ACUTRONIC

Acutrol3000® Real Time Data Communication

Technical Manual

TM-9374A

Acutrol3000® Real Time Data Communication

Technical Manual

TM-9374A

Prepared for:	Acutronic
Date Prepared:	March 1, 2005
Revision:	A

Table of Contents

1 Conventions 1

2 Acutrol3000 Real Time Data Processing Overview 3
2.1 Real time demand processing 3
2.2 Real time monitor processing 4
2.3 Demand Processing Variables 4
2.4 Scenarios 5
2.4.1 Overview 5
2.4.2 Typical Scenario 5

3 Acutrol3000 Real Time Data Transfer Protocol 9
3.1 General Real Time Data Transfer Configuration 9
3.1.1 Specifying the data format 9
3.1.2 Specifying the host period 10
3.1.3 Advanced Real-Time Data Transfer Configuration 11
3.1.3.1 Specifying the Host Timeout period 11
3.1.3.2 Specifying the network interrupt configuration 11
3.1.4 Setting the real-time interface on line 12
3.1.5 Setting the real-time interface off line 13
3.2 Demand Data 14
3.2.1 Demand Protocol Configuration. 14
3.2.1.1 Specifying the demand variables 14
3.2.1.1.1 Control Word 16
3.2.1.2 Specifying the Demand Mode 17
3.2.1.3 Setting the servo control mode 18
3.2.1.4 Demand Block Location 19
3.2.1.5 Demand Sync ID Location 20
3.2.1.6 Advanced Demand Configuration 21
3.2.1.6.1 Demand Delay 21
3.2.1.6.2 Demand Tracker Mode 21
3.2.1.6.3 Demand Translator Mode 22
3.2.1.6.4 Demand State Tracker Bandwidth and Damping 22
3.2.1.6.5 Demand Translator Bandwidth and Damping 23
3.2.2 Demand Protocol – Reflective Memory Interface 24
3.2.2.1 Transferring demand data 24
3.2.2.2 Demand Interrupts 24
3.2.3 Demand Protocol – Parallel Interface 24
3.3 Monitor Data 25
3.3.1 Monitor Protocol Configuration 25
3.3.1.1 Specifying the monitor variables 25
3.3.1.2 Specifying the monitor protocol 26
3.3.1.3 Monitor Block Location 26
3.3.1.4 Monitor Sync ID Location 27
3.3.1.5 Advanced Monitor Configuration 27
3.3.1.5.1 Monitor Delay 27
3.3.1.5.2 Monitor Translator Mode 28
3.3.1.5.3 Monitor Translator Bandwidth and Damping 29
3.3.1.5.4 Monitor Direct Read No Handshaking lockout window 29
3.3.2 Monitor Protocol – Reflective Memory Interface 30

3.3.2.1 Transferring monitor data using the ACT2000 protocol. 30
3.3.2.2 Transferring monitor data using the Direct Read No Handshaking (DRNHS) protocol 30
3.3.2.3 Host Computer interrupts (optional). 31
3.3.3 Monitor Protocol – Parallel Interface....31
3.4 Acutrol Command Language (ACL) Channel 31
3.4.1 ACL Channel Configuration 31
3.4.1.1 ACL Command Block Location 31
3.4.1.2 ACL Command Sync ID Location 32
3.4.1.3 ACL Response Block Location....33
3.4.1.4 ACL Response Sync ID Location....33
3.4.2 ACL Channel Protocol 34
3.4.2.1 ACL Channel Command Reflective Memory Interrupts 35

4 Acutrol3000 Real Time Variables 37

5 Acutrol3000 SCRAMNet+ Reflective Memory Interface 47
5.1 SCRAMNet+ Hardware....47
5.2 SCRAMNet+ Installation 48
5.2.1 Installation into the Acutrol3000 Chassis....49
5.2.2 Enabling the SCRAMNet+ Interface 50
5.3 SCRAMNet+ Configuration....51
5.3.1 SCRAMNet+ Node ID 51
5.3.2 SCRAMNet+ Ring Mode....52
5.4 SCRAMNet+ Features 53
5.5 SCRAMNet+ Usage....53
5.5.1 PCI SCRAMNet+ Setup....53
5.5.1.1 Installation 53
5.5.1.2 Board Configuration 53
5.5.1.3 EEPROM Configuration 54
5.5.2 CSR Setup – PCI 54
5.5.3 PCI SCRAMNet+ INITIALIZATION SEQUENCE USING SYSTRAN DLL 72
5.5.3.1 Initializing the Systran DLL....72
5.5.3.2 Initializing the Control Status Registers (CSRs)....73
5.5.3.3 Initializing the Auxiliary Control RAM (ACR) 74
5.5.3.4 Initializing the Shared Memory (SM) 75
5.5.3.5 Initializing Acutrol 76

6 Acutrol3000 VMIC Reflective Memory Interface 77
6.1 VMIC Hardware 77
6.2 VMIC Installation....78
6.2.1 VMIPCI-5565 Jumper Settings 78
6.2.2 Installation into the Acutrol3000 Chassis....80
6.2.3 Enabling the VMIC Interface....81
6.3 VMIC Configuration....82
6.4 VMIC Features....82
6.5 VMIC Usage....83
6.5.1 Initializing Acutrol 83

7 Acutrol3000 Parallel Interface....85
7.1 Parallel Interface Hardware 85
7.2 Parallel Interface Installation....85
7.2.1 Installation into the Acutrol3000 Chassis....85

7.2.2 Enabling the Parallel Interface 87
7.3 Parallel Interface Configuration 88
7.4 Parallel Interface Features 93
7.5 Parallel Interface Usage 93

Table of Figures

Figure 2-1 Acutrol3000 Real Time Data Flow....3

Figure 2-2 Acutrol3000 Real Time Data Flow....4

Figure 3-1 Real Time Interface Setup....10

Figure 3-2 VMIC Interface Setup....12

Figure 3-3 Remote Interface Control....13

Figure 3-4 Real Time Variable Data Block Configuration....16

Figure 3-5 Setting Servo Mode for Real Time Tracking....19

Figure 3-6 Setting Real Time Block Addresses....20

Figure 3-7 Configuring Tracker/Translator of Real Time Interface....23

Figure 5-1 SCRAMNet+ Card Inserted into Acutrol3000 Chassis....50

Figure 5-2 Acutrol3000 Rear Panel Showing SCRAMNet+ Interface....51

Figure 5-3 SCRAMNet+ Mode Setup....52

Figure 6-1- VMIPCI-5565 Jumper Locations....79

Figure 6-2 VMIC Card Inserted into Acutrol3000 Chassis....81

Figure 6-3 Acutrol3000 Rear Panel Showing VMIC Interface....82

Figure 7-1 PCI-6533 Card Inserted into Acutrol3000 Chassis....87

Figure 7-2 Acutrol3000 Rear Panel Showing Parallel Interface....88

Introduction

This manual describes the communication methods for a host controller to supply and receive real time data from an Acutrol3000 Motion Controller. It also describes the processing of the data once it has been transmitted to the Acutrol3000.

Currently there are three real-time interfaces to communicate to the Acutrol3000: SCRAMNet+ Reflective Memory Interface, VMIC Reflective Memory Interface, and a 16-bit parallel interface (DRV-11W).

The following topics are covered in this manual:

Section 2 provides an overview of the real time data processing.

Section 3 describes the methods used for passing real time data to the Acutrol3000.

Section 4 describes the system and axis variables that are specific to or related to real-time data processing.

Section 5 describes the hardware, installation, configuration, features, and usage of the SCRAMNet+ Interface.

Section 6 describes the hardware, installation, configuration, features, and usage of the VMIC Interface.

Section 7 describes the hardware, installation, configuration, features, and usage of the parallel interface.

Other documents that the user may find helpful:

TM-8004 Acutrol3000® Command Language (ACL)

TM-9388 Acutrol3000® User Guide

TM-9391 Acutrol3000® Variables

TM-9216 16-Bit Parallel Interface for the Acutrol® Measurement and Control System

SCRAMNet+ SC150 Network PCI Bus Hardware Reference, D-T-MR-PCI-#####-A-0-A7, Systran Corporation

VMIPCI-5565 Ultrahigh-Speed Fiber-Optic Reflective Memory with Interrupts Product Manual, VMIC Corporation

DAQ 653X User Manual, High-Speed Digital I/O Devices for PCI, PXI , CompactPCI, AT, EISA, and PCMCIA Bus Systems, 321464C-01, National Instruments Corporation

1 Conventions

The following conventions are used in this document:

`Monospace Bold`	Monospace Bold denotes text that is typed by the user, usually as part of an Acutrol Command Language (ACL) command. This font is also used for the names of memory locations in the reflective memory interface and the names of Acutrol3000 variables..
Tahoma Bold	Tahoma Bold text denotes items in the user interface, including touch panel names, dropdown menu items, etc.
→	The → symbol is used to direct the user through nested panel menu items in the user interface. For example, the sequence To **Setup→ Interfaces→Real Time** directs you to touch the **Setup** tab followed by the **Interfaces** tab followed by the **Real Time** tab.
`S_CONTROLWORD(100)`	This notation indicates a variable mnemonic, with the variable number in parentheses.
`x123`	Indicates a composite variable number. The axis number should be substituted for **`x`**. Use 0 as the axis number for system variables.
`<param>`	Angle brackets containing a name denote an ACL parameter. The user will normally substitute a value or a mnemonic when entering the command.
ACL:	Most actions and parameter specification in this manual can be performed using either ACL or the user interface. This icon denotes the ACL method.
GUI:	Most actions and parameter specification in this manual can be performed using either ACL or the user interface. This icon denotes the user interface method.
💡	This icon denotes a tip that alerts you to advisory information.
📝	This icon denotes a note that alerts you to important information.
!	This icon denotes a caution that advises you of precautions to take to avoid injury, data loss, or a system crash.

2 Acutrol3000 Real Time Data Processing Overview

This section provides an overview of the real time data processing functionality for the Acutrol3000.

2.1 Real time demand processing

Figure 2-1 shows a block diagram of the real time demand data flow in the Acutrol3000.

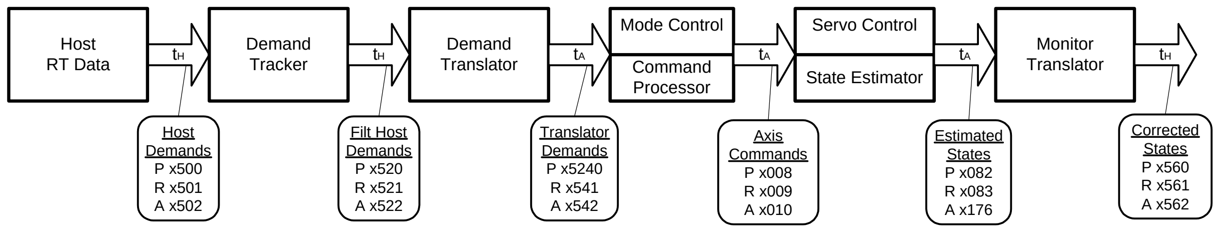

Host RT Data
tH
Demand Tracker
tH
Demand Translator
tA
Mode Control
Command Processor
tA
Servo Control
State Estimator
tA
Monitor Translator
tH
Host Demands P x500 R x501 A x502
Filt Host Demands P x520 R x521 A x522
Translator Demands P x5240 R x541 A x542
Axis Commands P x008 R x009 A x010
Estimated States P x082 R x083 A x176
Corrected States P x560 R x561 A x562

Figure 2-1 Acutrol3000 Real Time Data Flow

Figure 2-2 shows a block diagram of the real time demand data processing in the Acutrol3000.

At the beginning of each host frame, the Host Computer writes new demand data to the Acutrol3000. When using reflective memory, the host also sends a reflective memory interrupt to Acutrol (this happens automatically with the SCRAMNet+ interface).

Inside Acutrol the Host Time Tracker timestamps the data and sends it to the Host Demand Queue. At the start of the next Acutrol frame, Acutrol will recognize that new host demand data is available and the Host Demand Tracker will process all recorded Host Demand Vectors stored in the Host Demand Queue. The Host Demand Tracker filters the Host Demand Vectors to ensure that the Host Demand Vector is coherent. If the Host Demand Vector is of reduced order, the Host Demand Tracker also fills in missing motion states. The Host Time Tracker tracks the host frame rate to automatically compensate for drift or transmission variability.

Once the Host Demand Vector is processed through the Host Demand Tracker, the data is sent to the Multi-Rate Data Translator. The Multi-Rate Data Translator provides data interpolation at the Acutrol frame rate, calculating a new interpolated demand vector that is accurate at the end of the current Acutrol frame. The interpolated demand vector is then sent to the demand summer, where it is summed in with other demands according to the Acutrol demand mode.

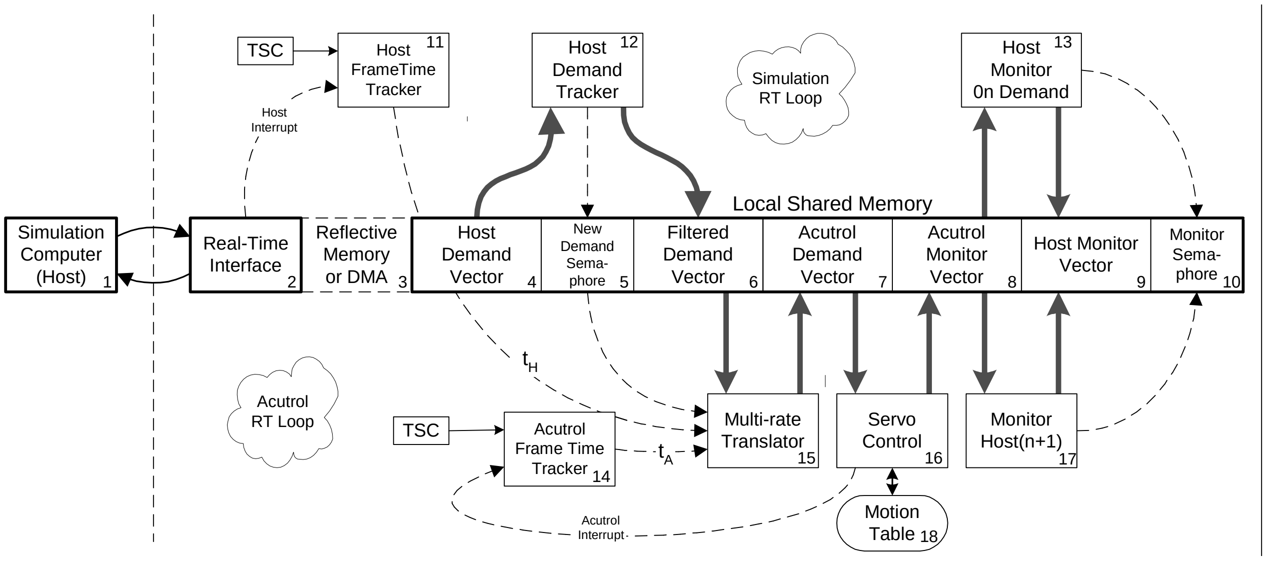


Figure 2-2 Acutrol3000 Real Time Data Flow

2.2 Real time monitor processing

At the end of each Acutrol frame, Acutrol time-skew corrects the motion state variables for each axis (see Figure 2-2). The time-skew correction is accurate at the next predicted Host Computer frame interrupt. The variables **`V_FHOST_POS_MON(x560), V_FHOST_RATE_MON(x561), V_FHOST_ACCEL_MON(x562),`** and **`V_FHOST_JERK_MON(x563)`** contain the time skew corrected data.

The Acutrol then copies all variables specified in the real-time configuration (see Section 3.3.1.1) into the monitor block record (subject to the selected monitor protocol constraints).

2.3 Demand Processing Variables

Appendix 1 contains a detailed block diagram of the Acutrol3000 Real Time Data Processing and shows the variables that are available for monitoring the real time data processing. The raw demand vector received from the Host Computer is available in variables **`V_HOST_POS_DMD(x500), V_HOST_RATE_DMD(x501), V_HOST_ACCEL_DMD(x502),`** and **`V_HOST_JERK_DMD(x503).`** Variables **`V_INVAR_1(x504)`** through **`V_INVAR_4(x507)`** are available as general-purpose demand variables. These variables can in turn be used as inputs to computational blocks.

The filtered host demand vector is available in variables **`V_FHOST_POS_DMD(x520)`** through **`V_FHOST_JERK_DMD(x523)`** and the interpolated demand vector is available in variables **`V_TRACK_POS_DMD(x540)`** through **`V_TRACK_POS_DMD(x543)`**.

Variable **`S_AVG_HOSTPERIOD(0715)`** is the average host period (in sec) computed by the host frame time tracker.

2.4 Scenarios

2.4.1 Overview

ACL commands can be transmitted via any of the available ACL command line interfaces: IEEE488, Reflective Memory, or TCP/IP.

2.4.2 Typical Scenario

A typical scenario would proceed as follows.

NOTE For each of the steps below, there are usually three alternatives. The host computer can send a command via an available ACL interface, the host computer can use the Acutrol3000 control word (see Section 3.2.1.1.1), or the action can be done manually using the GUI. Each available option is described under each step.

NOTE If using the control word, then the real time interface must be on-line for the action to take effect. Therefore you would need to do step 5) before step 2). Also note that once the real time interface is online, the timeout constraint (Section 3.1.3.1) must be honored by the host computer to avoid a Host Timeout error.

1) Setup realtime interface

Configure the real time protocol parameters as described in Section 3. If the desired real time interface configuration has already been saved in a configuration file, no action is needed here other than to load that configuration:

ACL:

```
:configure:restore ALL, 1
```

GUI:

Touch **Setup→System→Save/ Restore**

Enter desired version in the **File Version** field.

Hit **Restore** button.

2) Close interlocks

ACL:

```
:interlock:close  1
```

Repeat for remaining axes.

GUI:

Touch **Controls→Activate Servo**

Hit **Close** button for each axis (or hit **Close All**).

Hit **Execute** button.

CONTROL_WORD:

Set bits 3, 7, 11, 15, 19, 23 to close interlocks for axes 1 through 6 respectively.

3) Setup initial conditions for each axis

Normally the initial conditions would match the first real time data point so as to avoid any transients at startup. In this example the first real time data point is Position = Rate = Acceleration = 0.0.

ACL:

```
:demand:position  ALL, 0.0

:demand:rate  ALL, 0.0

:demand:accel  ALL, 0.0
```

GUI:

Touch **Motion** tab.

Enter **0.0** on keypad.

Touch **Position** field.

Repeat for **Rate** and **Acceleration.**

Hit **Enter All Axes** button.

4) Place the axes to be controlled in TRACK mode

ACL:

```
:mode:track  ALL
```

GUI:

Touch **Motion** tab.

Set **Mode Request** dropdown to **Track.**

Touch **Enter All Axes** button.

CONTROL_WORD:

Set bits <0-2>, <4-6>, <8-10>, <12-14>, <16-18>, <20-22> to 100 (binary) for axes 1 through 6 respectively.

5) Place the Acutrol realtime interface on-line

ACL:

```
:configure:real online
```

GUI:

Touch **Controls→Activate Remote**

Hit **OFF-LINE** button. The button will change to indicate **ON-LINE**.

6) Execute the demand and monitor protocols

The host computer should immediately begin following the demand and monitor protocols as defined in Sections 3.2.2 and 3.3.2 (reflective memory interface) or Sections 3.2.3 and 3.3.3 (parallel interface).

Proceed to step 7) when the test is complete.

NOTE If using a reflective memory interface, the host computer can monitor the **`DemandSyncID`** location (see Section 3.2.1.5) to know when the Acutrol is placed on-line. This location will change from **`0x0FF1E0FF`** to **`0x00000000`** at that time.

7) Place the Acutrol real time interface off-line

Since at this point the real time interface is on-line, the GUI controls will be locked out. Therefore there are only two alternatives to place the real time interface offline.

ACL:

```
:configure:real offline
```

CONTROL_WORD:

Clear bit 31.

8) Perform any post processing commands

At this point you may wish to stop all motions, open interlocks, etc. or prepare for the next scenario.

3 Acutrol3000 Real Time Data Transfer Protocol

The Acutrol3000 Real Time Data Transfer Protocol allows a Host Computer to demand motions from the Acutrol3000 and monitor Acutrol3000 motion states in real time. Data can be transferred to and from Acutrol using either a reflective memory interface or a parallel interface. Sections 3.2.2 and 3.3.1.5.4 describe the reflective memory demand and interface protocols respectively. See Sections 3.2.3 and 3.3.3 for a description of the parallel interface protocols.

NOTE Most of the real time configuration settings are not axis specific (exceptions are the demand tracker, monitor tracker, and monitor translator settings) and therefore are stored in the system configuration. Once configuration is complete, be sure and save the "system" configuration.

3.1 General Real Time Data Transfer Configuration

This section describes real time data transfer configuration settings that are common to all interfaces.

3.1.1 Specifying the data format

The demand and monitor protocols must use the same data format for all variables; you cannot mix formats. Three formats are available:

Binary – data is transferred as 32-bit signed integers.

Float – data is transferred as 32-bit IEEE 754 floating point values.

Double – data is transferred as 64-bit IEEE 754 floating point values.

Note that with the Parallel Interface only the **Binary** format is available.

ACL:

```
:configure:realtime:setup , , , , , , , <data format>
```

Specify the data format for the variables (Binary, Float, or Double).

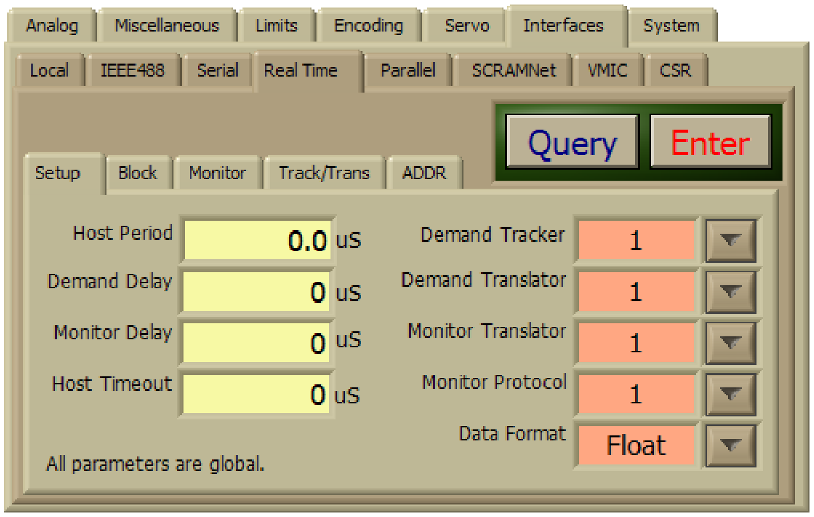
GUI:

See Figure 3-1 Real Time Interface Setup.

Touch **Setup→ Interfaces→Real Time→Setup**.

Specify the data format for the variables (Binary, Float, or Double).

Touch **Enter**.



Host Period 0.0 uS
Demand Delay 0 uS
Monitor Delay 0 uS
Host Timeout 0 uS
Demand Tracker 1
Demand Translator 1
Monitor Translator 1
Monitor Protocol 1
Data Format Float

All parameters are global.

Figure 3-1 Real Time Interface Setup

3.1.2 Specifying the host period

Acutrol normally will track the Host Period based on the frequency of demand interrupts. However, when starting the tracking protocol, it is useful to seed the host time tracker with a nominal value. Also, in the case where the host time tracker is turned off (see 3.2.1.6, Advanced Demand Configuration), then this value is used for the host period in any calculations.

ACL:

```
:configure:realtime:setup <HostPeriod>
```

Specify the <HostPeriod> in microseconds:

GUI:

See Figure 3-1 Real Time Interface Setup.

Touch **Setup→ Interfaces→Real Time→Setup**.

Enter the host period (microseconds) in the **Host Period** field.

Touch **Enter**.

3.1.3 Advanced Real-Time Data Transfer Configuration

This section contains advanced protocol setup information. Most users will be able to use the default values for these settings.

3.1.3.1 Specifying the Host Timeout period

Once the real time interface is on-line, the demand interrupts from the Host Computer are expected to occur at a nominal rate equal to the host frame rate. If a demand interrupt is not received within the **Host Timeout** period from when it was expected, a soft abort is issued[1]. Setting the **Host Timeout** period to 0.0 disables the host timeout function.

To specify the **Host Timeout** period:

ACL:

```
:configure:realtime:setup ,,,,,,,, <hostTimeoutPeriod>
```

Specify **`hostTimeoutPeriod`** in microseconds.

GUI:

See Figure 3-1 Real Time Interface Setup.

Touch **Setup→Interfaces→Real Time→Setup**.

Enter the host timeout period (microseconds) in the **Host Timeout** field.

3.1.3.2 Specifying the network interrupt configuration

The Acutrol will always send a network interrupt after writing data to the reflective memory. With certain interfaces, such as SCRAMNet, this happens automatically. Other interfaces, such as VMIC, require the Acutrol to explicitly configure and send an interrupt command to the Host Computer node.

Note that this functionality is optional. You only need to configure these settings if the Host Computer software is expecting to process network interrupts for each host frame and/or ACL command.

To specify the network interrupt configuration:

[1] Normally a **Host Timeout** will only occur in Track Mode. It is possible to cause a **Host Timeout** to occur in all modes by setting the **hostTimeoutAllModes** parameter to TRUE. This parameter is located in the **`<RT_SETUP..>`** node of the system.xml file. Note that if this parameter does not exist in the system.xml file it will default to TRUE.

ACL:

`:interface:vmic <channel>, <networkInterruptNode>, <networkInterrupt>, <networkInterruptData>`

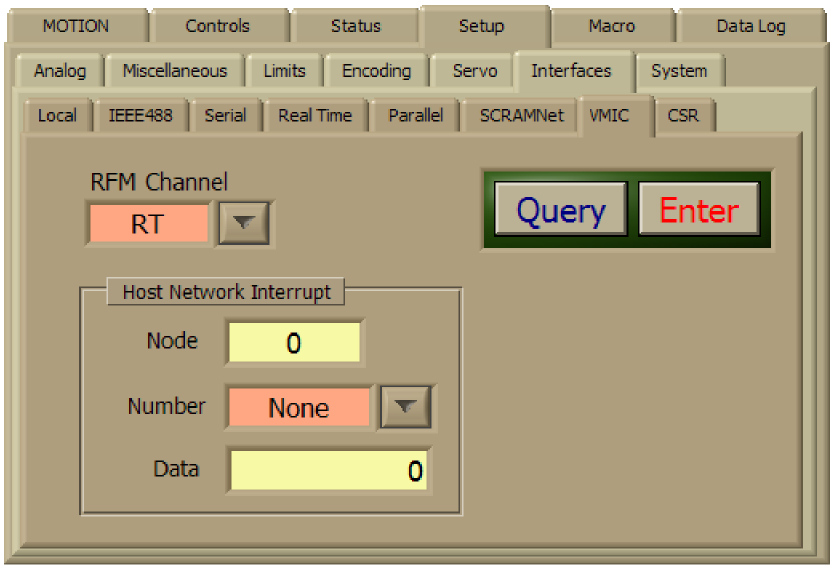
Specify the **`<channel>`** (RT or ACL), the **`<networkInterruptNode>`** (0..255), the **`<networkInterrupt>`** (1, 2, 3, or none), and the **`<networkInterruptData>`** (-32768..32767).

GUI:

See Figure 3-2 VMIC Interface Setup.

Touch **Setup→Interfaces→VMIC**.

In the **Host Network Interrupt** section specify the **Node**, **Number**, and **Data** for each channel (**RT** and **ACL**).



RFM Channel: RT

Node: 0

Number: None

Data: 0

Figure 3-2 VMIC Interface Setup

3.1.4 Setting the real-time interface on line

The demand and monitor protocols are started when the Host Computer or user sets the reflective memory interface “on-line”. When the Acutrol real-time interface is on line, the **`demandSyncID`** location (see Section 3.2.1.5) will read **`0x00000000`**, indicating that the Acutrol is ready to receive data.

ACL:

`:configure:realtime online`

GUI:

See Figure 3-3 Remote Interface Control.

Touch **Controls➔Activate Remote**.

Select **Online**.

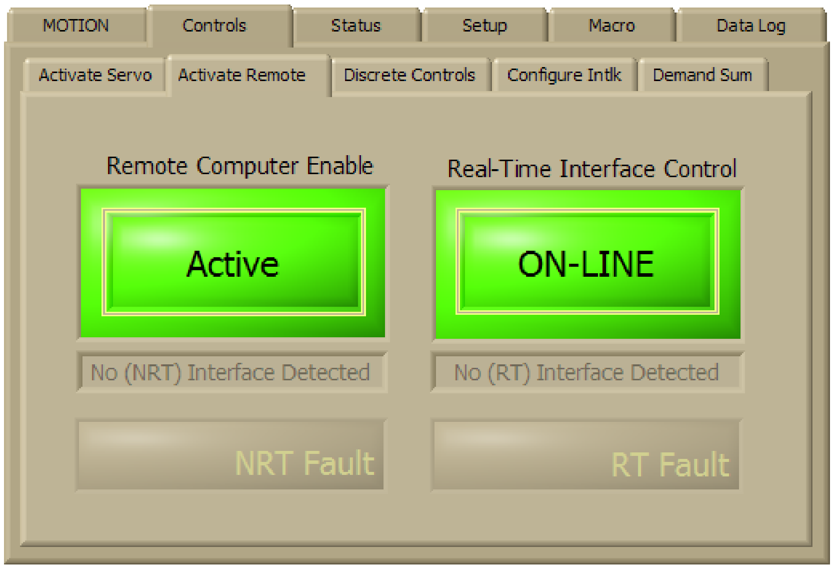


Figure 3-3 Remote Interface Control

3.1.5 Setting the real-time interface off line

The demand and monitor protocols are stopped when the Host Computer or user sets the reflective memory interface "off-line". When the Acutrol real-time interface is off line, the **`demandSyncID`** location (see Section 3.2.1.5) will read **`0x0FF1E0FF`**.

ACL:

```
:configure:realtime offline
```

GUI:

See Figure 3-3 Remote Interface Control.

Touch **Controls➔Activate Remote**.

Select **Offline**.

3.2 Demand Data

The demand protocol is used by the Host Computer to transfer demand data to Acutrol in real time.

3.2.1 Demand Protocol Configuration.

This section describes configuration parameters that are specific to the demand protocol.

3.2.1.1 Specifying the demand variables

Table 3-1 lists the variables that can be demanded of Acutrol. One or more of these variables can be specified.

In addition to the four motion state demand variables, Acutrol provides eight general-purpose demand variables. These variables can then be used as inputs to control algorithm computation blocks to allow additional real time flexibility.

Note that composite variable numbers are used here (composite variable numbers use the axis number as a prefix).

To specify the demand variables and the order that they will be written to the reflective memory:

ACL:

```
:configure:realtime:blockrecord  Demand, <var1>, <var2>…
```

List the demand variables numbers in the order that the variables will appear in the reflective memory.

GUI:

See Figure 3-4 Real Time Variable Data Block Configuration.

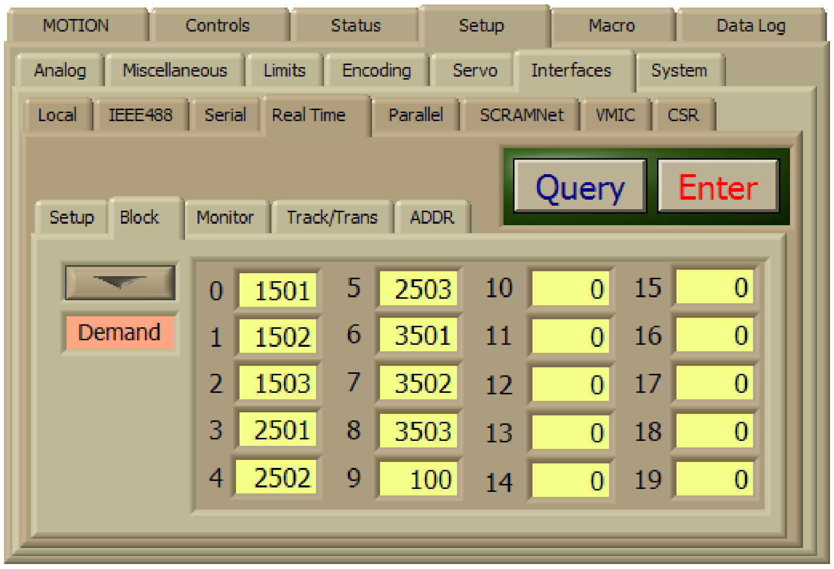
Touch **Setup→Interfaces→Real Time→Block Record.** Choose **Demand** from the drop-down menu.

Enter the demand variable numbers in the order that the variables will appear in the reflective memory.

Table 3-1 Demand Variables

Variable Number	Variable Mnemonic	Description	Applicable Modes
x500[2]	V_HOST_POS_DMD	Position Demand	TRACK
x501	V_HOST_RATE_DMD	Rate Demand	TRACK
x502	V_HOST_ACCEL_DMD	Acceleration Demand	TRACK
x503	V_HOST_JERK_DMD	Jerk Demand	TRACK
x504	V_INVAR_1	Input Variable #1	ALL
x505	V_INVAR_2	Input Variable #2	ALL
x506	V_INVAR_3	Input Variable #3	ALL
x507	V_INVAR_4	Input Variable #4	ALL
0100	S_CONTROLWORD	Control Word Variable	ALL
x000	V_POSN_DMD	Axis Position Demand	POSITION
x001	V_RATE_DMD	Axis Rate Demand	RATE, abs RATE
x002	V_ACCEL_DMD	Axis Acceleration Demand	N/A

[2] The notation x indicates the axis number. For example, to demand position on axis 1, use 1500.



0	1501	5	2503	10	0	15	0
1	1502	6	3501	11	0	16	0
2	1503	7	3502	12	0	17	0
3	2501	8	3503	13	0	18	0
4	2502	9	100	14	0	19	0

Figure 3-4 Real Time Variable Data Block Configuration

3.2.1.1.1 Control Word

The demand protocol allows the Host Computer to control certain functionality of the Acutrol using the reflective memory interface directly. This is done by adding the **`S_CONTROLWORD(0100)`** variable to the demand block data record.

There are certain restrictions on this variable:

- The **`S_CONTROLWORD`** variable must always be last in the list of demanded variables. Placing it in any other position will generate an error.

- The **`S_CONTROLWORD`** variable is always treated as an unsigned 32-bit integer, even if the real-time data format is set to FLOAT or DOUBLE.

On each Host Computer demand update, the Acutrol looks at the Control Word (if specified) and executes the requested changes. Table 3-2 shows the format of the 32-bit control word:

Table 3-2 - Control Word

31	30 - 24	23	22-20	19	19-16	15	14-12	11	10-8	7	6-4	3	2-0
Remote / Local	Reserved	Axis 6 Interlock	Axis 6 Mode	Axis 5 Interlock	Axis 5 Mode	Axis 4 Interlock	Axis 4 Mode	Axis 3 interlock	Axis 3 Mode	Axis 2 interlock	Axis 2 Mode	Axis 1 Interlock	Axis 1 Mode

Mode Control – The Acutrol will change the mode of each axis according to the value in the mode field.

Mode	Servo Mode
000	Position
001	Relative Rate
010	Absolute Rate
011	Synthesis
100	Track
101	Abort
110	Off
111	Illegal

Axis Interlock– Setting an interlock bit to “0” opens the interlock for that axis and disables the axis servo. Setting an interlock bit to “1” closes the interlock and enables the axis servo.[3]

Remote / Local – Setting this bit to “1” places Acutrol in remote mode. Setting it to “0” places it in local mode.

3.2.1.2 Specifying the Demand Mode

The track mode determines which motion demands are actually used by Acutrol. The possible track modes are:

- **Disabled** – All TRACK mode motion demands are ignored.
- **PRA** – Position, Rate, and Acceleration are used.
- **PR** – Position and Rate are used.
- **P** – Position is used.
- **RA** – Rate and acceleration are used
- **R** – Rate is used.

Note that if a complete motion vector is not specified, Acutrol will calculate the missing states and use a complete motion vector (position, rate, and acceleration) for control.

[3] In order for the Interlock Bit to Function, the **`allowControlWordIntlk`** parameter in the system.xml file must be set to **`TRUE`**.

ACL:

`:Configure:Realtime:Tracker` , <Demand Mode>.

Specify **`<DemandMode>`** (Disabled, PRA, PR, P, RA, or R).

GUI:

See Figure 3-7 Configuring Tracker/Translator of Real Time Interface.

Touch **Setup→Interfaces→Real Time→Tracker / Translator**.

Select the desired **`Demand Mode`**.

3.2.1.3 Setting the servo control mode.

Acutrol will only apply the real-time motion demands to a given axis when the servo control mode for that axis is set to **Track**. Regardless of the servo control mode, Acutrol will always honor the real-time data transfer protocol as long as the real-time interface is on line. If the interface is off line then no action will be taken.

To set the Acutrol servo mode to **Track**:

ACL:

`:mode All,Track`

- or -

`:mode:track` ALL

GUI:

See Figure 3-5 Setting Servo Mode for Real Time Tracking.

Touch **Motion**.

Set **Mode Request** to **Track**.

Touch **Enter All Axes**.

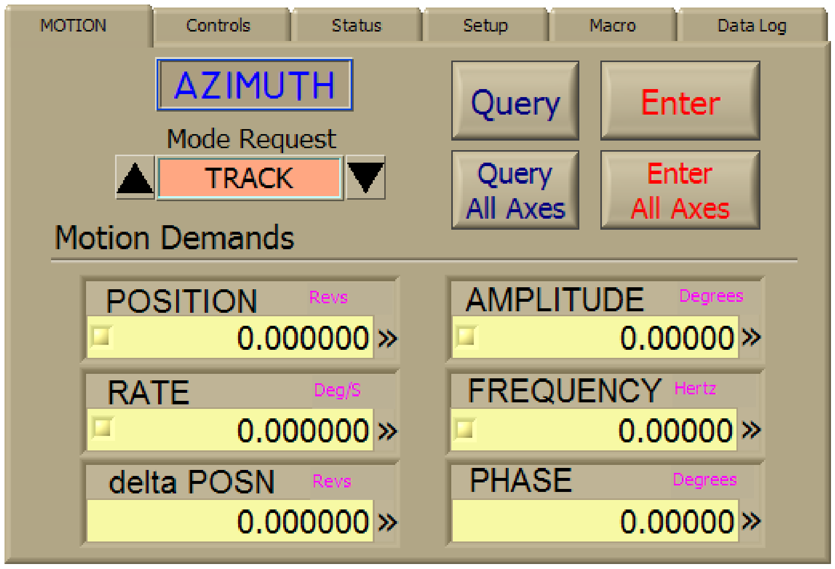


Figure 3-5 Setting Servo Mode for Real Time Tracking

3.2.1.4 Demand Block Location

Acutrol must know the location within the reflective memory address space for the demand block data. The restrictions on this address are a function of the specific reflective memory interface used (see Section 1 and Section 5.5.1). To specify the demand block location:

ACL:

```
:interface:reflectivememory:setup <DemandBlockAddress> …
```

Specify the **`<DemandBlockAddress>`** as either an integer or hex value.

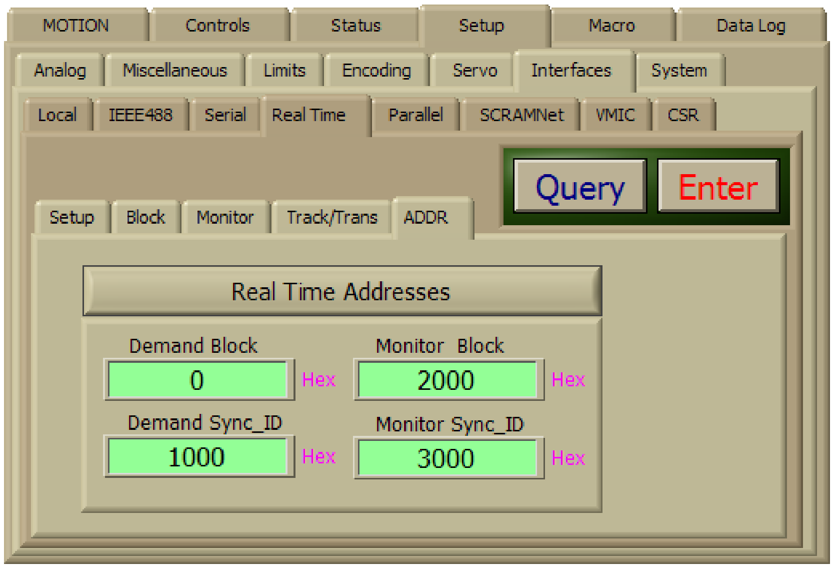
GUI:

See Figure 3-6 Setting Real Time Block Addresses.

Touch **Setup→Interfaces→Real Time→ADDR**. Select **Real Time Addresses.**

Enter the demand block address (hex) in the **Demand Block** field.

NOTE: Use the keyboard (**Kbd**) function, not the keypad, to enter hex addresses if the address contains non-numeric characters.



Demand Block: 0 Hex
Monitor Block: 2000 Hex
Demand Sync_ID: 1000 Hex
Monitor Sync_ID: 3000 Hex

Figure 3-6 Setting Real Time Block Addresses

3.2.1.5 Demand Sync ID Location

The **`DemandSyncID`** is used to marshal access to the demand block memory between Acutrol and the Host Computer. For certain interfaces, such as SCRAMNet+, writing to this address automatically triggers a reflective memory interrupt (see Section 3.2.2.2).

The restrictions on this address are a function of the specific reflective memory interface used (see Section 1 and Section 5.5.1). To specify the **DemandSyncID** address

ACL:

`:interface:reflectivememory:setup , <DemandSyncID>` …

Specify the **`<DemandSyncID>`** as either an integer or hex value.

GUI:

See Figure 3-6 Setting Real Time Block Addresses.

Touch **Setup→Interfaces→Real Time→ADDR**. Select **Real Time Addresses.**

Enter the demand sync ID address (hex) in the **Demand SyncID** field.

NOTE: Use the keyboard (**Kbd**) function, not the keypad, to enter hex addresses if the address contains non-numeric characters.

3.2.1.6 Advanced Demand Configuration

This section contains advanced protocol setup information. Most users will be able to use the default values for these settings.

3.2.1.6.1 Demand Delay

It is possible to adjust the reference point for time skew correction of the demanded motion states. This allows the user to correct for the time difference from when the demand is calculated during the host computer frame and sent over the remote interface to the time when the demand is valid and should be applied to the motion controller.

ACL:

```
:configure:realtime:setup  , , , , , <demandDelay>
```

Specify the **`demandDelay`** in microseconds.

GUI:

See Figure 3-1 Real Time Interface Setup.

Touch **Setup→Interfaces→Real Time→Setup**.

Enter the demand delay (microseconds) in the **Demand Delay** field.

3.2.1.6.2 Demand Tracker Mode

The demand state tracker has several possible algorithms for motion state tracking.

Disabled: No tracking applied, output vector = input vector. This is only valid when the Demand Mode (see Section 3.2.1.2) is set to "PRA" and the Host Computer is supplying all three motion states.

1: PRA integration with PR feedback loops and no PR input delay

2: PRA integration with PR feedback loops and PR input delay

ACL:

```
:configure:realtime:setup  , <trackMode>
```

Specify the **`trackMode`** (Disabled, 1, 2, or 3).

GUI:

See Figure 3-1 Real Time Interface Setup.

Touch **Setup→Interfaces→Real Time→Setup**.

Choose the track mode (Disabled, 1, or 2) using the **Tracker** drop-down menu.

Touch **Enter**.

3.2.1.6.3 Demand Translator Mode

Defines and/or enables the algorithm used for demand time skew correction:

Disabled: No translation, output vector = input vector

- **1:** Multi-rate State Synchronization with PRA input delay and PR feedback loops (uses current motion state, linear interpolation of states over customer frame w/ coherence filter)
- **2:** Multi-rate State Synchronization with PRA input filtering and PR feedback loops (uses predictor, gives no lead, predictor w/ coherence filter)
- **3:** Multi-rate State Synchronization with PRA input delay and PR feedback loops (uses previous motion state, linear interpolation of states over customer frame w/ coherence filter)
- **4:** Multi-rate State Synchronization with no PRA input delay and no PR feedback loops (Euler integration lead w/ coherence filter)
- **5:** Multi-rate State Synchronization with no PRA input delay and no PR feedback loops (Euler integration lead w/o coherence filter)

ACL:

```
:configure:realtime:setup  , , <demandTranslatorMode>
```

Specify the **demandTranslatorMode** (Disabled, 1, 2, 3, 4, or 5).

GUI:

See Figure 3-1 Real Time Interface Setup.

Touch **Setup→Interfaces→Real Time→Setup.**

Select the **Demand Translator** (Disabled, 1, 2, 3, 4, or 5).

3.2.1.6.4 Demand State Tracker Bandwidth and Damping

Defines the undamped natural frequency (ω_n) of the tracker and the damping factor (ξ) of the demand state tracker. These settings are axis specific and should be configured for each axis. Be sure and save the axis configuration after setting.

ACL:

```
:configure:realtime:tracker <axis>,,<bandwidth>,<damping>
```

Specify the **<bandwidth>** (Hz) and **<damping>**.

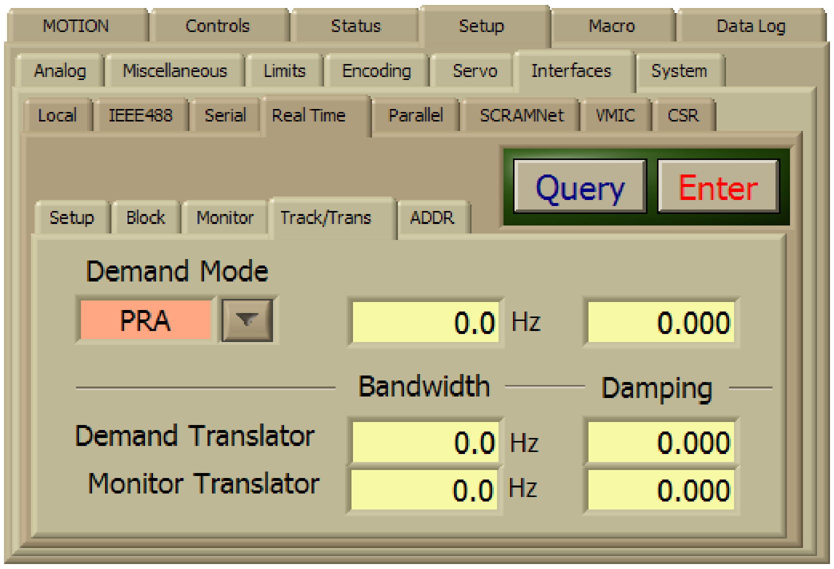
GUI:

See Figure 3-7 Configuring Tracker/Translator of Real Time Interface.

Touch **Setup→Interfaces→Real Time→Tracker / Translator.**

Enter the **Demand Mode Bandwidth** (Hz) and **Damping.**

Touch **Enter.**



Demand Mode: PRA, 0.0 Hz, 0.000

	Bandwidth	Damping
Demand Translator	0.0 Hz	0.000
Monitor Translator	0.0 Hz	0.000

Figure 3-7 Configuring Tracker/Translator of Real Time Interface

3.2.1.6.5 Demand Translator Bandwidth and Damping

Defines the undamped natural frequency (ω_n) of the tracker and the damping factor (ξ) of the demand translator. These settings are axis specific and should be configured for each axis. Be sure and save the axis configuration after setting.

ACL:

```
:configure:realtime:translator
      <axis>,<Demand>,<bandwidth>,<damping>
```

Specify the **`<bandwidth>`** (Hz) and **`<damping>`**.

GUI:

See Figure 3-7 Configuring Tracker/Translator of Real Time Interface.

Touch **Setup→Interfaces→Real Time→Tracker / Translator.**

Enter the **Demand Translator Bandwidth** (Hz) and **Damping.**

Touch **Enter**.

3.2.2 Demand Protocol – Reflective Memory Interface

This section describes the protocol to transfer demand data from a Host Computer to the Acutrol3000 using a reflective memory interface.

3.2.2.1 Transferring demand data

Acutrol sets location **`DemandSyncID`** to “0” when it is ready to receive demand data. The Host Computer should then use the following protocol on each host frame:

- Check that the **`DemandSyncID`** location is equal to 0x0000**.**
- Copy a frame of data to the reflective memory.
- Set the **`DemandSyncID`** to 0x8000.
- *VMIC only:* Send an interrupt to the Acutrol VMIC node.

NOTE: The Host Computer should only write to the frame data area if the **`DemandSyncID`** is clear (**`0x0000`**); otherwise, Acutrol may be in the process of reading the data. In the unlikely event that the **`DemandSyncID`** is not clear, the Host Computer has two options:

- Wait 50 microseconds and retry.
- Skip this frame

3.2.2.2 Demand Interrupts

The Acutrol must receive a reflective memory interrupt after the demand data is written to memory and the **`DemandSyncID`** is set. In the case of the SCRAMNet+ interface, this happens automatically when the board is configured appropriately (see Section 5). In the case of the VMIC interface, the Host Computer must explicitly configure and send a reflective memory interrupt to the Acutrol VMIC node (see Section 3.1.3.2).

3.2.3 Demand Protocol – Parallel Interface

See TM-9216 for a description of the protocol to transfer demand data from a Host Computer to the Acutrol3000 using a parallel interface.

3.3 Monitor Data

3.3.1 Monitor Protocol Configuration

This section describes configuration parameters that are specific to the monitor protocol.

3.3.1.1 Specifying the monitor variables

Any Acutrol variable can be monitored on a real time basis (see TM-9391 Acutrol3000® Variables). Note that composite[4] variable numbers are used here.

There is also a special variable, **`S_HOLE(0999)`**, that can be used to create a "hole" in the memory block data. When this variable is specified in the monitor block record, Acutrol will never write anything to that location. This allows additional flexibility in the way in which the virtual memory maps are laid out. For example, there may be locations within the monitor block record area that must be written to by the Host Computer. In this case, one would not want the Acutrol to overwrite those locations. Specifying variable **`0999`** will prevent this.

It is recommended that variable **`V_A3K_T (x229)`** always be specified as one of the monitor variables. This variable is the timestamp (in CPU tics) of the start of the Acutrol frame for the remainder of the data. The Host Computer can then check the interval from one frame to the next to verify that a frame was not skipped. This variable only needs to be specified for one axis, since it always has the same value for all axes.

To specify the monitor variables and the order that they will be written to the reflective memory:

ACL:

```
:configure:realtime:blockrecord  Monitor, <var1>, <var2>,
      <var3>, ,var4>, …
```

List the monitor variable numbers in the order you would like the variables to appear in the reflective memory.

GUI:

See Figure 3-6 Setting Real Time Block Addresses.

Touch **Setup→Interfaces→Real Time→Block Record.** Select **Monitor.**

[4] A composite variable includes both the axis number and the variable number. For example, to demand track mode position (x501) on axis 1, use 1500.

Enter the monitor variable numbers in the order you would like the variables to appear in the reflective memory.

3.3.1.2 Specifying the monitor protocol

The Acutrol3000 supports two reflective memory data transfer protocols for monitoring data.

Disabled – No monitor data is transferred.

ACUTROL 2000 Compatibility – This protocol is intended to be as compatible as possible with the Acutrol 2000. This protocol uses the **`MonitorSyncID`** location to handshake transfers to the Host Computer. Acutrol will only write data to the monitor data block if the **`MonitorSyncID`** location is 0x0000, meaning that the Host Computer has already read the previous data frame. Section 3.3.2.1 describes this protocol in detail.

Direct Read No Handshaking – In this protocol, Acutrol continually writes new monitor data to the **`MonitorBlock`** locations. In order to prevent data corruption, Acutrol will not write data during a specified “lockout window.” (see Section 3.3.1.5.4). Section 3.3.2.2 describes this protocol in detail.

To specify the monitor protocol:

ACL:

```
:configure:realtime:setup  ,,,, <monitorProtocol>
```

Specify the monitor protocol (Disabled, ACT2000, or DRNHS)

GUI:

See Figure 3-1 Real Time Interface Setup.

Touch **Setup→Interfaces→Real Time→Setup.**

Select **`Monitor Procotol`** (Disabled, ACT2000, or DRNHS).

Touch **Enter.**

3.3.1.3 Monitor Block Location

Acutrol must know the location in the reflective memory address space for the monitor block data. The restrictions on this address are a function of the specific reflective memory interface used (see Section 5 and Section 6). To specify the monitor block location:

ACL:

```
:interface:reflectivememory:setup  , ,
<MonitorBlockAddress> …
```

Specify the **`<MonitorBlockAddress>`** as either an integer or hex value.

GUI:

See Figure 3-4 Real Time Variable Data Block Configuration.

Touch **Setup→Interfaces→Reflective Memory.**

Enter the hex address value in the **Monitor Block** field.

3.3.1.4 Monitor Sync ID Location

The **`MonitorSyncID`** is used to marshal access to the monitor block between Acutrol and the Host Computer (except for the DRNHS protocol). Acutrol always writes this address last, after the Monitor Block is complete, so that for certain interfaces, such as SCRAMNet+, a reflective memory interrupt can be automatically triggered (see Section 3.2.2.2).

The restrictions on this address are a function of the specific reflective memory interface used (see Section 5 and Section 6). To specify the **MonitorSyncID** address:

ACL:

`:interface:reflectivememory:setup , , ,<MonitorSyncID>` …

Specify the **`<MonitorSyncID>`** as either an integer or hex value.

GUI:

Touch **Setup→Interfaces→Reflective Memory.**

Enter the hex address value in **Monitor Sync ID** field.

NOTE: Use the keyboard (**Kbd**) function, not the keypad, to enter hex addresses if the address contains non-numeric characters.

3.3.1.5 Advanced Monitor Configuration

This section contains advanced protocol setup information. Most users will be able to use the default values for these settings.

3.3.1.5.1 Monitor Delay

It is possible to adjust the reference point for time skew correction of the monitored motion data. This allows the user to adjust the time skew of the feedback to be valid at a specific time in the host frame.

ACL:

`:configure:realtime:setup , , , , , , <monitorDelay>`

Specify the **`<monitorDelay>`** in microseconds.

GUI:

See Figure 3-1 Real Time Interface Setup.

Touch **Setup→Interfaces→Real Time→Setup.**

Enter the **Monitor Delay** in microseconds.

3.3.1.5.2 Monitor Translator Mode

Defines and/or enables the algorithm used for monitor time skew correction:

Disabled: No translation, output vector = input vector

- **1:** Multi-rate State Synchronization with PRA input delay and PR feedback loops (uses current motion state, linear interpolation of states over customer frame w/ coherence filter)
- **2:** Multi-rate State Synchronization with PRA input filtering and PR feedback loops (uses predictor, gives no lead, predictor w/ coherence filter)
- **3:** Multi-rate State Synchronization with PRA input delay and PR feedback loops (uses previous motion state, linear interpolation of states over customer frame w/ coherence filter)
- **4:** Multi-rate State Synchronization with no PRA input delay and no PR feedback loops (Euler integration lead w/ coherence filter)
- **5:** Multi-rate State Synchronization with no PRA input delay and no PR feedback loops (Euler integration lead w/o coherence filter)

ACL:

`:configure:realtime:setup , , , <monitorTranslatorMode>`

Specify the **`<monitorTranslatorMode>`** (Disabled, 1, 2, 3, 4, or 5).

GUI:

See Figure 3-1 Real Time Interface Setup.

Touch **Setup→Interfaces→Real Time→Setup.**

Select the **Monitor Translator** (Disabled, 1, 2, 3, 4, or 5).

3.3.1.5.3 Monitor Translator Bandwidth and Damping

Defines the un-damped natural frequency (ω_n) of the tracker and the damping factor (ξ) of the monitor translator. These settings are axis specific and should be configured for each axis. Be sure and save the axis configuration after setting.

ACL:

```
:configure:realtime:translator
      <axis>,Monitor,<bandwidth>,<damping>
```

Specify the **`<bandwidth>`** (Hz) and **`<damping>`**.

GUI:

See Figure 3-7 Configuring Tracker/Translator of Real Time Interface.

Touch **Setup→Interfaces→Real Time→Tracker / Translator**.

Enter the **Monitor Translator Bandwidth** (Hz) and **Damping**.

Touch **Enter**.

3.3.1.5.4 Monitor Direct Read No Handshaking lockout window

This parameter defines the lockout window for the Direct Read No Handshaking (DRNHS) monitor protocol. In this protocol, the MonitorSyncID is not used. Rather, the Acutrol only writes to the monitor data block when it is outside of a window around the next expected Host Computer read.

The lockout window is a period of time bracketing the next expected Host Computer read. The size of the lockout window should be large enough to account for any jitter in the Host Computer frame rate.

Note that this lockout window is defined in the system.xml file as the **hostSafetyBuffer** parameter. The value is in μsec

ACL:

Currently there is no ACL command for this setting. This must be set manually in the system.xml file.

GUI:

Currently there is no GUI control for this setting. This must be set manually in the system.xml file.

3.3.2 Monitor Protocol – Reflective Memory Interface

This section describes the protocol to transfer data from the Acutrol3000 to a Host Computer using a reflective memory interface.

For all protocols, Acutrol writes the monitor data to the monitor block area on every Acutrol frame (subject to the constraints of the protocol). However, the data is time skew corrected so that it is accurate to the expected time of the next Host Computer frame.

3.3.2.1 Transferring monitor data using the ACT2000 protocol.

The Host Computer should set location **`MonitorSyncID`** to 0 when it is ready to receive monitor data. The Host Computer should then use the following protocol on each host frame:

- Check that the **`MonitorSyncID`** location is equal to 0x8000. (See NOTE below.)
- Copy a frame of data from the reflective memory.
- Set the **`MonitorSyncID`** to 0x0000. Acutrol will not write the monitor data if the **`MonitorSyncID`** is not 0x0000.

NOTE If the **`MonitorSyncID`** is not 0x8000, the Host Computer should not read the frame data area, since Acutrol may be in the process of writing the data. In the event that this situation occurs, the Host Computer has two options:

- Wait 50 microseconds and retry.
- Skip this frame

3.3.2.2 Transferring monitor data using the Direct Read No Handshaking (DRNHS) protocol

This protocol does not use handshaking. In order to prevent Acutrol from overwriting data while the Host Computer is reading it, Acutrol uses a “lockout window.” The lockout window is a period of time before and after the next expected Host Computer frame. Acutrol will not write to the monitor area during this window.

NOTE Use of this protocol is not recommended, except where the host frame rate is an order of magnitude or more less than the Acutrol frame rate. It is provided mainly as a convenience for non-critical (e.g., display only) monitoring applications.

The Host Computer should set location **`MonitorSyncID`** to 0x0000 when it is ready to receive monitor data. The Host Computer should then use the following protocol on each host frame:

- Copy a frame of data from the reflective memory.

3.3.2.3 Host Computer interrupts (optional).

Acutrol will always send a reflective memory interrupt after the monitor data is written to memory and the **`MonitorSyncID`** is set to 0x8000**.** The Host Computer can optionally enable this interrupt and transfer the data on this basis.

The procedure to do this varies depending on the specific reflective memory interface used (see Section 5 and Section 6).

3.3.3 Monitor Protocol – Parallel Interface

This section describes the protocol to transfer data from the Acutrol3000 to a Host Computer using a parallel interface.

See TM-9216 for a description of the protocol to transfer monitor data from an Acutrol3000 to the Host Computer using a parallel interface.

3.4 Acutrol Command Language (ACL) Channel

The Acutrol3000 reflective memory interface supports a second channel in addition to the real-time data channel. This second channel can be used to transmit Acutrol Command Language (ACL) commands and receive their responses. This allows Host Computers to control and query the Acutrol3000 without a second interface, such as IEEE-488 or TCP/IP.

NOTE: This capability is not available for the parallel interface.

3.4.1 ACL Channel Configuration

This section describes configuration settings for the ACL channel.

3.4.1.1 ACL Command Block Location

Acutrol must know the location in the reflective memory address space for the ACL command block data. The restrictions on this address are a function of the specific reflective memory interface used (see Section 5 and Section 6). To specify the demand block location:

ACL:

`:interface:reflectivememory:setup <DemandBlockAddress>` …

Specify the **`<ACLCommandBlockAddress>`** as either an integer or hex value.

GUI:

See Figure 3-6 Setting Real Time Block Addresses.

Touch **Setup→Interfaces→Real Time→ADDR** . Select **ACL Addresses**

Enter the ACL Command Block address (hex) in the **Command Block** field.

Note: You must use the keyboard function, not the keypad, to enter the addresses.

3.4.1.2 ACL Command Sync ID Location

The **`ACLCommandSyncID`** is used to marshal access to the ACL command block between Acutrol and the Host Computer. For certain interfaces, such as SCRAMNet+, writing to this address automatically triggers the reflective memory interrupt (see Section 3.4.2.1).

The restrictions on this address are a function of the specific reflective memory interface used (see Section 5 and Section 6). To specify the **ACLCommandSyncID** address.

ACL:

`:interface:reflectivememory:setup , <ACLCommandSyncID>` …

Specify the **`<ACLCommandSyncID>`** as either an integer or hex value.

GUI:

See Figure 3-6 Setting Real Time Block Addresses.

Touch **Setup→Interfaces→Real Time→ADDR** . Select **ACL Addresses**

Enter the ACL Command Sync ID address (hex) in the **Command_Sync_ID**field.

NOTE: Use the keyboard (**Kbd**) function, not the keypad, to enter hex addresses if the address contains non-numeric characters.

3.4.1.3 ACL Response Block Location

Acutrol must know the location in the reflective memory address space for the ACL response block data. The restrictions on this address are a function of the specific reflective memory interface used (see Section 5 and Section 6). To specify the demand block location:

ACL:

`:interface:reflectivememory:setup <DemandBlockAddress>` …

Specify the **`<DemandBlockAddress>`** as either an integer or hex value.

GUI:

See Figure 3-6 Setting Real Time Block Addresses.

Touch **Setup→Interfaces→Real Time→ADDR** . Select **ACL Addresses**

Enter the ACL Response Block address (hex) in the **Response Block** field.

NOTE: Use the keyboard (**Kbd**) function, not the keypad, to enter hex addresses if the address contains non-numeric characters.

3.4.1.4 ACL Response Sync ID Location

The **`ACLResponseSyncID`** is used to marshal access to the ACL response block between Acutrol and the Host Computer. For certain interfaces, such as SCRAMNet+, writing to this address automatically triggers the reflective memory interrupt.

The restrictions on this address are a function of the specific reflective memory interface used (see Section 5 and Section 6). To specify the **ACLResponseSyncID** address

ACL:

`:interface:reflectivememory:setup  , <ACLReponseSyncID>` …

Specify the **`<ACLResponseSyncID>`** as either an integer or hex value.

NOTE: Use the keyboard (**Kbd**) function, not the keypad, to enter hex addresses if the address contains non-numeric characters.

GUI:

See Figure 3-6 Setting Real Time Block Addresses.

Touch **Setup→Interfaces→Real Time→ADDR** . Select **ACL Addresses**

Enter the ACL Response Sync ID address (hex) in the **`Response_Sync_ID`** field.

NOTE: Use the keyboard (**`Kbd`**) function, not the keypad, to enter hex addresses if the address contains non-numeric characters.

3.4.2 ACL Channel Protocol

This section describes the protocol for sending ACL commands and receiving responses via the reflective memory ACL channel.

Acutrol sets location **`ACLComnmandSyncID`** to 0x0000 when it is ready to receive an ACL Command. The Host Computer should then use the following protocol to send an ACL Command to the Acutrol:

- Set the **`ACLResponseSyncID`** location equal to 0x0000. Check that the **`ACLCommandSyncID`** location is equal to 0x0000.

- Copy the ASCII text of the command string (e.g., ":read:variable 1234") to the ACL command block area of the reflective memory. Write the **`<NL>`** character (0x0A) to the location following the command. Write an ASCII null character (0x00) to the next location.

- Set the **`ACLCommandSyncID`** to 0x8000.

- Send an interrupt to the Acutrol node. (This happens automatically with SCRAMNet+).

- Wait for the **`ACLResponseSyncID`** location to be equal to 0x8000. Alternatively, the Host Computer can receive a reflective memory interrupt when this occurs.

- Read the response from the response data block area of the reflective memory. The first 32-bit word of this area is the error return from the ACL command (see Appendix C of *TM-8004, Acutrol3000 Command Language (ACL) Programming Manual* for a list of these codes). The second 32-bit word contains the length (in characters) of the response message. Following this is a null-terminated string (length characters long) that is the command response. The contents of this string are specific to the command that was sent. See *TM-8004 Acutrol3000® Command Language (ACL) Programming Manual* for details on ACL commands and their responses.

- Set the **`ACLResponseSyncID`** to 0x0000. Acutrol will not write the next response data if the **`ACLResponseSyncID`** is not 0x0000**.**

NOTE The Host Computer should only write to the ACL command area if the **`ACLCommandSyncID`** is 0x0000; otherwise, Acutrol may be in the process of reading the data. In the unlikely event that the **`ACLCommandSyncID`** is not clear, the Host Computer should wait 50 microseconds and retry.

NOTE If the **`ACLCommandSyncID`** is not 0x8000**,** the Host Computer should not read the frame data area, since Acutrol may be in the process of writing the data. In the event that this situation occurs, the Host Computer should wait 50 microseconds and retry.

NOTE The Host Computer should be sure to read the response for each command before sending the next command. Acutrol will not accept a new command until the response from the previous command has been read.

3.4.2.1 ACL Channel Command Reflective Memory Interrupts

Acutrol must receive an interrupt after the data is written to memory and the **`ACLCommandSyncID`** is set. In the case of the SCRAMNet+ interface, this happens automatically when the board is configured appropriately (see Section 5.3). In the case of the VMIC interface, the Host Computer must explicitly send an interrupt to the Acutrol VMIC node (see Section 6.3).

Acutrol will always signal a reflective memory interrupt to the Host Computer when it writes response data. It is the responsibility of the Host Computer to optionally receive this interrupt and then process it appropriately.

4 Acutrol3000 Real Time Variables

The following supervisor variables are specific to the real-time interfaces:

Variable Number	Description	Mnemonic	Data Type
s0040	Refl mem swap mode	S_RFM_SWAPMODE	DT_U8
s0041	Refl mem Node ID	S_RFM_NODE_ID	DT_U16
s0042	Refl mem endian mode	S_RFM_ENDIAN	DT_U16
s0043	Refl mem swap flag	S_RFM_SWAPFLAG	DT_U16
s0045	SCRAMNet Node Count	S_SCRAM_NODECOUNT	DT_U8
s0046	SCRAMNet Loopback Mode	S_SCRAM_LOOPMODE	DT_U8
s0047	SCRAMNet Timeout	S_SCRAM_TIMEOUT	DT_U16
s0050	keepOut count	S_KEEPOUT	DT_U32
s0054	hasSCRAMNet	S_HASSCRAMNET	DT_I32
s0055	hasVMIC	S_HASVMIC	DT_I32
s0056	hasParallel	S_HASPARALLEL	DT_I32
s0057	useHostTimeTracker	S_USEHOSTTIMETRACKER	DT_I32
s0058	useAIMTimeTracker	S_USEAIMTIMETRACKER	DT_I32
s0060	DmdBlock rfm addr	S_RFM_DMDBLK	DT_U32
s0061	DmdSync rfm addr	S_RFM_DMDSYNC	DT_U32
s0062	MonBlock rfmaddr	S_RFM_MONBLK	DT_U32
s0063	MonSync rfm addr	S_RFM_MONSYNC	DT_U32
s0064	Dmd Length	S_RFM_DMDLEN	DT_U32
s0065	Mon Length	S_RFM_MONLEN	DT_U32
s0066	AclCBlock rfm addr	S_RFM_CMDBLK	DT_U32
s0067	AclCSync rfm addr	S_RFM_CMDSYNC	DT_U32
s0068	AclRBlock rfm addr	S_RFM_RSPBLK	DT_U32
s0069	AclRSync rfm addr	S_RFM_RSPSYNC	DT_U32
s0100	Control word	S_CONTROLWORD	DT_U32
s0101	Control word enable	S_CTRLENABLE	DT_I32

Variable Number	Description	Mnemonic	Data Type
s0102	A2K status word	S_STATUSWORD	DT_U32
s150	Monitor skew frames	S_MONSKEWFRAME	DT_I32
s151	hostTimeoutAllModes	S_HOSTTOALLMODES	DT_I32
s0203	Realtime Interfaces event reg	S_EREG_RT	DT_U16
s0215	Parallel I/F status event reg	S_EREG_PARALLEL	DT_U16
s0216	Reflective mem I/F event reg	S_EREG_RFM	DT_U16
s0303	Realtime Interfaces condition reg	S_CREG_RT	DT_U16
s0314	TCPIP I/F status condition reg	S_CREG_TCPIP	DT_U16
s0315	Paralllel I/F status condition reg	S_CREG_PARALLEL	DT_U16
s0316	Reflective mem I/F status	S_CREG_RFM	DT_U16
s0501	Realtime monitor output[1]	S_RTMON_1	DT_F64
s0502	Realtime monitor output[2]	S_RTMON_2	DT_F64
s0503	Realtime monitor output[3]	S_RTMON_3	DT_F64
s0504	Realtime monitor output[4]	S_RTMON_4	DT_F64
s0505	Realtime monitor output[5]	S_RTMON_5	DT_F64
s0506	Realtime monitor output[6]	S_RTMON_6	DT_F64
s0507	Realtime monitor output[7]	S_RTMON_7	DT_F64
s0508	Realtime monitor output[8]	S_RTMON_8	DT_F64
s0509	Realtime monitor output[9]	S_RTMON_9	DT_F64
s0510	Realtime monitor output[10]	S_RTMON_10	DT_F64

Variable Number	Description	Mnemonic	Data Type
s0700	hostTracker	S_HT_NEEDSINIT	DT_I32
s0701	hostTracker	S_HT_CALCFLAG	DT_I32
s0703	hostTracker	S_HT_TICSPERSEC	DT_F64
s0704	hostTracker	S_HT_TSC_N_1	DT_TICS
s0705	hostTracker	S_HT_FP_N_1	DT_TICS
s0706	hostTracker	S_HT_FP_TICS	DT_TICS
s0707	hostTracker	S_HT_FP_SEC	DT_F64
s0708	hostTracker	S_HT_FT_N	DT_TICS
s0709	hostTracker	S_HT_FT_N_P1	DT_TICS
s0710	hostTracker	S_HT_CGAIN	DT_F64
s0711	hostTracker	S_HT_XGAIN	DT_F64
s0712	hostTracker	S_HT_RESET	DT_I32
s0715	AvgHostPd	S_AVGHOSTPERIOD	DT_F64
s0716	AvgHostPd-tics	S_AVGHP_TICS	DT_TICS
s0717	Next host time	S_NEXTHOST_TICS	DT_TICS
s0718	Current host time	S_CURRHOST_TICS	DT_TICS
s0720	aimTracker	S_AT_INITFLAG	DT_I32
s0721	aimTracker	S_AT_CALCFLAG	DT_I32
s0723	aimTracker	S_AT_TICSPERSEC	DT_F64
s0724	aimTracker	S_AT_TSC_N_1	DT_TICS
s0725	aimTracker	S_AT_FP_N_1	DT_TICS
s0726	aimTracker	S_AT_FP_TICS	DT_TICS
s0727	aimTracker	S_AT_FP_SEC	DT_F64
s0728	aimTracker	S_AT_FT_N	DT_TICS
s0729	aimTracker	S_AT_FT_N_P1	DT_TICS
s0730	aimTracker	S_AT_CGAIN	DT_F64
s0731	aimTracker	S_AT_XGAIN	DT_F64
s0732	aimTracker	S_AT_RESET	DT_I32
s0750	AvgAIMPd	S_AVGAIMPERIOD	DT_F64
s0751	AvgAIMPd-tics	S_AVGAP_TICS	DT_TICS
s0752	Next AIM time	S_NEXTAIM_TICS	DT_TICS

Variable Number	Description	Mnemonic	Data Type
s0753	Current AIM time	S_CURRAIM_TICS	DT_TICS
s0910	RFM mem map ver #	S_RFMMAPVER	DT_I32
s0911	RFM memory usage	S_RFMUSAGE1	DT_F32
s0912	RFM memory usage	S_RFMUSAGE2	DT_F32
s0913	RFM memory usage	S_RFMUSAGE3	DT_F32
s0914	RFM memory usage	S_RFMUSAGE4	DT_F32
s0915	RFM memory usage	S_RFMUSAGE5	DT_F32
s0916	RFM memory usage	S_RFMUSAGE6	DT_F32
s0917	RFM memory usage	S_RFMUSAGE7	DT_F32
s0918	RFM memory usage	S_RFMUSAGE8	DT_F32
s0999	Used to make a "hole" (int/float/double) in the memory map	S_HOLE	N/A

The following axis variables are used with the real-time interfaces:

Variable Number	Description	Mnemonic	Data Type
x194	RT Mon Posn var	V_RT_MON_POS_VAR	DT_F64
x195	RT Mon Rate var	V_RT_MON_RATE_VAR	DT_F64
x196	RT Mon Accl var	V_RT_MON_ACCEL_VAR	DT_F64
x500	host dmd position	V_HOST_POS_DMD	DT_F64
x501	host dmd rate	V_HOST_RATE_DMD	DT_F64
x502	host dmd accel	V_HOST_ACCEL_DMD	DT_F64
x503	host dmd jerk	V_HOST_JERK_DMD	DT_F64
x504	input variable #1	V_INVAR_1	DT_F64
x505	input variable #2	V_INVAR_2	DT_F64
x506	input variable #3	V_INVAR_3	DT_F64

Variable Number	Description	Mnemonic	Data Type
x507	input variable #4	V_INVAR_4	DT_F64
x508	input variable #5	V_INVAR_5	DT_F64
x509	input variable #6	V_INVAR_6	DT_F64
x510	input variable #7	V_INVAR_7	DT_F64
x511	input variable #8	V_INVAR_8	DT_F64
x520	filt'd host dmd pos	V_FHOST_POS_DMD	DT_F64
x521	filt'd host dmd rate	V_FHOST_RATE_DMD	DT_F64
x522	filt'd host dmd accel	V_FHOST_ACCEL_DMD	DT_F64
x523	filt'd host dmd jerk	V_FHOST_JERK_DMD	DT_F64
x524	filt'd dmd host time	V_FHOST_TIME	DT_F64
x525	hdf cdena	V_FHOST_CDENA	DT_I32
x526	hdf cdmode	V_FHOST_CDMODE	DT_I32
x527	hdf cdintlk	V_FHOST_CDINTLK	DT_I32
x528	hdf cdrem	V_FHOST_CDREMOTE	DT_I32
x530	new profile mode	V_NEWPROFILEMODE	DT_I32
x531	new lockout mode	V_NEWLOCKOUTMODE	DT_I32
x532	new intlk state	V_NEWINTLKSTATE	DT_I32
x540	track dmd position	V_TRACK_POS_DMD	DT_F64
x541	track dmd rate	V_TRACK_RATE_DMD	DT_F64
x542	track dmd accel	V_TRACK_ACCEL_DMD	DT_F64
x543	track dmd jerk	V_TRACK_JERK_DMD	DT_F64
x560	filt'd host monitor pos	V_FHOST_POS_MON	DT_F64
x561	filt'd host monitor rate	V_FHOST_RATE_MON	DT_F64
x562	filt'd host monitor accl	V_FHOST_ACCEL_MON	DT_F64
x563	filt'd host monitor jerk	V_FHOST_JERK_MON	DT_F64
x570	filt'd a2k-scaled pos	V_A2K_FHOST_POS_MON	DT_F64
x571	filt'd a2k-scaled rate	V_A2K_FHOST_RATE_MON	DT_F64
x572	filt'd a2k-scaled accl	V_A2K_FHOST_ACCEL_MON	DT_F64
x573	filt'd a2k-scaled jerk	V_A2K_FHOST_JERK_MON	DT_F64

Variable Number	Description	Mnemonic	Data Type
x600	dmd translator mode	V_DMDT_MODE	DT_I32
x601	dmd translator posFF	V_DMDT_POSFF	DT_F64
x602	dmd translator rateFF	V_DMDT_RATEFF	DT_F64
x603	dmd translator accelFF	V_DMDT_ACCELF	DT_F64
x604	dmd translator tDelay	V_DMDT_TDTICS	DT_TICS
x605	dmd translator tDelay	V_DMDT_TDSEC	DT_F64
x606	dmd translator fraction	V_DMDT_FRACTION	DT_F64
x607	dmd trans t_in_k_1	V_DMDT_TIK1TICS	DT_TICS
x608	dmd trans t_in dif	V_DMDT_TIDTICS	DT_TICS
x609	dmd translator t_in dif	V_DMDT_TIDSEC	DT_F64
x610	dmd translator K1	V_DMDT_K1	DT_F64
x611	dmd translator K2	V_DMDT_K2	DT_F64
x612	dmd translator p2	V_DMDT_P2	DT_F64
x613	dmd translator r2	V_DMDT_R2	DT_F64
x614	dmd translator a2	V_DMDT_A2	DT_F64
x615	dmd translator p(n)	V_DMDT_FHPN	DT_F64
x616	dmd translator p(n-1)	V_DMDT_FHPN_1	DT_F64
x617	dmd translator p3	V_DMDT_P3	DT_F64
x618	dmd trans neg diff	V_DMDT_NEGDIFF	DT_F64
x619	dmd trans neg diff cnt	V_DMDT_NEGCNT	DT_F64
x620	dmd translator p(n)	V_DMDT_PN	DT_F64
x621	dmd translator r(n)	V_DMDT_RN	DT_F64
x622	dmd translator a(n)	V_DMDT_AN	DT_F64
x623	dmd translator td limit	V_DMDT_TDLIM	DT_F64
x624	dmd translator offset (frames)	V_DMDT_TDOFFSET	DT_I32
x625	Dmd translator lead/lag (tics)	V_DMD_LEADLAG	DT_TICS
x630	mon translator mode	V_MONT_MODE	DT_I32

Variable Number	Description	Mnemonic	Data Type
x631	mon translator posFF	V_MONT_POSFF	DT_F64
x632	mon translator rateFF	V_MONT_RATEFF	DT_F64
x633	mon translator accelFF	V_MONT_ACCELFF	DT_F64
x634	mon translator tDelay	V_MONT_TDTICS	DT_TICS
x635	mon translator tDelay	V_MONT_TDSEC	DT_F64
x636	mon translator fraction	V_MONT_FRACTION	DT_F64
x637	mon trans t_in_k_1	V_MONT_TIK1TICS	DT_TICS
x638	mon trans t_in dif	V_MONT_TIDTICS	DT_TICS
x639	mon trans t_in dif	V_MONT_TIDSEC	DT_F64
x640	mon translator K1	V_MONT_K1	DT_F64
x641	mon translator K2	V_MONT_K2	DT_F64
x642	mon translator p2	V_MONT_P2	DT_F64
x643	mon translator r2	V_MONT_R2	DT_F64
x644	mon translator a2	V_MONT_A2	DT_F64
x645	mon translator p(n)	V_MONT_FHPN	DT_F64
x646	mon translator p(n-1)	V_MONT_FHPN_1	DT_F64
x647	mon translator p3	V_MONT_P3	DT_F64
x648	mon translator p4	V_MONT_P4	DT_F64
x649	mon translator p5	V_MONT_P5	DT_F64
x650	mon translator p(n)	V_MONT_PN	DT_F64
x651	mon translator r(n)	V_MONT_RN	DT_F64
x652	mon translator a(n)	V_MONT_AN	DT_F64
x653	mon translator td limit	V_MONT_TDLIM	DT_F64
x654	mon translator offset (frames)	V_MONT_TDOFFSET	DT_I32
x655	mon translator lead/lag (tics)	V_MON_LEADLAG	DT_TICS
x665	mon integrator dT(sec)	V_MON_DT	DT_F64
x666	a3k clock(double tics)	V_MONNOW_TICS	DT_F64

Variable Number	Description	Mnemonic	Data Type
x667	next host(double tics)	V_MONTHEN_TICS	DT_F64
x668	tics per sec	V_MONTPS	DT_F64
x670	monTranslator input pos	V_MON_IN_POS	DT_F64
x671	monTranslator input rate	V_MON_IN_RATE	DT_F64
x672	monTranslator input accl	V_MON_IN_ACCEL	DT_F64
x673	monTranslator input jerk	V_MON_IN_JERK	DT_F64
x676	DemandStateTracker	V_DST_TRACKMODE	DT_I32
x677	DemandStateTracker	V_DST_HASPOS	DT_I32
x678	DemandStateTracker	V_DST_HASRATE	DT_I32
x679	DemandStateTracker	V_DST_HASACCEL	DT_I32
x680	DemandStateTracker	V_DST_DEMANDMODE	DT_I32
x681	DemandStateTracker	V_DST_BANDWIDTH	DT_F64
x682	DemandStateTracker	V_DST_DAMPING	DT_F64
x683	DemandStateTracker	V_DST_ZETAGAIN	DT_F64
x684	DemandStateTracker	V_DST_INTEGMETH	DT_I32
x685	DemandStateTracker	V_DST_LIMPHI	DT_F64
x686	DemandStateTracker	V_DST_LIMPLO	DT_F64
x687	DemandStateTracker	V_DST_LIMR	DT_F64
x688	DemandStateTracker	V_DST_LIMA	DT_F64
x689	DemandStateTracker	V_DST_INPOS	DT_F64
x690	DemandStateTracker	V_DST_HATPOS	DT_F64
x691	DemandStateTracker	V_DST_HATRATE	DT_F64
x692	DemandStateTracker	V_DST_HATACCEL	DT_F64
x693	DemandStateTracker	V_DST_P2	DT_F64
x694	DemandStateTracker	V_DST_P3	DT_F64
x695	DemandStateTracker	V_DST_X	DT_F64
x696	DemandStateTracker	V_DST_A2	DT_F64
x697	DemandStateTracker	V_DST_A3	DT_F64
x698	DemandStateTracker	V_DST_RATEFF	DT_F64

Variable Number	Description	Mnemonic	Data Type
x699	DemandStateTracker	V_DST_ACCELFF	DT_F64

5 Acutrol3000 SCRAMNet+ Reflective Memory Interface

5.1 SCRAMNet+ Hardware

The SC150 is a high-speed reflective memory solution from Systran. It uses a standard PCI form factor, and operates in a ring topology. Connection to the network is via a pair (transmit/receive) of duplex (clock/data) fiber optic cable.

Key features of the SC150 are:

- High Speed. Messages are sent around the ring at 150 MBit/sec. Typical ring transport cycles are less than 10 µsec.
- Low Overhead. No bus contentions, no protocols to use
- Packets can vary in size up to 1024 bytes.
- Distance between nodes can be up to 300 meters with multimode fiber, and up to 3500 meters with single-mode fiber.
- The card contains both a 33 MHz/32-bit compatible PCI bus interface (PCI Spec 2.1)
- All communication to the network is via onboard hardware, i.e., no host processor involvement in the operation of the network. Hardware is included for configurable endian conversion for multiple CPU architectures on the same network.

  NOTE: By convention the SCRAMNet+ memory is big endian.
- Memory
  - Each node can have up to 8 Mbytes of memory that is dual ported to the SCRAMNet$^+$ ring.
  - Memory is updated at an exact deterministic interval.
  - Bit Error Rates are ~10^{-15}, meaning an error would only occur every 76 days at maximum usage.
  - In Platinum mode, any network timeouts or bit errors that do occur will cause automatic retransmission of message.
- Up to 256 nodes can be active in system

- Interrupts
    - Each node can generate and/or receive local and network interrupts when data is transmitted and/or received at a specific memory location.
    - This is programmable on a 32-bit word basis.
- Triggers
    - Each node can generate and/or receive local and network TTL Triggers when data is transmitted and/or received at a specific memory location.
    - Triggers generate a 26.64 ns TTL level compatible, non-terminated, output.
- Internal Programmable Counter/Timer
    - Free Run Mode. 26.66 nsec resolution
    - Ring Timer Mode. 26.66 nsec resolution. Can be used to measure transport delay times
    - Global Timer Mode. 1.7 µsec resolution. Synchronized Network Wide, can be used to Time-Tag Data
- Data Filter Mode
    - Can be used to minimize network traffic of repetitive messages
    - Data is only transmitted when it changes
- DWORD Data Integrity (HIPRO Mode)
    - Can be programmed to only transmit data only when all bytes of a DWORD have been written. This ensures data coherency for systems with bus widths smaller than their data words.
- Loopback Testing. Each Node has capabilities for Loopback Testing at various levels through the hardware.

5.2 SCRAMNet+ Installation

All configuration of the SCRAMNet+ interface is done through software. Therefore, installation of the Acutrol3000 SCRAMNet+ Memory Interface consists of only two steps:

- Installation into the Acutrol3000

- Enabling the SCRAMNet+ interface in the system.xml file

5.2.1 Installation into the Acutrol3000 Chassis

The installation into the Acutrol3000 Chassis is documented in Acutronic drawing 1202E52A. The SCRAMNet+ interface is supplied as a modification kit to the base chassis. Three components are supplied:

- **SC150 Card.** This card is installed into the leftmost PCI slot in the Acutrol3000 backplane and secured with a mounting screw. See Figure 5-1.
- **Rear Panel.** A modified rear panel is supplied for customer access to the fiber optic connections. The panel is mounted in place of the existing blank panel. See Figure 5-2.
- **Interconnecting cables.** Fiber Optic cables are supplied to go from the SC150 card to the bulkhead connections.

To install the SCRAMNet+ Interface:

1. Turn off and unplug the Acutrol.
2. Unscrew the front cover and fold out of the way.
3. Remove the top cover, being careful to disconnect the fan power cable.
4. Touch a metal part of the Acutrol chassis to discharge any static electricity that might be on your clothes or body.
5. Insert the SC150 card into the leftmost PCI slot. It can be a tight fit, be careful not to force the device into place.
6. Screw the mounting bracket of the SC150 card to the back panel rail of the Acutrol.
7. Remove the side bolts of the upper rail of the rear panel. Remove the upper rail. Replace the blank rear panel with the SCRAMNet+ Interface panel. Install the upper rail and the side bolts.
8. Install the fiber optic cables from the SC150 card to the rear panel.
9. Visually verify the installation is correct.
10. Replace the top cover, being careful to connect the fan power supply. Fold the front cover back into position and secure with the three retaining screws.
11. Plug in and turn on the Acutrol.

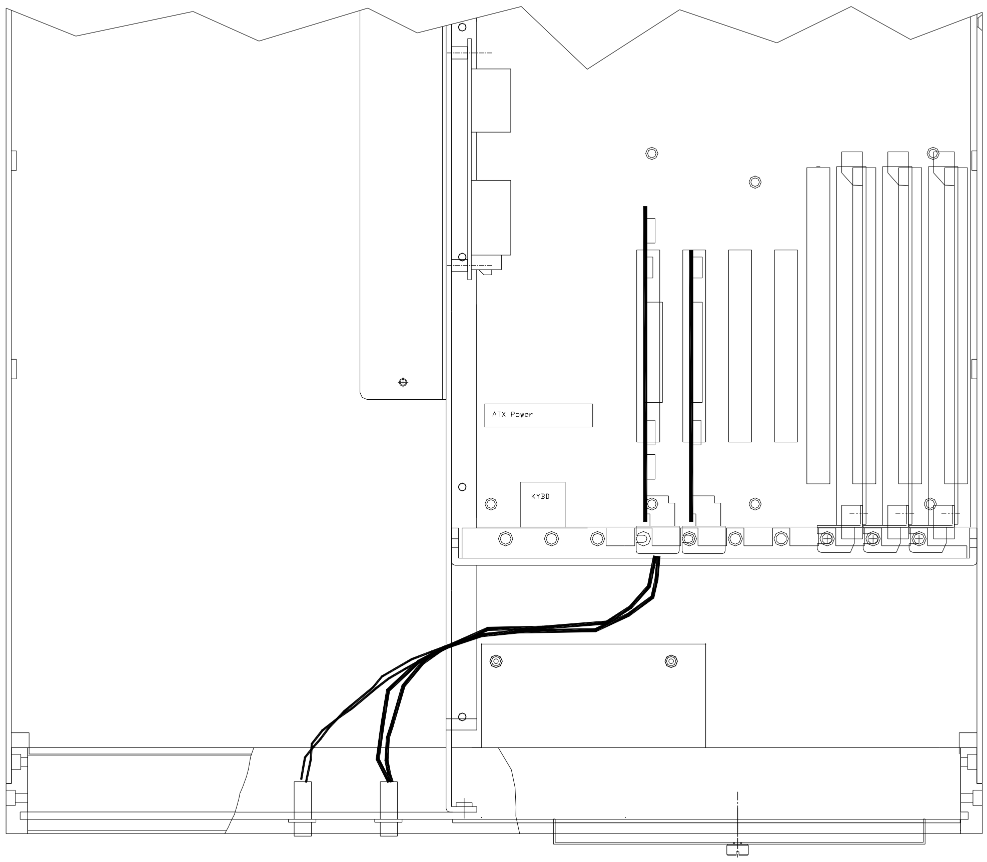


Figure 5-1 SCRAMNet+ Card Inserted into Acutrol3000 Chassis

5.2.2 Enabling the SCRAMNet+ Interface

Once the card has been installed into the chassis, the system software must be configured to acknowledge the board.

- Edit the system.xml file on the Real Time Computer and change the **hasSCRAMNet** field to “TRUE” from “FALSE”
- Edit the BIOS to configure PCI Slot 4 for Interrupt 12.
- Reboot the Acutrol

On startup the Acutrol will now look for the presence of the SCRAMNet+ board and report an error if either the hardware cannot be found or if the driver cannot be initialized.

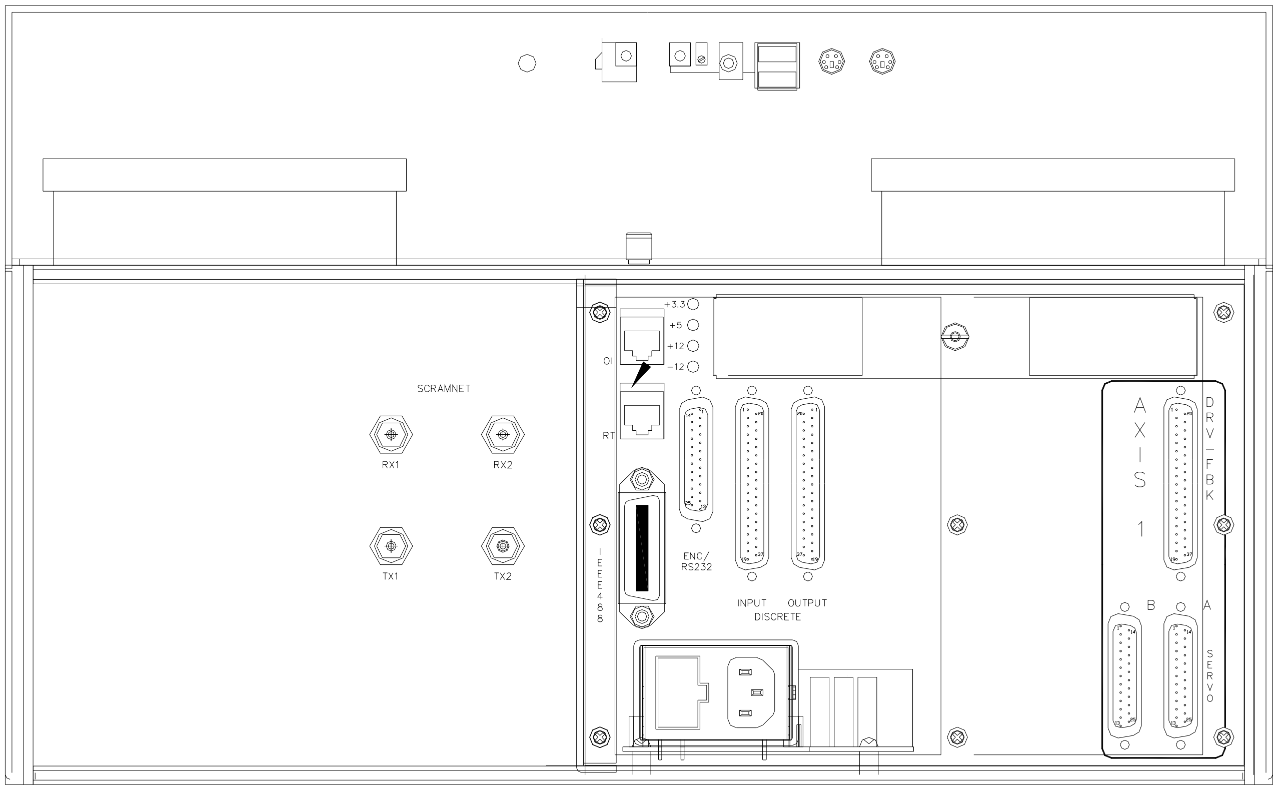


Figure 5-2 Acutrol3000 Rear Panel Showing SCRAMNet+ Interface

5.3 SCRAMNet+ Configuration

There are two parameters that must be configured for the SCRAMNet+ Interface. All other parameters are configured by the Acutrol3000 Real Time Software.

5.3.1 SCRAMNet+ Node ID

To set the SCRAMNet+ Node ID, edit the **nodeID** attribute of the **`<RFM_CONFIG`…`>`** element in the system.xml file. Valid values are 0…255.

5.3.2 SCRAMNet+ Ring Mode

The SCRAMNet+ Ring can be placed has several modes depending on the functionality to be implemented. These modes are: MONITOR, INSERT, FIBER, WIRE, and MECHANICAL. These refer to the connection of the SCRAMNet+ card to the SCRAMNet+ Network. The default condition is INSERT. See the SCRAMNet+ documentation for descriptions of the other modes.

To specify the SCRAMNet+ Ring Mode:

Interface:Scram:Mode Insert

ACL:

`:interface:Scram:Mode <mode>`

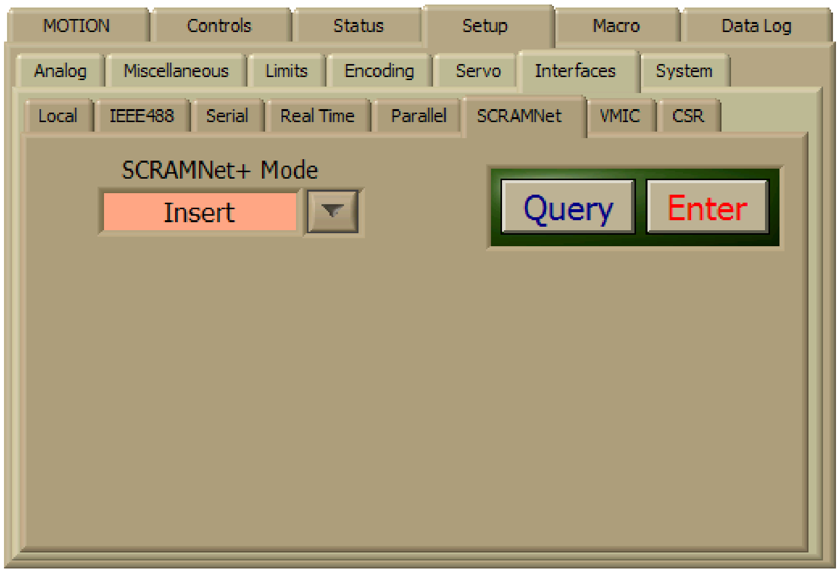
Specify the **`<mode>`** (Monitor | Wire | Mechanical | Fiber | Insert)

GUI:

See Figure 5-3 SCRAMNet+ Mode Setup.

Touch **Setup→Interfaces→SCRAMNet**.

In the **SCRAMNet+ Mode** pickbox, select the desired mode.



SCRAMNet+ Mode: Insert

Figure 5-3 SCRAMNet+ Mode Setup

5.4 SCRAMNet+ Features

The SCRAMNet+ Interface has all of the features of the Acutrol3000 Real Time Interface:

- Host Time Tracker
- Acutrol Time Tracker
- Host Demand Tracker
- Host Demand Translator
- Monitor Translator
- ACL Interface Block
- IEEE 754.1 Single and Double Precision Float and Binary Data Types

5.5 SCRAMNet+ Usage

This section gives an example customer (host computer) setup for a PCI SCRAMNet+ Card and using the Systran DLL. Setup is similar for other flavors of the SCRAMNet+ Card. Note that this section does not apply to the SCRAMNet+ card in the Acutrol3000 computer.

5.5.1 PCI SCRAMNet+ Setup

5.5.1.1 Installation

Refer to Section 4 of the Systran SCRAMNet+ Network PCI-150e Hardware Reference Manual.

5.5.1.2 Board Configuration

- Set EEPROM Write Jumper (J303).
    - Move J303 to “Enable” Position to allow EEPROM writes.
    - *Move J303 to “Disable” Position to disallow EEPROM writes after EEPROM Initialization.*
- Set EEPROM Read Jumper (J304)
    - Move J304 to “Enable” Position to allow EEPROM reads.

- o *Move J304 to "Disable" Position to disallow EEPROM reads after EEPROM initialization.*

5.5.1.3 EEPROM Configuration

The EEPROM configuration only needs to be completed once to configure the board. Program the EEPROM with the values set below.

NOTE: The factory default settings for the EEPROM are adequate for use in this system and need not be changed.

NOTE: It is quite easy to get confused with the memory map of the EEPROM programming. There is not a one-to-one correspondence between the EEPROM CSR locations and the I/O space CSR locations, so caution must be taken in interpreting the values. See Section 4.80 of the *SCRAMNet+ Network PCI-150e Hardware Reference* for details.

The EEPROM can be programmed with a software package purchased from Systran, or by manually toggling the Serial EEPROM bits in CSR8.

- o Set Node Identification (CSR3[15:8])
    - ▪ Set CSR3 to 0x0000.
- o Set Network Timeout (CSR5)
    - ▪ Two Cards + 20 m cable + 1 => Set CSR5 to 0x0004. (Factory Default is 0x0010)
    - ▪ Value = # Nodes In Ring + (Total Length of Cable in Meters̃50) + 1
    - ▪ Each bit equals 240 nsec delay

5.5.2 CSR Setup – PCI

Configure each of the CSR registers as shown on the following register maps.

CSR0

General SCRAMNet+ Enable and Reset (R/W)

Hex	Bin	Bit	Function	Name
3	1	0	Rx Enable	RX_ENB
	1	1	Tx Enable	TXEN
	0	2	Redund TxRx Toggle	RTT
	0	3	Host Interrupt Enable	HIE
0	0	4	Auxiliary Control Ram Enable	ACRE
	0	5	Interrupt on Memory Mask Match Enable	IMME
	0	6	Override RIE Flag	ORF
	0	7	Interrupt On Errors	IOE
1	1	8	Network Interrupt Enable	NIE
	0	9	Override TIE Flag	OTF
	0	10	Enable Tx Data Filter	DFEN
	0	11	Enable Lower 4 Kbytes for Data Filter	EN4K
8	0	12	RSEST Tx/Rx FIFO	RTRF
	0	13	RSEST Interrupt FIFO	RSTIF
	0	14	RSEST Tx FIFO	RTXF
	1	15	Insert Node	INSRT

CSR1

Error Indicators (R/W)

Hex	Bin	Bit	Function	Name
0	0	0	Tx FIFO Full	TXFF
	0	1	Tx FIFO Not Empty	TXFNE
	0	2	Tx FIFO 7/8 Full	TXFAF
	0	3	(Always 0)	
0	0	4	Interrupt FIFO Full	IFF
	0	5	Protocol Violation	PV
	0	6	Carrier Detect Fail	CDF
	0	7	Bad Message	BB
0	0	8	Receiver Overflow	RXO
	0	9	Transmit Retry	TXRTY
	0	10	Transmit Retry Time-Out	TO
	0	11	Redundant TxRx Fault	RTF
A	0	12	General Purpose Counter/Timer Overflow	GPCTO
	1	13	Redundant TxRx Link 1=A/0=B	RLAB
	0	14	Interrupts Armed – Write to re-arm	IARM
	1	15	Fiber Optic Bypass Not Connected	FOB

CSR2

Node Control (R/W)

Hex	Bin	Bit	Function	Name
0	0	0	Available to Host	
	0	1	Available to Host	
	0	2	Available to Host	
	0	3	Available to Host	
4	0	4	Available to Host	
	0	5	Available to Host	
	1	6	Disable Fiber Optic Loopback	FO_DIS
	0	7	Enable Wire Loopback	EN_WR_LPB
0	0	8	Disable Host to Memory Write	DIS_H_M_WR
	0	9	Enable Write of Own Slot to Memory	WOSEN
	0	10	Enable Interrupt on Receipt of Own Interrupt Slot	IOSEN
	0	11	1024 vs 256 variable size max (bytes)	LEN_LIMIT
6	0	12	Variable Length Messages on Network	VAR_LEN
	1	13	HIPRO Write Enable	HIPRO
	1	14	Allow Multiple Native Messages on Network	MULTIPLE_MSG
	0	15	No Network Error Correction	NO_ERR_CRCT

CSR3

Node Information* (R)

Hex	Bin	Bit	Function		Name
2	0	0	NN0	T_AGE0	NN0/T_AGE0
	1	1			NN1/T_AGE1
	0	2	Number		NN2/T_AGE2
	0	3	of	Transmit	NN3/T_AGE3
0	0	4	Nodes	Age	NN4/T_AGE4
	0	5			NN5/T_AGE5
	0	6			NN6/T_AGE6
	0	7	NN7	T_AGE7	NN7/T_AGE7
0	0	8	NID0	RXID0	NID0/RXID0
	0	9			NID1/RXID1
	0	10			NID2/RXID2
	0	11	NODE ID	Receive ID	NID3/RXID3
0	0	12	(Transmit ID)		NID4/RXID4
	0	13			NID5/RXID5
	0	14			NID6/RXID6
	0	15	NID7	RXID7	NID7/RXID7

* Depends on the setting of CSR8[0] ID_MUX

CSR4

Interrupt Address (LSP) (R)

Hex	Bin	Bit	Function	Name
0	0	0	Always 0	
	0	1	Always 0	
	0	2	RFA2	RFA2
	0	3		RFA3
0	0	4		RFA4
	0	5		RFA5
	0	6		RFA6
	0	7	Interrupt FIFO	RFA7
0	0	8	ADDRESS FIELD (LSW)	RFA8
	0	9		RFA9
	0	10		RFA10
	0	11		RFA11
0	0	12		RFA12
	1	13		RFA13
	1	14		RFA14
	0	15	RFA15	RFA15

CSR5

Interrupt Address (MSW) and Status* (R)

Hex	**Bin**	**Bit**	**Function**	**Name**
0	0	0	RFA16	RFA16
	0	1		RFA17
	0	2		RFA18
	0	3	Interrupt FIFO	RFA19
0	0	4	ADDRESS FIELD (MSW)	RFA20
	0	5		RFA21
	0	6	RFA22	RFA22
	0	7	Reserved	
0	0	8	Reserved	
	0	9	Reserved	
	0	10	Reserved	
	0	11	Reserved	
0	0	12	Reserved	
	1	13	Reserved	
	1	14	Retry Bit in Interrupt FIFO	(RF_RETRY)
	0	15	Interrupt FIFO Not Empty	(~RX_F_E)

*Writing the Transit Time-out Value to CSR5 stores it in shadow memory. Do not set this value to ‘0’. A value of ‘0’ prevents host-generated data from leaving the Transmit FIFO.

CSR6

Reserved (R)

Hex	Bin	Bit	Function	Name
0	0	0	Reserved	
	0	1	Reserved	
	0	2	Reserved	
	0	3	Reserved	
0	0	4	Reserved	
	0	5	Reserved	
	0	6	Reserved	
	0	7	Reserved	
0	0	8	Reserved	
	0	9	Reserved	
	0	10	Reserved	
	0	11	Reserved	
0	0	12	Reserved	
	1	13	Reserved	
	1	14	Reserved	
	0	15	Reserved	

CSR7

Reserved (R)

Hex	Bin	Bit	Function	Name
0	0	0	Reserved	
	0	1	Reserved	
	0	2	Reserved	
	0	3	Reserved	
0	0	4	Reserved	
	0	5	Reserved	
	0	6	Reserved	
	0	7	Reserved	
0	0	8	Reserved	
	0	9	Reserved	
	0	10	Reserved	
	0	11	Reserved	
0	0	12	Reserved	
	1	13	Reserved	
	1	14	Reserved	
	0	15	Reserved	

CSR8

General SCRAMNet+ SC150e Extended Control Register (R/W)

Hex	Bin	Bit	Function	Name
0	0	0	0: CSR3=NN0:7,NID0:7; 1: CSR3=TAGE0:7,RXID0:7	ID_MUX
	0	1	Disable Holdoff Feature	DIS_HOLD
	0	2	Chip Select to EEPROM	CSR_CS0
	0	3	Ext. Chip Select for AUX Microwire Peripheral	CSR_CS1
0	0	4	MICROWIRE DOUT Pin	CSR_DOUT
	0	5	EEPROM Program Enable	E_PRE
	0	6	Clock Line to Microwire Port	CSR_CK
	0	7	DIN Line connected to the Microwire DOUT Pins	E_DIN
8	0	8	Initiate Initiation Sequence - CSR Reset	CSR_RST
	0	9	Override Counter Mode	GPC_FRE
	0	10	Receive Interrupt Override	RX_INT_OVR
	1	11	1= Mechanical Switch Override, 0=Invoke COAX Loopback Mode	C_MECHSW
F	1	12	Memory Size Configuration	MC10
	1	13	Memory Size Configuration	MC11
	1	14	Memory Size Configuration	MC12
	1	15	Reserved (Always 1)	

CSR9

SCRAMNet+ SC150e Interrupt on Error Mask (R/W)

Hex	Bin	Bit	Function	Name
0	0	0	Transmit FIFO Full Mask	M_TX_F_F
	0	1	Transmit FIFO Not Empty Mask	M_TX_F_E
	0	2	Transmit FIFO 7/8 Full Mask	M_TX_F_AF
	0	3	Internal 82 bit BIST shift register output	BIST_STREAM
0	0	4	Receiver FIFO Full Mask	M_RX_F_F
	0	5	Protocol Violation Mask	M_PV
	0	6	Carrier Detect Fail Mask	M_CD_FAIL
	0	7	Bad Message Mask	M_BM
0	0	8	Receiver Overflow Mask	M_RX_OVR
	0	9	Transmitter Retry Mask	M_RETRY
	0	10	Transmitter Retry - Time-Out	M_RETRY_T_O
	0	11	Redundant Transmit/Receive Fault Mask	M_FAULT
0	0	12	Interrupt on General Purpose Counter Overflow	M_COUNT_OVR
	0	13	General Purpose Counter/Timer Modes	M_INC_TRIGS
	0	14	General Purpose Counter/Timer Modes	M_INC_ERRS
	0	15	Fiber Optic Bypass Not Connected Mask	M_FO_BYPASS

CSR10

DEC_IGNORE (R/W)

Hex	Bin	Bit	Function	Name
0	0	0	0=DEC System, 1=Non-DEC System	DEC_IGNORE
	0	1	Reserved	
	0	2	Reserved	
	0	3	Reserved	
0	0	4	Reserved	
	0	5	Reserved	
	0	6	Reserved	
	0	7	Reserved	
0	0	8	Reserved	
	0	9	Reserved	
	0	10	Reserved	
	0	11	Reserved	
0	0	12	Reserved	
	1	13	Reserved	
	1	14	Reserved	
	0	15	Reserved	

CSR11

Reserved (R)

Hex	Bin	Bit	Function	Name
0	0	0	Reserved	
	0	1	Reserved	
	0	2	Reserved	
	0	3	Reserved	
0	0	4	Reserved	
	0	5	Reserved	
	0	6	Reserved	
	0	7	Reserved	
0	0	8	Reserved	
	0	9	Reserved	
	0	10	Reserved	
	0	11	Reserved	
0	0	12	Reserved	
	1	13	Reserved	
	1	14	Reserved	
	0	15	Reserved	

CSR12

SCRAMNet+ SC150e Virtual Paging Register (R/W)

Hex	Bin	Bit	Function	Name
0	0	0	Enables Virtual Paging when Set	VP
	0	1	Always 0	
	0	2	Always 0	
	0	3	Always 0	
0	0	4	Always 0	
	0	5	VPA12	VPA12
	0	6		VPA13
	0	7		VPA14
0	0	8		VPA15
	0	9	VIRTUAL	VPA16
	0	10	PAGE	VPA17
	0	11	NUMBER	VPA18
0	0	12		VPA19
	1	13		VPA20
	1	14		VPA21
	0	15	VPA22	VPA22

CSR13

General Purpose Counter/Timer (R)

Hex	Bin	Bit	Function	Name
0	0	0	RD_COUNT0	RD_COUNT0
	0	1		RD_COUNT1
	0	2		RD_COUNT2
	0	3		RD_COUNT3
0	0	4		RD_COUNT4
	0	5		RD_COUNT5
	0	6	GENERAL	RD_COUNT6
	0	7	PURPOSE	RD_COUNT7
0	0	8	COUNTER/	RD_COUNT8
	0	9	TIMER	RD_COUNT9
	0	10	REGISTER	RD_COUNT10
	0	11		RD_COUNT11
0	0	12		RD_COUNT12
	1	13		RD_COUNT13
	1	14		RD_COUNT14
	0	15	RD_COUNT15	RD_COUNT15

CSR14

Reserved (R)

Hex	Bin	Bit	Function	Name
0	0	0	Reserved	
	0	1	Reserved	
	0	2	Reserved	
	0	3	Reserved	
0	0	4	Reserved	
	0	5	Reserved	
	0	6	Reserved	
	0	7	Reserved	
0	0	8	Reserved	
	0	9	Reserved	
	0	10	Reserved	
	0	11	Reserved	
0	0	12	Reserved	
	1	13	Reserved	
	1	14	Reserved	
	0	15	Reserved	

CSR15

Reserved (R)

Hex	Bin	Bit	Function	Name
0	0	0	Reserved	
	0	1	Reserved	
	0	2	Reserved	
	0	3	Reserved	
0	0	4	Reserved	
	0	5	Reserved	
	0	6	Reserved	
	0	7	Reserved	
0	0	8	Reserved	
	0	9	Reserved	
	0	10	Reserved	
	0	11	Reserved	
0	0	12	Reserved	
	1	13	Reserved	
	1	14	Reserved	
	0	15	Reserved	

ACR			
?	0	0	RIE
	X	1	TIE
	0	2	EXT TRG 1
	0	3	EXT TRG 2
?	Y	4	HIPRO ENB
	0	5	Reserved
	0	6	Reserved
	0	7	Reserved

X=1 for Demand SYNC ID, 0 for all else

Y=1 for Demand Block, 0 for all else

5.5.3 PCI SCRAMNet+ INITIALIZATION SEQUENCE USING SYSTRAN DLL

Initializing the SCRAMNet+ consists of five parts: Initializing the Systran DLL, Initializing the Control Status Registers (CSRs), initializing the Auxiliary Control Ram (ACR), initializing the reflective memory, and initalizing the Acutrol.

NOTE: All numerical values are given in HEX unless specifically noted.

5.5.3.1 Initializing the Systran DLL

The Systran DLL is initialized with the following calls:

Systran DLL Call	Description
`char _sp_scram_init(void);`	Read Configuration and map registers into memory
`void _scr_reset_mm(void);`	Reset Node
`unsigned char _SetScrTransactionType(unsigned char 0x10);`	Set Byte-Swapping method to Byte to perform little endian to big endian conversion

5.5.3.2 Initializing the Control Status Registers (CSRs)

The CSRs have been mapped into memory via the PCI chip set. Each register is 16 bits wide. Set the CSRs as shown below.

Systran DLL Call	Description
`void _scr_csr_write(unsigned short int 0x000C, unsigned short int 0x0000);`	Set CSR12 to 0000 (Not using Virtual Paging)
`void _scr_csr_write(unsigned short int 0x000A, unsigned short int 0x0001);`	Set CSR10 to 0001 (Non-DEC system)
`void _scr_csr_write(unsigned short int 0x0000, unsigned short int 0x0000);`	Set CSR09 to 0000 (Not using interrupts)
`void _scr_csr_write(unsigned short int 0x0008, unsigned short int 0x0800);`	Set CSR08 to 0800 (Enable the Mechanical Switch Override and enable Holdoff Mode)
`void _scr_csr_write(unsigned short int 0x0002, unsigned short int 0x6040);`	Set CSR02 to 6040 (Use Platinum BURST Mode, disable PLUS modes, Enable HIPRO write, and disable the fiber optic loopback)
`void _scr_csr_write(unsigned short int 0x0000, unsigned short int 0x0000);`	Set CSR00 to 0 to disable SCRAMNet+ till ready

5.5.3.3 Initializing the Auxiliary Control RAM (ACR)

Command	Description
`void _scr_acr_write(unsigned long addr, unsigned char 0x0000);`	Clear all ACR Memory.
`void _scr_acr_write(unsigned long addr, unsigned char 0x0010);`	Set every 32-bit word in the Demand Buffer to 0x0010 to enable HIPRO Mode.
`void _scr_acr_write(unsigned long 0xTBD, unsigned char 0x0002);`	Set the Demand SYNC ID to 0x0002 to enable the Transmit Interrupt Enable (TIE) bit.

5.5.3.4 Initializing the Shared Memory (SM)

Command	Description
`void _scr_csr_write(unsigned short int 0x0000, unsigned short int 0xF000);`	Set CSR0 to F000 to insert the node and initiate reset of the FIFOs.
`void _scr_csr_write(unsigned short int 0x0000, unsigned short int 0x8103);`	Set CSR0 to 8103 to insert the node, toggle the reset of the FIFOs, disable the Data Filter, enable Network (Transmit) Interrupts, and enable network activity.
`unsigned short int _scr_csr_read(unsigned short int 0x0001);`	Read CSR1 to check for any existing error conditions. Check specifically for Carrier Detect Fail CSR1[6]
`unsigned long _WriteSCRLong(unsigned long 0x00000200/4, unsigned long 0x00000000);`	Write a value to SCRAMNet+ memory from at least one node. This enables all powered node transmitters and checks for fiber optic ring integrity.
`unsigned short int _scr_csr_read(unsigned short int 0x0001);`	Read CSR1 again to check for any new error conditions. If no errors are present, then the initialization was successful. Check specifically for Carrier Detect Fail CSR1[6]

5.5.3.5 Initializing Acutrol

Command	Description
`unsigned long _WriteSCRLong(unsigned long addr, unsigned long 0x00000000);`	Set every 32-bit word in the Demand Buffer to 0x00000000. This sets the Position Demand, Rate Demand, and Acceleration Demand to zero for each axis.
`unsigned long _WriteSCRLong(unsigned long addr, unsigned long 0x00000000);`	Set Position Mode for Axis 1, Axis 2, and Axis 3. Open the Interlocks and set the system to Local Mode
`unsigned long _WriteSCRLong(unsigned long addr, unsigned long 0x80000000);`	Set Demand SYNC ID to 0x80000000 to transfer data.

6 Acutrol3000 VMIC Reflective Memory Interface

6.1 VMIC Hardware

The VMIPCI-5565 is a very high-speed reflective memory solution from VMIC. It uses a standard PCI form factor, and operates in a ring topology. Connection to the network is via a duplex (transmit/receive) fiber optic cable.

Key features of the VMIPCI-5565 are:

- Very high-speed fiber-optic network. Communication occurs at 2.12 GBaud serially.
- Packets are dynamically sized, from 4 to 64 data bytes per packet. Transfer rate range from 43 Mbyte/sec (4 byte packet) to 174 Mbyte/sec (64 byte packet)
- Distance between nodes can be up to 300 meter with multimode fiber, and up to 10 kilometer with single-mode fiber. (Redundant Mode of Operation are available)
- The card contains both a 66 MHz/64-bit and a 33 MHz/32-bit compatible PCI bus interface, 3.3 or 5.5 V logic level.
- All communication to the network is via onboard hardware, i.e., no host processor involvement in the operation of the network. Hardware is included for configurable endian conversion for multiple CPU architectures on the same network.

  NOTE: By convention the VMIC memory is little endian.
- Up to 256 unique nodes can exist on the same network; each can contain 64 Mbyte or 128 Mbyte SDRAM reflective memory with parity.
- Access to the reflective memory is via programmed I/O or two independent DMA channels.
- The card can generate four general-purpose network interrupts with 32 bits of data each.

6.2 VMIC Installation

Installation of the Acutrol3000 VMIC Reflective Memory Interface consists of three steps:

- Configuring the Jumper Settings on the VMIPCI-5565
- Installation into the Acutrol3000
- Enabling the VMIC interface in the system.xml file

6.2.1 VMIPCI-5565 Jumper Settings

There are two jumpers that need to be configured on the VMIPCI-5565 prior to operation:

- **Jumper E4 (Node ID).** Each node in network must have a unique node ID. See Figure 6-1on for the location of jumper block E4. Jumper block E4 can accommodate 8 jumper shunts, which correspond to 8 node ID select signal lines. The 8 node ID select lines permit any binary node ID from 0x00 to 0xFF. Jumper block E4, pins 1 and 2 correspond to the least significant node ID line and pins 15 and 16 correspond to the most significant node ID line. Installing a jumper shunt sets the binary node ID line low (0), while removing a jumper shunt sets the binary node ID line high (1). For example, a Node ID of 0x00 would require installation of all eight jumpers and a Node ID of 0xFF would require no jumpers to be installed.

- **Jumper E7 (Operating Mode).** Jumper E7 controls the operating mode of the board. The default factory configuration is all jumper shunts installed (except pins 7 and 8). The details of these modes are beyond the scope of the document and are explained in the VMIC documentation. These modes are hardware settings and do not affect the operation with the Acutrol3000. They do have a pronounced effect on the network and the operation of all other nodes must be considered before adjusting these settings. Figure 6-1 lists the jumpers and their functions.

NOTE: Pins 7 and 8 of Jumper E7 are currently reserved and should not be used.

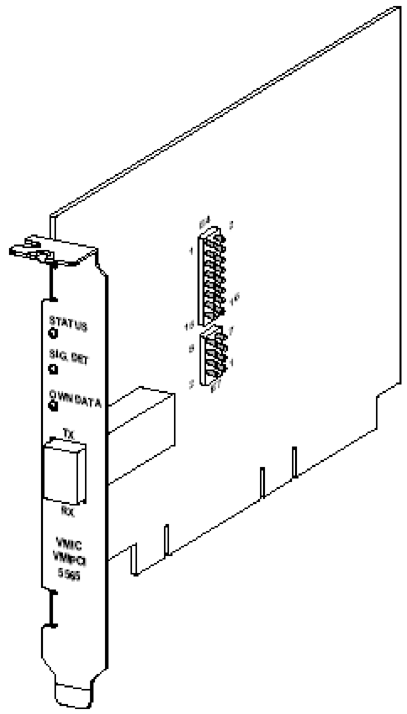


Figure 6-1- VMIPCI-5565 Jumper Locations

Jumper E7 Pins	Installed/Omitted	Function/Mode Selected
1 to 2	Installed	Non-redundant (Fast) network transfer mode selected
	Omitted	Redundant network transfer mode selected
3 to 4	Installed	Rogue Master 0 function disabled
	Omitted	Rogue Master 0 function enabled
5 to 6	Installed	Rogue Master 1 function disabled
	Omitted	Rogue Master 1 function enabled
7 to 8	None	**These pins are RESERVED, NO jumper shunt should be installed.**

Table 6-1 - VMIPCI-5565 Jumper E7 Settings

6.2.2 Installation into the Acutrol3000 Chassis

The installation into the Acutrol3000 Chassis is documented in Acutronic drawing 1202E51A. The VMIC interface is supplied as a modification kit to the base chassis. Three components are supplied:

- **VMIPCI-5565 Card.** This card is installed into an empty PCI slot in the Acutrol3000 backplane and secured with a mounting screw. See Figure 6-2
- **Rear Panel.** A modified rear panel is supplied for customer access to the fiber optic connections. The panel is mounted in place of the existing blank panel. See Figure 6-3.
- **Interconnecting cables.** Fiber Optic cables are supplied to go from the VMIPCI-5565 card to the bulkhead connections.

To install the VMIC Interface:

1. Turn off and unplug the Acutrol.
2. Unscrew the front cover and fold out of the way.
3. Remove the top cover, being careful to disconnect the fan power cable.
4. Touch a metal part of the Acutrol chassis to discharge any static electricity that might be on your clothes or body.
5. Insert the VMIC card into the leftmost PCI slot. It can be a tight fit, be careful not to force the device into place.
6. Screw the mounting bracket of the VMIC card to the back panel rail of the Acutrol.
7. Remove the side bolts of the upper rail of the rear panel. Remove the upper rail. Replace the blank rear panel with the VMIC Interface panel. Install the upper rail and the side bolts.
8. Install the fiber optic cables from the VMIC card to the rear panel.
9. Visually verify the installation is correct.
10. Replace the top cover, being careful to connect the fan power supply. Fold the front cover back into position and secure with the three retaining screws.
11. Plug in and turn on the Acutrol.

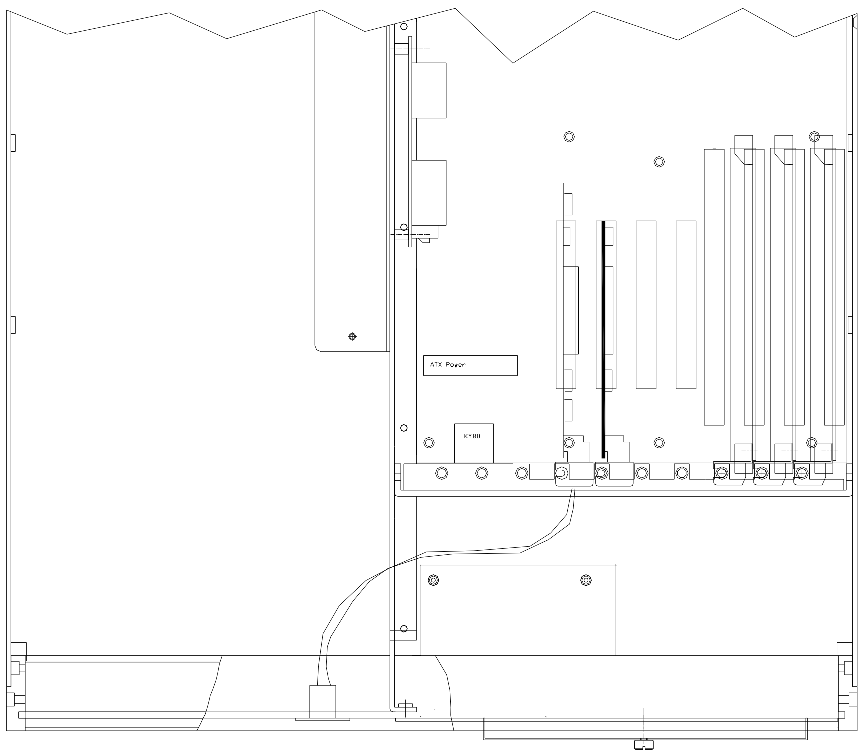


Figure 6-2 VMIC Card Inserted into Acutrol3000 Chassis

6.2.3 Enabling the VMIC Interface

Once the card has been installed into the chassis, the system software must be configured to acknowledge the board.

- Edit the system.xml file on the Real Time Computer and change the **hasVMIC** field to “TRUE” from “FALSE”
- Edit the BIOS to configure PCI Slot 4 for Interrupt 12.
- Reboot the Acutrol

On startup the Acutrol will now look for the presence of the VMIC board and report an error if either the hardware cannot be found or if the driver cannot be initialized.

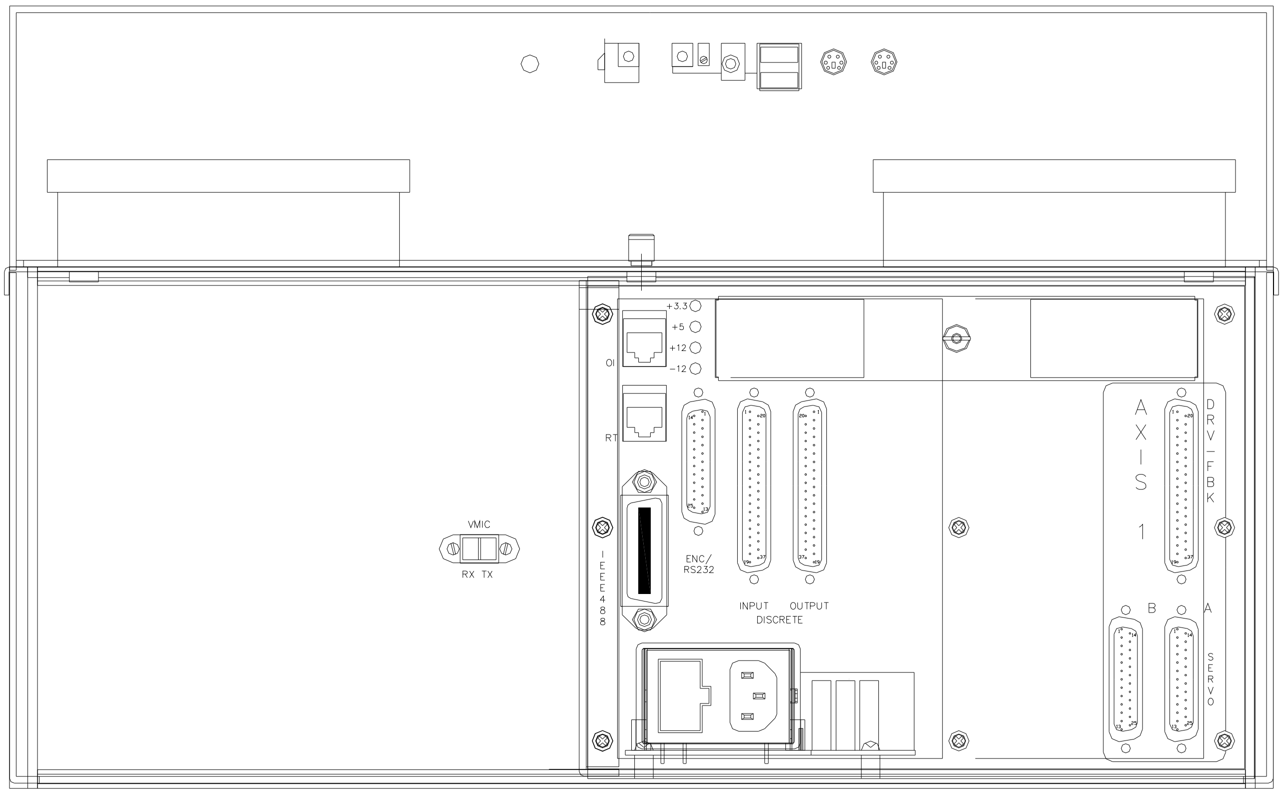


Figure 6-3 Acutrol3000 Rear Panel Showing VMIC Interface

6.3 VMIC Configuration

The only configurable option for the VMIC interface is the Host Computer Interrupts. See Section 3.3.2.3. All other configuration of the VMIC interface is completed by the Acutrol3000 Real Time Software.

6.4 VMIC Features

The VMIC Interface has all of the features of the Acutrol3000 Real Time Interface:

- Host Time Tracker
- Acutrol Time Tracker

- Host Demand Tracker
- Host Demand Translator
- Monitor Translator
- ACL Interface Block
- IEEE 754.1 Single and Double Precision Float and Binary Data Types

6.5 VMIC Usage

Due to the simplicity of the VMIC interface, there are no user configurable registers to program. The only initialization required is for the Acutrol.

6.5.1 Initializing Acutrol

The following pseudo-code demonstrates initialization of the Acutrol.

Command	**Description**
`unsigned long Write(unsigned long addr, unsigned long 0x00000000);`	Set every 32-bit word in the Demand Buffer to 0x00000000. This sets the Position Demand, Rate Demand, and Acceleration Demand to zero for each axis.
`unsigned long Write(unsigned long addr, unsigned long 0x00000000);`	Set Position Mode for Axis 1, Axis 2, and Axis 3. Open the Interlocks and set the system to Local Mode
`unsigned long Write(unsigned long addr, unsigned long 0x80000000);`	Set Demand SYNC ID to 0x80000000.
`VMIC Interrupt(Acutrol_Node, Interrupt 1, 0)`	Send Network Interrupt 1 to Acutrol Node ID to transfer demand data. Send a 0 in the data field.

7 Acutrol3000 Parallel Interface

7.1 Parallel Interface Hardware

The Parallel Interface uses a National Instruments PCI-6533 Digital I/O Card. The National Instruments PCI-6533 (DIO-32HS) is a high-speed, 32-bit, parallel digital I/O interface for PCI. The NI PCI-6533 incorporates the NI DAQ-DIO ASIC, a 32-bit general-purpose digital I/O interface specifically designed to deliver high performance on plug-in digital I/O boards. The 6533 performs single-point I/O, pattern I/O, and high-speed data transfer using a wide range of handshaking protocols at speeds up to 76 Mbytes/s. You can operate the 32 lines as individually configurable single-line I/O, or as 8, 16, or 32-bit ports for pattern I/O and handshaking.

7.2 Parallel Interface Installation

All configuration of the PCI-6533 interface is done through software. Therefore, installation of the Acutrol3000 Parallel Interface consists of only two steps:

- Installation into the Acutrol3000
- Enabling the Parallel interface in the system.xml file

7.2.1 Installation into the Acutrol3000 Chassis

The installation into the Acutrol3000 Chassis is documented in Acutronic drawing 1202E52A. The SCRAMNet+ interface is supplied as a modification kit to the base chassis. Three components are supplied:

- o **PCI-6533 Card.** This card is installed into an empty PCI slot in the Acutrol3000 backplane and secured with a mounting screw. See Figure 7-1.
- o **Rear Panel.** A modified rear panel is supplied for customer access to the fiber optic connections. The panel is mounted in place of the existing blank panel. See Figure 7-2.
- o **Interconnecting cables.** Ribbon cables are supplied to go from the PCI-6533 card to the bulkhead connector on the rear panel.

To install the Parallel Interface:

1. Turn off and unplug the Acutrol.
2. Unscrew the front cover and fold out of the way.
3. Remove the top cover, being careful to disconnect the fan power cable.
4. Touch a metal part of the Acutrol chassis to discharge any static electricity that might be on your clothes or body.
5. Insert the PCI-6533 card into the leftmost PCI slot. It can be a tight fit, be careful not to force the device into place.
6. Screw the mounting bracket of the PCI-6533 card to the back panel rail of the Acutrol.
7. Remove the side bolts of the upper rail of the rear panel. Remove the upper rail. Replace the blank rear panel with the Parallel Interface panel. Install the upper rail and the side bolts.
8. Install the 68-to-50 pin adapter into the connector on the PCI-6533. Install the ribbon cable from the PCI-6533 adapter to the rear panel. Fold the ribbon cable as shown.
9. Visually verify the installation is correct.
10. Replace the top cover, being careful to connect the fan power supply. Fold the front cover back into position and secure with the three retaining screws.
11. Plug in and turn on the Acutrol.

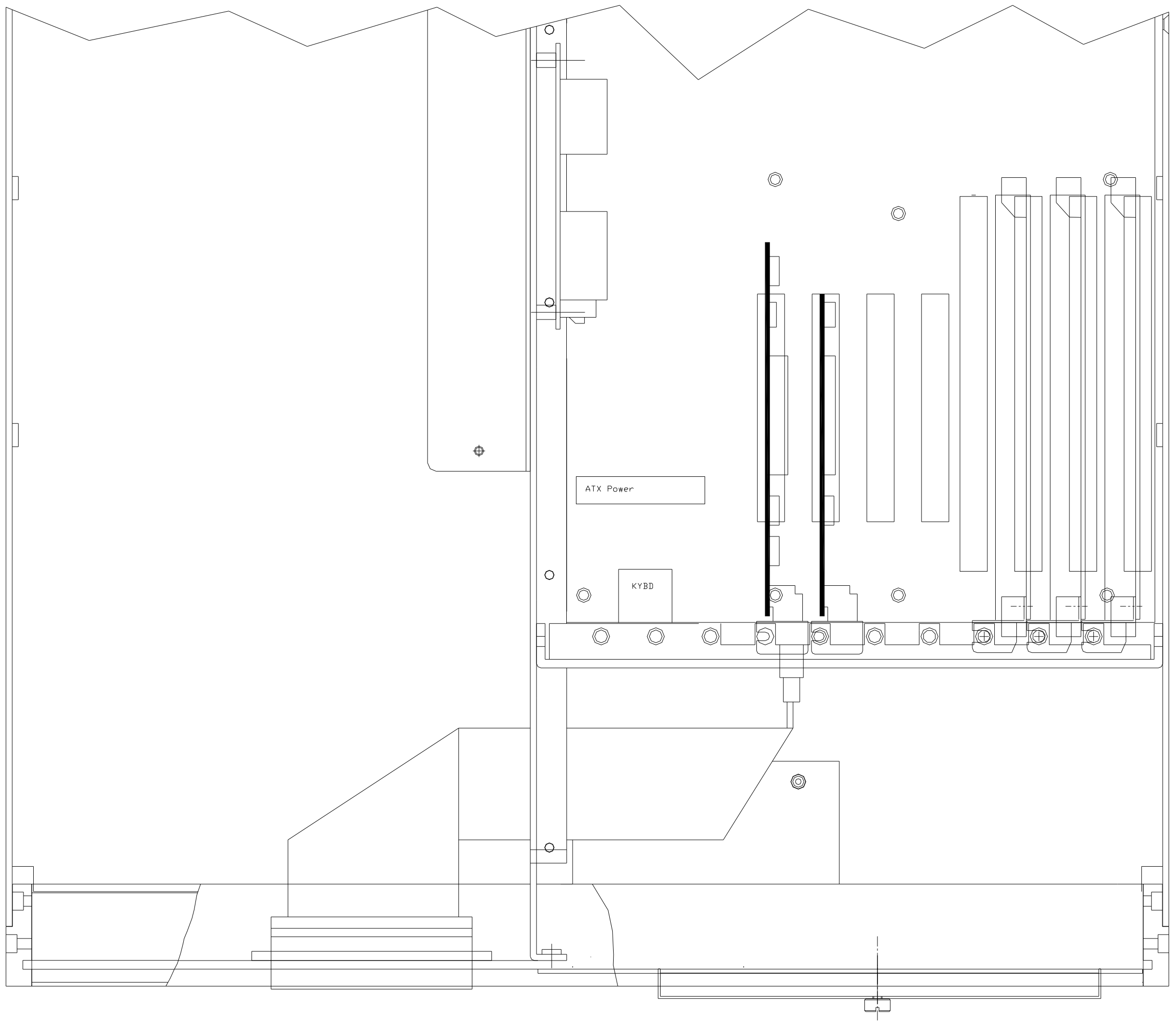


Figure 7-1 PCI-6533 Card Inserted into Acutrol3000 Chassis

7.2.2 Enabling the Parallel Interface

Once the card has been installed into the chassis, the system software must be configured to acknowledge the board.

- Edit the system.xml file on the Real Time Computer and change the **hasParallel** field to “TRUE” from “FALSE”
- Edit the BIOS to configure PCI Slot 4 for Interrupt 12.
- Reboot the Acutrol

On startup the Acutrol will now look for the presence of the PCI-6533 board and report an error if either the hardware cannot be found or if the driver cannot be initialized.

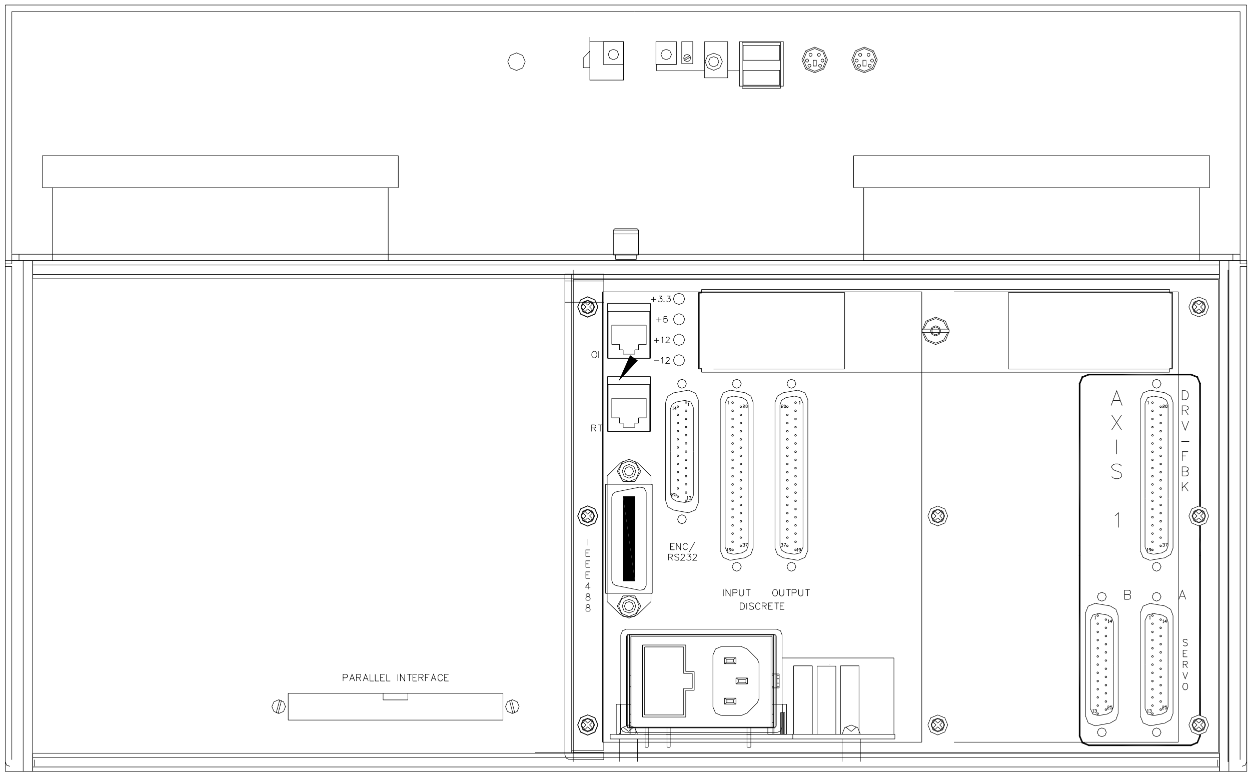


Figure 7-2 Acutrol3000 Rear Panel Showing Parallel Interface

7.3 Parallel Interface Configuration

There are seven parameters that must be configured for the Parallel Interface. These parameters are configured with the **`<PARALLEL_CONFIG>`** element in the system.xml file. All other parameters are configured by the Acutrol3000 Real Time Software.

`syncID` Use this to set the SYNC ID for the parallel interface.

`exchangePins`. Use this parameter to swap the handshake lines.

The **`readDelay`** and **`writeDelay`** parameter keys affect the read and write data settling time. Longer cables or special handshaking considerations may require additional data settling times. Enter an integer value from 0–7. These parameter fields are scaled in hundreds of nanoseconds. 0 corresponds to no data settling time, 1 corresponds to 100 ns, and 7 to 700 ns of data settling time.

The **`activeLevel`** parameter controls the active level of the parallel interface handshake signals. These are the REQ1, ACK1, REQ2, and ACK2 signals as defined by National Instruments for the NI-6533 board.

For compatibility, there is a one-to-one correspondence between the bits of the **`activeLevel`** parameter and the HI/LO parameter of the Acutrol 2000 (See *TM-9216 16-Bit Parallel Interface for the Acutrol® Measurement and Control System.*). This is shown in Table 2. When an NI-6533 register bit is 0, the corresponding handshake signal will be a logic high when asserted. When the bit is 1, the handshake signal will be a logic low when asserted.

Every allowed bit combination of the **`activeLevel`** parameter is shown in Table 3. This table also defines the decimal values that must be entered to obtain a particular configuration. The default value for the **`activeLevel`** parameter is highlighted in the table.

Either Table 2 or Table 3 may be used to determine the proper value to command for the **`activeLevel`** parameter.

Table 2: Acutrol 2000 HI/LO Parameter Compatibility

	Monitor (Group 2)		Demand (Group 1)	
`activeLevel` Parameter	Bit 3	Bit 2	Bit 1	Bit 0
Acutrol 2000 HI/LO parameter	INVRQ2	INVACK2	INVRQ1	INVACK1

Table 3: **activeLevel** Parameter Values

activeLevel Parameter		Handshake Active Level[5]			
Decimal	Binary 3 2 1 0	Monitor REQ2	Monitor ACK2	Demand REQ1	Demand ACK1
0	0 0 0 0	High	High	High	High
1	0 0 0 1	High	High	High	Low
2	0 0 1 0	High	High	Low	High
3	0 0 1 1	High	High	Low	Low
4	0 1 0 0	High	Low	High	High
5	0 1 0 1	High	Low	High	Low
6	0 1 1 0	High	Low	Low	High
7	0 1 1 1	High	Low	Low	Low
8	1 0 0 0	Low	High	High	High
9	1 0 0 1	Low	High	High	Low
10	1 0 1 0	Low	High	Low	High
11	1 0 1 1	Low	High	Low	Low
12	1 1 0 0	Low	Low	High	High
13	1 1 0 1	Low	Low	High	Low
14	1 1 1 0	Low	Low	Low	High
15	**1 1 1 1**	**Low**	**Low**	**Low**	**Low**

The **handshakeMode** parameter controls the handshaking mode of the parallel interface handshake signals. Three handshaking modes are available: level, leading-edge, and trailing-edge.

For compatibility with the Acutrol 20000, there is a one-to-one correspondence between the bits of the Acutrol 2000 LEV/PUL parameter and the **handshakeMode** parameter. This is shown in Table 4. When a **handshakeMode** PULSEx register bit is a 0, then the corresponding group will use level mode handshaking. When the PULSEx register bit is a 1 and the EDGEx bit is a 0, then the group will use leading-edge handshaking. The final combination is PULSEx and EDGEx both set to 1 which causes trailing-edge handshaking.

Every allowed bit combination of the **handshakeMode** parameter is shown in Table 5. This table also defines the decimal values that must be entered to obtain a particular configuration. The default value for the **handshakeMode** parameter is highlighted in the table.

[5] The text “Low” refers to a signal that is asserted (true) when it is a logical low (approximately 0 V). The text “High” refers to a signal that is asserted (true) when it is a logical high (approximately 5 V).

Either Table 4 or Table 5 may be used to determine the proper value to command for the **handshakeMode** parameter.

Table 4: Acutrol 2000 LEV/PUL Parameter Compatibility

	Monitor (Group 2)		Demand (Group 1)	
LEV/PUL Parameter	Bit 3	Bit 2	Bit 1	Bit 0
handshakeMode Parameter	PULSE2	EDGE2	PULSE1	EDGE1

Table 5: **handshakeMode** Parameter Values

handshakeMode Parameter		Handshake Mode[6]	
Decimal	Binary 3 2 1 0	Monitor	Demand
0	0 0 0 0	Level	Level
1	0 0 0 1	Level	Level
2	0 0 1 0	Level	Leading
3	0 0 1 1	Level	Trailing
4	0 1 0 0	Level	Level
5	0 1 0 1	Level	Level
6	0 1 1 0	Level	Leading
7	0 1 1 1	Level	Trailing
8	1 0 0 0	Leading	Level
9	1 0 0 1	Leading	Level
10	1 0 1 0	Leading	Leading
11	1 0 1 1	Leading	Trailing
12	1 1 0 0	Trailing	Level
13	1 1 0 1	Trailing	Level
14	**1 1 1 0**	**Trailing**	**Leading**
15	1 1 1 1	Trailing	Trailing

[6] The text “Level” refers to the level handshaking mode. The text “Leading” refers to the leading edge-pulse handshaking mode. The text “Trailing” refers to the trailing-edge pulse handshaking mode.

The **`dataSettlingMode`** parameter controls the data settling mode when leading-edge handshaking is used. Two modes may be set, delay before or after the pulse leading-edge. The **`readDelay`** and **`writeDelay`** parameters determine the actual delay used.

For compatibility with the Acutrol 2000, there is a one-to-one correspondence between the bits of the Acutrol 2000 LPULSE parameter and the **`dataSettlingMode`** parameter. This is shown in Table 6. When the **`dataSettlingMode`** LPULSEx bit is 0, the data settling delay is added before the pulse leading-edge and the width of the pulse is fixed. When the LPULSEx bit is 1, the data settling delay is added after the leading-edge and the pulse width is lengthened (a long pulse).

Every allowed bit combination of the **`dataSettlingMode`** parameter is shown in Table 7. This table also defines the decimal values that must be entered to obtain a particular configuration. The default value for the **`dataSettlingMode`** parameter is highlighted in the table.

Either Table 6 or Table 7 may be used to determine the proper value to command for the LPULSE parameter.

Table 6: Acutrol 2000 LPULSE Parameter Compatibility

LPULSE Parameter	Monitor (Group 2) Bit 1	Demand (Group 1) Bit 0
`dataSettlingMode` Parameter	LPULSE2	LPULSE1

Table 7: **`dataSettlingMode`** Parameter Values

`dataSettlingMode` Parameter		Pulse Width[7]	
Decimal	Binary 1 0	Monitor	Demand
0	0 0	Fixed	Fixed
1	0 1	Fixed	Long
2	**1 0**	**Long**	**Fixed**
3	1 1	Long	Long

7.4 Parallel Interface Features

The SCRAMNet+ Interface is designed for compatibility with the Acutrol2000 and has a limited subset of the features of the Acutrol3000 Real Time Interface:

- Host Time Tracker
- Acutrol Time Tracker
- Host Demand Tracker
- Host Demand Translator

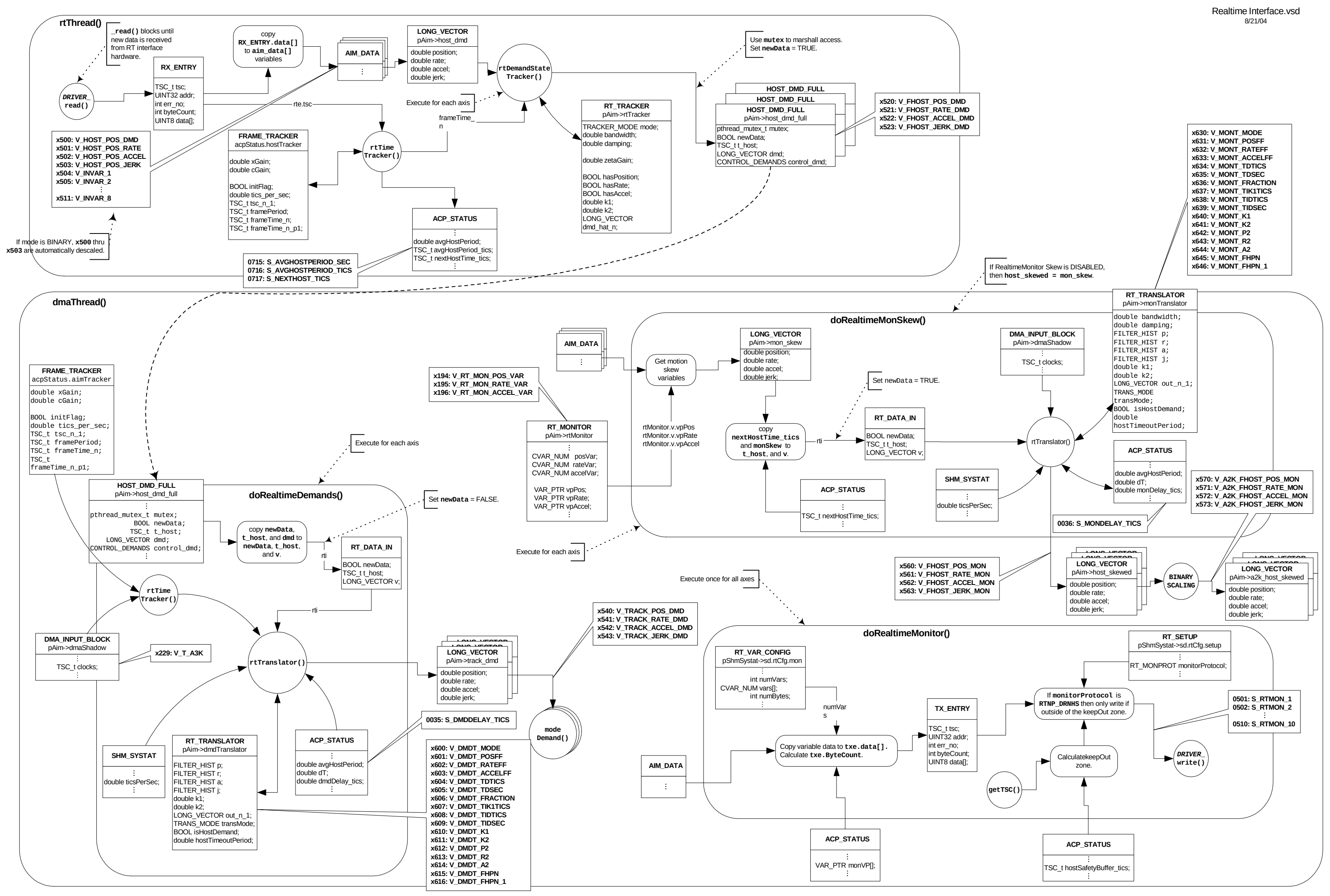
7.5 Parallel Interface Usage

See *TM-9216 16-Bit Parallel Interface for the Acutrol® Measurement and Control System.*

7 The text “Fixed” refers to the data settling delay mode of leading-edge pulse handshaking in which the pulse width is fixed (the delay is added before the leading-edge of the pulse). The text “Long” refers to the data settling delay mode in which the pulse width is lengthened (the delay is added after the leading-edge of the pulse).

ACUTRONIC

Appendix 1 – Acutrol3000 Real Time Data Processing Block Diagram



Realtime Interface.vsd
8/21/04
rtThread()
_read() blocks until new data is received from RT interface hardware.
DRIVER_ read()
RX_ENTRY
TSC_t tsc; UINT32 addr; int err_no; int byteCount; UINT8 data[];
copy RX_ENTRY.data[] to aim_data[] variables
AIM_DATA
LONG_VECTOR pAim->host_dmd
double position; double rate; double accel; double jerk;
rtDemandState Tracker()
Use mutex to marshall access. Set newData = TRUE.
HOST_DMD_FULL
HOST_DMD_FULL pAim->host_dmd_full
pthread_mutex_t mutex; BOOL newData; TSC_t t_host; LONG_VECTOR dmd; CONTROL_DEMANDS control_dmd;
x520: V_FHOST_POS_DMD
x521: V_FHOST_RATE_DMD
x522: V_FHOST_ACCEL_DMD
x523: V_FHOST_JERK_DMD
x500: V_HOST_POS_DMD
x501: V_HOST_POS_RATE
x502: V_HOST_POS_ACCEL
x503: V_HOST_POS_JERK
x504: V_INVAR_1
x505: V_INVAR_2
x511: V_INVAR_8
rte.tsc
FRAME_TRACKER acpStatus.hostTracker
double xGain; double cGain; BOOL initFlag; double tics_per_sec; TSC_t tsc_n_1; TSC_t framePeriod; TSC_t frameTime_n; TSC_t frameTime_n_p1;
rtTime Tracker()
Execute for each axis
frameTime_n
RT_TRACKER pAim->rtTracker
TRACKER_MODE mode; double bandwidth; double damping; double zetaGain; BOOL hasPosition; BOOL hasRate; BOOL hasAccel; double k1; double k2; LONG_VECTOR dmd_hat_n;
ACP_STATUS
double avgHostPeriod; TSC_t avgHostPeriod_tics; TSC_t nextHostTime_tics;
If mode is BINARY, x500 thru x503 are automatically descaled.
0715: S_AVGHOSTPERIOD_SEC
0716: S_AVGHOSTPERIOD_TICS
0717: S_NEXTHOST_TICS
If RealtimeMonitor Skew is DISABLED, then host_skewed = mon_skew.
x630: V_MONT_MODE
x631: V_MONT_POSFF
x632: V_MONT_RATEFF
x633: V_MONT_ACCELFF
x634: V_MONT_TDTICS
x635: V_MONT_TDSEC
x636: V_MONT_FRACTION
x637: V_MONT_TIK1TICS
x638: V_MONT_TIDTICS
x639: V_MONT_TIDSEC
x640: V_MONT_K1
x641: V_MONT_K2
x642: V_MONT_P2
x643: V_MONT_R2
x644: V_MONT_A2
x645: V_MONT_FHPN
x646: V_MONT_FHPN_1
dmaThread()
doRealtimeMonSkew()
AIM_DATA
Get motion skew variables
LONG_VECTOR pAim->mon_skew
double position; double rate; double accel; double jerk;
DMA_INPUT_BLOCK pAim->dmaShadow
TSC_t clocks;
RT_TRANSLATOR pAim->monTranslator
double bandwidth; double damping; FILTER_HIST p; FILTER_HIST r; FILTER_HIST a; FILTER_HIST j; double k1; double k2; LONG_VECTOR out_n_1; TRANS_MODE transMode; BOOL isHostDemand; double hostTimeoutPeriod;
x194: V_RT_MON_POS_VAR
x195: V_RT_MON_RATE_VAR
x196: V_RT_MON_ACCEL_VAR
RT_MONITOR pAim->rtMonitor
CVAR_NUM posVar; CVAR_NUM rateVar; CVAR_NUM accelVar; VAR_PTR vpPos; VAR_PTR vpRate; VAR_PTR vpAccel;
rtMonitor.v.vpPos
rtMonitor.v.vpRate
rtMonitor.v.vpAccel
Set newData = TRUE.
copy nextHostTime_tics and monSkew to t_host, and v.
rti
RT_DATA_IN
BOOL newData; TSC_t t_host; LONG_VECTOR v;
rtTranslator()
ACP_STATUS
double avgHostPeriod; double dT; double monDelay_tics;
SHM_SYSTAT
double ticsPerSec;
ACP_STATUS
TSC_t nextHostTime_tics;
0036: S_MONDELAY_TICS
x570: V_A2K_FHOST_POS_MON
x571: V_A2K_FHOST_RATE_MON
x572: V_A2K_FHOST_ACCEL_MON
x573: V_A2K_FHOST_JERK_MON
FRAME_TRACKER acpStatus.aimTracker
double xGain; double cGain; BOOL initFlag; double tics_per_sec; TSC_t tsc_n_1; TSC_t framePeriod; TSC_t frameTime_n; TSC_t frameTime_n_p1;
Execute for each axis
HOST_DMD_FULL pAim->host_dmd_full
pthread_mutex_t mutex; BOOL newData; TSC_t t_host; LONG_VECTOR dmd; CONTROL_DEMANDS control_dmd;
doRealtimeDemands()
copy newData, t_host, and dmd to newData, t_host, and v.
Set newData = FALSE.
RT_DATA_IN
BOOL newData; TSC_t t_host; LONG_VECTOR v;
Execute for each axis
x560: V_FHOST_POS_MON
x561: V_FHOST_RATE_MON
x562: V_FHOST_ACCEL_MON
x563: V_FHOST_JERK_MON
LONG_VECTOR pAim->host_skewed
double position; double rate; double accel; double jerk;
BINARY SCALING
LONG_VECTOR pAim->a2k_host_skewed
double position; double rate; double accel; double jerk;
Execute once for all axes
rtTime Tracker()
DMA_INPUT_BLOCK pAim->dmaShadow
TSC_t clocks;
x229: V_T_A3K
rtTranslator()
x540: V_TRACK_POS_DMD
x541: V_TRACK_RATE_DMD
x542: V_TRACK_ACCEL_DMD
x543: V_TRACK_JERK_DMD
LONG_VECTOR pAim->track_dmd
double position; double rate; double accel; double jerk;
0035: S_DMDDELAY_TICS
mode Demand()
SHM_SYSTAT
double ticsPerSec;
RT_TRANSLATOR pAim->dmdTranslator
FILTER_HIST p; FILTER_HIST r; FILTER_HIST a; FILTER_HIST j; double k1; double k2; LONG_VECTOR out_n_1; TRANS_MODE transMode; BOOL isHostDemand; double hostTimeoutPeriod;
ACP_STATUS
double avgHostPeriod; double dT; double dmdDelay_tics;
x600: V_DMDT_MODE
x601: V_DMDT_POSFF
x602: V_DMDT_RATEFF
x603: V_DMDT_ACCELFF
x604: V_DMDT_TDTICS
x605: V_DMDT_TDSEC
x606: V_DMDT_FRACTION
x607: V_DMDT_TIK1TICS
x608: V_DMDT_TIDTICS
x609: V_DMDT_TIDSEC
x610: V_DMDT_K1
x611: V_DMDT_K2
x612: V_DMDT_P2
x613: V_DMDT_R2
x614: V_DMDT_A2
x615: V_DMDT_FHPN
x616: V_DMDT_FHPN_1
doRealtimeMonitor()
RT_VAR_CONFIG pShmSystat->sd.rtCfg.mon
int numVars; CVAR_NUM vars[]; int numBytes;
numVars
RT_SETUP pShmSystat->sd.rtCfg.setup
RT_MONPROT monitorProtocol;
If monitorProtocol is RTNP_DRNHS then only write if outside of the keepOut zone.
TX_ENTRY
TSC_t tsc; UINT32 addr; int err_no; int byteCount; UINT8 data[];
Copy variable data to txe.data[]. Calculate txe.ByteCount.
0501: S_RTMON_1
0502: S_RTMON_2
0510: S_RTMON_10
DRIVER_ write()
CalculatekeepOut zone.
getTSC()
AIM_DATA
ACP_STATUS
VAR_PTR monVP[];
ACP_STATUS
TSC_t hostSafetyBuffer_tics;