

% generator
g.H = [1;1]; g.pt1.kp = [2;2]; g.pt1.t1 = [.2;.2]; g.vdroop =
[0.1;0.1];

% propulsion
pr.p = [.55;.55]; pr.t1 = [.1;.1]; pr.vT1 = [.01;.01]; pr.vmin =
[.8;.8];

% loads
% type 1
lt.a.vmin1 = [.7]; lt.a.vmin2 = [.7]; lt.a.p = [.1]; lt.a.t1 =
[.1]; lt.a.vT1_1 = [.01]; lt.a.vT1_2 = [.01];
```

B.4 S-functions Used in PCAN Simulink Blocks

```
/* S-function for PCAN_IN block. */

/* COPYRIGHT (C) November 2005  APERC, WVU
(karl.schoder@mail.wvu.edu)
*/
#define S_FUNCTION_NAME sfun_pcan_in2
#define S_FUNCTION_LEVEL 2

#ifdef MATLAB_MEX_FILE
#include "mex.h"      /* needed for declaration of mexErrMsgTxt */
#endif

#ifndef MATLAB_MEX_FILE
#include <stdio.h>
#include <stdlib.h>
#include <libpcan.h>
#include <fcntl.h>    // O_RDWR #endif
#include "simstruc.h"

#define NUMBER_OF_ARGS          (3)
#define CAN_ID_ARG       ssGetSFcnParam(S,0)
#define BAUDRATE_ARG     ssGetSFcnParam(S,1)
#define SAMP_TIME_ARG    ssGetSFcnParam(S,2)
#define NO_I_WORKS              (1)
#define CAN_ID_I_IND            (0)

static void *h = NULL;
static int h_cnt = 0;
int CANMsg[12] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};


static void mdlInitializeSizes(SimStruct *S) {
    ssSetNumSFcnParams(S, NUMBER_OF_ARGS);
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
        return; /* Parameter mismatch will be reported by Simulink */
    }
    ssSetNumOutputPorts(S,5);
    ssSetOutputPortWidth(S, 0,1); /* status */
    ssSetOutputPortWidth(S, 1,1); /* ID     */
    ssSetOutputPortWidth(S, 2,1); /* type   */
    ssSetOutputPortWidth(S, 3,1); /* length */
    ssSetOutputPortWidth(S, 4,8); /* data   */
    ssSetNumContStates(S, 0);
    ssSetNumDiscStates(S, 0);
    ssSetNumSampleTimes(S, 1);
    ssSetNumIWork(S,NO_I_WORKS);
}


static void mdlInitializeSampleTimes(SimStruct *S) {
    ssSetSampleTime(S, 0, mxGetPr(SAMP_TIME_ARG)[0]);
    ssSetOffsetTime(S, 0, 0.0);
}


#define MDL_START
static void mdlStart(SimStruct *S) {
#ifndef MATLAB_MEX_FILE
    int_T can_id   = mxGetPr(CAN_ID_ARG)[0];
    int_T baudrate = mxGetPr(BAUDRATE_ARG)[0];
    ssSetIWorkValue(S,CAN_ID_I_IND, can_id);
    __u16 wBTR0BTR1;
    int nExtended = CAN_INIT_TYPE_ST;

    switch (baudrate)
    {
        case 10 :
{
    wBTR0BTR1 = CAN_BAUD_10K;
    break;
        }
        case 20 :
{
    wBTR0BTR1 = CAN_BAUD_20K;
    break;
        }
        case 50 :
{
    wBTR0BTR1 = CAN_BAUD_50K;
    break;
        }
        case 100 :
{
    wBTR0BTR1 = CAN_BAUD_100K;
    break;
        }
        case 125 :
{
    wBTR0BTR1 = CAN_BAUD_125K;
    break;
        }
        case 250 :
{
    wBTR0BTR1 = CAN_BAUD_250K;
    break;
        }
        case 500 :
{
    wBTR0BTR1 = CAN_BAUD_500K;
    break;
        }
        case 1000 :
{
    wBTR0BTR1 = CAN_BAUD_1M;
    break;
        }
        default :
{
    wBTR0BTR1 = CAN_BAUD_100K;
        }
    }

    if(h==NULL){

        int erno=0;
        h = LINUX_CAN_Open("/dev/pcan0", O_RDWR);
        if (h){
            CAN_Status(h);
            if (wBTR0BTR1){
                erno = CAN_Init(h, wBTR0BTR1, nExtended);
                if (erno)
                {
                    perror("request: CAN_Init()");
                    exit(1);
                }
            }
        }
        else
        {
            erno = nGetLastError();
            perror("request: CAN_Open()");
            exit(1);
        }
    }
    h_cnt++;
#endif
}


static void mdlOutputs(SimStruct *S, int_T tid) {
#ifndef MATLAB_MEX_FILE
    real_T    *status  = ssGetOutputPortRealSignal(S,0); /* status */
    real_T    *msgid   = ssGetOutputPortRealSignal(S,1); /* ID     */
    real_T    *msgtype = ssGetOutputPortRealSignal(S,2); /* type   */
    real_T    *msglen  = ssGetOutputPortRealSignal(S,3); /* length */
    real_T    *msgdata = ssGetOutputPortRealSignal(S,4); /* data   */

    TPCANMsg  MyMsg;
    int       errno = 0;
    int       i;

    if(errno = CAN_Read(h, &MyMsg)){
        perror("request: CAN_Read()");
        exit(1);
    }
    else{
        CANMsg[1] = MyMsg.ID;
        CANMsg[2] = MyMsg.MSGTYPE;
        CANMsg[3] = MyMsg.LEN;
        for(i=0;i<MyMsg.LEN;i++){
                CANMsg[4+i] = MyMsg.DATA[i];
    }
    for(i=MyMsg.LEN;i<8;i++){
        CANMsg[4+i] = 0;
    }
    for(i=0;i<8;i++){
        msgdata[i] = CANMsg[4+i];
    }
    }
    CANMsg[0] = CAN_Status(h);

    status[0]  = CANMsg[0];
    msgid[0]   = CANMsg[1];
    msgtype[0] = CANMsg[2];
    msglen[0]   = CANMsg[3];
#endif
}


static void mdlTerminate(SimStruct *S)
{
#ifndef MATLAB_MEX_FILE
    h_cnt--;
    if(h_cnt==0) CAN_Close(h);
#endif
}

#ifdef  MATLAB_MEX_FILE     /* MEX-file? */
#include "simulink.c"       /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"        /* Code generation registration function */
#endif
```

```
/* S-function for PCAN_OUT block. */

/* COPYRIGHT (C) November 2005 APERC, WVU
(karl.schoder@mail.wvu.edu)
*/

#define S_FUNCTION_NAME  sfun_pcan_out2 #define S_FUNCTION_LEVEL 2

#ifdef MATLAB_MEX_FILE #include "mex.h"      /* needed for
declaration of mexErrMsgTxt */ #endif

#ifndef MATLAB_MEX_FILE
#include <stdio.h>
#include <stdlib.h>
#include <libpcan.h>
#include <fcntl.h>    // O_RDWR
#include <sys/ioctl.h>
#endif

#include "simstruc.h"

#define NUMBER_OF_ARGS          (3)
#define CAN_ID_ARG        ssGetSFcnParam(S,0)
#define BAUDRATE_ARG      ssGetSFcnParam(S,1)
#define SAMPLETIME_ARG    ssGetSFcnParam(S,2)
```

```
#define NO_I_WORKS              (1)
#define CAN_ID_I_IND            (0)
#define BYTE                   __u8

static int h_cnt = 0;
static void *h = NULL;


static void mdlInitializeSizes(SimStruct *S) {
    ssSetNumSFcnParams(S, NUMBER_OF_ARGS);
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
        return; /* Parameter mismatch will be reported by Simulink */
    }
    if (!ssSetNumInputPorts(S, 3)) return;
    ssSetInputPortWidth(S, 0, 1);
    ssSetInputPortWidth(S, 1, 1);
    ssSetInputPortWidth(S, 2, 8);
    ssSetInputPortDirectFeedThrough(S, 0, 1);
    ssSetInputPortDirectFeedThrough(S, 1, 1);
    ssSetInputPortDirectFeedThrough(S, 2, 1);

    if (!ssSetNumOutputPorts(S, 0)) return;

    ssSetNumContStates(S, 0);
    ssSetNumDiscStates(S, 0);
    ssSetNumSampleTimes(S, 1);
    ssSetNumIWork(S,NO_I_WORKS);
}


static void mdlInitializeSampleTimes(SimStruct *S) {
    real_T sampletime   = mxGetPr(SAMPLETIME_ARG)[0];
    ssSetSampleTime(S, 0, sampletime);
    ssSetOffsetTime(S, 0, 0.0);
}


#define MDL_START
static void mdlStart(SimStruct *S) {
#ifndef MATLAB_MEX_FILE
    int_T can_id   = mxGetPr(CAN_ID_ARG)[0];
    int_T baudrate = mxGetPr(BAUDRATE_ARG)[0];
    ssSetIWorkValue(S,CAN_ID_I_IND, can_id);
    int nType;
    __u16 wBTR0BTR1;
    int    nExtended = CAN_INIT_TYPE_ST;

    nType = HW_USB;

    switch (baudrate)
{
        case 10 :
{
    wBTR0BTR1 = CAN_BAUD_10K;
    break;
        }
        case 20 :
{
    wBTR0BTR1 = CAN_BAUD_20K;
    break;
        }
        case 50 :
{
    wBTR0BTR1 = CAN_BAUD_50K;
    break;
        }
        case 100 :
{
    wBTR0BTR1 = CAN_BAUD_100K;
    break;
        }
        case 125 :
{
    wBTR0BTR1 = CAN_BAUD_125K;
    break;
        }
        case 250 :
{
    wBTR0BTR1 = CAN_BAUD_250K;
    break;
        }
        case 500 :
{
    wBTR0BTR1 = CAN_BAUD_500K;
    break;
        }
        case 1000 :
```

```
{
    wBTR0BTR1 = CAN_BAUD_1M;
    break;
        }
        default :
{
    wBTR0BTR1 = CAN_BAUD_100K;
        }
    }

    if(h==NULL){

        int erno=0;
        h = LINUX_CAN_Open("/dev/pcan0", O_RDWR);
        if (h){
            CAN_Status(h);
            if (wBTR0BTR1){
                erno = CAN_Init(h, wBTR0BTR1, nExtended);
                if (erno)
                {
                    perror("request: CAN_Init()");
                    exit(1);
                }
            }
        }
        else
        {
            erno = nGetLastError();
            perror("request: CAN_Open()");
            exit(1);
        }
    h_cnt++;
    }
#endif
}


static void mdlOutputs(SimStruct *S, int_T tid) {
#ifndef MATLAB_MEX_FILE
    InputRealPtrsType MSGid   = ssGetInputPortRealSignalPtrs(S,0);
    InputRealPtrsType MSGlen  = ssGetInputPortRealSignalPtrs(S,1);
    InputRealPtrsType MSGdata = ssGetInputPortRealSignalPtrs(S,2);
    int_T             id   = ssGetIWorkValue(S,CAN_ID_I_IND);
    double            u;
    int_T                ii;
    int_T                erno = 0;
    TPCANMsg             myMsg;
    myMsg.ID        = (long int) *MSGid[0];
    myMsg.MSGTYPE = MSGTYPE_STANDARD;
    myMsg.LEN       = *MSGlen[0];
    for(ii=0;ii<myMsg.LEN;ii++){
        myMsg.DATA[ii] = *MSGdata[ii];
    }

    int_T baudrate = mxGetPr(BAUDRATE_ARG)[0];
    ssSetIWorkValue(S,CAN_ID_I_IND,can_id);

    int nType;
    __u16 wBTR0BTR1;
    int    nExtended = CAN_INIT_TYPE_ST;

    nType = HW_USB;

    switch (baudrate)
{
        case 10 :
{
    wBTR0BTR1 = CAN_BAUD_10K;
    break;
        }
        case 20 :
{
    wBTR0BTR1 = CAN_BAUD_20K;
    break;
        }
        case 50 :
{
    wBTR0BTR1 = CAN_BAUD_50K;
    break;
        }
        case 100 :
{
    wBTR0BTR1 = CAN_BAUD_100K;
    break;
        }
        case 125 :
{
    wBTR0BTR1 = CAN_BAUD_125K;
    break;
        }
        case 250 :
{
    wBTR0BTR1 = CAN_BAUD_250K;
    break;
        }
        case 500 :
{
    wBTR0BTR1 = CAN_BAUD_500K;
    break;
        }
        case 1000 :
{
    wBTR0BTR1 = CAN_BAUD_1M;
    break;
        }
        default :
{
    wBTR0BTR1 = CAN_BAUD_100K;
        }
    }

    if(h==NULL){

        int erno=0;
        h = LINUX_CAN_Open("/dev/pcan0", O_RDWR);
        if (h){
            CAN_Status(h);
            if (wBTR0BTR1){
                erno = CAN_Init(h, wBTR0BTR1, nExtended);
                if (erno)
                {
                    perror("request: CAN_Init()");
                    exit(1);
                }
            }
        }
        else
        {
            erno = nGetLastError();
            perror("request: CAN_Open()");
            exit(1);
        }
    }

    if ((erno = CAN_Write(h, &myMsg)))
    {
        perror("request: CAN_Write()");
        exit(1);
    }
    return; // erno;
#endif
}

static void mdlTerminate(SimStruct *S) {
#ifndef MATLAB_MEX_FILE
    int_T         id   = ssGetIWorkValue(S,CAN_ID_I_IND);

    h_cnt--;
    if(h_cnt==0) CAN_Close(h);
#endif
}

#ifdef  MATLAB_MEX_FILE     /* MEX-file? */
#include "simulink.c"       /* MEX-file interface mechanism */
#else #include "cg_sfun.h" /* Code generation registration function */
#endif
```