FREQUENTLY ASKED QUESTIONS.................................................................................................... 59

APPENDIX A: INSTALLING REDHAT LINUX 6.1............................................................................ 63
  Preliminaries ................................................................................................................................. 63
  GRAPHICAL MODE ......................................................................................................................... 63
    Partition and Format Hard Disk ............................................................................................... 63
    Select Components to Install ................................................................................................. 63
    Hardware Identification ........................................................................................................... 63
    LILO Configuration .................................................................................................................. 64
    Network Configuration ............................................................................................................ 64
    User Account Creation .......................................................................................................... 64
    Packages for Installation ....................................................................................................... 64
    Video Card/Monitor Setup ..................................................................................................... 65
  TEXT MODE .................................................................................................................................. 66
    Partition and Format Hard Disk ............................................................................................... 66
    LILO Configuration .................................................................................................................. 66
    Network Configuration ............................................................................................................ 67
    Hardware Identification ........................................................................................................... 67
    User Account Creation .......................................................................................................... 67
    Package Group Selection ....................................................................................................... 67
  NETWORK PROBLEMS ................................................................................................................ 69
    Hardware Check ....................................................................................................................... 69
    Configuration Check ............................................................................................................... 69
    Driver Check ............................................................................................................................ 69
    Rebuild the Kernel ................................................................................................................. 69
    Reconfiguring the LILO ............................................................................................................ 70

APPENDIX B: INSTALLING PRE-PATCHED RT-LINUX (V2.0) .................................................... 71
  Preliminaries ................................................................................................................................. 71
  Extracting the Linux 2.2.13 Kernel pre-patched with RT-Linux 2.0 ............................................ 71
  Configuring and Building New Kernel Image .............................................................................. 71
  Configuring the LILO .................................................................................................................. 72
  Installing RT-Linux Modules ..................................................................................................... 72
  CONFIGURING SHARED MEMORY .......................................................................................... 73
    Shared Memory Pool Configuration ........................................................................................ 73
  RTLT Licenses ............................................................................................................................. 75

APPENDIX C: MATLAB 5.3 - RED HAT LINUX 6.1 COMPATIBILITY ISSUES ....................... 77

APPENDIX D: RTLT SUPPORTED I/O BOARDS ........................................................................... 80
  MultiQ I/O Board ....................................................................................................................... 80
  ServoToGo I/O Board ................................................................................................................ 80
  Mini PMAC Board ....................................................................................................................... 80
  ATI Force/Torque Sensor System .............................................................................................. 81
  PCL812 Board ............................................................................................................................. 81

APPENDIX E: LOADING I/O BOARD SERVER MODULES........................................................ 83
  Loading the MultiQ-2 Server Module ....................................................................................... 83
  Loading the MultiQ-3 Server Module ....................................................................................... 84
  Loading the ServoToGo Server Module .................................................................................. 84
  Loading the ATI-FT Server Module ....................................................................................... 85
  Loading the PMAC Server Module ......................................................................................... 85
  Loading the PCL812 Server Module ....................................................................................... 86

APPENDIX F: WINDOWS 98, QNX AND LINUX MULTIPLE BOOT INSTALLATION ............ 88
Introduction

MATLAB is a software environment that allows a user to easily integrate computation and visualization tasks. The primary advantage of MATLAB lies in that problems and solutions are expressed in familiar mathematical notations. Due to fact that numerous toolboxes and other software packages that have been developed for MATLAB, it is quickly becoming the tool of choice for computation, algorithm development, modeling, simulation, data analysis, visualization, engineering graphics, and application development (including graphical user interface development).

Simulink is a software package for modeling, simulating, and analyzing dynamic systems in the MATLAB environment. Simulink supports both linear and nonlinear systems that are modeled in continuous time and discrete time. The graphical user interface (GUI) in Simulink is used to create block diagram models. Various block-set libraries provide pre-configured blocks and connectors that can be incorporated into a model by simple drag and drop operations. Different types of sources and sinks in these libraries allow the user to apply different inputs and analyze the output data. Another salient feature of Simulink is that the models are hierarchical. The user can view the system model at the high-level and then progressively access lower-level blocks to examine more details. After the model is defined, the user can simulate the response of the system by selecting appropriate integration methods. Simulink also allows for on-line parameter tuning in order to assess the change in system response and perform a “what-if” exploration. Scopes and other display blocks allow the user to view the simulation results while the simulation is still running. In addition, the simulation results can be exported to the MATLAB workspace for post-processing and visualization.

Real-Time Workshop (RTW) is an automatic C language code generator for Simulink. RTW produces C code directly from the Simulink models and automatically constructs a file that can be executed in real time in various environments. In conjunction with RTW, Simulink provides a powerful front-end for developing executable code with minimal amount of computer skills. That is, the block diagram interface of Simulink coupled to the RTW code generator allows the user to concentrate on the modeling and control issues as opposed to programming issue.

Real-Time Linux Target (RTLT) is a software package developed by Quality Real Time Systems that gives the user the ability to implement a Simulink block diagram on a standard Intel PC in hard real-time. Specifically, RTLT is a set of source files, device driver libraries, a template makefile, and an MEX-file interface that uses RTW to automatically generate C code from a user defined Simulink block diagram. The C code is first generated and compiled on a PC running RT-Linux. A target for running the generated code is then built on the same PC. During the execution of a Simulink block diagram, RTLT captures sampled data from one or more input channels using standard I/O boards (e.g., A/D channels, digital lines, and encoder lines, etc.). RTLT then provides the data to the block diagram model. The Simulink block diagram model then processes the data accordingly. RTLT then outputs the processed data via one or more output channels (e.g., D/A channels). A custom Simulink block library and four different hardware I/O board drivers are also provided. The user can also observe the behavior of any signal during or after the real-time run via the Simulink Scope blocks. If the user builds the
Simulink code in the external mode, the user can perform on-line parameter tuning during real-time execution.

**Typographical Conventions**

The following typographical conventions are used in this guide.

- **Boldface** indicates keys, menu names, page names in dialog boxes, and GUI controls.
- **Italics** indicate file names.
- **Monospace** type indicates commands and arguments typed at the Linux or MATLAB command-line prompts.
- "Within double quotes" indicates the text the user enters in fields of various dialog boxes.
- The `#` sign at the beginning of a command line indicates that the command is to be typed at the Linux prompt, unless specified otherwise.
- The `>>` sign at the beginning of a command line indicates that the command is to be typed at the MATLAB prompt, unless specified otherwise.
Getting Started

This section discusses how to install and configure Real-Time Linux Target. In order to install RTLT, one must be the super-user/root with system administration privileges.

System Requirements

The hardware required for implementing control algorithms using RTLT includes:

A Host Computer (i.e., a Pentium computer with 16 MB+ main memory), and

Data acquisition boards (i.e., the MultiQ board, the ServoToGo board, the Mini PMAC board, the Advantech PCL812 board, and the ATI F/T Sensor System). A detailed description of the features of each board is provided in Appendix G.

A dongle/hardware lock provided by QRTS upon purchase of RTLT plugs into the parallel port of the host computer. RTLT will not function correctly, if this dongle is not plugged in to the parallel port.

The software required for implementing control algorithms using RTLT includes:

The Linux operating system, Red Hat Linux Version 6.1,

The real time Linux patch RT-Linux 2.0, which is included in the RTLT package,

MATLAB Version 5.3, Release 11, for Linux,

Simulink Version 3.0, Release 11, for Linux,

Real-Time Workshop 3.0, Release 11, for Linux, and

RTLT Version 1.2 (a Quality Real Time Systems product). Note the C development packages should be selected while installing Red Hat Version 6.1.

Note: If RTLT 1.1 is already installed, the user may directly upgrade to RTLT 1.2 as explained in the section "Upgrading RTLT".

Installing RTLT

In order to install RTLT, the user must first install Linux, MATLAB, Simulink, and RTW. A Linux shell script is used to ensure that RT-Linux 2.0 is installed and all RTLT files are extracted and saved in the appropriate directories. In order to install RTLT successfully, the user must ensure that Linux and MATLAB version 5.3 (release 11) are correctly installed (see Appendix A and B for guides on how to install Linux and RT-Linux, respectively). The rest of this guide assumes that Linux OS and MATLAB have been correctly installed.
What is Included in the RTLT Package

**QRTS Base Package** (*baseqrts.tgz*): This is a prerequisite package for any QRTS software product installation. It sets up environment variables and path that will be used by other QRTS software.

**RTLT 1.2** (*rtl1.2.tgz*): RTLT 1.2, libc5 packages, and Install/Uninstall scripts.

**Pre-patched RT Linux 2.0** source file (*rtl2patch.tgz*).

**SDK Package(s)**: QRTS provides SDK(s) for several supported I/O boards. Currently, QRTS provides MultiQ SDK for Quanser MultiQ board, STG SDK for ServoToGo board, the MPMAC SDK for the Mini PMAC board, PCL821 SDK for the PCL812 board, and the ATI F/T SDK for the ATI Force/Torque Sensor System. The SDK package(s) may also be included in your RTLT package.

*A Quick-Install procedure is also available for the advanced user at the end of this section.*

**QRTS Base Package Installation**

The user must install the QRTS base package before installing RTLT. If the QRTS base package is already installed, there is no need to reinstallation.

**The Environment Variable “QRTS”**

The environment variable `QRTS` must be defined before you can install any QRTS software. `$QRTS` should be set to the directory where the QRTS software products are to be installed, for example, `/usr/qrts`.

The user now follows the steps listed below:

The user creates a directories `/usr/qrts/bin` by typing the following at the Linux prompt

```
# mkdir /usr/qrts
# mkdir /usr/qrts/bin
```

The environment variable `QRTS` and the path `/usr/qrts/bin` are setup as follows:

For bash shell, the user opens the file `/etc/profile` and appends the following line to the file, (Note: Be sure not to leave any additional space when typing this line)

```
export QRTS=/usr/qrts
```

The user then changes the line

```
PATH="$PATH:/usr/X11R6/bin"
```

to

```
PATH="$PATH:/usr/qrts/bin"
```
PATH="$PATH:/usr/X11R6/bin:.:${QRTS}/bin"

This adds /usr/qrts/bin and the current directory into the path.

For the C shell, the user opens file /etc/csh.cshrc or /etc/csh.login and appends the following line to the file:

```
setenv QRTS "/usr/qrts"
```

The user then changes the line

```
setenv PATH "$PATH:/usr/X11R6/bin"
```

To
```
setenv PATH "$PATH:/usr/X11R6/bin:.: ${QRTS}/bin"
```

This adds /usr/qrts/bin and the current directory into path.

The user now logs out and logs back in

**Installing the QRTS base package**

Copy baseqrts.tgz to a temporary directory, for example, /tmp, by typing the following at the Linux prompt.

```
# cp baseqrts.tgz /tmp
```

Change directories to tmp

```
# cd /tmp
```

Unzip the zipped file as follows

```
# tar -xzf baseqrts.tgz.
```

A subdirectory baseqrts is created.

Change directories to baseqrts

```
# cd baseqrts
```

The user now executes the installation script that guides the user through the installation.

```
# qinstall
```

Upon successful installation, the following message is displayed on the monitor.

```
----- Congratulations, the QRTS base package intalled successfully. -----
```

It should be noted that the QRTS base package is to be installed only once.
Extracting and Executing RTLT Installation Files

In order to install RTLT, the user must already have successfully installed MATLAB and the required components (Simulink and Real-Time Workshop). The user also sets an environment variable called MATLAB to the MATLAB root directory.

The Environment Variable “MATLAB”

For example, if MATLAB is installed in /usr/local/matlab, for the bash shell, the user appends the following line to the file /etc/profile

```
export MATLAB=/usr/local/matlab
```

For the C shell, the user appends the following line to file /etc/csh.cshrc or /etc/csh.login

```
setenv MATLAB "/usr/local/matlab"
```

You should now logout and log back in for this new environment variable to take effect.

Installing the RTLT 1.2 package

The RTLT installation script automatically installs RT-Linux 2.0. Therefore, it is sufficient for the user to ensure that Linux and MATLAB are correctly installed. It should be noted that even if RT-Linux 2.0 is already installed, the user must boot the system into Linux and reinstall RT-Linux 2.0 as per the procedure given below. Moreover, the user must run the installation script within the GUI desktop (X Windows, Gnome, KDE, etc.)

Copy rtlt1.2.tgz and rtl2patch.tgz to a temporary directory, for example, /tmp, by typing the following at the Linux prompt.

```
# cp rtlt1.2.tgz /tmp
# cp rtl2patch.tgz /tmp
```

If the installation package includes SDK packages, then the user also copies the SDK packages (for example, mqsdk.tgz for the MultiQ SDK,) into /tmp. You may want to copy all the CD files by simply typing

```
# cp *.* /tmp
```

This facilitates automatic installation of the SDK packages alongside RTLT.

Unzip the zipped file as follows

```
# cd /tmp
# tar -xzvf rtlt1.2.tgz.
```
The user now executes the installation script that guides the user through the installation.

```
# ./install_rtlt
```

The RTLT installation script in addition to installing RT-Linux and RTLT sets up the shared memory pool. To this end, the system memory is checked first. If it fails, the user reboots the PC and runs the installation script again. By default, the pre-patched RT-Linux kernel does not enable any network cards, sound cards, etc. Hence, to enable some specific hardware, the user types "y" when the prompted by the installation script to run the configuration utility.

The installation script now rebuilds the new kernel. If some warning messages appear on the screen during this stage of installation, the user should disregard it. Interested readers should refer to Appendix B for details on RT-Linux 2.0 installation and shared memory configuration.

It is strongly recommended that when prompted by the installation script, the user type “y” to install the libc5 environment.

The installation script automatically solves incompatibilities between MATLAB 5.3 and Red Hat Linux6.1. A discussion about the compatibility issues is available in Appendix C of this document.

SDK packages present in the same directory are also automatically installed.

This Linux shell script ensures that all RTLT files are extracted and saved in the appropriate directories.
A table describing the directory structure with the relevant files is shown below

<table>
<thead>
<tr>
<th>Directory</th>
<th>Filenames</th>
</tr>
</thead>
<tbody>
<tr>
<td>rtw/c/grt/</td>
<td>lrt.tlc, lrt.tmf, lrt_main.c</td>
</tr>
<tr>
<td>rtw/c/lib/lnx86/</td>
<td>lrtwLib.a</td>
</tr>
<tr>
<td>rtw/c/src/</td>
<td>lcommon.h, lupdown.h, lupdown.c</td>
</tr>
<tr>
<td></td>
<td>lode1.c, lode2.c, lode3.c, lode4.c, lode5.c</td>
</tr>
<tr>
<td>rtw/c/tlc/</td>
<td>lcodegenentry.tlc, lcommonentry.tlc, lcommonsetup.tlc, lsrtbody.tlc, lsrtexport.tlc, lsrthdr.tlc, lsrtmap.tlc</td>
</tr>
<tr>
<td></td>
<td>lsrtbody.tlc, lsrtexport.tlc, lsrthdr.tlc, lsrtmap.tlc</td>
</tr>
<tr>
<td></td>
<td>lsrtreg.tlc, lsrtwide.tlc</td>
</tr>
<tr>
<td>rtw/c/tools/</td>
<td>rlinuxtools.mk</td>
</tr>
<tr>
<td>simulink/src/</td>
<td>iob_adc.c, iob_adc.mexlx, iob_adc.c, iob_adc.mexlx</td>
</tr>
<tr>
<td></td>
<td>iob_digin.c, iob_digin.mexlx, iob_digout.c, iob_digout.mexlx</td>
</tr>
<tr>
<td></td>
<td>iob_encoder_r.c, iob_encoder_r.mexlx, iob_encoder_w.c, iob_encoder_w.mexlx</td>
</tr>
<tr>
<td>toolbox/rtw/</td>
<td>lext_comm.mexlx, rtlilib.mdll, tlc.mexlx, lmcheckcfg.p</td>
</tr>
<tr>
<td></td>
<td>lmcheckoutDriver.p, lmFindIO.p, lmEnc.p, lmReadlist.p</td>
</tr>
<tr>
<td>rtlit</td>
<td>driverlist.cfg</td>
</tr>
<tr>
<td>rtlit/include</td>
<td>lmotor.h</td>
</tr>
<tr>
<td>rtlit/lib/</td>
<td>libLMotor.a</td>
</tr>
</tbody>
</table>
Quick Install for the Advanced User

QRTS Base Package Installation Quick Start

Create a QRTS root directory, where QRTS software products can be installed. For example, /usr/qrts
Set environment variable QRTS to this directory, for example, $QRTS=/usr/qrts
Add $QRTS/bin in PATH
Unzip the installation files using the command `tar -xzf baseqrts.tgz`
Change to the base QRTS directory using command `cd baseqrts`
Install the QRTS base package using the command `./qinstall`

RTLT Installation Quick Start

The user must already have installed the QRTS base package and Matlab5.3.
The user boots into Linux
Copy files rtlt1.2.tgz, rtl2patch.tgz, and any SDK packages into /tmp
Set environment variable MATLAB to MATLAB root directory, for example,
 $MATLAB=/usr/local/matlab
Unzip the installation files using the command `tar -xvf rtlt1.2.tgz`
Install RTLT using the command `./install_rtlt`
Answer "y" to the RT-Linux configuration question about enabling specific hardware (for example, network card).
Answer "y" to the libc5-related question.

Upgrading RTLT 1.1 to RTLT 1.2

If RTLT 1.1 is already installed, it is simple to upgrade to RTLT 1.2 as outlined in the procedure below

Copy the update file update.tgz into a temporary directory /tmp.
Change active directories to /tmp by typing at the Linux prompt `cd /tmp`
Unzip the file by using `tar -xzf update.tgz`
Run the upgrade script `update_rtlt`. 

Creating the Real-Time Model

The user creates the real-time target by using the same methodology as that used in constructing a simulation. The hardware client-server architecture of RTLT automatically interfaces the user’s block diagram to the digital information provided by the installed I/O boards. Specifically, the RTLT library block set contains several I/O blocks that are used for I/O data interface. At the MATLAB console, the user accesses the RTLT library block set by typing

>> rtltlib

Through the use of a pop-up menu shown below, this command allows the user to drag the desired block into the Simulink model and then make the necessary connections.

After the user has placed the I/O blocks in the Simulink model, the user can edit the I/O block properties by double clicking on the block. For example, the user can set or edit the input channel that the data is read from as shown below.
I/O Blocks

Hardware Adapter

Typically, the Hardware Adapter block is the first block that is accessed if the user’s Simulink model includes an I/O board. The Hardware Adapter block defines the software driver that is used for data acquisition. By double clicking on the Hardware Adapter block, the user can access the Block Parameters: Hardware Adapter property sheet.

- Presently, the I/O Board Driver pop-up menu can be set to the following supported I/O boards: MultiQ2, MultiQ3, ServoToGo, Mpmac, PCL812, Feedback_Enc, Feedback_Pend, and ATIFT. RTLT allows the user to write a customized I/O device driver by selecting "Other" from the I/O Board Driver pop-up menu. For more details, the user is referred to the chapter "Driver Development Kit".
- The user then enters a name in the I/O Board Server Name edit box. The user must use the same name in the I/O Board Server Name edit box during the construction of other I/O blocks to access data from the same I/O board. For example, if the server name "iobs0" is used to indicate a MultiQ-2 board, then to access a desired A/D channel on this particular I/O board, the user must enter "iobs0" in the I/O Board Server Name edit box in the Block Parameters: Analog To Digital Converter property sheet.
- The Base Address edit box contains a hexadecimal number that is unique to an I/O board (note two I/O boards must have two different base addresses). As a standard practice, the user should use the factory preset base address for the I/O board. The user may be able to change the base address of the I/O board by modifying on-board jumper switches.
The Frequency edit box corresponds to how fast the user desires the I/O board driver to run. By default, the sampling frequency is set to 1000 Hz. The sampling frequency of the I/O board server should be the same as the sampling frequency in user’s model (See Fixed step size edit box discussed below in the Setting the Real-Time Options section).

The Find I/O Board by pop-up menu indicates how the I/O board will be identified. Specifically, it determines whether an I/O board is identified by its base address, or a board number. For example, the MultiQ2 and MultiQ3 boards require that the user specify the base address, while the ServoToGo board requires a board number. Hence, the user selects either "Base Address" or "Board Number" according to the type of I/O board selected. Some I/O boards may auto-detect the available I/O port. In such cases, the user should select "Auto-Detected", which is the third option in the Find I/O Board by pop-up menu. The user should NOT use the "Auto-Detected" option for the MultiQ2, MultiQ3 or ServoToGo boards.

The user must input the base address in the Base Address edit box if the Find I/O board by menu is set to "Base Address". Alternatively, if the user sets the Find I/O board by menu is set to "Board Number", the user must enter an appropriate board number (an integer) in the Board Number edit box.

The user may pass more parameters to the I/O board server via the Other Parameters edit box. For example, the MultiQ2 board provides 8 A/D channels and 8 D/A channels. By default, all these channels will be accessed by the I/O board server. If the user wants to use only 2 A/D and 3 D/A channels on the board, the user may specify it by overwriting the default values. More details about the boards supported by RTLT are available in the Appendix.

Note that if an I/O board server is shared by multiple real-time targets it is not recommended that the user overwrite the default values unless the user is absolutely certain that the new values are suitable for all real-time targets.
• By default, **Auto-Loading** and **Auto-Unloading** check boxes are both enabled. Hence, after the real-time target begins executing, the I/O board server will automatically be loaded; furthermore, after the real-time model stops, I/O board server will automatically be unloaded provided no other real-time model is using the same I/O board server (note that one I/O board server maybe shared by multiple real-time models at the same time). If the user disables **Auto-Loading** and **Auto-Unloading** check boxes, then user must manually load and unload the server via Linux commands, `insmod` and `rmmod`, respectively.

**Analog to Digital Converter (ADC)**

The ADC block provides the data available at the specified A/D channel to the Simulink model. The **Block Parameters: Analog to Digital Converter** property sheet allows the user to specify an **I/O Board Server Name** and the **Input Channel** on the board that corresponds to the location of the input data. The channel specified should be available on the I/O board. If an invalid channel is specified, then channel 0 will be used (this convention applies to all I/O blocks).

For the ATI Force/Torque sensor board, the ADC block is used to read the force/torque values. The forces along the X, Y and Z-axes are mapped to A/D channels 0, 1, and 2, respectively. The torque values about the X, Y and Z-axes are mapped to A/D channels 3, 4, and 5, respectively. For example, if the user wants to read the value of the force along the Z axis, the user types 2 in the **Input Channel** edit box.

![Block Parameters: Analog To Digital Converter](image)

**Digital to Analog Converter (DAC)**

The DAC block writes data from the Simulink model to the specified D/A channel. Similar to the ADC, the **Block Parameters: Digital to Analog Converter** property sheet allows the user to specify an **I/O Board Server Name** and the **Output Channel** on the board that corresponds to the location of the output data.
Digital Input and Digital Output

The Digital Input and Digital Output blocks write and read data, respectively, from the Simulink model to the specified digital channel. The Block Parameters property sheet of the Digital Input and Digital Output blocks are specified in a manner similar to that for the DAC and ADC blocks.

Read Encoder

The Read Encoder block provides the data available at the specified encoder channel to the Simulink model. The Block Parameters are specified in a manner similar to that of the DAC and ADC blocks. An additional feature of the block includes the Reset this encoder channel automatically check box. Enabling this check box allows the user to reset the specified encoder channel at the start of the control run. The check box Reset this encoder channel to is also included to allow the user to specify the initial value of the specified encoder channel. If the Reset this encoder channel automatically check box is not enabled, then the encoder channel will not be reset at the start of the control run, and hence, the encoder channel will retain the value of the previous run. With regard to resetting the encoder, the user should note the following exception. If the user utilizes the Set Encoder block (see below for details) to set the encoder channel to some specified value, then the value may not be correctly loaded unless the user disabled the Enable Auto-Unloading check box in the Hardware Adapter block.

Set Encoder

The Set Encoder block allows the user to set the value of an encoder to some value provided some specified condition occurs. Specifically, when the enable signal, located at the top of the block, is positive, the input from the left side of the block is written to the encoder channel. The I/O Board Server Name and the Encoder Channel must be appropriately specified.
Reset FT

The Reset FT block allows the user to reset the bias array of the ATI Force/Torque sensor system. This allows removal of the loading condition at run time. When the enable signal, located at the top of the block is positive, the bias array of the ATI Force/Torque sensor system will be reset to 0. This block can only be used with the ATI Force/Torque sensor board.
Setting the Real-Time Options

In order to set the Real-Time Options, the user must access the Simulation Parameters property sheet by selecting Simulation from the Simulink model menubar, and then selecting Parameters from the pull-down menu. The user can then access the necessary information in the Solver, Workspace I/O, Diagnostics, and Real-Time Workshop pages.

Solver Page

To access this page, the user must select Solver in the Simulation Parameters property sheet. Before building the RTLT model, the user must set the fixed-step solver step size. To this end, the user must click on pop-up menu next to Type under Solver options group title and select "Fixed-step". An appropriate value for the Fixed step size edit box is the sampling period. Thus, if the control algorithm is executed at 1000 Hz, then the user must set the Fixed step size edit box to "0.001". Currently RTLT does not support multi-tasking; hence, the Mode pop-up menu must be set to "Single Tasking". The user can also set the Start time and Stop time by accessing the appropriate edit boxes under Simulation time group title. By default these values are set to

Workspace I/O Page

To access this page, the user must select the Workspace I/O in the Simulation Parameters property sheet. The Decimation edit box is used to indicate how often data is logged. A proper choice of decimation can dramatically decrease the size of log file. By default, Decimation is set to "1", and hence, log data will be stored for every time step. That is, given a decimation setting of 1 and sample time of 1000 Hz, then 1000 data points will be logged every second for every logged variable. Given a decimation of 2 and sample time of 1000 Hz, only 500 data points will
be logged every second for every logged variable. Other parameters in the Workspace I/O page are not used by RTLT (hence, do not make any other modifications).

**Diagnostics Page**

To access this page, the user must select the Diagnostics in the Simulation Parameters property sheet. This page is used to set the desired action for many types of events or conditions that can be encountered during a simulation. Under normal circumstances, the user should not modify the default settings. For details on how to modify options in the Diagnostics page, the user is referred to the Simulink manual.

**Real-Time Workshop Page**

To access this page, the user must select the Real-Time Workshop in the Simulation Parameters property sheet. The options in this field allow the user to select the target file for code generation and specify build options.
The user must specify the **System target file** for RTLT on the **Real-Time Workshop** page of the **Simulink Parameters** property sheet as *lrt.tlc*. An easy way to do this is to click on the **Browse** button that will pop up the **System Target File Browser**. The user should select *lrt.tlc* and click **OK**. The user should not enable the **Inline parameters** check box. This option does not work with the external mode.

The template makefile is used during the automatic build procedure. During the automatic build procedure, the `make_rtw` command extracts information from the template makefile *lrt.tmf* and generates the template makefile *model_name.mk*. To this end, the user must set the **Template makefile** edit box to "lrt.tmf". The standard `make_rtw` command supplied with the Real-Time Workshop should be used as the **Make command**.
• If the user desires to execute the Simulink model on-line in Simulink, the user must generate the code in the External mode. This action is accomplished by accessing the Options property sheet and then enabling the External mode check box. If the External Mode check box is not enabled, then the code will be generated in Standalone mode wherein on-line running/tuning will not be available in Simulink.

![Code Generation Options](Image)

• The other options in the Real-Time Workshop page are not used by RTLT; hence, the user should not modify any other options.
Building the Real-Time Model

In this section, we discuss how to configure the data logging options and build a real-time model. Data logging and plotting in RTLT is achieved using the Simulink Scope. The standard Simulink X-Y Graph, Display, To File, and To Workplace blocks are not supported in RTLT. Hence, in order to log data, the output of the block must be connected to a Scope. The user can specify a variable name for a signal by adding a label to it. To add a label to a signal, the user double clicks on the line that connects into the Scope. The user must then enter a valid variable name (no spaces are allowed in a variable name). If the user does not specify a name, a default Simulink variable name (e.g., "root_Gain1") will be used.

Data Logging

Data logging/plotting in RTLT is achieved via a Scope. Note that X-Y Graph, Display, To File, and To Workplace blocks are not supported in RTLT. Hence, in order to log data, the output of the block must be connected to a Scope. An example of logging the encoder data is shown below.

The user can specify a variable name for a signal by adding a label to it. To add a label to a signal, double click on the line that connects into the Scope. Enter a valid variable name (no spaces allowed in a variable name). If the user does not specify a name, the default Simulink variable name (e.g., "root_Gain1") will be used.

Merged Signals

In order to display multiple plots overlapping each other, the user can use the Mux (multiplexer) block in the Signals and Systems blockset as shown above. It should be noted that the block diagram on the left does not allow the user to assign a specific name to each variable. However, if the block diagram on the right is used, variables can be assigned names (for example, AD1, and AD2).
Vector Variable

If the variable to be plotted/logged is a vector, the user can use the Demux (demultiplexer) block in the Signals & Systems blockset as shown above. The user only has to name the input vector variable (Output in the above example) once, and the vector components will automatically be saved with a numerical suffix. For example, in the above block diagram, the components of Output will be saved as Output_1, Output_2, and Output_3.

Output of a Subsystem

The output of a subsystem is actually the output of the last block within that subsystem.

For example, in the above figure, the block diagram on the right constitutes the subsystem in the block diagram on the left. For RTLT to plot and log the variable "Voltage" correctly, the variable name "Voltage" must be given within the subsystem as shown in the block diagram on the right. If the variable name is assigned to the line entering the scope, the variable will still be plotted but will not be logged with the name "Voltage".

Configuring the Scope

In order to configure the Scope, the user must double click on the Scope block in the Simulink model. The Properties button in the Scope toolbar is used to access the scope properties. In the General page under Scope Properties property sheet, the Number of axes refers to the number of subplots in the scope. The user must enter an appropriate value in this edit box (do not modify any other fields). In the Data History page under the Scope Properties property sheet, the user
should enable the **Limit rows to last** check box and enter the number of data points to be stored in the adjacent edit box. The number of data points is counted backwards from the end of the run. If the user desires to store all the data points, the user should disable the **Limit rows to last** check box (do not modify any other field). The user can obtain more details on the Simulink Scope by referring to the MATLAB Real-Time Workshop Manual.

**Data Logging Options**

To set the data logging options, the user must access the **External Mode Control Panel**. The user then can access the **External Signal & Triggering** property sheet by clicking on the **Signal & triggering** button in the **External Control Panel**. Under **Signal selection** group title, the user can click on the **Select all** button to select all the scopes or highlight the scope that the user desires to access and then enable the **on** check box. A scope is selected if an 'X' appears in the leftmost column under the **Signal selection** group title. The **Source** pop-up menu under the **Trigger** group title should be set to "manual". The **Mode** pop-up menu should be set to "one-shot". The **Delay** edit box is usually set to 0.

The **Duration** edit box specifies the number of time interval for which the external mode logs data after logging starts. The value of the **Duration** edit box should be equal to the (Stop time – Start time) x (1/Fixed step size). For example, if the user executes a real-time model at 1000 Hz for 50 seconds and desires to log data for the entire 50 seconds, then **Duration** is set to 50,000. If the user sets the **Decimation** field to a value of 1, then all the 50,000 data points will be logged.
for every variable connected to a Scope block. If the user sets the Decimation edit box to 10, then only 5000 data points will be logged (1000 Hz/100 = 10 points per second).

If the user desires to log data when the control program starts to execute, the user should enable on the Arm when connect to target check box. The user can also start the data logging process by clicking on the Arm trigger button in Tools/External Mode Control Panel after the real-time model starts executing. For more details with regard to configuring External Signal & Triggering, the user is referred to the MATLAB Real-Time Workshop User's Guide.

Data Archiving

To set the data archiving options, the user must access the Tools/External Mode Control Panel. The user can then click on the Data archiving button to open the External Data Archiving property sheet. Under the Data archiving group title, the user can enable the Enable archiving check box to activate the automated data archiving feature of the external mode. If Enable archiving check box is disabled, no log file will be created. If the Enable archiving check box is enabled, the user must provide a valid Directory name and a File name for archiving purposes. If Increment file after one-shot check box is enabled, the filename will be automatically incremented after each run. For example, if the log filename is "dioLog", then successive log files will be saved as dioLog1.m, dioLog2.m, and so on, until the user disables the Increment file after one-shot check box or opens a different Simulink model. Other options
under the **Data archiving** group title are not used by RTLT; hence, the user should not modify any other fields.

![MotorControl: External Data Archiving](image)

**Automatic Build Process**

The automatic build process is started by selecting **Tools** from the Simulink model menubar and then selecting **RTW Build** from the pull-down menu. After this action, RTLT automatically builds the real-time target according to the following steps:

1. RTLT creates `<model_name>.c` and `<model_name>.h` C source files, and configuration file `<model_name>_cfg.cfg`.
2. The build command for real-time applications (`make_rtw`) is issued which creates `<model_name>.mk` makefile from the template file (`lrt.tmf`).
3. The make utility (`nmake`) builds the `<model_name>_rtl.o` (or `rt_<model_name>.o` for standalone targets) real-time target using the makefile created in step 2.

If there are any errors in the Simulink model or an RTLT settings, a dialog box will appear with warnings and errors. If there are no errors, then the user is ready to execute the real-time target. There are two methods available for executing the real-time target: External Mode and Standalone Mode. The External Mode is usually used for tuning and prototyping the controller since the user remains within the Simulink environment. The Standalone mode is usually used after the control prototyping has been completed (i.e., the Standalone mode could be used in an industrial setting as part of an embedded system without the MATLAB software installed).
Executing the Real-Time Model

This section describes how to execute the real-time model.

After building the Simulink model (see previous section) using Tools / RTW Build, the user is now ready to execute the control.

Note that if the user changes any element in the model, he must re-build the model. The user may not need to rebuild the parameters when the user changes some parameters; however, there are some limitations. For more details, refer to the Real-Time Workshop User’s Guide. If the IO Board Server Name or the Channel number is changed on-line, the user must rebuild the model.

In External Mode

To execute the real-time target in the External Mode, the user connects to the real-time target by accessing selecting Tools from the Simulink model menubar and then selecting External Mode from the pull-down menu. In the External Model Control Panel, the user clicks on the Connect button.

To begin execution, the user then clicks on the Start real-time code button.
To stop execution of the real-time target, the user can click either the Stop real-time code button or the Disconnect button. In the External Mode Control Panel, the check box batch download under Parameter tuning group title controls the behavior of parameter tuning. If the batch download check box is enabled, then the new parameters will not be downloaded automatically to the real-time target until the user clicks the Download button. This feature is useful in a situation where the user needs to change multiple parameters and desires that these changes take effect instantaneously; however, there are some limitations to this capability. For more details, the user is referred to the MATLAB Real-Time Workshop User's Guide.

If the user modifies or changes any element in the Simulink model, the user must rebuild the real-time target. In some cases, the user may not need to rebuild the real-time target if the user makes modifications to parameter values; however, there are some limitations. For more details, refer to the Real-Time Workshop User's Guide. If the I/O Board Server Name or the Channel Number is modified or changed, the user must rebuild the real-time target.
In Standalone Mode

In Standalone mode, the real-time target cannot be executed or tuned on-line in the Simulink environment; however, two script files supplied with RTLT can be used to start and stop the real-time target. Specifically, the script `load_<model_name>` is used to load the standalone real-time target and execute it. If the user chooses to execute the standalone module indefinitely, the value -1 is passed as the parameter to the script as follows:

```
# load_<model_name> -1
```

If any other parameter value or no parameter is specified after `load_<model_name>` then the default stop time is the time set in the Simulation Parameters property sheet (e.g., if the default stop time is 20, `load_<model_name>` will execute the real-time target for 20 seconds. The standalone real-time target can be stopped at anytime by executing the script `unload_<model_name>`. This script will stop the real-time target and unload the real-time target from the Linux kernel. The real-time target will stop at the desired stop time automatically if it is not manually stopped and unloaded; however, the real-time target will remain in the Linux kernel until the user executes the unload script.
Example: PD Control of a DC Motor

Step 1: Draw the Simulink Block Diagram
The first step is to draw the Simulink block diagram. The SIMULINK library shown below appears if the user types, at the MATLAB prompt

```matlab
>> simulink
```

To construct the model, the double click on the appropriate block-sets to acquire the desired mathematical functions. Now drag and drop the desired blocks into the Simulink workspace. For more details, refer to the Simulink User’s guide.

To access the RTLT library, type the following command at the MATLAB prompt

```matlab
>> rtltlib
```

The RTLT library shown below will appear.

Detailed discussions about each of these blocks are presented in the section entitled, "Creating the Real-Time Model."

An example block diagram for a proportional-derivative (PD) control of a DC motor using high-gain current feedback is shown below. The angular position of the motor shaft is measured via an
optical encoder. This quadrature signal is fed into encoder channel 0 on a MultiQ board. The motor current is measured via a Hall-effect current sensor that outputs a voltage proportional the current. This voltage is read through A/D channel 0 on the MultiQ board. The voltage is output to the motor via the D/A channel on the MultiQ board.

**Step 2: Initialize the Simulation Parameters**

In **Solver** page of the **Simulation Parameters** property sheet, the user enters the Start time, Stop time, and Solver options. In this example, the control is executed for 10 seconds at 2000 Hz (*i.e.*, the fixed step size is set to $1/2000 = 0.0005$).
In the **Workspace I/O** page of **Simulation Parameters** property sheet, the user now sets the **Decimation**. In this example, the decimation is set to 1.

In the **Real-Time Workshop** page of **Simulation Parameters** property sheet, the user clicks on **Browse** button and selects the system target file as `lrt.tlc`. This action will automatically select the correct option for the **Template makefile** and the **Make command**.
The user now clicks on the Options button in the Real-Time Workshop page of Simulation Parameters property sheet to enable the External mode check box. If the External Mode check box is not enabled, then the code will be built in Standalone Mode.

Step 3: Initialize the I/O Board Server

The user now double clicks on the Hardware Adapter Block in the Simulink block diagram to initialize the I/O board server. The user selects the appropriate I/O board server by using the I/O Board Driver pop-up menu (in this example, the "multiq2"). The user also enters appropriate information under the I/O Board Server Name edit box (in this example, "iobs0"), Base Address edit box (in this example, "0x320"), and the Frequency edit box (in this example, 2000 Hz). Note that Fixed Step Size edit box in the Solver page of Simulation Parameters property
sheet must be the reciprocal of the value of the **Frequency** edit box in the **Block Parameters: HW Adapter** property sheet.

The user now enables the **Enable Auto-Loading** check box and the **Enable Auto-Unloading** check box so that the real-time target code is loaded automatically when the user clicks on the **Connect** button in the **External Control Panel**. Therefore, the real-time target is unloaded automatically when the user clicks on the **Disconnect** button in the **External Control Panel**.

### Step 4: Set the Block Properties

The user now double clicks on the A/D, D/A, Scope, and Encoder Channel blocks to edit the respective **Block Properties** property sheets. For example, the user should ensure that the correct I/O board server name is entered under the **I/O Board Server Name** edit box and that the correct channel number is entered under the **Input Channel** edit box. (For a detailed discussion on the block parameter settings, the user is referred to the section entitled, Creating the Real-Time Model.)
Step 5: Set Data Logging and Archiving Options

The user now clicks on the **Signal & Triggering** button in the **External Control Panel**. In most circumstances, the user should retain the default values of **Trigger Source** ("manual"), **Trigger Mode** ("one-shot"), and **Trigger Delay** ("0"). The user should set the **Duration** edit box to the desired number of data points to be logged (in this example, "20,000" which is the product of the sampling frequency = 2000 and the duration = 10 divided by the decimation = 1).
For this example, the **Arm when connect to target** check box is enabled. The Scope block has been selected by clicking on the **Select all** button (indicated by the "X" to the left of "Scope").

**Step 6: Build the Target and Execute**

The automatic build process is started by first selecting **Tools** from the Simulink model menubar and then selecting **RTW Build** from the pull-down menu. After this action, RTLT automatically builds the real-time target. To execute the real-time target in the External Mode, the user connects to the real-time target by accessing selecting **Tools** from the Simulink model menubar and then selecting **External Mode** from the pull-down menu. In the **External Model Control Panel**, the user clicks on the **Connect** button to load the real-time target.
To begin execution, the user then clicks on the **Start real-time code** button.

To stop execution of the real-time target, the user can click either the **Stop real-time code** button or the **Disconnect** button.

**Driver Development Kit**

**Introduction**

The RTLT Driver Development Kit (DDK) provides the user with a platform to write customized drivers for different I/O boards. This is facilitated by the client-server architecture of RTLT and the shared memory based communication between the I/O board driver and the real-time model. It should be noted that the I/O board driver itself is a real-time task running periodically within the RT-Linux environment.
In order to write an I/O board driver using the RTLT-DDK, the three main issues that need attention are:

- Hardware Interface
- Shared Memory Access
- Real-time, Periodic Execution of the Driver

The RTLT-DDK provides a framework for the user to write the driver by setting up the shared memory access and creating the real-time task, both of which are fairly complex tasks. Hence, the user focuses only on the hardware interface. Thus, the RTLT-DDK simplifies the task of writing and implementing a customized driver by allowing the user to focus only on I/O hardware-specific issues.

The RTLT-DDK provides a set of template files and a startup script. The startup script uses the template files to generate C++ source files and a makefile. These generated C++ source files include source files for the hardware interface class, I/O board server class, and the RT-Linux drive module. For example, if the user uses the DDK to develop a driver for a board called `myBoard`, the following files are automatically generated:

- `myBoard.hpp`: is the header file for the hardware interface class. The user defines the properties and behaviors of the specific board in this file. For example, `readADC()` method may be used to read the A/D channels.

- `myBoard.cpp`: deals with the implementation of the hardware interface class.

- `myServer.hpp`: is the header file for the I/O Board Server class. Each I/O board is a subset of the I/O Board Server class. The I/O Board Server provides only one method, i.e., `doMessageLoop()`, which executes periodically in real-time.

- `myServer.cpp`: deals with the implementation of the I/O Board Server class for “myBoard”. The constructor calls the hardware interface to initialize the I/O board. The input-output operations are performed within the `doMessageLoop()` method because it is executed periodically in real-time. For example, the `readADC()` method of the hardware interface class may be called from within the `doMessageLoop()` method to read A/D channels of “myBoard” and store the result in shared memory.

- `myDriver.cpp`: deals with the implementation of the RT-Linux module. The I/O board driver is implemented as a Linux kernel module. This file provides the interface for a kernel module. In `init_module()`, the I/O Board Server class is instantiated to facilitate I/O board initialization. Subsequently, a real-time task is created, which calls the `doMessageLoop()` method. Once the driver module is loaded, the driver is executed at a fixed frequency, which is passed as a load-time parameter when the driver module is loaded in the Linux kernel.

Since RTLT employs the client-server architecture, any user-customized I/O board driver development with RTLT-DDK can be used in conjunction with RTLT. This is done
simply by entering the location of the driver module file in the Hardware Adapter block found in the RTLT library.

**Hardware Adapter Block for the DDK**

After the RTLT-DDK is installed, an additional parameter is added to the HW Adapter block properties. Specifically, the Driver File Name (Full Path) edit box is incorporated to allow the user to specify the location of the customized I/O board driver. The user must note that this edit box is enabled only if the I/O Board Driver is set to “Others”. That is, if the I/O Board Driver is set to an available driver (for example, “ServoToGo”, “MultiQ2”, “MultiQ3”, etc.), the Driver File Name (Full Path) edit box is disabled. An example illustrating the usage of this additional parameter is shown in a later section of this document.

![Hardware Adapter Block](image)

**Installing the RTLT-DDK**

Copy ddk.tgz to a temporary directory, for example,

```bash
# cp ddk.tgz /tmp
```

Change the current directory to /tmp

```bash
# cd /tmp
```
Untar the package as

```bash
# tar -xzf ddk.tgz
```

Upon extracting the files within `ddk.tgz`, a directory `ddk` will be created under current directory (for example, `/tmp/ddk`). Change the current directory to the newly created `ddk` directory.

```bash
# cd ddk
```

Run DDK installation script `qinstall`

```bash
# ./qinstall
```

The DDK installation script creates a subdirectory `$QRTS/ddk`. Furthermore, a guide shell-script `$QRTS/bin/startddk`. Sample driver files for the CIODIO digital I/O board are copied into the `$QRTS/ddk/sample/ciodio` directory.

### Compiling and Using the Customized Driver

A customized driver is developed on an RTLT-DDK platform as follows:

The user runs the `startddk` script in a desired directory (for example, `/home/user1`).

```bash
# startddk
```

The `startddk` script generates the basic C++ code and makefile for the customized driver.

The script now prompts the user for a driver name. The user may specify the name based on a specific board. For example, the user may type “myIO” and hit the Enter key to continue installation.

A subdirectory with the same name, that is, “myIO” is created and all the auto generated C++ source code and the `makefile` are copied into this directory. The following files are generated

- `myIOBoard.hpp` – header file for the hardware interface class
- `myIOBoard.cpp` – hardware interface implementation
- `myIOServer.hpp` – header file for the I/O Board server
- `myIOServer.cpp` – I/O Board server implementation
- `myIODriver.cpp` – RT-Linux module code

The user now modifies the files according to the driver functionality requirement. Once the files are modified, the user compiles these programs by using the make utility in Linux that uses the `makefile`
# make

After compiling the above source code, the RT-Linux module `myIOModule.o` is generated along with other intermediate object files. To use the `myIOModule.o` driver, the user sets the I/O Board Driver pull-down menu to “Others” and types the full path of the `myIOModule.o` file in the Driver File Name (Full Path) edit box (for example, the file `/home/user1/myIO/myIOModule.o`)

This newly generated driver is incapable of performing I/O operations because it is just a "framework". The user must add hardware-specific code into these auto-generated "framework" files. The following example for the CIO-DIO24/CTR3 board illustrates how the DDK is used to write the I/O board driver.
Example: RTLT-DDK-Based Customized Driver for the CIO-DIO24/CTR3 Board

The driver for CIO-DIO24/CTR3 board (hereon, referred to simply as the CIODIO board) manufactured by Computer Boards, Inc. is discussed as an example of an RTLT-DDK-based driver.

The Hardware Adapter block properties for the CIODIO board are shown below. Note that in this example, the driver module is available at /usr/qrts/modules/cioidoModule.o. The full path is therefore entered in the Driver File Name (Full Path) edit box as shown below.

The source files and the makefile that are used to generate this driver module are given in the following section. The source files are compiled using the make utility in Linux as discussed in the previous section.
Source Files for the CIODIO Board

Hardware Interface Class Header: ciodioBoard.hpp

`--- FILE HEADER ---
component   : DDK
file        : ciodioBoard.hpp
description : Sample hardware interface for CIODIO board
author      : Zhigao Yao
date        : May, 2000
---

#ifndef INCLUDED_HWI_HPP
#define INCLUDED_HWI_HPP

#ifndef __LINUX__
#define __LINUX__
#endif

#ifndef __KERNEL__
#define __KERNEL__
#endif

#include <linux/kernel.h>
#include <linux/version.h>

/* I don’t want vmalloc.h and page.h be included */
#if LINUX_VERSION_CODE > 0x20036
    #define __LINUX_VMALLOC_H
    #define __ASSEMBLY__
    #define _PAGE_PCD 0x10
    #include <asm/pgtable.h>
#endif

#include <asm/io.h>
#include "Status.hpp"

/* Hardware interface is hardware specific. The template file
* just provides a framework, the user needs to add customized
* code based on the specific hardware. */
class ciodioBoard
{
    private: // ----- Private data members
        int d_baseAddress;   // IO board base address
        int    d_portAReg;  
        int    d_portBReg;  
        int    d_portCReg;  
        int    d_configReg; 
        int    d_counter0Reg;
        int    d_counter1Reg;
        int    d_counter2Reg;
        int    d_counterControlReg;

...
public: // ----- Public members
    Status d_status;
    int d_portADirection;
    int d_portBDirection;
    int d_portCDirection;

public: // ----- Public methods
// ----- Creators
ciodioBoard ( int baseAddress,
               int portADirection=1,
               int portBDirection=1,
               int portCDirection=1 );

~ciodioBoard ();

int readPortA();
int readPortB();
int readPortC();
void writePortA( unsigned char value );
void writePortB( unsigned char value );
void writePortC( unsigned char value );

};

#endif
#include "ciodioBoard.hpp"

extern "C"
{
    #include <linux/ioport.h>
}

// Constructor
ciodioBoard::ciodioBoard( int baseAddress,
                            int portADirection=1,
                            int portBDirection=1,
                            int portCDirection=1 )
{
    unsigned char portConfig = 0x80;
    int errCode;

    d_baseAddress  = baseAddress;

    // TO DO - Add other initialization code here

    // check baseAddress
    errCode = check_region( baseAddress, 8 );
    if ( errCode < 0 )
    {
        printk("Base Address is invalid. Please make sure no other
                devices are using it.\n"));
        d_status.setStatusError();
        return;
    }

    request_region( baseAddress, 8, "ciodio" );

    d_portAReg         = d_baseAddress;
    d_portBReg         = d_baseAddress + 1;
    d_portCReg         = d_baseAddress + 2;
    d_configReg        = d_baseAddress + 3;
    d_counter0Reg      = d_baseAddress + 4;
    d_counter1Reg      = d_baseAddress + 5;
    d_counter2Reg      = d_baseAddress + 6;
    d_counterControlReg = d_baseAddress + 7;

    d_portADirection   = portADirection;
    d_portBDirection   = portBDirection;
    d_portCDirection   = portCDirection;
// Set port I/O configuration
if ( portADirection )
    portConfig |= 0x10;
else
    portConfig &= 0xEF;
if ( portBDirection )
    portConfig |= 0x02;
else
    portConfig &= 0xFD;
if ( portADirection )
    portConfig |= 0x09;
else
    portConfig &= 0xF6;
outw( portConfig, d_configReg );
d_status.setStatusOk();
}
ciodioBoard::~ciodioBoard( void )
{
    // reset all ports as input ports
    outw( 0x9B, d_configReg );

    // release io ports
    release_region( d_baseAddress, 8 );
}
int ciodioBoard::readPortA()
{
    return ( inb(d_portAReg) );
}
int ciodioBoard::readPortB()
{
    return ( inb(d_portBReg) );
}
int ciodioBoard::readPortC()
{
    return ( inb(d_portCReg) );
}
void ciodioBoard::writePortA( unsigned char value )
{
    return ( outb(value, d_portAReg) );
}
void ciodioBoard::writePortB( unsigned char value )
{
    return ( outb(value, d_portBReg) );
}
void ciodioBoard::writePortC( unsigned char value )
{
return ( outb(value, d_portCReg) );
}
Server Class Header: ciodioServer.hpp

///------------------------ FILE HEADER -------------------------------
/// component   : DDK
/// file        : ciodioServer.hpp
/// description : Declaration for the server class.
/// author      : Zhigao Yao
/// date        : May, 2000
///
///--------------------------------------------------------------------
#ifndef INCLUDED_SERVER_HPP
#define INCLUDED_SERVER_HPP

#include "IOBoardServer.hpp"
#include "ciodioBoard.hpp"

class ciodioServer : public IOBoardServer 
{
    private:
        ciodioBoard *d_board;
        int d_portADirection;
        int d_portBDirection;
        int d_portCDirection;

    public:
        // ----- creators
        ciodioServer( int baseAddress,
                      double frequency,
                      int numAdc,
                      int numDac,
                      int numDigin,
                      int numDigout,
                      int numEncoders,
                      char *serverName,
                      char *timerServerName,
                      int adcIrq,
                      int priority,
                      int portADirection,
                      int portBDirection,
                      int portCDirection );
        virtual ~ciodioServer();

        // ----- MANIPULATORS
        virtual int doMessageLoop();

};
#endif
Server Class Implementation: ciodioServer.cpp

///------------------------ FILE HEADER -------------------------------
/// component   : DDK
/// file        : ciodioServer.cpp
/// description : Implementation for the server class.
/// author      : Zhigao Yao
/// date        : May, 2000
///
///--------------------------------------------------------------------
#ifndef __KERNEL__
#define __KERNEL__
#endif

#define new _new_
#include <linux/malloc.h>
#undef new

#ifdef __cplusplus
}
#endif
#include "ciodioServer.hpp"

//Overload "new" operator
inline void *operator new (size_t, void *p) {return p;}

ciodioServer::ciodioServer ( int baseAddress,
    double frequency,
    int numAdc,
    int numDac,
    int numDigin,
    int numDigout,
    int numEncoders,
    char *serverName,
    char *timerServerName,
    int adcIrq,
    int priority,
    int portADirection=1,
    int portBDirection=1,
    int portCDirection=1)
    : IOBoardServer (frequency, timerServerName, numAdc, numDac, numEncoders,
        numDigin, numDigout, serverName, priority)
{
    // TO DO - Add code here for other server initialization code.
// adcIrq and priority is not used in this version. Leave as is.
adcIrq = -1;
priority = -1;

// initialize hardware interface
d_board = (ciodioBoard *)kmalloc(sizeof(ciodioBoard), GFP_KERNEL);
if ( !d_board )
{
        printk("Can not allocate kernel memory to initialize
               hardware interface.\n");
        setStatusError();
        return;
}

// parameters of constructor may vary for different boards
new (d_board) ciodioBoard( baseAddress,
                portADirection,
                portBDirection,
                portCDirection ) ;

// check if hardware interface is correct
if ( d_board->d_status.isStatusError() )
{
        setStatusError();
        return;
}

// TO DO - Add code here for other server initialization code
d_portADirection = portADirection;
d_portBDirection = portBDirection;
d_portCDirection = portCDirection;
}
ciodioServer::~ciodioServer()
{
        // TO DO - Add other cleanup code here

        // cleanup hardware interface
        if ( d_board )
        {
                d_board->~ciodioBoard();
                kfree(d_board);
        }

        // Explicitly call parent’s destructor or it will not be called
        this->IOBoardServer::~IOBoardServer();
}

int ciodioServer::doMessageLoop ()
{
        // TO DO - Add code here to do IO in each running step
        int inputChannel = 0;
        int outputChannel = 0;

        if ( d_portADirection )
                d_diginValue[inputChannel++] = (unsigned char)d_board->readPortA();
        else
d_board->writePortA( d_digoutValue[outputChannel++] );

if ( d_portBDirection )
    d_diginValue[inputChannel++] = (unsigned char)d_board-
>readPortB();
else
    d_board->writePortB( d_digoutValue[outputChannel++] );

if ( d_portCDirection )
    d_diginValue[inputChannel++] = (unsigned char)d_board-
>readPortC();
else
    d_board->writePortC( d_digoutValue[outputChannel++] );

return 0;
}
RT-Linux Driver Module: ciodioDriver.cpp

///------------------------ FILE HEADER -------------------------------
/// component   : DDK
/// file        : ciodioDriver.cpp
/// description : Driver module, for RTLinux2.0
/// author      : Zhigao Yao
/// date        : May, 2000
///
///--------------------------------------------------------------------

#ifndef __KERNEL__
#define __KERNEL__
#endif

#ifndef MODULE
#define MODULE
#endif

#include "builtins++.h"
extern "C"
{
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/version.h>
#include <linux/errno.h>

#include <rtl.h>
#include <rtl_time.h>
EXPORT_NO_SYMBOLS;

#include <rtl_sched.h>
#include <rtl_fifo.h>
#include "ciodioServer.hpp"

// For "placement new"
inline void * operator new (size_t, void *p ) { return p; }
inline void * operator new[] (size_t, void *p ) { return p; }

// Specify load-time parameters of the module, the users can
// add/remove more based on their specific driver
MODULE_PARM(name,"s");
MODULE_PARM(baseAddress,"i");
MODULE_PARM(frequency,"i");
MODULE_PARM(portADirection, "i");
MODULE_PARM(portBDirection, "i");
MODULE_PARM(portCDirection, "i");

// Default value for load-time parameters
static char *name = "iobs0";
static int baseAddress = 0;
static int frequency = 1000;
static int portADirection = 1;
static int portBDirection = 1;
static int portCDirection = 1;

static ciodioServer *q;
pthread_t ioManager;

// Shared Memory location to store usage information.
static int *pNumOfUsage;
static char shm_name[64];

// The function which will be called in every running step
void *mainLoop (void *t)
{
    while(1)
    {
        pthread_wait_np();
        q->doMessageLoop();
    }
}

// All initialization code should be here
int init_module()
{
    int period;
    struct sched_param p;
    pthread_attr_t attr;
    int retval;
    int numOfDigin = 0;
    int numOfDigout = 0;

    /* Setup Checkin point for IOBoard Clients. Every IOBoard Client,
    * once being created, must check in here. A counter will
    * increase
    * by one for each IOBoard Client. When an IOBoard Client is
    * released,
    * it should checkout. Then the counter will decrease by one.
    * When the counter is zero, it’s safe to unload the IOBoard
    * Server.
    */
    sprintf( shm_name, "%s_Usage", name );
    pNumOfUsage = (int *)shmmgmt_k_create( shm_name, sizeof(int) );
    if ( pNumOfUsage == NULL )
    {
        printk("Access shared memory error!\n");
        return 1;
    }
    *pNumOfUsage = 0;

    if ( frequency == 0 )
    {
        printk("frequency can not be zero! Changed to 100Hz\n");
        frequency = 100;
    }

    /* There is no "new" and "delete" in Kernel mode. So we need to
* "placement new" to initiate an object in kernel. We need to allocate
* memory for the object first, then initiate the object in the allocated
* space.
*/
q = (ciodioServer *)kmalloc( sizeof(ciodioServer), GFP_KERNEL );
if ( !q )
{
    printk("Can not allocate memory for I/O Board Server\n!");
    shmmgmt_k_delete(shm_name);
    return 1;
}
if ( portADirection )
    numOfDigin += 8;
else
    numOfDigout += 8;
if ( portBDirection )
    numOfDigin += 8;
else
    numOfDigout += 8;
if ( portCDirection )
    numOfDigin += 8;
else
    numOfDigout += 8;
new (q) ciodioServer( baseAddress, (double)frequency, 0, 0,
numOfDigin, numOfDigout, 0, name, 0, -1, -1, portADirection,
portBDirection, portCDirection );

    // Check if the server is initialized correctly
if ( q->isStatusError() )
{
    printk("Can not initiate IOBoardServer\n");
    goto EXIT;
}

/* The following code is used to initiate a RTLinux task
* for I/O Board Server.
* Note:
*  1 - The timer counter is different in RTL1.0 and RTL2.0
*  2 - The setup of stack size of the RT task is tricky. Seems
*      that 32k is a good size in this case. By default, pthread_create will setup only 3K stack for a RT task.
*  3 - priority is different in RTL1.0 and RTL2.0. In RTL1.0,
*      1 is the highest priority while in RTL2.0, 1 is the lowest
*/

    period = HRTICKS_PER_SEC/frequency;
    pthread_attr_init(&attr);
    p.sched_priority = 99; // Set the server priority to 99, which is highest
    pthread_attr_setschedparam(&attr, &p);
/ Set stack size to 32KB. May adjust to 64KB if necessary. 
pthread_attr_setstacksize(&attr, 32000);
pthread_create(&ioManager, &attr, mainLoop, (void *)1);
// Allow float-point calculation in RT task 
pthread_setfp_np(ioManager, 1);
// Start driver 
retval = pthread_make_periodic_np(ioManager, gethrtime(), period);
if (retval)
{
    printk("Can not start I/O Board Server!\n");
go to EXIT;
}
printk("ciodioDriver %s loaded successfully.\n", name);
return 0;

EXIT:
    shmmgnt_k_delete(shm_name);
    q->~ciodioServer();
    kfree(q);
    return 1;
}

// Any cleanup code should be here
void cleanup_module()
{
pthread_delete_np(ioManager);

    /* Destructor need to be called explicitly in kernel */
    q->~ciodioServer();

    if ( *pNumOfUsage != 0 )
        printk("%s report: I’m still being used by %d IO block(s).
Unloading me is unsafe!\n", 
            shm_name, *pNumOfUsage);
    shmmgnt_k_delete( shm_name );
    kfree(q);
    printk("ciodioDriver %s unloaded.\n", name);
}

} //end extern "C"
Makefile for Driver Module

INCLUDE=-I/usr/include/linux -I/usr/src/rtl/include -I$(QRTS)/include -I.
LIB=-L$(QRTS)/lib -L./
FLAG=-Wall -fno-exceptions -fno-rtti
DEFINITION=-D__LINUX__ -D__KERNEL__
CC=g++
TARGET=ciodio

all: $(TARGET)Module.o

$(TARGET)Module.o: $(TARGET)Driver.o lib$(TARGET)Server.a lib$(TARGET)Board.a
    ld -r -o $(TARGET)Module.o $(TARGET)Driver.o $(LIB) $(TARGET)Server $(TARGET)BoardServer $(TARGET)SharedMemory_K $(TARGET)Board

$(TARGET)Driver.o: $(TARGET)Driver.cpp
    g++ $(TARGET)Driver.cpp $(TARGET)Driver.o $(INCLUDE) $(FLAG) -c $(TARGET)Driver.o $(DEFINITION)

lib$(TARGET)Server.a: $(TARGET)Server.o
    ar -r lib$(TARGET)Server.a $(TARGET)Server.o

$(TARGET)Server.o: $(TARGET)Server.cpp
    g++ $(TARGET)Server.cpp $(INCLUDE) $(FLAG) -c $(TARGET)Server.o $(DEFINITION)

lib$(TARGET)Board.a: $(TARGET)Board.o
    ar -r lib$(TARGET)Board.a $(TARGET)Board.o

$(TARGET)Board.o: $(TARGET)Board.cpp
    g++ $(TARGET)Board.cpp $(TARGET)Board.o $(INCLUDE) $(FLAG) -O2 -c $(TARGET)Board.o $(DEFINITION)

clean:
    rm -f $(TARGET)Board.o
    rm -f lib$(TARGET)Board.a
    rm -f $(TARGET)Server.o
    rm -f lib$(TARGET)Server.a
    rm -f $(TARGET)Driver.o
    rm -f $(TARGET)Module.o
Frequently Asked Questions

In this section, answers to Frequently Asked Questions are provided. Furthermore, useful tips/hints suggested by frequent RTLT users are also provided.

How many real-time targets can the user run/tune on-line simultaneously?

Only one model can be run/tuned on-line at one time in RTLT. If the user wants to run multiple real-time targets concurrently, the user must build them as Standalone real-time targets, and run them in the background. However, no on-line parameter modification/logging will be available.

What is the maximum frequency that RTLT can run at?

It really depends on the conversion time for the A/D and how many channels (A/D, D/A) the user uses, as well as, the I/O board driver and RTLT are kernel modules that run in the background. If the frequency selected is too high, the foreground tasks such as the Graphical User Interface will run very slow.

Can the user run a simulation in Simulink with RTLT blocks?

No. The RTLT blocks cannot give the user correct input/output information during simulations. One possible solution is to use other source/sink blocks from Simulink for input/output operations during simulation. When the user is ready to implement the model on an actual system, replace them with actual RTLT I/O blocks and build the model.

RTLT seems to be running but I cannot obtain any signals on the I/O boards

It is likely that the user configuration file is not in the "current" directory. Check if the file <model_name>_cfg.cfg is in the current working directory. To check the current working directory type, at the MATLAB prompt,

```matlab
>> pwd
```

Also, ensure that the I/O Board is installed correctly on the system and the HW Adapter is correctly configured.

Can the user make the real-time program stop prematurely, if some condition is satisfied?

Yes. In RTLT, the Stop Simulation block in the Sinks menu of Simulink is setup such that if the input to the Stop Simulation block is 1, then the real-time target will stop executing.

For example, if the user wants the real-time target to stop executing if the voltage is higher than 10 volts, then the user can construct the following sub-model, where Hit Crossing block is available in the Signals & Systems menu of Simulink.
The signal I obtain is very noisy; how can I avoid this in RTLT?

First, trace the source of the high-frequency noise in the signal.

- If this signal is input through the A/D channels, check the sensor that generates the signal and the cable that carries the signal to the A/D channel. For example, a loosely mounted motion sensor may output a noisy signal. Also, ensure that the user is using a shielded cable to carry the sensor output to the I/O board.

- If the noise is introduced within your SIMULINK block diagram, it may be due to an approximate derivative computation. It is a well-established fact that numerical derivatives (such as backwards’ difference) result in increased noise in the output signal.

The software solution is to preferably use a second-order low pass filter, constructed by using the **Transfer Function** block in the **Continuous** menu of Simulink.

Also, ensure that the cut-off frequency is set appropriately high so that the user does not eliminate useful signal information.
Advanced filters are also available in the DSP block set in SIMULINK. However, the DSP block set must be purchased separately.

**The signal I read in slowly drifts with time. How can I solve this in RTLT?**

First, check to see if the signal exactly represents the behavior of the physical quantity being measured. If this is the case, then RTLT cannot help the user. On the other hand, for example, if the signal drifts but the physical quantity it represents is constant, then the user may want to use a high-pass filter. A first order high-pass filter can be constructed by using the **Transfer Function** block in the **Continuous** menu of Simulink. More advanced high-pass filters are available in the DSP block set (which has to be purchased separately).
The logged variables are not those that were selected. Some other variables are logged instead.

It is likely that this problem is caused by incorrect variable names in the Simulink block diagram. The following rules must be followed for variable names:

- No spaces are allowed within any variable name
- Variable names must not contain a carriage return
- There must not be a line with a blank label on it (it is likely that it is just a carriage return).

Another procedure to check is via the header file <model_name>.h. Open the header file in any editor, and go to the line with the structure definition for "BlockIO". On the line below that you will see the definition for a string "variableNameTable". The variable names in the string have a one-to-one relationship with the field names in the structure BlockIO. Find the variable you want to log in the structure BlockIO and the offset of the variable from the beginning of the structure. Then in the string "variableNameTable", find the name with the same offset. This name should be the same as the variable name in the structure (as well as the label you have specified in the diagram). If they do not match, then you have an invalid variable name. Ensure that you modify it based on the above rules.
Appendix A: Installing RedHat Linux 6.1

Preliminaries
1. Insert the RedHat Linux 6.1 bootable floppy and installation CD.
2. The installer will ask to install system in different modes. Hit <Enter> for default graphical mode, type "text" at boot: for text mode, or type "expert" and hit <Enter>.
3. Choose a Language: "English".
4. Choose a Keyboard: "US"
5. Installation Method: "Local CD-ROM".

Graphical Mode
We use the default graphical mode for installation if the installer is able to identify the settings of the video card.

*If the settings of the video card cannot be determined, the "windows" type screen will not appear and the installer will ask the user to reboot the PC. Once you reboot the PC, type "text" in step 2.*

*If you are using the Revolution 3D Ticket to Ride video card, then you will probably have to go the text mode route. If you are using the Kingston Network Card, you will probably have initial network problems.*

6. Would you Like to Install a New System or Upgrade: "Install".
7. Installation Class: "Custom".
8. Do you have an SCSI adapters: "No".

Partition and Format Hard Disk
9. Which tool do you want to use to partition the disk: Disk Druid (is default)
10. Add New Partition: Mount Point = / (which means root directory), Size (Megs) = 2323, Type = Linux Native; hit OK (/dev/hda1).
11. Add New Partition: Mount Point = don't type, Size (Megs) = 118, Type = Linux Swap. Now, the value of the Mount Point field will be automatically set to "Swap Partition". Hit OK. (/dev/hda5).
12. Active Swap Space: What partitions would you like to use for swap space? I selected /dev/hda5. I also checked the box, "Check for bad sectors while formatting".
13. Partitions to Format: What partitions would you like to format. The only option is /dev/hda1 which I selected. I also checked the box, "Check for bad sectors while formatting".

Select Components to Install
14. Components to Install: Everything (the only things you can knock off are the games and IPX/Netware Connectivity).
15. Message: A complete log of your installation will be in /tmp/install.log after rebooting the system.
16. Message: Running. Making ext2 filesystem on /dev/hda1. This will take quite some time and the message will not change on the screen.
17. The installer will now start installing a bunch of files (total of 383 packages at 521 MB).

**Hardware Identification**
18. Probing Result: the probe found some type of PS/2 mouse on port psaux. Hit OK.
19. What type of Mouse do you have? "Generic 3 button PS/2 mouse".
20. Do you want to setup networking? "Yes".

**LILO Configuration**
21. Would you like to create a boot disk on your system? No. (The boot disk is useful if you don’t want to install LILO on your system).
22. Where do you want to install LILO? Two options, /dev/hda (master boot record) or /dev/hda1 (first sector of boot partition). I selected the default of master boot record.
23. A few file systems will need to pass special options to the kernel at boot time for the system to function properly. If you need to pass boot options to the kernel, enter it now. If you don’t need any or are not sure, leave it blank. So, I left it blank and hit OK.

**Network Configuration**
24. Hostname: "crb185.ces.clemson.edu"
25. In Network Configuration:
   - Deselect Boot/DHCP.
   - IP Address: "130.127.172.195".
   - Netmask: "255.255.255.0".
   - Default Gateway: "130.127.172.2".
   - Primary Nameserver: "130.127.37.11".
   - Secondary Nameserver: "130.127.200.15".
   - Ternary Nameserver: "130.127.8.8".

**User Account Creation**
27. Pick a Root Password, and re-enter it.
28. Authentication Configuration. Do not check "Enable NIS". Check "Use Shadow Passwords" and "Enable MD5 Passwords".

**Packages for Installation**
30. Would you like to configure a printer? No.
**Video Card/Monitor Setup**

31. PCI Probe: found PCI Entry: 215CT222, X Server Mach 64 (this is the video card)
33. Screen Configuration: Xconfigurator will setup the default resolution and color depth by probing the PCI video card. The screen will blink several times.
34. Probing Finished: Xconfigurator successfully probed the video card. And the default video mode is 16 bits per pixel and 1024x768. I hit Use Default.
35. Starting X: Xconfigurator will now start X to test your configuration.
36. If you can see X windows clearly, Xconfigurator will ask if you want to start X on booting. I clicked Yes. (And the mouse worked here too).
37. Congratulations! Installation is complete.
38. Now reboot the PC after taking the floppy out of the drive.
Text Mode

1. Installation Class: select Custom Workstation.

Partition and Format Hard Disk

2. Disk Druid utility is used to partition and format the hard disk.

Modify Partitions

If you have other valid partitions (e.g., QNX partition on /dev/hda2), then ensure that you do not delete the other partitions. You may want to delete the Linux native partition and add it again as a new partition. Do not change the Swap partition.

3. Disk Druid shows a list of Current Partitions.
4. Select the appropriate partition (e.g., /dev/hda1), and click on Edit to modify properties.
5. Enter the mount directory (e.g., "/").
6. You may have another Linux native partition that may be mounted on ("/home"). Hit OK.

New Partitions

7. Add New Partition: Mount Point = / (which means root directory), Size (Megs) = 2323, Type = Linux Native; Hit OK (/dev/hda1).
8. Add New Partition: Mount Point = don't type, Size (Megs) = 118, Type = Linux Swap. Now, the value of the Mount Point field will be automatically set to "Swap Partition". Hit OK. (/dev/hda5).
9. Active Swap Space: What partitions would you like to use for swap space? I selected /dev/hda5. I also checked the box, "Check for bad sectors while formatting".
10. Partitions to Format: What partitions would you like to format. The only option is /dev/hda1 which I selected. I may want to check the box, "Check for bad sectors while formatting".

LILO Configuration

11. A few file systems will need to pass special options to the kernel at boot time for the system to function properly. If you need to pass boot options to the kernel, enter it now. If you don’t need any or are not sure, leave it blank. So, I left it blank and hit OK.
12. Where do you want to install LILO? Two options, /dev/hda (master boot record) or /dev/hda1 (first sector of boot partition). I selected the default of master boot record.
13. Select the operating system to be used by the Boot Manager. Select Linux.
Network Configuration

14. Hostname: "crb07.ces.clemson.edu".
15. In Network Configuration:
   - Deselect Bootp/DHCP.
   - IP Address: "130.127.172.195".
   - Netmask: "255.255.255.0".
   - Default Gateway: "130.127.172.2".
   - Primary Nameserver: "130.127.37.11".

Hardware Identification

16. What type of Mouse do you have? I selected "Generic 2 button Serial mouse".
17. Select the Time-Zone: "US/Eastern".
18. Set Hardware Clock to GMT: "No".

User Account Creation

19. Pick a Root Password, and re-enter it.
20. Authentication Configuration. Do not check "Enable NIS". Check "Use Shadow Passwords" and "Enable MD5 Passwords".

Package Group Selection

22. You may want to check the Select Individual Packages option and install specific components of each of the above packages. Do this only if you are certain that the components you are not installing do not affect normal operation. It is not required to check the Select Individual Packages box.

Video/Monitor Setup

23. Create a boot disk: "No."
24. PCI Probe: found PCI Entry: Number Nine Revolution 3D (T2R) (this is the video card)
25. Monitor Setup: What type of Monitor do you have? "View Sonic E771." If your monitor is not listed, select "Custom". You will have to specify the horizontal scan rate (e.g., 50-90) and the screen size (e.g., 1024 x 768).
26. Screen Configuration: Xconfigurator will setup the default resolution and color depth by probing the PCI video card. The screen will blink several times.
27. Probing Finished: Xconfigurator successfully probed the video card.
28. Xconfigurator Probing: For the Revolution 3D and the ELSA Erazor II boards, the Xconfigurator returns a failure message. Hit "Skip". This is because of a faulty driver in the RedHat Linux 6.1 CD.
For the Revolution 3D board, you will have to manually download the older file from RedHat Linux 6.0 CD (XFree86-I128-3.3.3.1-49) or obtain the latest version from the Update directory of the XFree for I128 (from www.xfree.org).

For the ELSA Erazor II video board, you need to install the SVGA driver (XFree86-SVGA-3.3.5-3.rpm)

29. The installer will format the file system and install the selected packages.
30. Congratulations! Installation is complete.
31. Now reboot the PC after taking the floppy out of the drive.
32. However, you have not successfully configured your video card/monitor.
33. Boot up into text mode.
34. Copy the XFree86-I128-3.3.3.1-49.rpm to any directory as follows:
   - mount /dev/cdrom
   - cp /mnt/cdrom/RedHat/RPMS/Xfree86-I28-3.3.3.1-49.rpm /tmp
   - cd /tmp
35. rpm -i --force XFree86-I128-3.3.3.1-49.rpm to install the correct rpm.
36. cd /usr/src/linux
37. Xconfigurator, and select the appropriate values for your setup.
38. Once the probing is completed (as in steps 24 to 27), it will give you a default setting of 32 bit color depth at 1024 x 768. Select "Let Me Choose" and select the 16 bit color depth at 1280 x 1024.
39. Reboot and X windows will automatically startup at run level 5.
40. Note that if you are having X window problems, you can boot in text mode by typing at the LILO prompt: linux 3, where 3, indicates run level 3.
41. You have now finished installation.
Network Problems

1. If you have network problems (i.e., you cannot ping, or telnet).

Hardware Check

2. Check if the hardware connections are correct, and ensure that the hub and the back of you network card show activity (i.e., LEDs blink irregularly).

Configuration Check

3. cd /etc
4. vi resolv.conf
domain ces.clemson.edu
search ces.clemson.edu clemson.edu
nameserver 130.127.37.11
nameserver 130.127.200.15
nameserver 130.127.8.8
5. vi hosts
   127.0.0.1 localhost.localdomain localhost crb07.ces.clemson.edu
   130.127.172.17 crb07.ces.clemson.edu

Driver Check

6. Ensure that you have the correct device driver (.c file). For the new Kingston networks cards, the version of tulip.c is 104241 bytes. The old version is 91540 bytes, and will not work with the new network cards.
7. To copy the latest driver, you must have the floppy disk with the latest tulip.c in the drive before you mount it.
   • cd /usr/src/linux/drivers/net
   • cp tulip.c tulip.c.orig
   • rm tulip.o (this file may or may not exist)
   • mount -t msdos /dev/fd0 /mnt/floppy
   • cp /mnt/floppy/tulip.c tulip.c
8. After changing the network driver, you will have to rebuild the kernel. It is also a good time to reconfigure the kernel using (make xconfig).
9. cd /usr/src/linux
10. make xconfig
11. In the Linux Kernel Configuration Window that appears, click on Ethernet (10 or 100 Mbit) button
12. Ensure that Ethernet is activated (i.e., "y" is checked).
13. Scroll down and ensure that EISA, VLB, PCI and on board controllers is activated.
14. Ensure that DECchip Tulip (dc21x4x) PCI support is activated.
15. Also check the Networking options and Network device support.

Rebuild the Kernel

16. If you have had to change anything, then click on Save and Exit.
17. Now you have to rebuild the kernel.
18. `cd /usr/src/linux`
19. `make clean`
20. `make dep` (dependencies).
21. `make bzImage` (creates the new kernel image in `/usr/src/linux/arch/i386/boot`).
22. `cp /usr/src/linux/arch/i386/boot/bzImage /boot/bzImage` (copy this image to the boot directory).

**Reconfiguring the LILO**
23. `cd /etc`
24. `vi lilo.conf`
25. Change from "image=/boot/vmlinuz-2.2.12-20" to "image=/boot/bzImage".
26. Save the file.
27. `lilo`
   Added linux * (will be displayed).
28. Reboot the PC into linux.
29. Your network should work. If it still does not work.
30. Click on the Gnome start button (it is in the same place where the Windows start button would be), then click on System and Control Panel.
31. Click on the network icon.
32. In the dialog window, click on Interfaces.
33. Ensure that eth0 is active. If not, activate it, and reboot.
34. Your network should now work.
Appendix B: Installing Pre-Patched RT-Linux (v2.0)

Preliminaries

1. You must be root.
2. It is very important to note that depending on your shell, you can execute a script in your current directory in either of the two ways
   
   `script` or
   
   `./script`

Extracting the Linux 2.2.13 Kernel pre-patched with RT-Linux 2.0

3. Obtain the `rtlinux-2.0-prepatched.tgz` file from QRTS. You may ftp it or obtain it on a cdrom. Ftp it into any directory.
4. `cp rtlinux-2.0-prepatched.tgz /usr/src`
5. `tar -xzvf rtlinux_package.tgz`
6. `cd /usr/src`
7. `rm linux` (deletes the symbolic link to linux-2.2.12)
   
   `/usr/src/rtlinux-2.0` contains among other directories, linux and rtl. Use these directories for booting (instead of `/usr/src/linux` which is a symbolic link to the older `/usr/src/linux-2.2.12 kernel`).
8. Create the following symbolic links
   
   `ln -s ./rtlinux-2.0/linux linux`
   
   `ln -s ./rtlinux-2.0/rtl rtl`

Configuring and Building New Kernel Image

9. `cd /usr/src/linux`
10. `make xconfig`
11. This will open up the Linux Kernel Configuration (note these settings reflect the configuration on crb03 and not the factory default).
12. Click on Processor type and features: The processor type is configured to be a Pentium processor family. For AMD K6 and K6-3D, select Pentium/K6/TSC. Also, ensure that the ‘Yes’ radio button is checked for the Hard Real-Time option and the ‘No’ radio button is checked for the Symmetric Multiprocessing option.
13. Click on Loadable Module Support: Ensure that the ‘No’ radio button is selected for the Set Version Information on All Symbols for Modules.
14. Ensure that your Video Card and Network Card are properly selected.
15. Also, ensure the Hard Real Time Support is "Yes".
16. If you had network driver problems when installing RedHat Linux 6.1, then you must repeat the entire Network Problem Solution procedure before building this new kernel image (see last section of Installing RedHat Linux 6.1 manual).
17. `make clean`
18. `make dep` (dependencies).
19. `make bzImage` (creates the new kernel image in `/usr/src/linux/arch/i386/boot/`).
20. `cp /usr/src/linux/arch/i386/boot/bzImage /boot/rtl2Image` (copy this image to the boot directory).

### Configuring the LILO

21. `vi /etc/lilo.conf`

   Find the line `timeout=50`
   Add these lines
   ```
   image=/boot/rtl2Image
   <tab>label=rtllinux (this is the name it will show at LILO during boot)
   <tab>root=/dev/hda1 (this is where your Linux stuff is installed)
   <tab>read-only
   <tab>append="mem=127M" (this is to set 1 MB shared memory pool for RT-Linux; the number here is the total size of the main memory - 1 MB)
   ```

22. Save this file.

23. `lilo` (activate this new configuration).

24. You should see the following
    
    Added rtllinux *
    Some other message may also be displayed

25. `reboot` into rtlinux.

### Installing RT-Linux Modules

26. `cd /usr/src/rtl`

27. `make all` (makes all the RT-Linux modules).

28. `make install` (copies RT-Linux modules into `/lib/modules/2.2.13-RTL2.0/misc` directory and creates FIFO).

29. `cd /lib/modules/2.2.13-RTL2.0/misc`

30. `rtl_time.o, rtl_sched.o, rtl_fifo.o and rtl_posixio.o` should be loaded into the kernel before RT-Linux tasks can run.
    
    It is recommended to load them every-time the system boots up. You can do this by creating a script file with the load commands in the file `/etc/rc.d/rc.local`

31. In the script file append at the bottom (in THIS order) of the file `rc.local`

    ```
    insmod -f /lib/modules/2.2.13-RTL2.0/misc/rtl_time.o
    insmod -f /lib/modules/2.2.13-RTL2.0/misc/rtl_sched.o
    insmod -f /lib/modules/2.2.13-RTL2.0/misc/rtl_posixio.o
    insmod -f /lib/modules/2.2.13-RTL2.0/misc/rtl_fifo.o
    ```
    
    Save this file.

32. `cd /boot`

33. `cp /usr/src/linux/System.map System.map.2.2.13-RTL2.0`

    By default System.map is the symbolic link to the System map for 2.2.12 kernel. We need to remove this symbolic link and make a new one for the 2.2.13-RTL2.0 kernel.

34. `rm System.map`

35. `ln -s System.map.2.2.13-RTL2.0 System.map`

36. `cd /usr/src/linux`

37. `make dep` (you do not have to rebuild the image).

38. `reboot` into RT-Linux.
This completes the installation of RT-Linux

**Configuring Shared Memory**

**Shared Memory Pool Configuration**

Shared Memory Pool is a block of physical memory that is reserved during boot up. RTLT uses a shared memory pool to facilitate the implementation of hardware client-server architecture.

**Set Shared Memory Pool**

The Shared Memory Pool is reserved at the top of the physical memory by passing a "mem=" argument to the kernel during boot up. To this end, the user must edit the configuration file `lilo.conf` in the `/etc` directory. Add a line with the "append" keyword. The size of the Shared Memory Pool should be less than 4 MB in an Intel Pentium-class or above; furthermore, it should be a multiple of 4 K (4096) so that it can be mapped into user space.

For example, if the system has 128 MB of physical memory with 1 MB of shared memory, then the Linux kernel will use 128 - 1 = 127 MB of physical memory and the base address of the shared memory pool will be 0x7F00000. The relevant part of the `/etc/lilo.conf` file will contain

```
image=/boot/bzImage
label=rtlinux
root=/dev/hda1
read-only
```

Add the line with the "append" keyword. The relevant part of the `/etc/lilo.conf` file will now read as follows

```
image=/boot/bzImage
label=rtlinux
root=/dev/hda1
read-only
append="mem=127M"
```

Save the file and type the following to activate the new configuration, and then reboot the PC to reserve the Shared Memory Pool.

```
# lilo
```

**Shared Memory Manager**

The shared memory manager should be loaded once the shared memory pool is setup. The shared memory manager module `shmmgmt.o` is loaded by typing the following at the Linux command prompt

```
# insmod shmmgmt.o shm_info=<Starting Address>,<Size>
```

where `<Starting Address>` refers to the starting address of the shared memory pool and is typed in hexadecimal (`e.g., 0x7F00000`). The `<Size>` (`e.g., 1048576`) refers to the actual size of the shared memory pool. Usually, a 1MB size of shared memory pool is sufficient.

It should be noted that the values of these two parameters are based on the actual system configuration.
Also, a sample script file is available at $QRTS/modules/loadshmm

The user should set the system up to load prerequisite modules every time the PC boots up into RT-Linux, the user can add them into a script file /etc/rc.d/rc.local

**Configuration Examples**

Users should change the parameters according to the shared memory configuration.

Note: The size of Shared Memory Pool should be less than 4M in Pentium class or higher system. Looks like 4190208 bytes in the biggest. However, 1M is enough most of the time.

**Script for 1M Shared Memory out of 128M Main Memory**

```
insmod shmmgnt.o shm_info=0x7F00000,1048576
```

In *lilo.conf*:

```
append="mem=127M"
```

**Script for 3M Shared Memory out of 128M Main Memory**

```
insmod shmmgnt.o shm_info=0x7D00000,3145728
```

In *lilo.conf*:

```
append="mem=126980K"
```

**Script for 4M-4K(4092K) Shared Memory out of 128M Main Memory**

```
insmod shmmgnt.o shm_info=0x7C01000,4190208
```

In *lilo.conf*:

```
append="mem=126980K"
```

**Script for 1M Shared Memory out of 64M Main Memory**

```
insmod shmmgnt.o shm_info=0x3F00000,1048576
```

In *lilo.conf*:

```
append="mem=63M"
```

**Script for 4M-4K(4092K) Shared Memory out of 64M Main Memory**

```
insmod shmmgnt.o shm_info=0x3C01000,4190208
```

In *lilo.conf*:

```
append="mem=61444K"
```

**Script for 1M Shared Memory out of 32M Main Memory**

```
insmod shmmgnt.o shm_info=0x1F00000,1048576
```

In *lilo.conf*:

```
append="mem=31M"
```
**RTLT Licenses**

The QRTS hardware key protects RTLT. The device must be plugged to a parallel port at all times. The license daemon that is included in the base QRTS package must be started during boot up. For example, if $QRTS=/usr/qrts, then the user appends the following line to the end of the `/etc/rc.d/rc.local` file:

```
/usr/qrts/licenses/qrtslicense start
```

**Setup Shared Memory Pool**

The user must know how much memory is available in the PC and determine how much is to be reserved for use as the shared memory pool (usually 1MB is sufficient).

- **Pass argument** "mem=<size>" **to kernel**, where
  
  `<size>` = Total memory - shared memory.

- **For example**, if you have 128M memory and 1M shared memory, then `mem=127M`.

- **Open `/etc/lilo.conf`**, add this line to the RT-Linux boot section:
  
  `append="mem=127M"
  
- **Run the LILO command** `lilo`

- **Reboot the PC.**

- **Load the QRTS shared memory management module (`shmmgmt.o`).** This module can be found in `$QRTS/modules`.
  
  `insmod shmmgmt.o shm_info=<baseAddress of SharedMemory Pool>,<Size>`

  - **baseAddress** is the start address of the shared memory pool
  - **Size** is the size, in byte, of the shared memory pool

  **For example**, if you use 1MB shared memory out of 128MB main memory:
  
  `insmod shmmgmt.o shm_info=0x7F00000,1048576`

  A sample script `loadshmm` can be found with the shared memory management module.

**License Issue**

Sample commands that constitute the last lines of the file `/etc/rc.d/rc.local` are shown below:

```
insmod /lib/modules/2.2.13-RTL2.0/misc/rtl_time.o
insmod /lib/modules/2.2.13-RTL2.0/misc/rtl_sched.o
insmod /lib/modules/2.2.13-RTL2.0/misc/rtl_posixio.o
insmod /lib/modules/2.2.13-RTL2.0/misc/rtl_fifo.o
insmod /usr/qrts/modules/shmmgmt.o shm_info=0x7F00000,1048576
```

`/usr/qrts/licenses/qrtslicense start`

The above file assumes the existence of the MultiQ SDK, a total RAM of 128MB, and a shared memory of 1 MB.
Appendix C: MATLAB 5.3 - Red Hat Linux 6.1 Compatibility Issues

There is some incompatibility between MATLAB 5.3 and Red Hat 6.1 because MATLAB 5.3 was constructed with the Red Hat 4.2 development environment. Specifically, Red Hat 4.2 environment used version 5 of the C and Math libraries while version 6.0 of Red Hat uses version 6 of the C and Math libraries. To ensure compatibility, the user should follow the procedure outlined below.

The installation script of RTLT is configured to automatically fix this incompatibility. During RTLT installation, the user types ‘y’ to select this automatic fix option. If the user types ‘n’, then QRTS provides a script file that may be executed later to fix the incompatibility. This script file is called `rtlt_setup_libc5` in the directory `$QRTS/bin/`

The following details are provided for interested users.

**Step 1: Obtaining a Libc5 Development Environment**

You will need to be root to perform this installation. Please read all the instructions carefully before proceeding. These packages may be copied off the RTLT installation package provided. Alternatively, the user may ftp it off a QRTS web-site. For more details on how to ftp from a QRTS web-site, contact Quality Real-Time Systems at http://www.qrts.com.

The following packages are required:
- `gcc-libc5-2.7.2.3-1.i386.rpm`
- `libc5-devel-5.4.38-3.i386.rpm`
- `libc5-5.4.38-3.i386.rpm`
- `gcc-libc5-c++-2.7.2.3-1.i386.rpm` (optional, only for g++)

Note that only one file in the above packages overlaps with an existing file. The user may make a copy as follows:

```
# cp /etc/nsswitch /etc/nsswitch.orig
```

Since the directory names in the packages differ slightly, it is recommended that a link be created, thus ensuring all files exist in one place. Also, the first directory listed below should already be named in `/etc/ld.so.conf`, so if the user makes the link shown below, the user will not have to modify that file.

```
# ln -s /usr/i486-linux-libc5 /usr/i486-linuxlibc5
```

Now install the packages as follows:

```
# rpm -i --force gcc-libc5-2.7.2.3-1.i386.rpm
# rpm -i --force libc5-devel-5.4.38-3.i386.rpm
# rpm -i --force libc5-5.4.38-3.i386.rpm
```
The --force option ensures that existing installations are overwritten.

The specifications for this gcc are broken and use the wrong ld-linux.so. To see this, try to compile a simple program

```c
#include <stdio.h> /* not a Linux command */

void main(void)
{
    printf("Hello World.\n");
}
```

Compile the above file as follows

```
# i486-linuxlibc5-gcc -o hello hello.c
```

And run

```
# ./hello
```

This will generate the error message, "libc.so.5 cannot open shared object file" because it has been linked against the wrong ld-linux.so file. To ascertain the version of the ld-linux.so file, type

```
# strings hello | grep ld-linux
```

This will display /lib/ld-linux.so.2, which is wrong. To fix it, type

```
# cd /usr/lib/gcc-lib/i486-linuxlibc5/2.7.2.3
# mv specs specs.orig
# cat specs.orig | sed -e "s/ld-linux.so.2/ld-linux.so.1/" > specs
```

At this point the user should be able to build libc5-compatible binaries.

```
# i486-linuxlibc5-gcc -o hello hello.c
# ./hello
```

"Hello World!!" will now be displayed on the screen.

**Step 2: Building MEX files with a Libc5 Development Environment**

In mexopts.sh, replace "gcc" with "i486-linuxlibc5-gcc" in the lnx86 section.

```bash
CC='i486-linuxlibc5-gcc'
LD='i486-linuxlibc5-gcc'
```

Also, make the above changes to the following files in /usr/local/matlab/bin:
gccopts.sh, cxxopts.sh, mbuildopts.sh, matopts.sh, and engopts.sh.

Ensure that if the user program links with other libraries (like libdl, or libjpeg) that the programs are actually linking against libc5 versions of these libraries. (Hint: put them in /usr/i486-linux-libc5/lib, and make sure that directory appears at the top of ld.so.conf.)

As a final check on any mex file, type the following command

```bash
>> !/usr/bin/ldd mexfile.lnx86
```

Now look for dependencies on libc.so.6 or libm.so.6. If present, the mex file will probably not work. Find out what library is introducing this dependency and replace it with a libc5 version.
Appendix D: RTLT Supported I/O Boards

MultiQ I/O Board
The MultiQ-2 board is an ISA bus data acquisition and control that is equipped with the features listed below. The MultiQ-3 board has the same features as the MultiQ-2 board with up to 8 quadrature encoder inputs.

- 8 multiplexed analog inputs, 13 bit, +/- 5 volts range
- 8 analog outputs, 12 bit, +/-5 volts range
- 8 digital inputs
- 8 digital outputs
- Up to 6 quadrature encoder inputs into 24 bit counters (optional) with A and B inputs
- 3 real-time clocks that can tie to interrupts on the bus

ServoToGo I/O Board
The ServoToGo Model 1 board is an ISA bus data acquisition and control that is equipped with the features listed below. The ServoToGo Model 2 board has the same features as the Model 1 board with improved implementation of the index pulse functions and the watchdog timer.

- 8 analog inputs, 13 bit, configurable as +/- 10 volts or +/- 5 volts
- 8 analog outputs, 13 bit, +/- 10 volts with a sign bit digital output that is Opto-22 compatible
- 8 digital inputs, 32 bit
- 8 digital outputs, 32 bit
- Up to 8 quadrature encoder inputs into 24 bit counters (extendible to 32 bits in software) with A, B, and index signal inputs; the index pulse can be configured to be normally high or normally low
- Interval and Watchdog timer

Mini PMAC Board
The Mini PMAC Board is an ISA bus data acquisition and control board with an onboard 20 MHz DSP CPU and PLC programming capability, and has the following features

- Flash-backed memory
- 2 optically isolated 16-bit DAC outputs
- Four 3-channel single-ended/differential quadrature encoder inputs
- 2 sets of optically isolated flag inputs, 12V to 24V, sinking or sourcing
- Display port for liquid crystal and vacuum-fluorescent displays
- "Thumbwheel" port for multiplexed I/O accessories or 16 direct TTL I/O
- "JS1" port for ACC-28 A/D converter interface
- "Opto" port for 16 general-purpose I/O, 5V to 24V levels
- Optional 2 V/F converters
- Optional 2 V/F converters
- Can be used as analog inputs (12-bit resolution)
- Can be used for pulse outputs for stepper drives
- Optional on-board DPRAM
- Optional buffered expansion port for high-speed communication to accessory boards
- Standard 60-pin JMACH connector compatible with other PMACs (some open pins)

**ATI Force/Torque Sensor System**

The ATI Industrial Automation ISA bus interface card provides an easy interface between a computer and an ATI Industrial Automation force/torque sensor. The high speed Digital Signal Processor (DSP)-based interface card performs operations similar to ATI's stand-alone RS-232-based controller at a much faster rate. Digital discrete I/O capability is available for signaling user-programmed F/T thresholds. The ATI F/T Sensor system has the following features:

- Overload protection: The F/T transducer is extremely rugged and durable. All transducer models can withstand high overload.
- High signal-to-noise ratio: Silicon gages provide a signal 75 times stronger than conventional foil gages. This signal is amplified resulting in near-zero noise distortion.
- High-speed output of over 3000 Hz with ISA controller exceeds requirements for most applications.
- Tool transformations translate and/or rotate the F/T reference frame.
- Peak analysis allows maximum and minimum F/T values to be detected and stored.
- Biasing provides a convenient way to offset tool weight.
- Increased system throughput is possible by reducing the number of axis output.
- Programmable F/T thresholds provide fast-response F/T condition monitoring.
- Temperature compensation option insures accuracy over a wide temperature range.

**PCL812 Board**

The Advantech PCL-812PG is a high-performance, multi-function data acquisition card for PCs. Key features include:

- 16 single-ended analog input channels
- An industrial standard 12-bit successive approximation converter (HADC574Z) to convert analog inputs. The maximum A/D sampling rate is 30 KHz in DMA mode
- Software programmable analog bipolar input rates, +/- 10V, +/- 5V, +/- 2.5 V, +/- 1.25 V, +/- 0.625V and +/-0.3125V
- Three A/D trigger modes: software trigger, programmable pacer trigger, and external pulse trigger
- Intel 8253-5 programmable timer/counter provides pacer output (trigger pulse) at the rate of 0.5 MHz to 35 minutes/pulse. The timer time base is 2 MHz. One 16-bit counter channel is reserved for user configuration applications
- Two 12-bit monolithic multiplying D/A output channels. An output range from 0 to +5V or 0 to +10V can be created by using the on-board -5V or -10V reference. The precision reference
is derived from the A/D converter reference. External AC or DC references can be used to generate other D/A output ranges.

- 16 TTL/DTL compatible digital input, and 16 digital output channels
Appendix E: Loading I/O Board Server Modules

This document presents a systematic procedure to load I/O board server modules.

The following I/O board server modules are supported:

- MultiQ-2 and MultiQ-3 I/O board server modules
- ServoToGo S-8 I/O board server module
- ATI Force Torque board server module
- PMAC I/O board server module
- Advantech PCL812 I/O board server module

Loading the MultiQ-2 Server Module

The MultiQ-2 Server takes the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>String</td>
<td>Iobs0</td>
<td>IOBoard Server Name</td>
</tr>
<tr>
<td>baseAddress</td>
<td>Integer</td>
<td>0x320</td>
<td>MultiQBoard Base Address</td>
</tr>
<tr>
<td>numOfADC</td>
<td>Integer</td>
<td>8</td>
<td>Number of A to D Channels used</td>
</tr>
<tr>
<td>numOfDAC</td>
<td>Integer</td>
<td>8</td>
<td>Number of D to A Channels used</td>
</tr>
<tr>
<td>numOfDigin</td>
<td>Integer</td>
<td>8</td>
<td>Number of Digital Input channels used</td>
</tr>
<tr>
<td>numOfDigout</td>
<td>Integer</td>
<td>8</td>
<td>Number of Digital Output channels used</td>
</tr>
<tr>
<td>numOfEncoder</td>
<td>Integer</td>
<td>8</td>
<td>Number of Encoder Channels used</td>
</tr>
<tr>
<td>frequency</td>
<td>Integer</td>
<td>1000</td>
<td>Running Frequency of the MultiQ2 Server</td>
</tr>
</tbody>
</table>

If no parameters are specified, default values will be used. The user can always assign new values to these parameters.

For example,

```
insmod multiq2.o -o server0 name=server0
insmod multiq2.o -o server1 name=server1 baseAddress=0x330 numOfADC=6 frequency=2000
insmod multiq2.o -o myDriver name=myDriver frequency=100
```

To load the multiq2 server with default value, you may simply type,

```
insmod multiq2.o -o iobs0
```

When you `lsmod`, the module name will be listed as "iobs0"
Loading the MultiQ-3 Server Module

MultiQ3 Server takes the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>String</td>
<td>Iobs0</td>
<td>IOBoard Server Name</td>
</tr>
<tr>
<td>baseAddress</td>
<td>Integer</td>
<td>0x320</td>
<td>MultiQBoard Base Address</td>
</tr>
<tr>
<td>numOfADC</td>
<td>Integer</td>
<td>8</td>
<td>Number of A to D Channels used</td>
</tr>
<tr>
<td>numOfDAC</td>
<td>Integer</td>
<td>8</td>
<td>Number of D to A Channels used</td>
</tr>
<tr>
<td>numOfDigin</td>
<td>Integer</td>
<td>8</td>
<td>Number of Digital Input channels used</td>
</tr>
<tr>
<td>numOfdigout</td>
<td>Integer</td>
<td>8</td>
<td>Number of Digital Output channels used</td>
</tr>
<tr>
<td>numOfEncoder</td>
<td>Integer</td>
<td>8</td>
<td>Number of Encoder Channels used</td>
</tr>
<tr>
<td>frequency</td>
<td>Integer</td>
<td>1000</td>
<td>Running Frequency of the MultiQ3 Server</td>
</tr>
</tbody>
</table>

If no parameters are specified, default values will be used. The user can always assign new values to these parameters.

For example,

insmod multiq3.o -o server0 name=server0

insmod multiq3.o -o server1 name=server1 baseAddress=0x330 numOfADC=6 frequency=2000

insmod multiq3.o -o myDriver name=myDriver frequency=100

To load the multiq3 server with default value, you may simply type,

insmod multiq3.o -o iobs0

When you `lsmod`, the module name will be listed as "iobs0"

Loading the ServoToGo Server Module

ServoToGo server takes these parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>String</td>
<td>Iobs0</td>
<td>IOBoard Server Name</td>
</tr>
<tr>
<td>boardIndex</td>
<td>Integer</td>
<td>0</td>
<td>IOBoard actually used</td>
</tr>
<tr>
<td>baseAddress</td>
<td>Integer</td>
<td>0x320</td>
<td>MultiQBoard Base Address</td>
</tr>
<tr>
<td>numOfADC</td>
<td>Integer</td>
<td>8</td>
<td>Number of A to D Channels used</td>
</tr>
<tr>
<td>numOfDAC</td>
<td>Integer</td>
<td>8</td>
<td>Number of D to A Channels used</td>
</tr>
</tbody>
</table>

If no parameters are specified, default values will be used. The user can always assign new values to these parameters.

For example,

insmod multiq3.o -o server0 name=server0

insmod multiq3.o -o server1 name=server1 baseAddress=0x330 numOfADC=6 frequency=2000

insmod multiq3.o -o myDriver name=myDriver frequency=100

To load the multiq3 server with default value, you may simply type,

insmod multiq3.o -o iobs0

When you `lsmod`, the module name will be listed as "iobs0"
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td>fts0</td>
<td>ATI-FT Server Name</td>
</tr>
<tr>
<td>baseAddress</td>
<td>Integer</td>
<td>0x280</td>
<td>ATI-FT Board Base Address</td>
</tr>
<tr>
<td>frequency</td>
<td>Integer</td>
<td>1000</td>
<td>Running Frequency of the ATI-FT Server</td>
</tr>
<tr>
<td>mask</td>
<td>Integer</td>
<td>0x3F</td>
<td>Data mask of the ATI-F/T board</td>
</tr>
<tr>
<td>autoReset</td>
<td>Integer</td>
<td>1</td>
<td>If autoReset=1, the bias will be reset to zero every time the ATIFT server is loaded. If autoReset=0, then the auto-reset feature is disabled</td>
</tr>
</tbody>
</table>

If no parameters are specified, default values will be used. The user can always assign new values to these parameters.

For example,

```
insmod servo2go.o [-o <module name>] [ parameter1=value1] [parameter2 = value2]
```

```
insmod -o iobs0 servo2go.o adcRange=10
```

**Loading the ATI-FT Server Module**

ATI-FT Server takes the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td>fts0</td>
<td>ATI-FT Server Name</td>
</tr>
<tr>
<td>baseAddress</td>
<td>Integer</td>
<td>0x280</td>
<td>ATI-FT Board Base Address</td>
</tr>
<tr>
<td>frequency</td>
<td>Integer</td>
<td>1000</td>
<td>Running Frequency of the ATI-FT Server</td>
</tr>
<tr>
<td>mask</td>
<td>Integer</td>
<td>0x3F</td>
<td>Data mask of the ATI-F/T board</td>
</tr>
<tr>
<td>autoReset</td>
<td>Integer</td>
<td>1</td>
<td>If autoReset=1, the bias will be reset to zero every time the ATIFT server is loaded. If autoReset=0, then the auto-reset feature is disabled</td>
</tr>
</tbody>
</table>

If no parameters are specified, default values will be used. The user can always assign new values to these parameters.

For example,

```
insmod atift.o -o iobs0 name=iobs0
```
insmod atift.o -o iobs0 name=iobs0 frequency=100
insmod atift.o -o iobs0 name=iobs0 baseAddress=0x300 frequency=100
insmod atift.o -o iobs0 name=iobs0 autoReset=0

To load the ATI-FT server with default value, you may simply type,

insmod atift.o -o fts0

When you `lsmod`, the module name will be listed as "fts0"

**Loading the PMAC Server Module**

PMAC Server takes the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td>iobs0</td>
<td>PMAC server name</td>
</tr>
<tr>
<td>baseAddress</td>
<td>Integer</td>
<td>0x210</td>
<td>The base address of the PMAC board</td>
</tr>
<tr>
<td>frequency</td>
<td>Integer</td>
<td>100</td>
<td>The frequency at which to update I/O</td>
</tr>
<tr>
<td>numAdc</td>
<td>Integer</td>
<td>0</td>
<td>The number of ADC channels to use</td>
</tr>
<tr>
<td>numDac</td>
<td>Integer</td>
<td>2</td>
<td>The number of DAC channels to use</td>
</tr>
<tr>
<td>numEncoders</td>
<td>Integer</td>
<td>4</td>
<td>The number of Encoder channels to use</td>
</tr>
</tbody>
</table>

If no parameters are specified, default values will be used. The user can always assign new values to these parameters.

For example,

insmod pmac.o -o iobs0 name=iobs0 frequency=250

**Loading the PCL812 Server Module**

The PCL812 server takes these parameters:

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>baseAddress</td>
<td>0x220</td>
<td>The base address of the board to use.</td>
</tr>
<tr>
<td>adcRange</td>
<td>5</td>
<td>The ADC input range (in volts) as set by jumper JP9</td>
</tr>
<tr>
<td>dac1Reference</td>
<td>5000</td>
<td>The reference voltage for DAC 1 (in MILLIVOLTS). If JP3 is set to INT, pass in 5000 or 10000 depending on the setting of JP8. If JP3 is set to EXT, pass in the value of the external reference voltage. This parameter is ignored by the Feedback-specific modules.</td>
</tr>
<tr>
<td>dac2Reference</td>
<td>5000</td>
<td>The reference voltage for DAC 2 (in MILLIVOLTS). If JP4 is set to INT, pass in 5000 or 10000 depending on the setting of JP8. If JP4 is set to EXT, pass in the value of the external reference voltage. This parameter</td>
</tr>
</tbody>
</table>
is ignored by the Feedback-specific modules.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>adcGain</td>
<td>1</td>
<td>The desired gain for the ADC preamplifier. Must be 1, 2, 4, 8, or 16. Gains other than 1 will reduce the input range by an appropriate amount (e.g. adcRange=5, adcGain=2 means the input signal must be between -2.5v and 2.5v).</td>
</tr>
<tr>
<td>frequency</td>
<td>100</td>
<td>The frequency at which to update I/O.</td>
</tr>
<tr>
<td>name</td>
<td>iobs0</td>
<td>The name of the server.</td>
</tr>
<tr>
<td>numAdc</td>
<td>16</td>
<td>The number of ADC channels to use. The default is 16. Using fewer channels will allow higher sample frequencies.</td>
</tr>
</tbody>
</table>

If no parameters are specified, default values will be used. The user can always assign new values to these parameters.

For example,

Load the server for the Feedback, Inc., pendulum experiment as iobs0:

```bash
insmod -o iobs0 feedback_pend.o
```

Load the server with the encoder interface for the Feedback Inc., experiment assuming +/-10v ADC input setting:

```bash
insmod -o iobs0 feedback_enc.o adcRange=10
```

Load the standard server as iobs1 with a sample rate of 1000Hz:

```bash
insmod -o iobs1 pcl812.o frequency=1000 name=iobs1
```
Appendix F: Windows 98, QNX and Linux Multiple Boot Installation

This document supplements the Microsoft Windows 98 Installation Manual (available with Microsoft Windows), the QNX Installation Manual (available with QNX OS), and the Linux Installation Manual (free download at: http://qrts.com/manuals/rtlinuxtarget/htm/appendix_install_linux6-1.htm)

Preliminaries

In order to boot into any operating system (Windows 98, QNX, and/or Linux), the bootable partition of the operating system (i.e., the ‘C:\’ drive for Windows 98, everything except the /home directory for Linux/QNX partition) should be present within the first 1024 cylinders of the hard disk.

For a multiple boot installation the operating systems MUST be installed in the following order:

1. Windows 98: 4 GB is sufficient.
2. QNX: 2 GB is sufficient for QNX, QMotor, and related software
3. Linux (uses LILO): at least 2 GB is required for Linux, RT-Linux, RTLT, and related software. An extended partition may be created to mount the /home directory. Note that proper Linux installation is a pre-requisite for RT-Linux and RTLT installation.

When installing a triple boot system of Windows 98, QNX, and Linux, it is necessary to install the operating systems in the order mentioned above because it is simpler to configure LILO Loader in Linux to boot into Windows 98, QNX, Linux, and RT-Linux. The procedure of configuring the QNX loader to allow for boot up options in Windows 98 and/or Linux is more involved and should be avoided. The user should also ensure that during Linux installation, the LILO is written to the Master Boot Record.

When installing a double boot system of QNX and Linux, the Windows 98 Installation procedure is simply neglected. That is, the user starts with the QNX installation and proceeds as per the instructions given in this document. Upon completion of the QNX installation, the user installs Linux followed by RT-Linux.

When installing a single boot system of QNX or Linux, the user follows directly, the instructions given in the QNX and Linux installation manuals, respectively.

Windows 98 Installation

1. After switching the PC on, the user hits the <del> or <F5> key to enter into the CMOS setup
2. The user then sets the boot-seek order to CDROM, Floppy, then Hard Disk (contact the system administrator for information specific to particular CMOS settings)
3. The user now inserts the Windows 98 Installation CD into the CDROM drive.
4. The user then selects the Save and Exit option to write the new settings to the CMOS and the system will reboot.
5. Windows 98 installation is automatically initiated.
6. When prompted, the user provides the size of the Windows 98 installation (for example, 2 GB). Note that a partition per se is not created but only a part of the hard disk is used for Windows. The unused space in the hard disk is then partitioned in QNX and later in Linux installations. The Windows 98 installation appears as a partition /dev/hd0 in QNX and /dev/hda1 in Linux installations.
7. Follow the Windows 98 installation guide to proceed and complete installation.

The user should note that the Windows 98 partition is of the FAT32 (LBA) type (as opposed to Windows NT that uses the NTFS).

**QNX Installation**

1. After switching the PC on, the user hits the <del> or <F5> key to enter into the CMOS setup.
2. The user then sets the boot-seek order to CDROM, Floppy, then Hard Disk (contact the system administrator for information specific to particular CMOS settings).
3. The user now inserts the QNX Installation CD in the CDROM drive.
4. The user then selects the Save and Exit option to write the new settings to the CMOS.
5. The system now boots off the CD into QNX.
6. The user executes `fdisk` (to partition the hard disk) when prompted. The user should be able to see the Windows 98 partition (as /dev/hd0). Note that the Windows 98 reference name changes to /dev/hda1 during Linux installation.
7. The user now creates the QNX partition by typing in the required size in MB. For a 2 GB QNX partition, the user types 2048. This creates the QNX partition.
8. The user writes the partition table hitting “Save and Continue.”
9. The QNX installation now continues. For more information, refer to the QNX installation guide.

**Linux Installation**

1. If you installed only Windows without QNX, you will need to use Partition Magic or some other method to resize the Windows partition to 2 MB before attempting to install Linux. Linux can not resize an existing partition of Windows.
2. After switching the PC on, the user hits the <del> or <F5> key to enter into the CMOS setup.
3. The user sets the boot-seek order to Floppy, Hard Disk, then CDROM (contact the system administrator for information specific to particular CMOS settings of the machine).
4. The user inserts the Linux Installation Floppy into the Floppy drive and the Red Hat Linux 6.1 CD into the CDROM drive.
5. The user then selects the Save and Exit option to write the new settings to the CMOS.
6. The user now follows exactly the procedure listed at http://qrts.com/manuals/rlinuxtarget/htm/appendix_install_linux6-1.htm (this information is
also available in Appendix A of the Real-Time Linux Target Manual) until the hard disk partition data is displayed.

7. The user checks to see whether the Windows 98 partition and the QNX partition are seen. If the aforementioned order is specified, the Windows 98 partition is assigned /dev/hda1, the QNX partition is assigned /dev/hda2, and the Linux partition is assigned /dev/hda3.

8. It is important to note that this bootable Linux partition must be within the first 1024 cylinders. The cylinder information is also provided in the partition table display.

9. As explained in the Linux installation guide, the user sets the Linux Swap partition.

10. If space beyond 1024 cylinders is available, the user sets the extended partition as a Linux partition and mounts the /home directory in this partition. The ‘/’ mount must be done in the /dev/hda3 partition. If this is not correctly implemented the system may not boot into Linux correctly.

11. Once the hard disk is partitioned, the user continues with the Linux installation as elaborated in the Linux installation guide.

12. A sample /etc/lilo.conf file for a triple-boot system with Windows 98, QNX and Linux is shown below.

### Sample LILO Configuration File

A sample /etc/lilo.conf file is shown below:

```plaintext
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
prompt
timeout=50
default=linux

image=/boot/vmlinuz-2.2.12-20
    label=linux
    initrd=/boot/initrd-2.2.12-20.img
    read-only
    root=/dev/hda3

other=/dev/hda2
    label=qnx

other=/dev/hda1
    label=win98
```

This allows Windows 98 to boot from partition /dev/hda1, QNX OS to boot from partition /dev/hda2, and Linux/RT-Linux to boot from /dev/hda3.

### Checking Partitions

Another way to check the partitions is to boot into Linux, login as root, and type at the Linux prompt
# fdisk /dev/hda

The partition table is displayed on the screen by typing ‘p’ at the command prompt. For example, shown below is the partition table of a hard disk with 1584 cylinders.

Disk /dev/hda: 255 heads, 63 sectors, 1584 cylinders
Units = cylinders of 16065 * 512 bytes

<table>
<thead>
<tr>
<th>Device</th>
<th>Boot</th>
<th>Start</th>
<th>End</th>
<th>Blocks</th>
<th>Id</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>/dev/hda1</td>
<td></td>
<td>1</td>
<td>128</td>
<td>1028128+</td>
<td>c</td>
<td>Win95 FAT32 (LBA)</td>
</tr>
<tr>
<td>/dev/hda2</td>
<td>129</td>
<td>651</td>
<td>4200997+</td>
<td>4d</td>
<td>QNX4.x</td>
<td></td>
</tr>
<tr>
<td>/dev/hda3</td>
<td>*</td>
<td>652</td>
<td>1023</td>
<td>2988090</td>
<td>83</td>
<td>Linux</td>
</tr>
<tr>
<td>/dev/hda4</td>
<td>1024</td>
<td>1313</td>
<td>2329425</td>
<td>5</td>
<td>Extended</td>
<td></td>
</tr>
<tr>
<td>/dev/hda5</td>
<td>1024</td>
<td>1040</td>
<td>136521</td>
<td>82</td>
<td>Linux swap</td>
<td></td>
</tr>
<tr>
<td>/dev/hda6</td>
<td>1041</td>
<td>1313</td>
<td>2192841</td>
<td>83</td>
<td>Linux</td>
<td></td>
</tr>
</tbody>
</table>

Note that Linux, QNX and Windows are within the first 1024 cylinders. The Linux Swap partition another Linux partition (that mounts /home) is in the space between cylinders 1024 and 1313. The * indicates the active boot partition, which is easily changed by configuring LILO.