RUG3 TECHNICAL MANUAL

Version 2.05A

Rugid Computer
6305 Elizan Dr. NW
Olympia, WA 98502
360-866-4492, Fax 360-866-8074
Web site: **www.rugidcomputer.com**
email: **support@rugidcomputer.com**

TABLE OF CONTENTS

Table of Figures

List of Tables

CHAPTER 1…INTRODUCTION

Welcome to the latest in RUGID's series of small remote terminal units (RTU's). The RUG3 unit is the latest in a long line of RTUs designed for remote data acquisition and control applications. It incorporates advanced hardware and software techniques so you can implement your application in the minimum time. Numerous hardware and software safeguards are incorporated into the design so you can be assured that the unit will continue to operate for many years in the most demanding field environment.

This document will help get your RUG3 unit up and running within minutes of unpacking it. If you are new to the RUG3, we encourage you to read the "Getting Started" chapter that follows so you will understand how to use the support software and sample applications.

Minimum Equipment Required

In order to use the RUG3 you will need to run RUGID's R3SETUP.EXE program on a PC-compatible computer running Windows 95 or later, or Windows NT, and having at least a 600 X 800 SVGA screen. The R3SETUP.EXE program is available for download free of charge from our web site address below. For program loading, you will also need a serial cable with a DB9 female connector on one end and a 3.5 mm plug connector on the other. This is available from us, part number R3CBL232.

Factory Support

If you have any trouble with your unit or have questions regarding anything in this manual, feel free to call or write us at the addresses and phone numbers below:

Mailing address:	RUGID Computer 6305 Elizan Dr. NW Olympia, WA 98502
Phone number (8AM to 5 PM PST):	(360) 866-4492
Fax number (24Hrs/day):	(360) 866-8074
Web site:	www.rugidcomputer.com
Email:	support@rugidcomputer.com

Introduction

CHAPTER 2…GETTING STARTED

Introduction

This chapter presents the basics of getting the RUG3 running. It includes installing the support software, powering up the RUG3, loading configuration files to the RUG3, and implementing a simple application to give you a feel for how the support software works. The section on implementing a simple example application is extremely detailed to the extent that virtually every mouse move, click and resulting screen are shown. This is to make sure that we have left nothing out that may lead to confusion later. If you are already familiar with the RUG3, you may want to skip this chapter.

Installing RUG3 Support Software

In order to load configuration files and the RUG3 operating system, and to modify the RUG3 operating system, you will need the R3SETUP.EXE program. This program is included on request at no extra charge on CD with each RUG3 unit. It is also available for download at no charge from out web site. Follow this procedure to install the program:

1) Make sure you have Windows 9X/NT/2000/XP/ME/Vista (herein after referred to as Windows) running on a PC compatible computer with a screen having at least 600 X 800 pixel resolution.
2) Insert the R3SETUP disk into your 3.5 inch floppy drive or CDROM drive, whichever applies; or download the program from our web site: www.rugidcomputer.com/Downloads. Double click on the SetupXXX.exe program, where XXX is the revision number such as 205. The program should self install into the directory C:\Programs\RUG3.
3) To start the program, go to START, PROGRAMS, RUG3 and double click the R3 icon.

The R3SETUP program and its various utilities will self install and leave an icon in your main “Programs” list. During installation, you will be asked to confirm the destination of the files, or to indicate an alternate destination. Feel free to alter the destinations, but be aware that this document assumes that the locations are left at the defaults.

Applying Power to the RUG3

RUG3 units can be powered from 120 VAC using a standard 12 VDC wall transformer. Simply wire the wall transformer's output wires (clip off any connector as necessary) as shown in Figure 1. An 800 ma transformer should be sufficient for most applications. Note that if the RUG3's measured battery voltage falls below 11.0 VDC, internal logic will turn off the loop supply and will not allow it to be turned on until measured battery voltage rises above 12.0 VDC. This is to protect against the unit drawing excessive current in an attempt to maintain full power to the loop supply loads. Total RUG3 power draw, in milliamps, can be estimated from the measurements presented in the following table, or from this equation, assuming a 12VDC power source:

Pwr in ma=
2.28+0.68*(if have LCD)
+19.9*(if LCD backlight ON)
+2.93*(if modem transmitting)
+17.5*(if loop supply ON)
+57.0*(number of 4-20 ma loops ON)
+39.0*(number of relays ON).

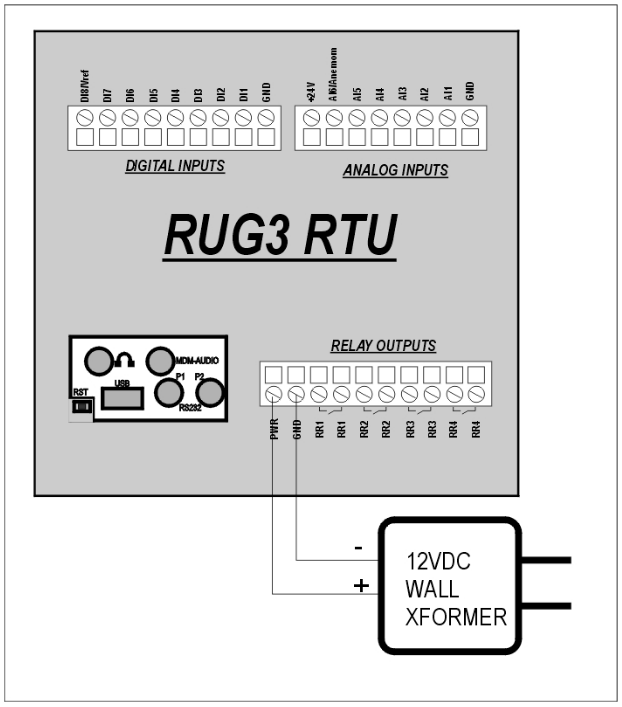


Figure 1 Powering the RUG3 from 120VAC

For board revisions 11 and later, you can also power the unit from the USB port with these restrictions: the 10 amp relays will not pull in and the loop supply will not work (they need external 12 VDC). Relay outputs on units with solid state relays (SSR) will work off USB.

For board revisions 10 and earlier, if you use the USB port, be sure to power the unit from 12 VDC before plugging in the USB cable; and remove the USB cable before removing 12 VDC.

The board revision is noted on the unit's label.

Table 1 RUG3 Current Requirements from 13.5 VDC supply

ITEM	CURRENT DRAW, ma.	NOTES:
Unit unpowered	0.002 internal lithium battery	Timekeeping mode
Unit powered by 13.5 VDC	2.28	Full operation, no loads on
LCD, no backlight	0.68	
LCD backlight	22.2	
RS232 port	4.76	Only when cable connected
Poll using internal modem	2.93	
Loop supply, no load	17.5	
Loop supply, 120 ma load	360.0	
Relay, 10 amp	39.0	Current draw per relay
Solid state relay	3.0	Current draw per SSR
Max draw, everything on	538.0	LCD, loop supply, all relays on

Preparing to Communicate with the RUG3

The figure below illustrates how you connect a PC to the RUG3. With either the illustrated USB or serial port connection, you can load configuration files, load the RUG3's operating system, and send screens of information from the RUG3 to the PC for observation by an operator. In either case, you must supply power to the RUG3 as illustrated above. Note that when the USB cable is installed, the RS232 port #1 is disabled. You cannot use both simultaneously.

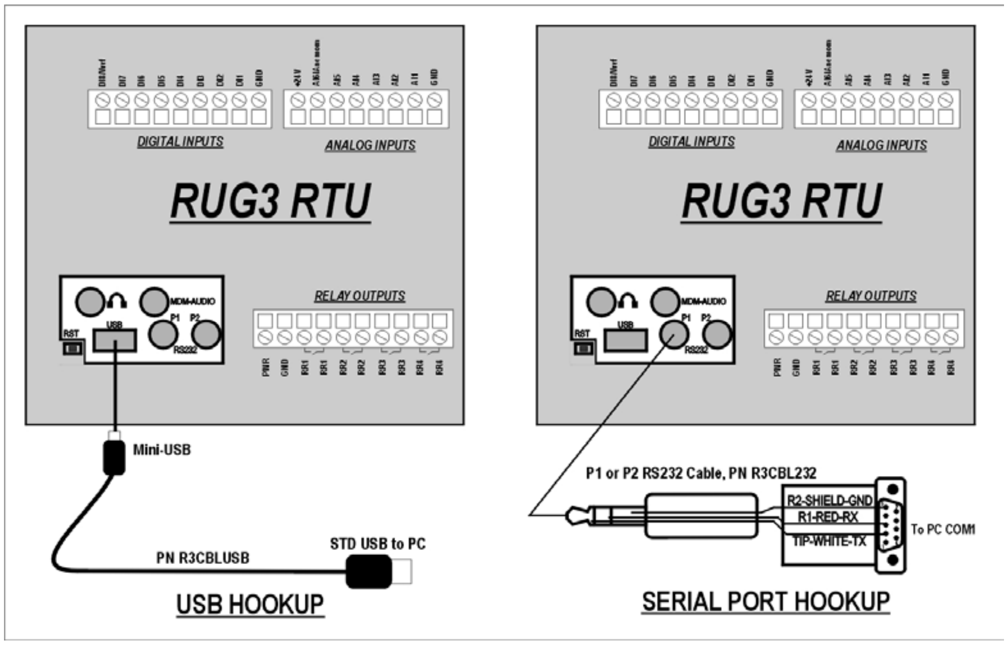


Figure 2 PC Serial Hookup to RUG3

RUG3 BASIC OPERATION

When you power up the RUG3, it will perform various tests to determine operating system and memory integrity. If it determines that everything is OK, it will return to the same tasks it was doing before it was last powered down. Normally, this will mean that it will begin running the user's program. If it finds an error in the operating system, it will issue the message "Waiting for OS load…" using the RS232 programming port. This should rarely happen, but if it does, you must reload the operating system using the R3SETUP program. If it finds an error in the user configuration file (user program), or if no file is present, then the unit will present the following display on the unit's RS232 port:

```
*** SYSTEM MENU ***
Version 205
No program

5 Set clock/calendar
7 Adjust LCD contrast, now =130
8 Raw I/O data
Choose option…
```

If this is the case, then a new configuration file must be loaded from the R3SETUP program.

Getting Started

If the program is found to be intact, the program will start running and the unit will present the following menu on the serial port:

```
*** SYSTEM MENU ***

Version 205
Program: R3Burnin.rgd  03/31/04 12:13:16 PM

1 Display Menu
2 Setpoints Menu
3 Log on
4 Log off
5 Set clock/calendar
6 Future
7 Adjust LCD contrast, now =130
8 Raw I/O data

Choose option…
```

This menu header indicates the installed operating system's revision (2.05) and the user configuration file name and date of installation. Here are the options from the above screen. They will be discussed in detail later.

1) Display menu…presents list of displays for this port from which you can select.
2) Setpoints menu…presents list of setpoints that you can observe and alter
3) Log on…enables you to log on for setpoint alteration
4) Log off…disables ability to alter setpoints
5) Set clock/calendar…enters a prompted system for setting the realtime clock/calendar. Simply enter those items that need to be changed, or hit [ENTER] to skip to the next item of clock setting.
6) Future
7) Adjust LCD contrast. You will be shown the present LCD contrast value (0 to 255). Enter a new value to adjust LCD contrast. Low numbers give light contrast; high numbers give darker contrast. A value of 130 usually gives good contrast.
8) Raw I/O data…causes unit to show raw I/O data (DI's, relay states and analog input raw counts).

Configuring the RUG3

Once you have applied power to the RUG3 and the serial cable is hooked up, you can load a project's configuration file into the unit and observe its operation. Before you can do that though, you must design the project and generate its configuration file. Notice that we refer to a project's configuration file, rather its program. Unlike earlier RUGID units that were programmed in BASIC, the RUG3 operating system contains building blocks, that we refer to as "modules", that are already programmed, debugged and compiled in the C language and are part of the RUG3's operating system. Our job as programmers now becomes one of connecting together the modules (i.e., designing the project) to do the task we wish the RUG3 to perform. When we tell the R3SETUP program to compile our project, it produces a configuration file for loading into the RUG3.

The RUG3 Menu System

When the RUG3 starts running its program it will attempt to present the same display that was being shown at the end of the prior run. If it can't, it will present the following menu, which is referred to as the system menu, and is the top level menu in a running RUG3. This menu also is presented whenever you press the minus [-] key on the keyboard.

System menu for serial ports:

```
*** SYSTEM MENU ***

Version 205
Program: R3Burnin.rgd  03/31/04 12:13:16 PM

1 Display Menu
2 Setpoints Menu
3 Log on
4 Log off
5 Set clock/calendar
6 Future
7 Adjust LCD contrast, now =100
8 Raw I/O data

Choose option…
```

System menu for the RUG3 2 X 16 LCD:

```
1DSP,2SP,3LOGON
5CLK,7LCD
```

If you have already logged on and the logon timer has not timed out, or, if the logon feature is not employed, then option 3 above will not be shown, and you will be allowed to change setpoints if you wish. Notice that the program is running when the above menu is displayed; and that changes you make to setpoints, or clock values will take effect immediately.

Accessing Displays

When you press key [1] from the system menu above, the RUG3 will present a display list of this form from which you may select a display to be presented. If you are using the serial port or USB port, the entire list will be shown; in the case of the 2 X 16 LCD, only two display choices at a time will be shown. The list below is simply an example; the list you see will be different:

```
*** DISPLAY LIST ***

0 Levels
1 Summary display
2 Flow rates
3 Run times
4 Pump status
5 Weather info
6 Pump history

Choose display…
```

The entries in this list come from the **Display Title for Menus** box at the top of each display definition page in your project as defined using R3SETUP. There is a separate list for each port for which you have defined displays. If the list contains more than 10 entries, then additional pages will be available from which to choose a display on the RUG3. To access a particular display, hit the numeric key corresponding to the display you want. To access another page of display titles, hit the [ENTER] key. If you select a particular display, the RUG3 will immediately present the display you selected. Thereafter, each time you hit the [ENTER] key, the RUG3 will present the next display in the list and cycle back to the beginning of the list after the last display. At any time, if you hit the minus [-] key, the RUG3 will return you to the system summary menu. For the RUG3's 2X16 LCD, use the up and down arrow keys to scroll up and down the lines of any display.

Accessing Setpoints Using Serial Ports

Hitting key [2] from the system menu will cause the RUG3 to present the first page of setpoints. The setpoints are arranged alphabetically by R3SETUP before loading into the RUG3. If the setpoint list contains more than 10 entries, then additional pages will be available from which to choose a setpoint. To access a particular setpoint, hit the numeric key corresponding to the setpoint you want. To access another page, hit the [ENTER] key. If you select a particular setpoint, and you have logged on for setpoint access, the RUG3 will immediately present the setpoint you selected and prompt for a new entry. You can then either enter a new value followed by the [ENTER] key, or press the [ENTER] key alone to exit from altering that setpoint. Hitting the MINUS [-] key will return you to the system menu.

Accessing Setpoints Using the 2X16 LCD:

When using the RUG3's keyboard and 2-line LCD, hitting key [2] from the system menu will cause the RUG3 to present the first setpoint. Setpoints are arranged alphabetically by R3SETUP before loading into the RUG3. The prompt for the setpoint will be presented on the top line of the display. The existing setpoint value will then be presented on the second line. To move to the next setpoint in the list, hit the [ENTER] key or the [Down Arrow] key. To move to the previous setpoint, hit the [Up Arrow] key. To change the value of the setpoint that is displayed, hit the [CLR] key to erase the existing value, then key in the new value followed by the [ENTER] key. Hitting the MINUS [-] key will return you to the system menu.

Logging On For Setpoint Access

If the programmer has engaged logon security in the configuration file, hitting key [3] from the system menu will cause the RUG3 to prompt you for your access code. This code is set using the **SysSetup** module in R3SETUP program. It can either be a fixed number, such as 714 in the example below; or it can be taken from a setpoint in which case it can be modified by anyone who knows it and knows what title it has been given. After you enter the correct access code, the RUG3 will allow you to alter setpoints. If you are not logged on, the RUG3 will allow you to examine setpoints but not change them. Once you are logged on, you will remain logged on until the logon timer expires. If you alter a setpoint, the logon timer will be restarted. The logon timer timeout period is set by the programmer in the **SysSetup** module.

Logging Off Setpoint Access

Hitting key [4] from the system menu will cause the RUG3 to disable setpoint access until you log on again.

Setting the Clock Calendar

Hitting key [5] initiates the clock/calendar setting process. The system will prompt you for each element of the clock/calendar (day, month, year, hour, minute, second, day of week). To change each entry, simply enter the new value when prompted followed by the [ENTER] key. To skip an entry without altering that element of the clock/calendar, hit the [ENTER] key without entering a value.

Adjusting LCD Contrast

Hitting key [7] from the system menu enables you to adjust the LCD contrast. When you hit key [7] from the system menu, you will be prompted for a new contrast setting. Full range is 0 to 255; but normal range is usually around 130 depending on LCD temperature. A value of 130 usually gives acceptable contrast.

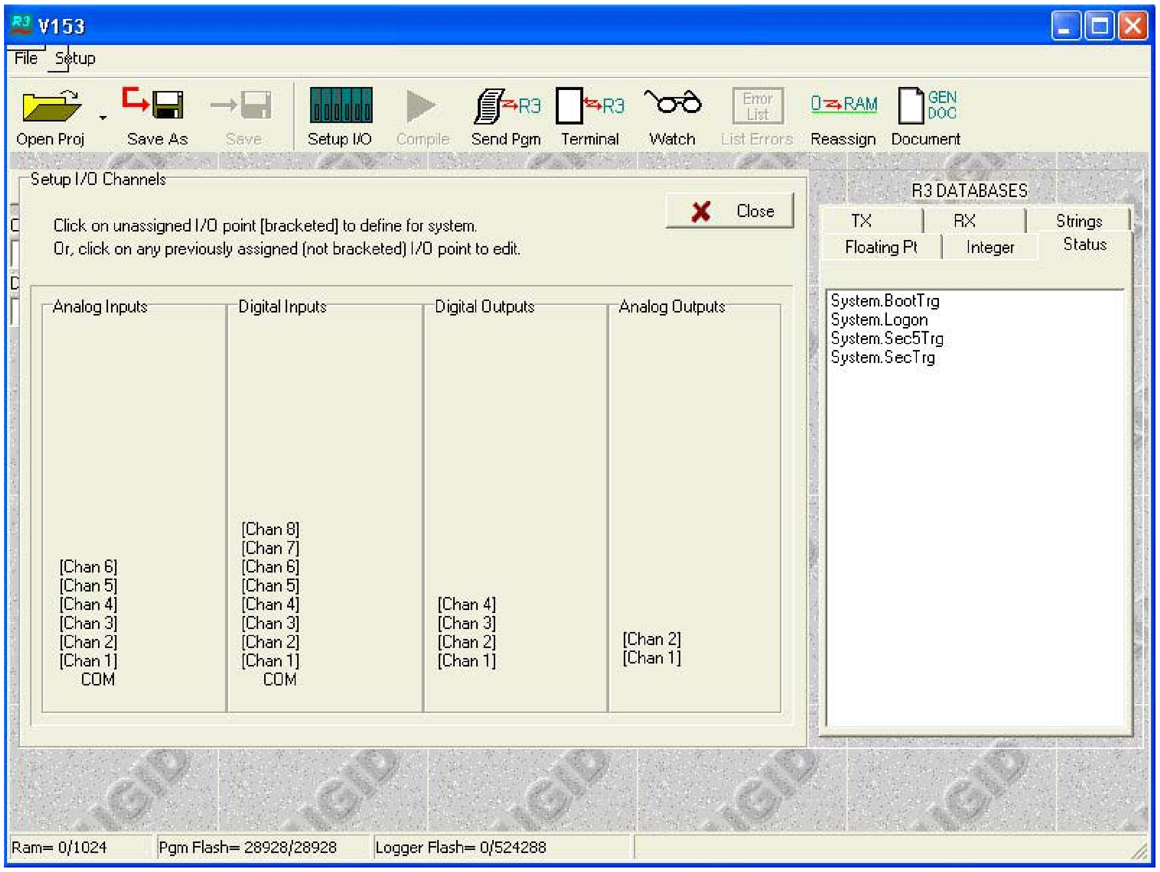
A Simple Application

To illustrate how you configure the RUG3 to do something useful, let's configure a RUG3 to read an analog value that we'll assume is a tank level and generate high and low alarms based on the measured tank level. We'll send the alarms to a pair of relays on the RUG3. We'll also make the RUG3 show the tank level and alarm states on its LCD display.

Specifying RUG3 Hardware

To begin, start the R3SETUP program you installed on your Windows system above. To do this, click on the R3 icon on your desktop:

or click **START…Programs…** and select "R3SETUP". You should see a screen appear as illustrated below. This is the screen where you tell the RUG3 what inputs and outputs you wish to use and what you wish to call each of them. Your screen should look like this:



Setup I/O Channels

Click on unassigned I/O point [bracketed] to define for system.
Or, click on any previously assigned (not bracketed) I/O point to edit.

Analog Inputs	Digital Inputs	Digital Outputs	Analog Outputs
	[Chan 8]		
	[Chan 7]		
[Chan 6]	[Chan 6]		
[Chan 5]	[Chan 5]		
[Chan 4]	[Chan 4]	[Chan 4]	
[Chan 3]	[Chan 3]	[Chan 3]	
[Chan 2]	[Chan 2]	[Chan 2]	[Chan 2]
[Chan 1]	[Chan 1]	[Chan 1]	[Chan 1]
COM	COM		

System.BootTrg
System.Logon
System.Sec5Trg
System.SecTrg

Ram= 0/1024 Pgm Flash= 28928/28928 Logger Flash= 0/524288

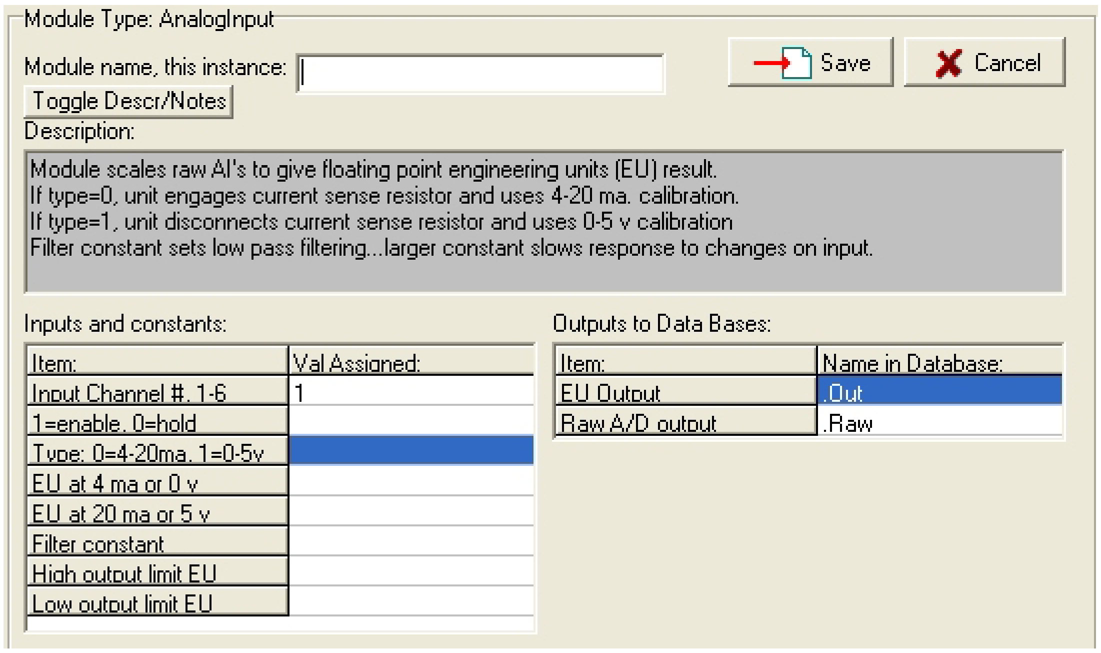
Figure 3 I/O Setup Page

This indicates that the RUG3 has available six analog inputs, eight digital inputs, four relay outputs, and two analog outputs. The default channel names in brackets ('[Chan 1], [Chan 2], etc') indicate that the channels have not yet been named by you.

Specifying I/O Assignments

We must now assign names to the I/O points and set some scale factors and other properties so that we can refer to them as we connect them together. First, let's set up an analog input. Click on the analog input's **Chan 1** field. You should see the following screen appear. It allows you to name the analog input and set parameters for it.

Getting Started



Module name, this instance:

Description:

Module scales raw AI's to give floating point engineering units (EU) result.
If type=0, unit engages current sense resistor and uses 4-20 ma. calibration.
If type=1, unit disconnects current sense resistor and uses 0-5 v calibration
Filter constant sets low pass filtering...larger constant slows response to changes on input.

Inputs and constants:

Item:	Val Assigned:
Input Channel #, 1-6	1
1=enable, 0=hold	
Type: 0=4-20ma, 1=0-5v	
EU at 4 ma or 0 v	
EU at 20 ma or 5 v	
Filter constant	
High output limit EU	
Low output limit EU	

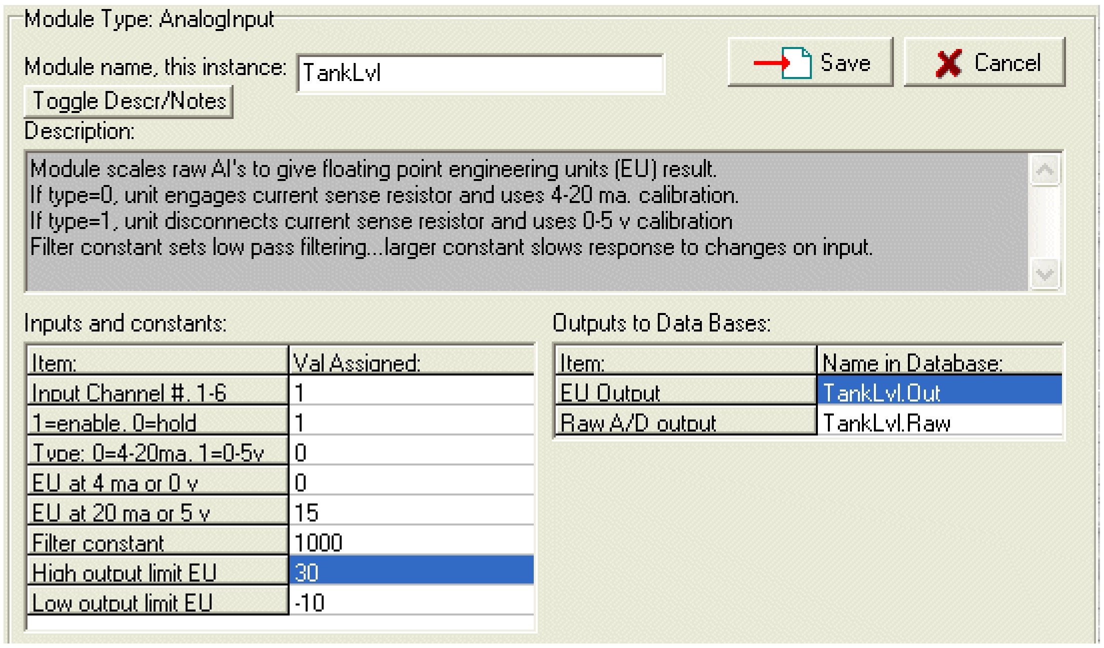
Outputs to Data Bases:

Item:	Name in Database:
EU Output	.Out
Raw A/D output	.Raw

Figure 4 Analog Input Configuration Panel

This is the parameter entry screen for the first analog input channel. This is one of over 100 pre-programmed modules available for your use in the RUG3 system. They will all have more or less the same appearance but with different input and output properties and descriptions. Your cursor should be sitting in the edit box where you enter the name of this channel. If not, click in the box to the right of ‘Module name, this instance’. Using your PC’s keyboard, enter the name “TankLvl”. Notice that as you do so, the cell to the right of **EU output** in the outputs list box will immediately reflect what you are typing. The name that appears there (**TankLvl.OUT**) will become the name of the measurement coming from this analog input after it has been converted to floating point and placed in the RUG3’s floating point data base. Immediately below the module name entry cell is a larger cell labeled ‘Description’ that holds a brief description of the uses of the input properties below it. Down below, in the ‘Inputs and constants’ list, you will set the properties defining how this analog input will behave to produce a useful analog value for use by the rest of the program. The first input item, ‘Input Channel #, 1-6’, has already been filled in with a ‘1’ to indicate that you are working on the analog input module for channel 1. Click on the cell to the right of the **1=enable, 0=hold** item and enter a ‘1’ to designate that you want the module to calculate its EU (engineering units) value on each scan. Now, select the cell to the right of the **Type: 0=4-20ma 1=0-5v** field and enter a ‘0’ to designate that this is a 4-20 ma. type analog input. In the next two cells below this one enter values of 0.0 for **EU at 4 ma or 0 v** and 15.0 for **EU at 20 ma or 5** v. This will specify that 4 ma. results in a tank level of 0.0; and that 20 ma. results in a tank level of 15.0 feet. Next, click on the cell to the right of the title **Filter sec** and enter the value “1000”. This sets the analog input’s filter time constant to 1000 scans to slow down its response to transients. Finally, set the high and low output limits to 30.0 and -10.0, respectively to give some under/overrange space. Your panel should look like the one below.

Getting Started



Module Type: AnalogInput

Module name, this instance: TankLvl

Description:

Module scales raw AI's to give floating point engineering units (EU) result.
If type=0, unit engages current sense resistor and uses 4-20 ma. calibration.
If type=1, unit disconnects current sense resistor and uses 0-5 v calibration
Filter constant sets low pass filtering...larger constant slows response to changes on input.

Inputs and constants:

Item:	Val Assigned:
Input Channel #, 1-6	1
1=enable, 0=hold	1
Type: 0=4-20ma, 1=0-5v	0
EU at 4 ma or 0 v	0
EU at 20 ma or 5 v	15
Filter constant	1000
High output limit EU	30
Low output limit EU	-10

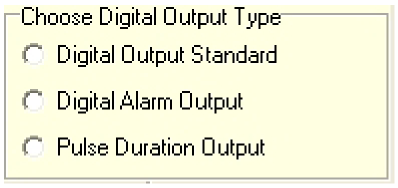
Outputs to Data Bases:

Item:	Name in Database:
EU Output	TankLvl.Out
Raw A/D output	TankLvl.Raw

Figure 5 AI Module With Properties Set

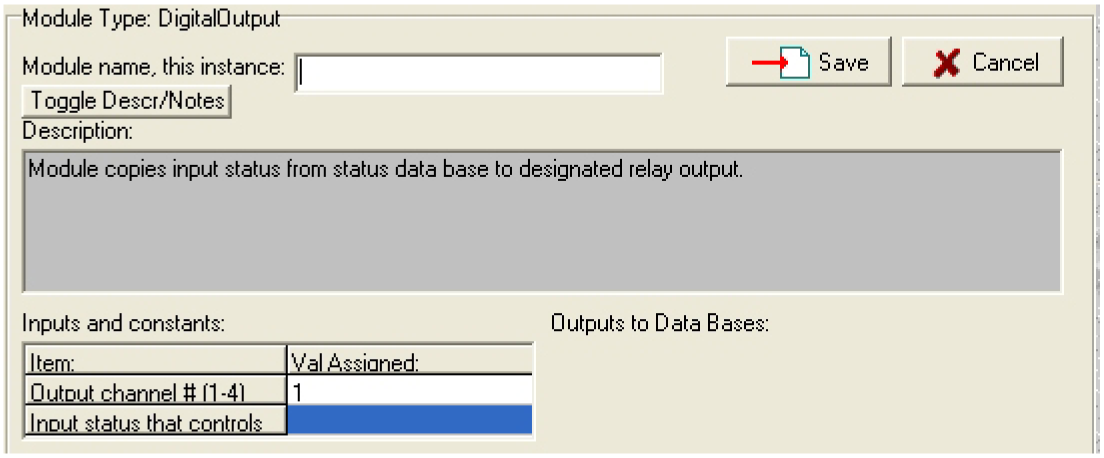
OK, we're done, click on the **Save** button and we will return to the screen showing our I/O. It should look the same as before except that the name for card 1's channel 1 entry should have changed to **TankLvl**.

Now we'll set up relay output channel 1 to be our tank low alarm. Click on the digital output channel 1 entry and you should get a channel type selection panel such as shown below:



Choose Digital Output Type
- Digital Output Standard
- Digital Alarm Output
- Pulse Duration Output

Select **Digital Output Standard**, and the new module screen should appear as shown below:



Module Type: DigitalOutput

Module name, this instance:

Description:

Module copies input status from status data base to designated relay output.

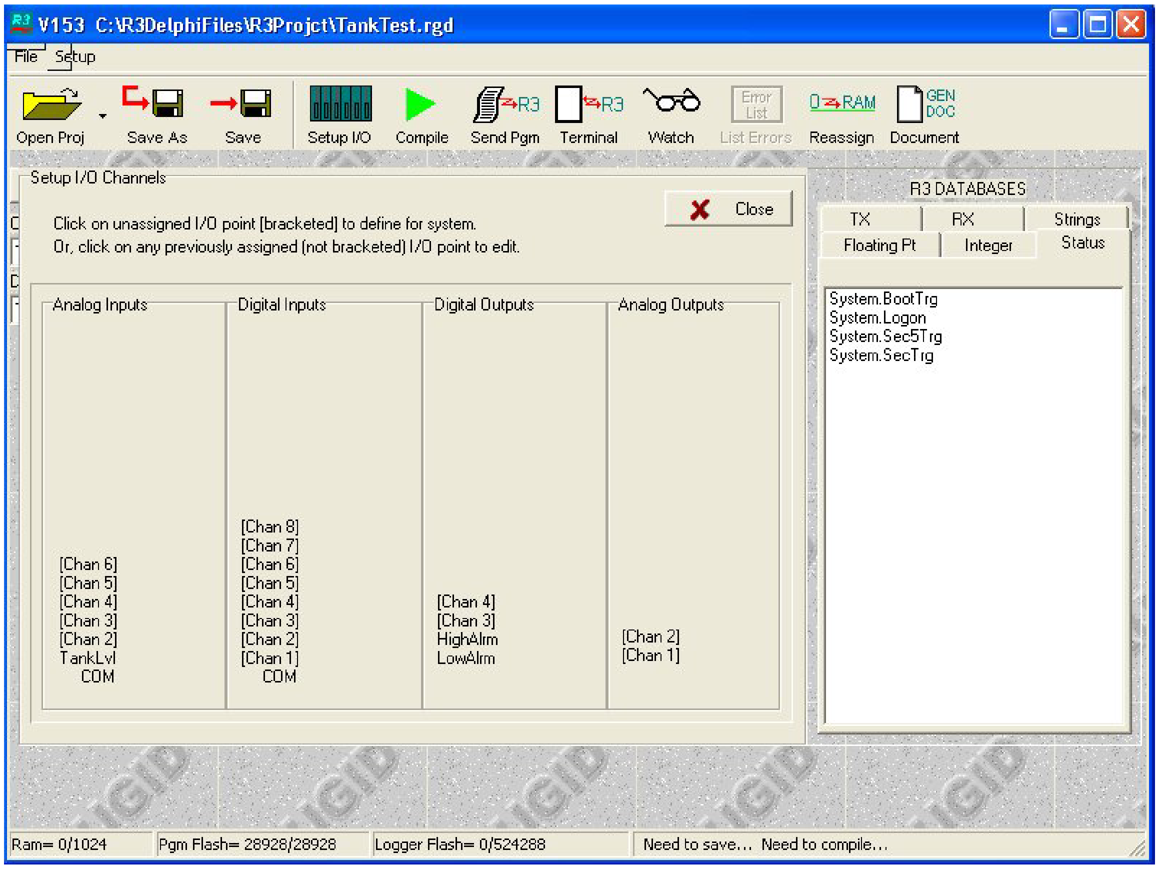
Inputs and constants:

Item:	Val Assigned:
Output channel # (1-4)	1
Input status that controls	

Outputs to Data Bases:

Figure 6 Digital Output Configuration Panel

Here, we just want to give it a module name of "LowAlrm" and click on **Save**. Similarly, click on card 2's Chan 2 point and name it "HiAlrm". We're now done with identifying I/O for this project. Our I/O representation should look like this:



Setup I/O Channels

Click on unassigned I/O point [bracketed] to define for system.
Or, click on any previously assigned (not bracketed) I/O point to edit.

Analog Inputs	Digital Inputs	Digital Outputs	Analog Outputs
	[Chan 8]		
	[Chan 7]		
[Chan 6]	[Chan 6]		
[Chan 5]	[Chan 5]		
[Chan 4]	[Chan 4]	[Chan 4]	
[Chan 3]	[Chan 3]	[Chan 3]	
[Chan 2]	[Chan 2]	HighAlrm	[Chan 2]
TankLvl	[Chan 1]	LowAlrm	[Chan 1]
COM	COM		

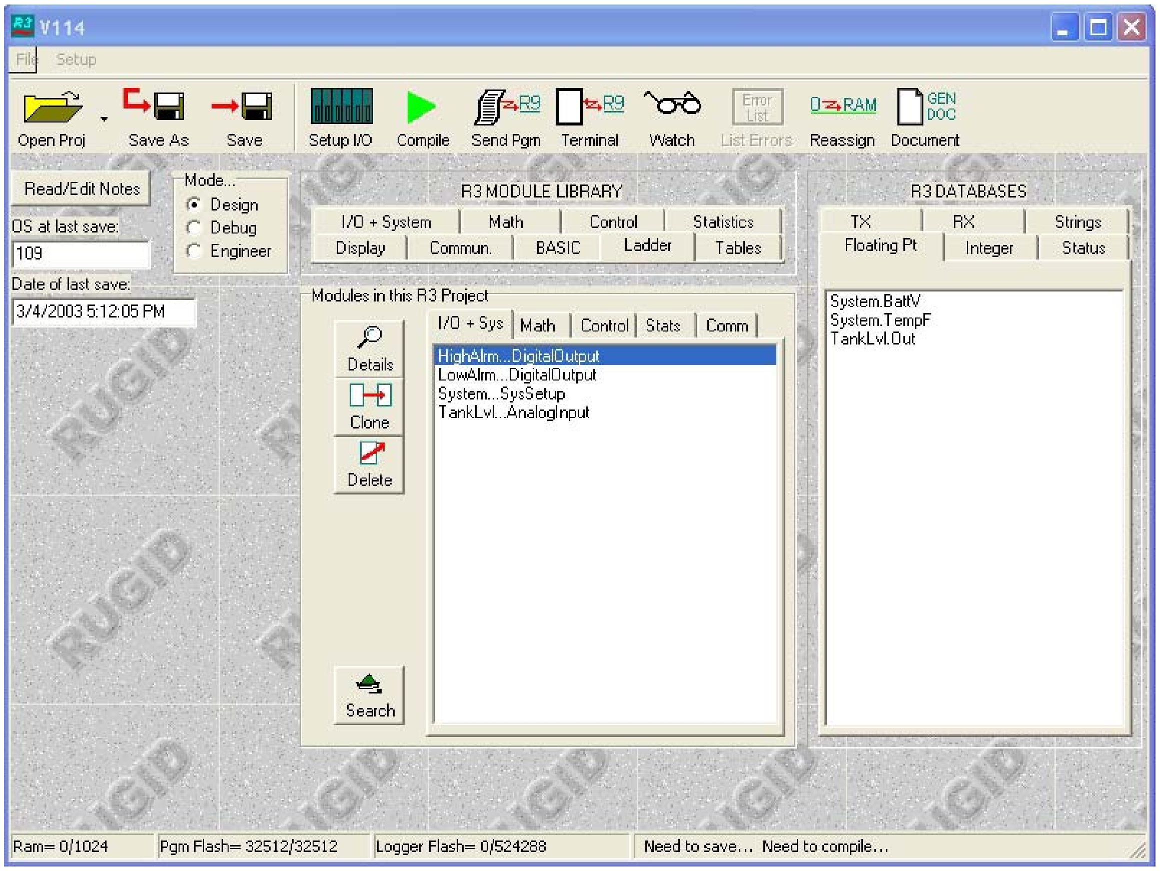
System.BootTrg
System.Logon
System.Sec5Trg
System.SecTrg

Ram= 0/1024 | Pgm Flash= 28928/28928 | Logger Flash= 0/524288 | Need to save... Need to compile...

Figure 7 I/O Channels With I/O Modules Installed

Notice that the first analog input channel has the name "TankLvl" and that the first two digital output channels are named "LowAlrm" and "HighAlrm" respectively. The remaining channels have their names in braces indicating that they are unused. We're ready to leave this screen and do some other things. Click on the large **Close** key in the upper right hand corner and you'll get the main project design screen illustrated in the figure below:

Getting Started



Mode... [✓ Design] Debug Engineer

OS at last save: 109

Date of last save: 3/4/2003 5:12:05 PM

R3 MODULE LIBRARY: I/O + System | Math | Control | Statistics | Display | Commun. | BASIC | Ladder | Tables

R3 DATABASES: TX | RX | Strings | Floating Pt | Integer | Status

Modules in this R3 Project: I/O + Sys | Math | Control | Stats | Comm

HighAlrm...DigitalOutput
LowAlrm...DigitalOutput
System...SysSetup
TankLvl...AnalogInput

System.BattV
System.TempF
TankLvl.Out

Ram= 0/1024 | Pgm Flash= 32512/32512 | Logger Flash= 0/524288 | Need to save... Need to compile...

Figure 8 Main Project Panel

Notice a few things here. In the middle of the screen are two sets of tabs…**R3 MODULE LIBRARY**, and **MODULES in this R3 Project**. On the right side of the screen is a third set of tabs labeled **R3 DATABASES**. Here is what they mean:

R3 MODULE LIBRARY Preprogrammed modules we can use to design our project.
MODULES in this R3 Project…Modules we have configured and installed in our project.
R3 DATABASES…The outputs of modules we have named and installed in our project.

Now, click on the tab labeled **I/O + System** in the **Modules in this R3 Project** set of tabs. In the middle of our screen should appear a list of I/O modules in our project. In our case, so far we have:

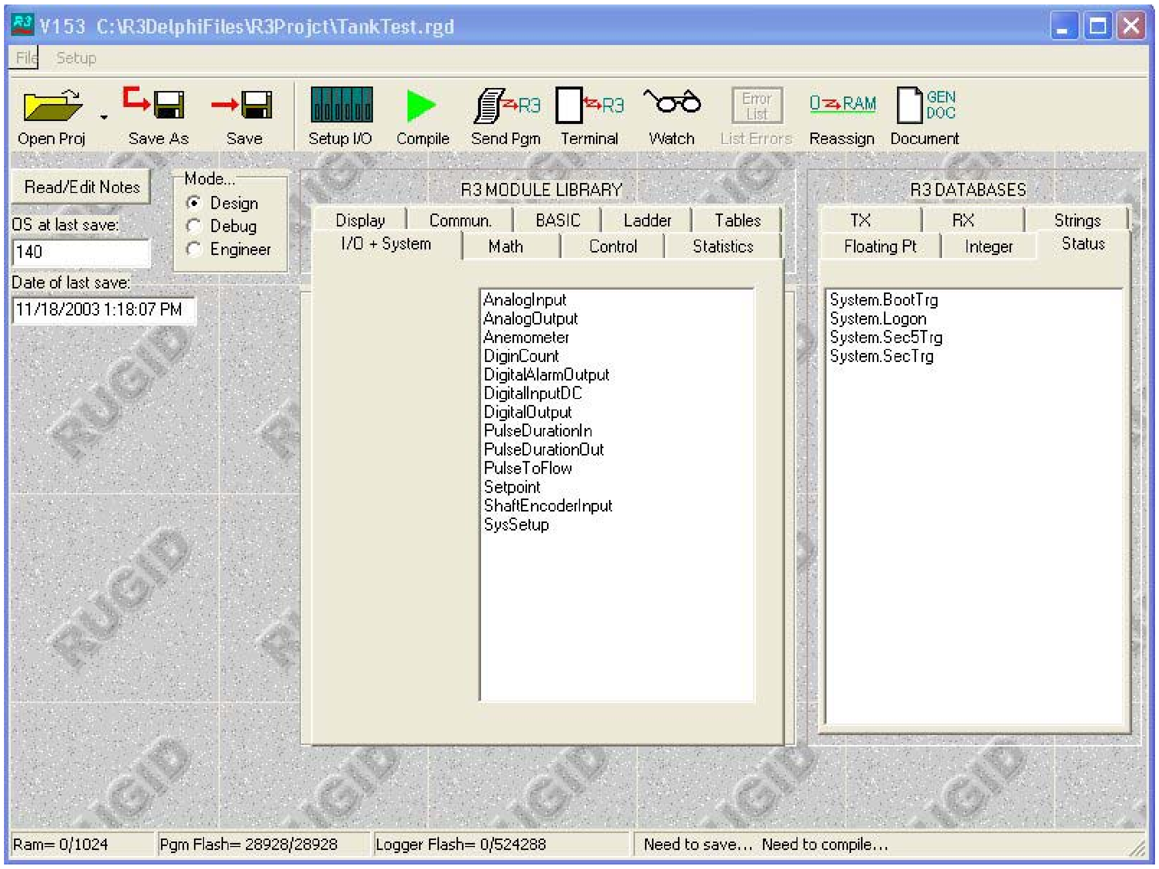
HighAlrm…Digital output
LowAlrm…Digital output
System…SysSetup
Tanklvl…Analog input

We configured the **HighAlrm**, **LowAlrm** and **TankLvl** modules; the system automatically installed the **System** module, since all projects need it. Notice that each name in this list is composed of the name we gave the I/O module (e.g., "TankLvl"), followed by its module type (e.g., "AnalogInput").

On the right of our screen we have the RUG3 databases for our project. Only one is visible at a time. Click on the **Floating Pt** tab to see the floating point database, as shown. It should have an entry named **TankLvl**, the name we gave our first analog input module, plus two outputs from the system module called **System.BattV** and **System.TempF**. Each time a module is added to the project, its outputs become entries in the databases. In the case of the analog input module, the analog input module takes the raw count from the board's A/D converter, scales it as we have specified to the range of 0.0 to 15.0 feet, low pass filters it, and places it in the floating point database. Therefore, our tank level is now in the floating point database **where it can be used as an input by any other module.**

Setting Up Setpoints

Now we need a pair of high and low alarm setpoints. At the top of the main project screen above, notice the set of tabs labeled **R3 MODULE LIBRARY**. This is where the preprogrammed modules reside for us to use in designing our projects. Modules and other tools for configuring our project are divided into functional categories such as Math, Control, etc. You can get to a module by first clicking on the tab representing the type of function you want, and then selecting the specific module from a list. For example, we want a setpoint module, which is in the **I/O + System** list. Click on the **I/O + System** tab in the **R3 MODULE LIBRARY**. (A setpoint is really an input point…it just comes from an operator.) You should see a list of I/O modules appear in the middle of the screen as illustrated below:



OS at last save: 140

Date of last save: 11/18/2003 1:18:07 PM

Mode... [✓ Design] Debug Engineer

AnalogInput
AnalogOutput
Anemometer
DiginCount
DigitalAlarmOutput
DigitalInputDC
DigitalOutput
PulseDurationIn
PulseDurationOut
PulseToFlow
Setpoint
ShaftEncoderInput
SysSetup

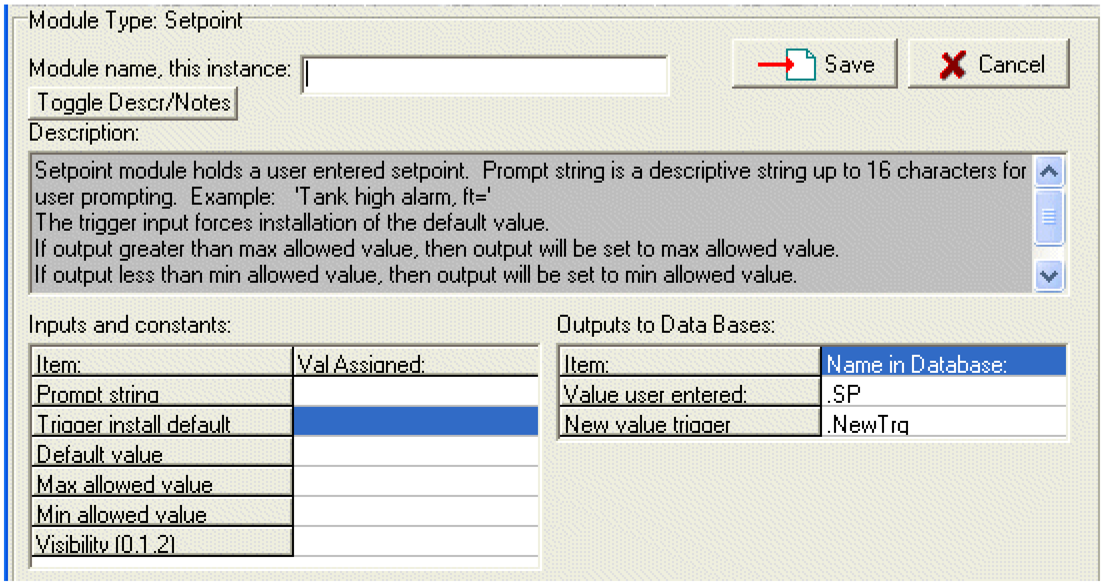
System.BootTrg
System.Logon
System.Sec5Trg
System.SecTrg

Ram= 0/1024 | Pgm Flash= 28928/28928 | Logger Flash= 0/524288 | Need to save... Need to compile...

Figure 9 Selecting a Setpoint Module from Module Library

Click on **Setpoint** and you will get the setpoint module's entry screen as presented below:

Getting Started



Setpoint module holds a user entered setpoint. Prompt string is a descriptive string up to 16 characters for user prompting. Example: 'Tank high alarm, ft='
The trigger input forces installation of the default value.
If output greater than max allowed value, then output will be set to max allowed value.
If output less than min allowed value, then output will be set to min allowed value.

Inputs and constants:

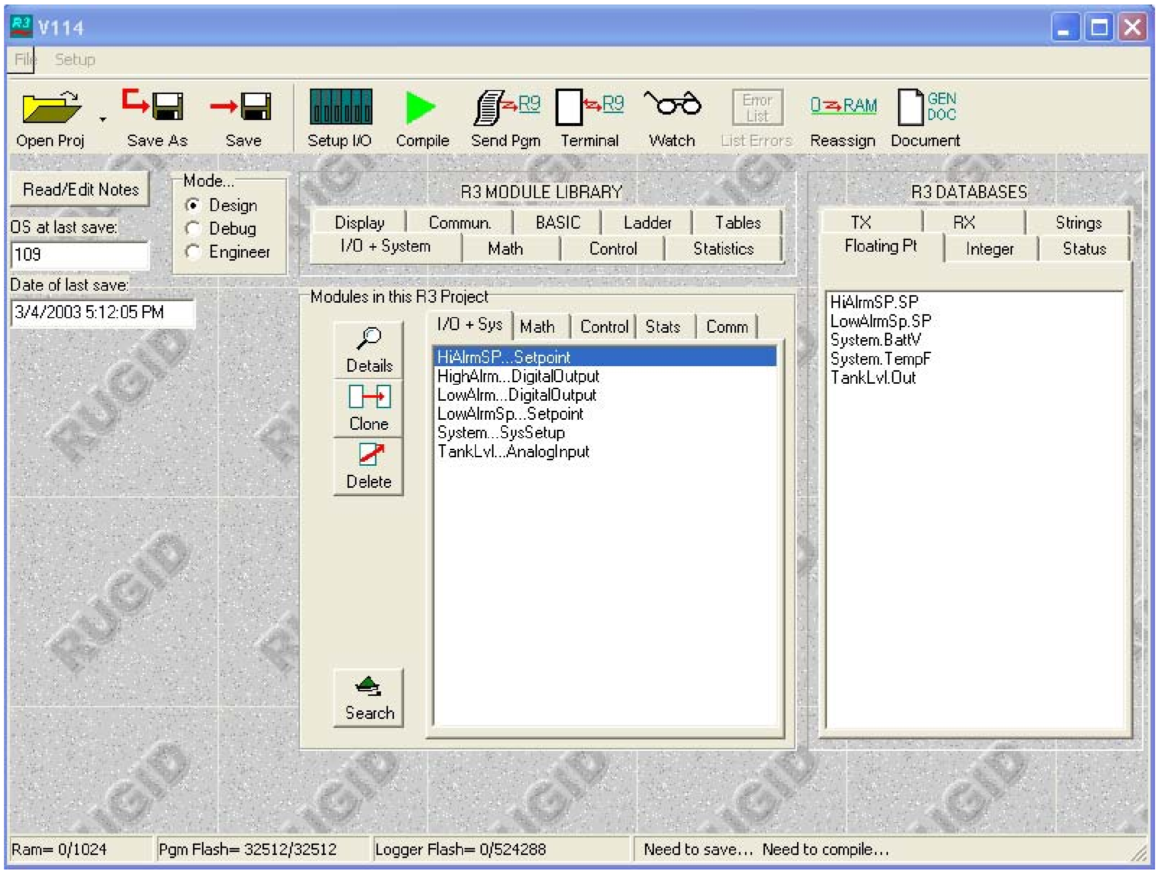
Item:	Val Assigned:
Prompt string	
Trigger install default	
Default value	
Max allowed value	
Min allowed value	
Visibility (0,1,2)	

Outputs to Data Bases:

Item:	Name in Database:
Value user entered:	.SP
New value trigger	.NewTrg

Figure 10 Setpoint Module Configuration Panel

This is the type of screen you will get whenever you select one of the modules from the module library. Each module consists of a module name, which you will supply, a description, a set of inputs and constants, and a set of outputs. The cursor should be in the setpoint screen's name edit box, so let's name this setpoint "LowAlrmSP". Do this by typing "LowAlrmSP" into the name edit box. When you have entered the name, the module's output name will become **LowAlrmSP.SP.** Now, click on the cell to the right of **Prompt string** in the **Inputs and Constants** list box. A new edit box with our cursor in it will appear just below where we entered the module's name. This is where you must enter the setpoint prompt that the operator will see when he goes to the setpoint list in the RUG3's menu system. This must adequately describe what we want the operator to enter for a setpoint. Therefore, enter "Tank low SP ft=". We'll not enter the other properties now. Note that in general, unnecessary input properties can be left blank. Now click the **Add to Project** button to save this setpoint to the project. Notice that after we save this module to the project, our new setpoint appears in the floating point data base. Similarly, select another setpoint module from the **I/O + System** tabbed list and install the high alarm setpoint.



OS at last save: 109

Date of last save: 3/4/2003 5:12:05 PM

Mode: [✓ Design] Debug Engineer

Modules in this R3 Project:
HiAlrmSP...Setpoint
HighAlrm...DigitalOutput
LowAlrm...DigitalOutput
LowAlrmSp...Setpoint
System...SysSetup
TankLvl...AnalogInput

R3 DATABASES (Floating Pt):
HiAlrmSP.SP
LowAlrmSp.SP
System.BattV
System.TempF
TankLvl.Out

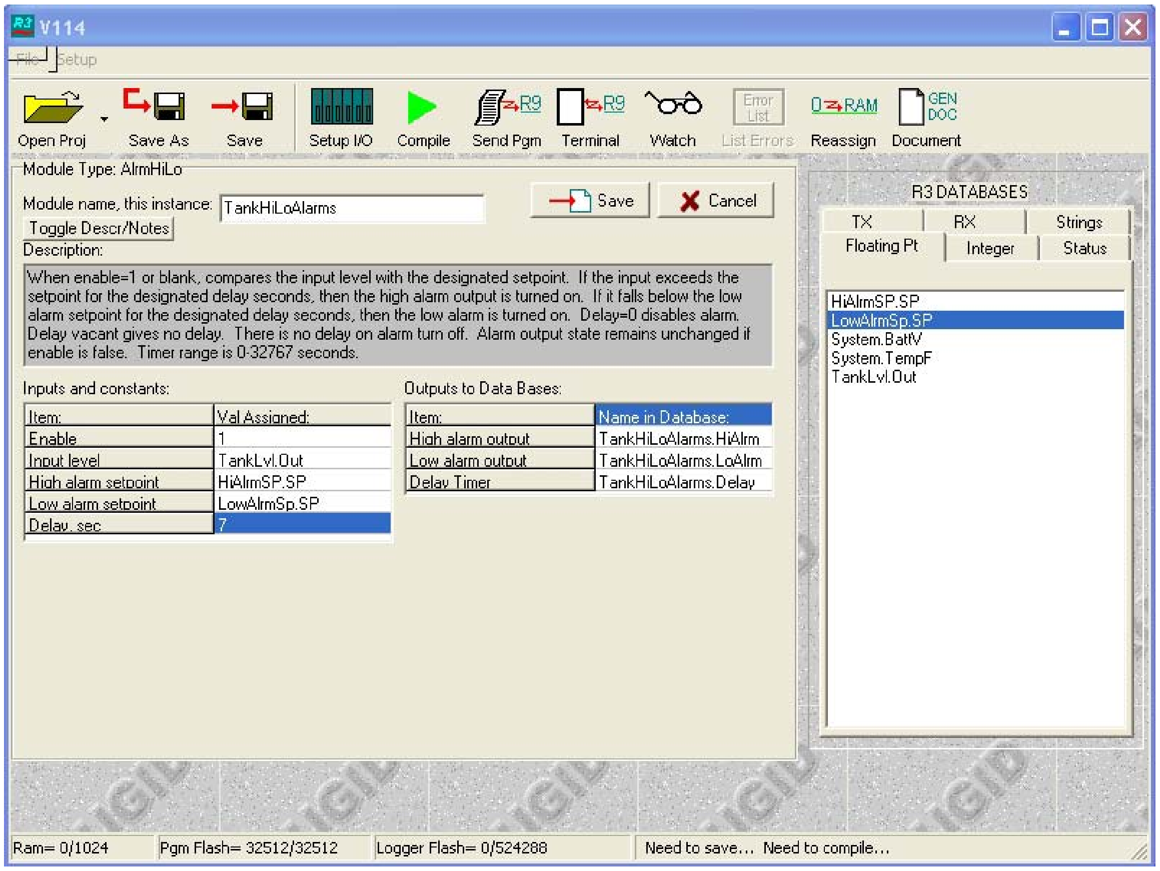
Ram= 0/1024 | Pgm Flash= 32512/32512 | Logger Flash= 0/524288 | Need to save... Need to compile...

Figure 11 Example Project With Setpoints Configured

Connecting Modules Together

We now have six modules in our project and five entries in our floating point data base. It's time to begin hooking these modules together and engaging other modules that perform tasks that our project needs. We do that by selecting modules from the module library and taking the outputs of our previously configured modules, which are in one of the data bases, and dragging them into the inputs of modules that do the work we want done. The first module we need is the **AlrmHiLo** module from the **Control** tab. Click on the **Control** tab in the **R3 MODULES LIBRARY**. Now, select **AlrmHiLo** from the control module list. The **AlrmHiLo** module configuration panel should appear. Type "TankHiLoAlarms" in its module name edit box. This module will compare its input with its high alarm setpoint input and if the input exceeds the setpoint for the specified delay time, it will turn on its high alarm output. Similarly, it will also compare its input with the low alarm setpoint and turn on the low alarm output if the input falls below the low alarm setpoint for more than the delay setpoint. We want our tank level to be this module's input. To make that assignment, we must drag our **TankLVL.OUT** entry from the floating point data base over to the **Input** cell in our module and drop it. Go ahead and do that now. Similarly, drag the **HiAlrmSP.SP** entry from the floating point data base over to this module's **High alarm setpoint** cell. And drag the **LoAlrmSP.SP** entry from the floating point database and drop it in the **Low alarm setpoint** cell. Finally, enter the value "7" into the **Delay sec** cell. This sets the amount of time the tank level must violate the setpoints before this module will turn on either alarm output. Your module should look like this:

Getting Started



Module Type: AlrmHiLo

Module name, this instance: TankHiLoAlarms

Description:

When enable=1 or blank, compares the input level with the designated setpoint. If the input exceeds the setpoint for the designated delay seconds, then the high alarm output is turned on. If it falls below the low alarm setpoint for the designated delay seconds, then the low alarm is turned on. Delay=0 disables alarm. Delay vacant gives no delay. There is no delay on alarm turn off. Alarm output state remains unchanged if enable is false. Timer range is 0-32767 seconds.

Inputs and constants:

Item:	Val Assigned:
Enable	1
Input level	TankLvl.Out
High alarm setpoint	HiAlrmSP.SP
Low alarm setpoint	LowAlrmSp.SP
Delay, sec	7

Outputs to Data Bases:

Item:	Name in Database:
High alarm output	TankHiLoAlarms.HiAlrm
Low alarm output	TankHiLoAlarms.LoAlrm
Delay Timer	TankHiLoAlarms.Delay

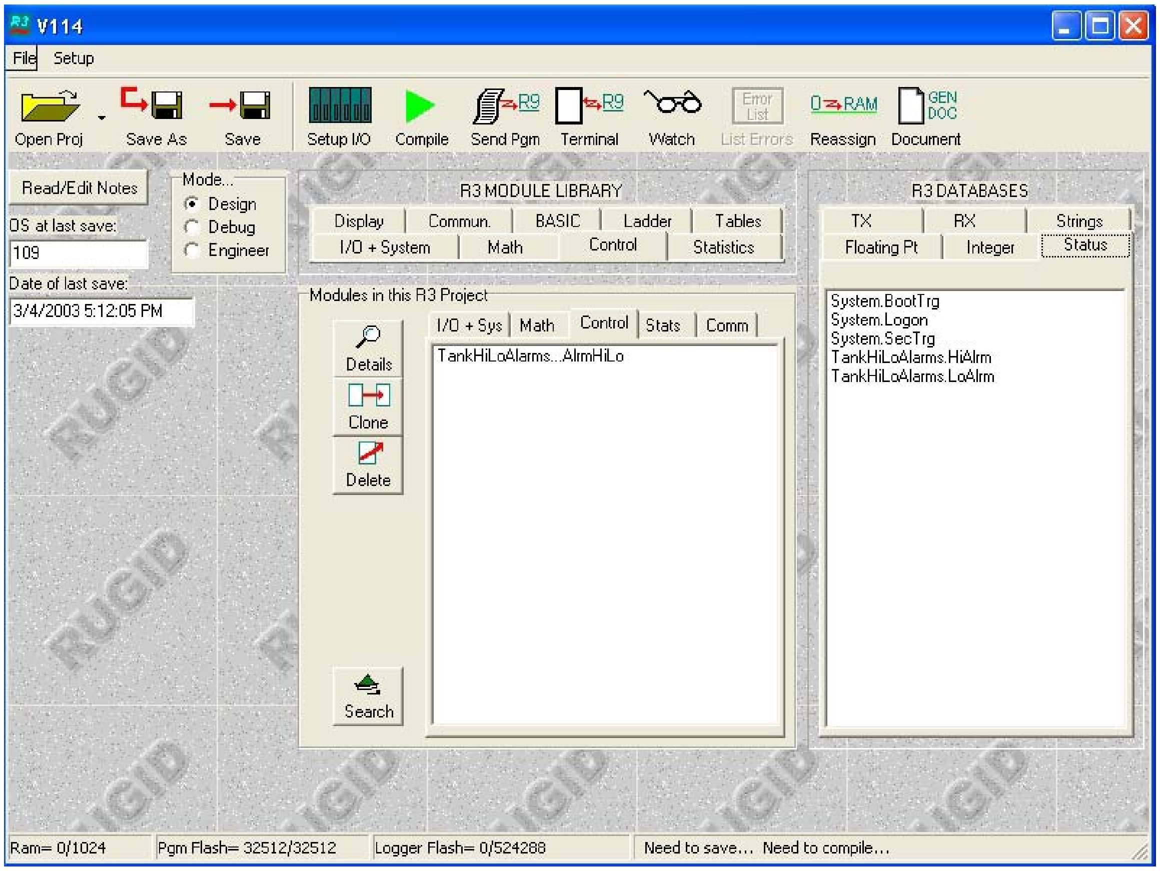
HiAlrmSP.SP
LowAlrmSp.SP
System.BattV
System.TempF
TankLvl.Out

Ram= 0/1024 Pgm Flash= 32512/32512 Logger Flash= 0/524288 Need to save... Need to compile...

Figure 12 Configured HiLo Alarm Generator Module

Notice that for each module input cell, you can enter either a constant value into the input cell, drag a signal from a database into the cell, or leave it blank. If you leave it blank, it will be given a value that is usually transparent to the module's math (0.0 or 1.0). However, it is best to enter all values to be certain. Also, the compiler will handle value type conversions such as floating to integer, so you need not be concerned with matching the type of signal a module is expecting with a particular database. The exception is that module inputs that expect strings must have either string constants or strings taken from the string database. Basically, the **AlrmHiLo** module we just configured compares its input (tank level) with its alarm setpoints, and, if the alarm condition exists for at least 7 seconds, it will turn on its corresponding alarm output. Now, click the save button to save this module to the project.

Your project screen should now look like this, where the new **AlrmHiLo** module is shown in the list of control modules, and the high and low alarms are shown in the status database along with some system statuses:



Mode...
- (•) Design
- ( ) Debug
- ( ) Engineer

OS at last save: 109

Date of last save: 3/4/2003 5:12:05 PM

Modules in this R3 Project:

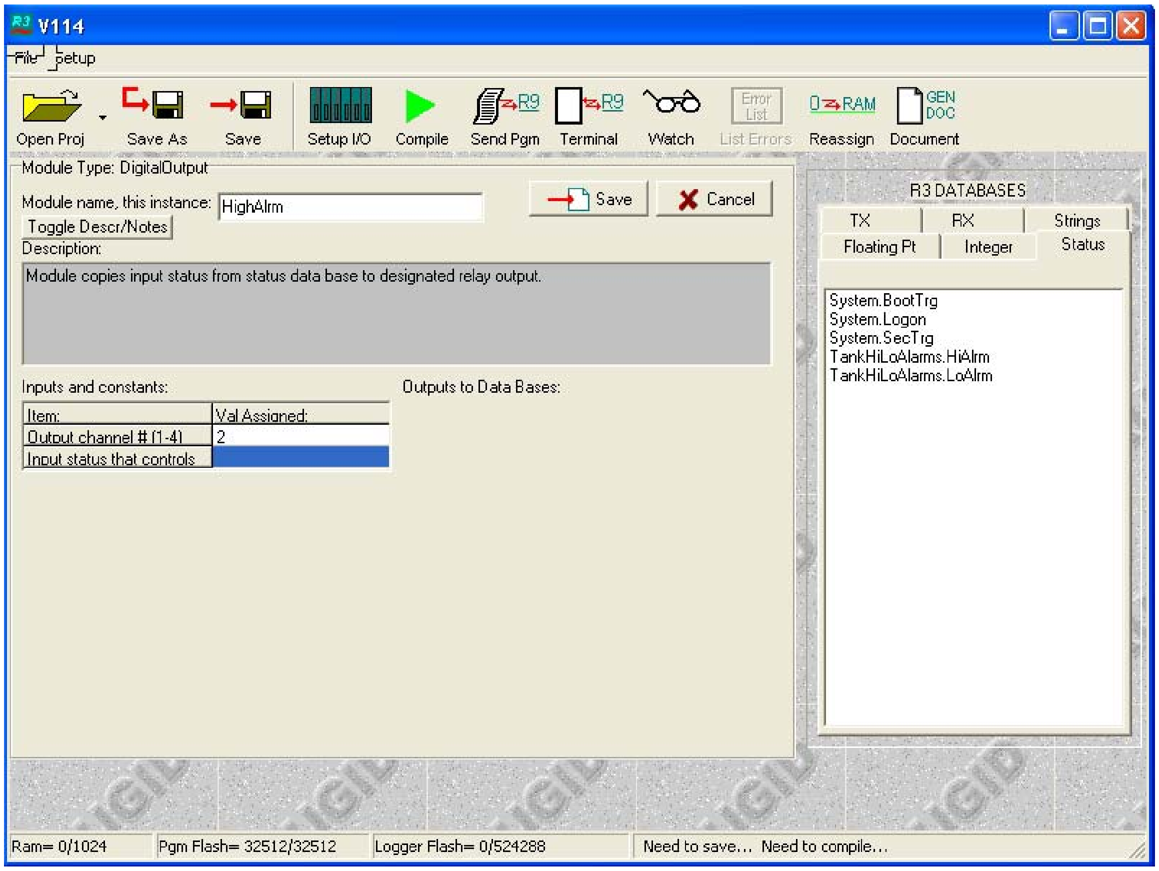
TankHiLoAlarms...AlrmHiLo

System.BootTrg
System.Logon
System.SecTrg
TankHiLoAlarms.HiAlrm
TankHiLoAlarms.LoAlrm

Ram= 0/1024 | Pgm Flash= 32512/32512 | Logger Flash= 0/524288 | Need to save... Need to compile...

Figure 13 Project Screen After Alarm Generators Added

If at any time you wish to modify the setup of any module in the system, you can simply select the module from the project module list in the middle of the screen and click on the **Details** button. We want to do that now so that we can connect the alarm generator outputs to our relays. In the project module list in the middle of the screen, click on the **I/O + Sys tab**, then click on **HiAlrm…Digital Output**, then click on **Details**. The relay output module we set up at the beginning should appear. Notice it is missing its input specifier. This is because at the time we assigned the analog input and relay outputs, we had not yet specified the alarm generators. Also, before we can drag the output from the appropriate alarm generator to the input of the relay output we must select the status data base instead of the floating point data base that is presently showing. Do this by clicking on the **Status** tab over to the right in the data base window. Your screen should look like this.



Module Type: DigitalOutput

Module name, this instance: HighAlrm

Toggle Descr/Notes

Description:

Module copies input status from status data base to designated relay output.

Inputs and constants:

Item:	Val Assigned:
Output channel # (1-4)	2
Input status that controls	

Outputs to Data Bases:

System.BootTrg
System.Logon
System.SecTrg
TankHiLoAlarms.HiAlrm
TankHiLoAlarms.LoAlrm

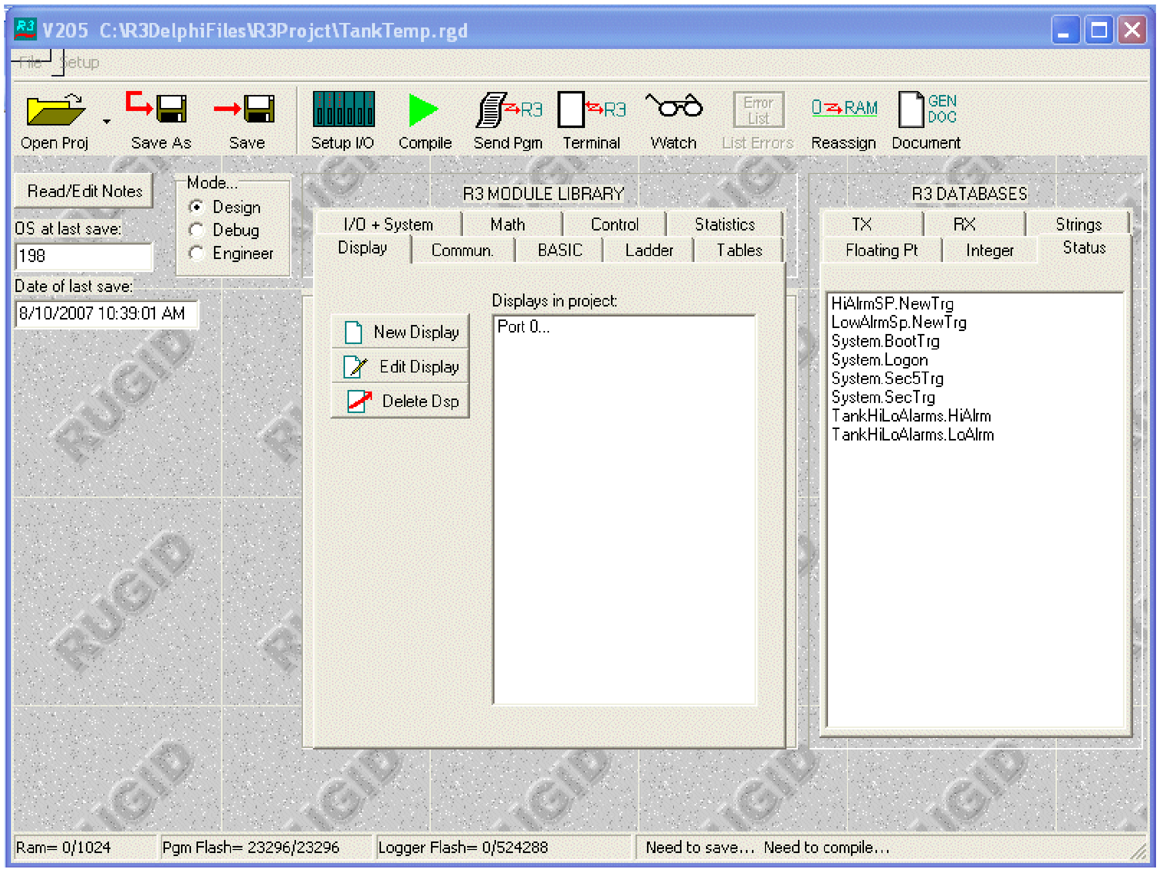
Ram= 0/1024 | Pgm Flash= 32512/32512 | Logger Flash= 0/524288 | Need to save... Need to compile...

Figure 14 Digital Output Configuration Page With Status Data Base Showing

Now drag the **TankHiLoAlarms.HiAlrm** entry from the status database over to the relay output module's **Input status that controls** cell. This will cause the **TankHiLoAlarms.HiAlrm** status output to control the relay. Similarly, call up the **LowAlrm…Digital output** module from the project module list and drag the **TankHiLoAlarms.LoAlrm** point from the status database over to the module's control input. At this point, this configuration could be compiled and sent to the RUG3 to run. It would read the analog input and control the relays properly. Before we do that, let's add a display to finish the project.

LCD Display Setup

Adding a display to the system will make an operator's job a lot easier than if the unit has no display since we can show him at a glance what the tank level is and how the setpoints are set. To add a display, first click on the **Display** tab in the **R3 MODULE LIBRARY** window. Your screen will change to this:



OS at last save: 198

Date of last save: 8/10/2007 10:39:01 AM

Mode... (● Design) Debug Engineer

Displays in project:

Port 0...

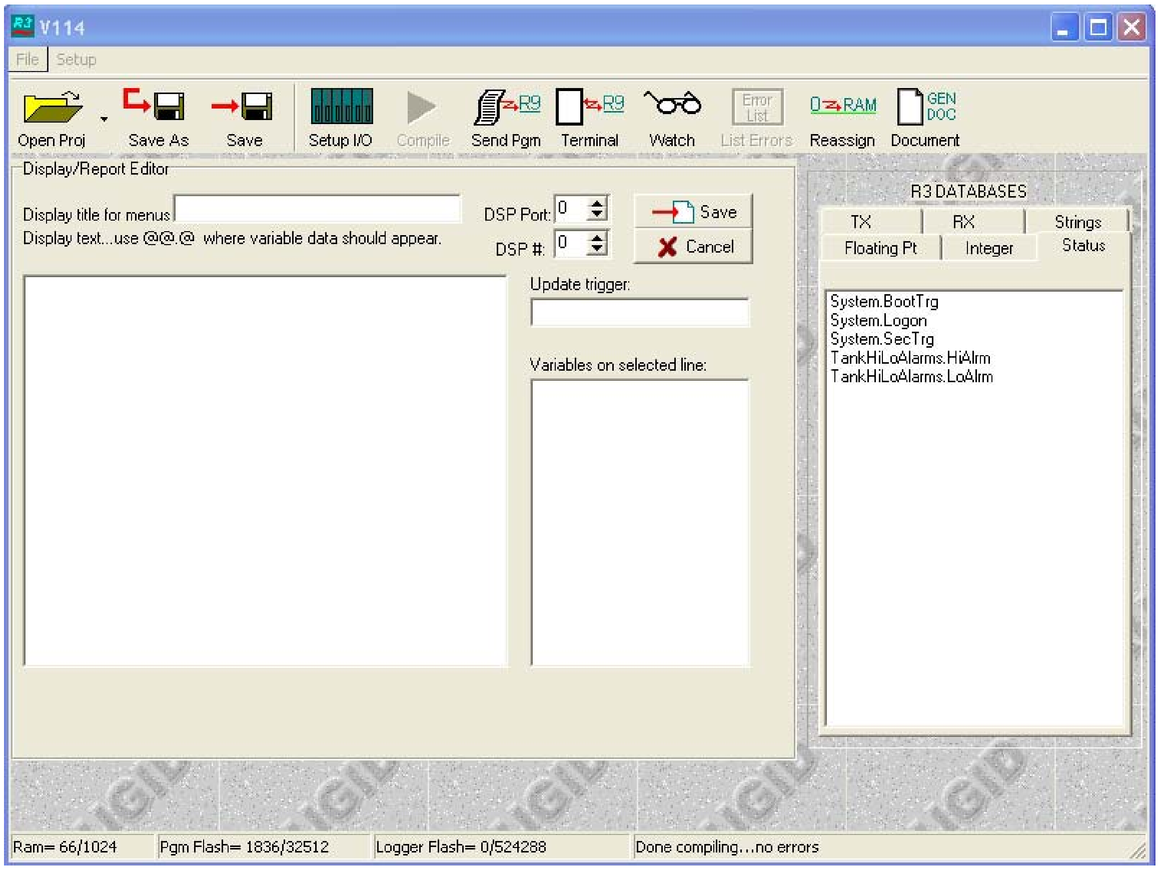
HiAlrmSP.NewTrg
LowAlrmSp.NewTrg
System.BootTrg
System.Logon
System.Sec5Trg
System.SecTrg
TankHiLoAlarms.HiAlrm
TankHiLoAlarms.LoAlrm

Ram= 0/1024 | Pgm Flash= 23296/23296 | Logger Flash= 0/524288 | Need to save... Need to compile...

Figure 15 Display Selection Tab in Module Library

There are not yet any displays defined, so the display list is blank. Click on **New Display** and the display definition screen will appear:

Getting Started



System.BootTrg
System.Logon
System.SecTrg
TankHiLoAlarms.HiAlrm
TankHiLoAlarms.LoAlrm

Ram= 66/1024 | Pgm Flash= 1836/32512 | Logger Flash= 0/524288 | Done compiling…no errors

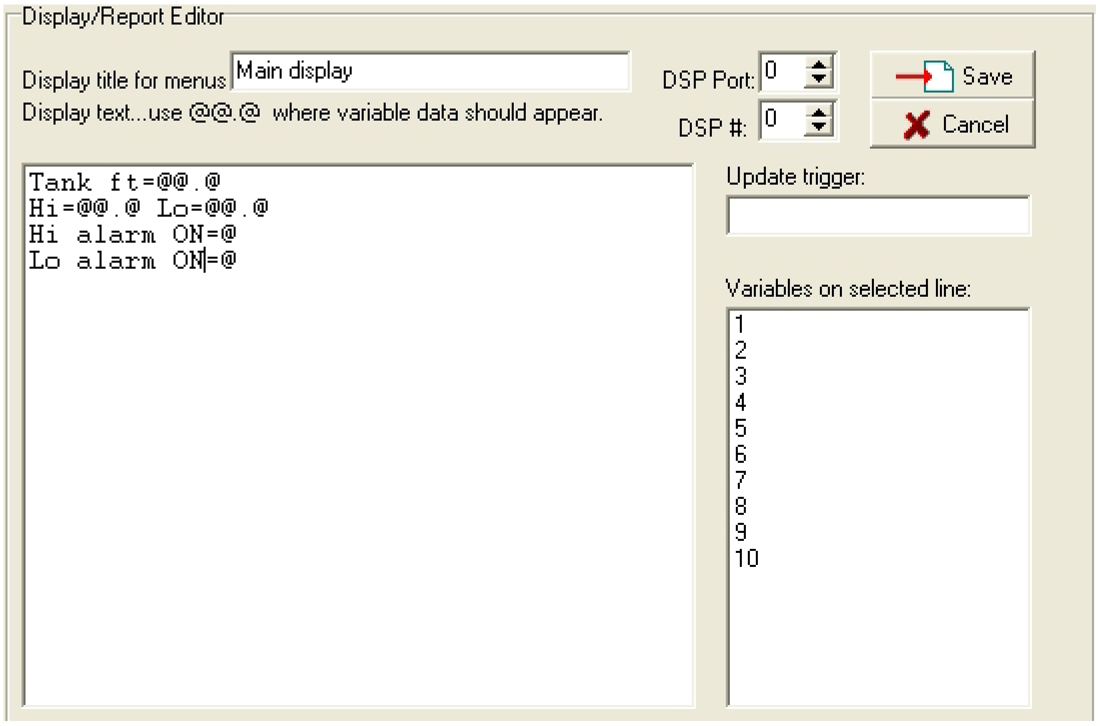
Figure 16 Display Editing Panel

First, we must name the display. The name we enter will be used in display lists presented to the operator. To do this, in the **Display title for menus** edit box above the large window, type in "Main display". The large window to the left is where you will enter your actual display definition. Now, move the cursor to the upper left corner of that large window and click. We're now ready to enter our display data. Type the following into the display window:

```
Tank ft=@@.@
Lo=@@.@ Hi=@@.@
Hi alarm ON=@
Lo alarm ON=@
```

Your screen should look like this:

Getting Started


Display/Report Editor

Display title for menus: Main display

Display text...use @@.@ where variable data should appear.

DSP Port: 0

DSP #: 0

Update trigger:

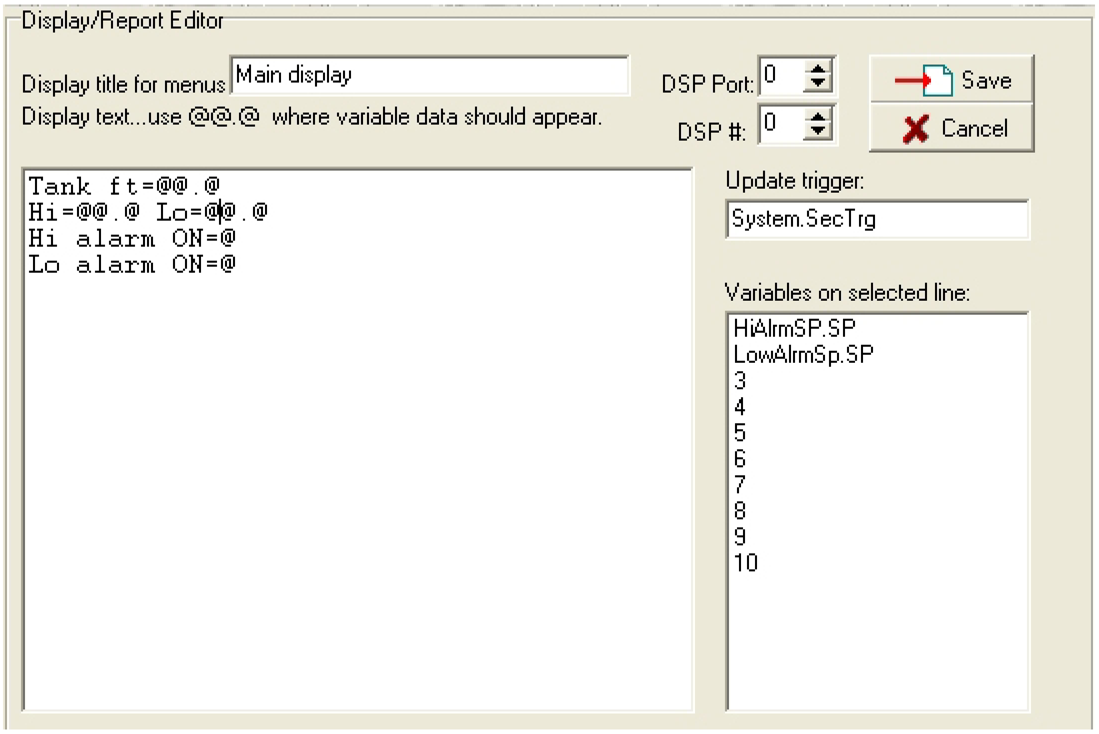
Variables on selected line:

```
Tank ft=@@.@
Hi=@@.@ Lo=@@.@
Hi alarm ON=@
Lo alarm ON=@
```

```
1
2
3
4
5
6
7
8
9
10
```

Figure 17 Display Defined

Notice the use of @@.@ symbols. Basically, anywhere you want active data from a data base to appear on the LCD, you specify the location and format by using the @@.@ characters. '@' symbols and decimal points are regarded as one field with the location of the decimal point specifying where the decimal point will usually be placed on the LCD screen. Also, notice that since the RUG3's LCD has two lines of 16 characters each, we must pay close attention not to exceed the 16 character line length. We can have as many lines as we want for any single display (up to 100 lines) but only show two lines at any time. Now, we must tell the RUG3 what data to present in each of the @@.@ type fields. We do that as follows. First, click on the "Tank ft=@@.@" line. Now notice the list box to the right of our display window with the numerals 1…10 down its left side. That is where the tags from the databases go. For each @@.@ field in our selected display line we must drag in one entry from a data base and drop it into the list box beginning with the top entry in the list box corresponding to the leftmost @@.@field in our LCD display line. Go ahead and drag **TankLvl.OUT** into the top line of the **Variables on selected line** list box. (You may have to first select the floating point database on the right.) Now click on our display's second line, where we wish to show the two alarm setpoints. For each of the high alarm setpoint and low alarm setpoint in the large window, drag the corresponding alarm setpoint from the floating point data base and drop it on the #1 line and #2 line of the list box (**HiAlrmSP.SP** and **LoAlrmSP.SP**, respectively). Remember to click on the desired LCD line in the large window before dragging an entry from the database. Now click on the LCD line that reads "Hi alarm ON=@". On this line, we must drag the high alarm entry from the status database (**TankHiLoAlrms.HiAlrm**) and drop onto the top location in the variables list box. Similarly, drag the low alarm entry from the status database (**TankHiLoAlrms.LoAlrm**) and drop onto the top location in the variables list box. We still have one more thing to do before we're done…we must specify the display's update trigger. With the RUG3, the display being presented updates only when triggered, enabling the programmer to specify the timing of display updates. A trigger is an event, generated by a module, that is true for only one program scan. Using a trigger, the programmer can make the display update based on time, or based on an event such as a communications reception or keystroke, etc. We'll have the display refresh once per second. To do that, from the status data base, drag the variable named **System.SecTrg** into the display's update trigger cell. This signal is installed by default into all projects. Your screen should now look like this:


Display/Report Editor

Display title for menus: Main display

Display text...use @@.@ where variable data should appear.

DSP Port: 0

DSP #: 0

```
Tank ft=@@.@
Hi=@@.@ Lo=@@.@
Hi alarm ON=@
Lo alarm ON=@
```

Update trigger: System.SecTrg

Variables on selected line:

```
HiAlrmSP.SP
LowAlrmSp.SP
3
4
5
6
7
8
9
10
```

Figure 18 Display Panel Showing Two Setpoints Installed on Line 2

Now click the **Save** button to save this screen.

Saving the Project

Now, before anything else, let's save the project. Do this by clicking on the **Save As** menu item in the tool bar at the top of the form. Choose a name such as "TankTest" for the file and click OK.

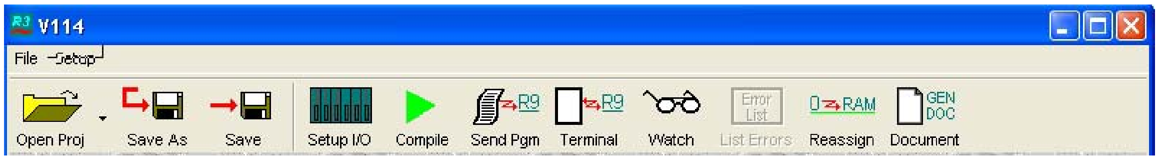
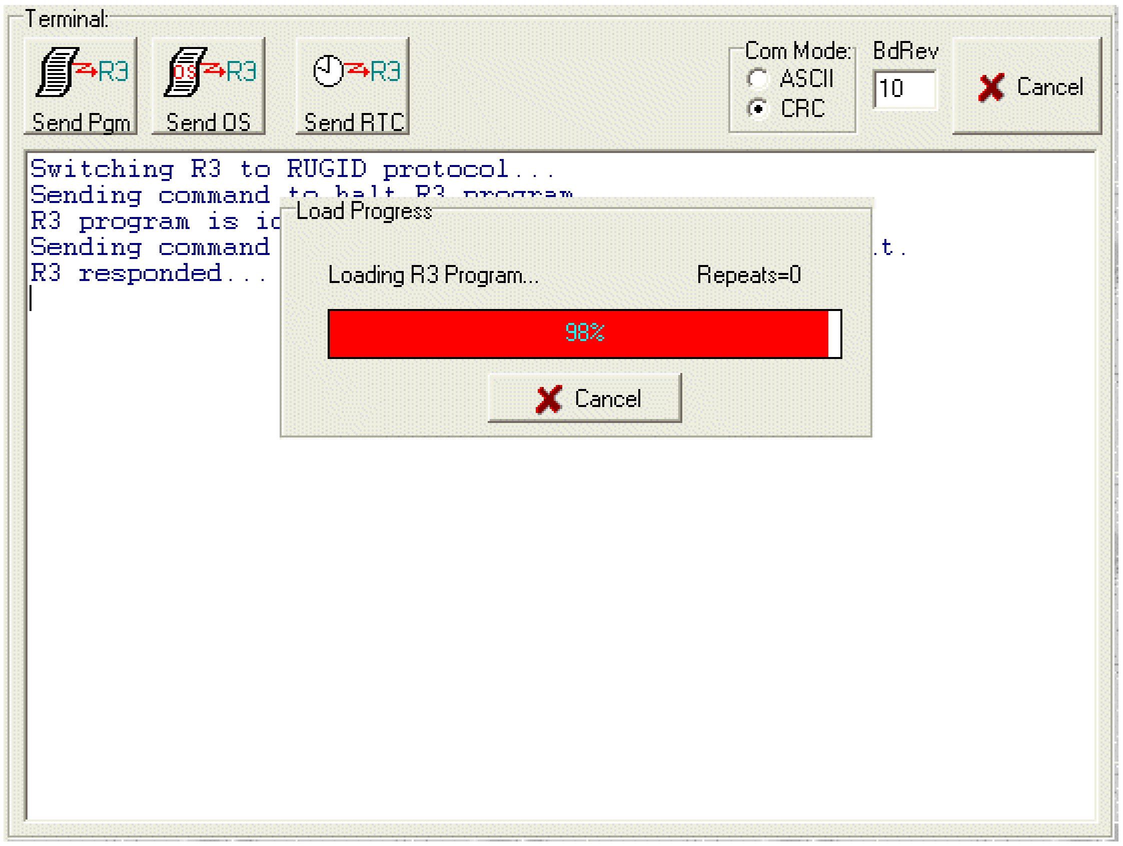


Figure 19 R3SETUP Toolbar

Compiling the Project

To compile the project, click the green **Compile** button on the toolbar. When done, a panel will show any errors the program encountered. There should be no errors in this setup. If there are any errors, you must correct them before the program will let you download the project to the RUG3. If no errors are present, click on the **SendPgm** button to send the project to the RUG3. Note that you do not have to click the **Compile** button before sending the program to the RUG3, R3SETUP will compile automatically before sending the program. The terminal screen will appear and will show you the PC's progress in sending the configuration file to the RUG3. It will look like the screen below:

Getting Started



```
Switching R3 to RUGID protocol...
Sending command to halt R3 program
R3 program is id
Sending command                        .t.
R3 responded...
```

Figure 20 Example File Load Progress Screen

When done with loading, the RUG3 will begin running the program. Look at the RUG3's LCD. You should see the following screen. If this screen is not visible, hit the [-] key to present it. Remember that from any display, you should be able to get back to the system menu by hitting the [-] key.

System Menu:

```
1DSP,2SP
5CLK,7LCD
```

From this display you can press key [1] to access the display menu and select the display we designed. You could also press key [2] to access the setpoints that control the high and low alarms that are routed to the two relays we used. Note that we did not use the setpoint default settings, so the setpoints may have wild values. You should set the setpoints to reasonable values such as 4.0 feet for low alarm, and 14.0 feet for high alarm. That's the end of the tutorial to illustrate how you design a configuration file for the RUG3. If you look at the lists in the module library, you will notice a large number of useful preprogrammed modules. You can install any of these into your program to implement a range of control strategies. No matter how complex your project is, the process is the same…select and name a module, then drag items from the databases into each module's inputs. As you save each module, its outputs become entries in the databases for use by other modules

CHAPTER 3…HARDWARE

Overview-RUG3

Each RUG3 consists of a single board with I/O connections at the top and bottom. Units are available with or without steel enclosure and with and without LCD display and keyboard and other options as identified in the following table. Note that a two channel optically isolated analog output option is available for any of the listed models simply by appending an 'O' to the part number.

Table 2 RUG3 Models

MODEL	FEATURES	MOUNTING
RUG3B, board only		Mounting holes at board corners
RUG3BL, board with LCD	LCD only, no keyboard	Mounting holes at board corners
RUG3C, steel case	None	DIN rail
RUG3D, steel case with LCD	LCD and keyboard	DIN rail
RUG3P, steel case with flange	LCD and keyboard	Through panel cutouts
OPTIONS:		
O	Analog outputs, 2 chan, 4-20 ma.	Internal
M	Modem, 300 baud	Internal
L	LCD, 2 line X 16 char.	Internal
MR	Mechanical relays, 10 amp	Internal
SR	Solid state relays, 0.5a/48VDC	Internal

Refer to Chapter 10 for unit dimensions and mounting requirements.

Hardware

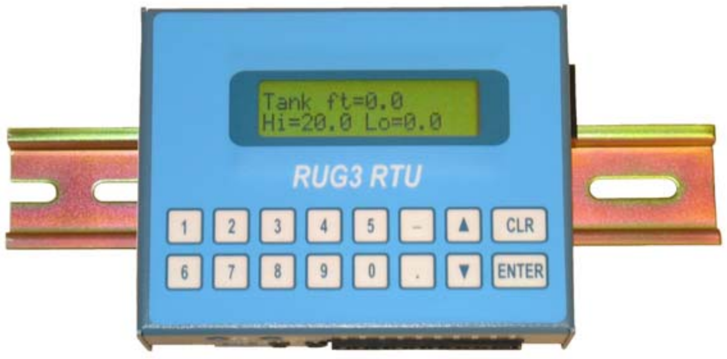


Figure 21 RUG3D Face Photo

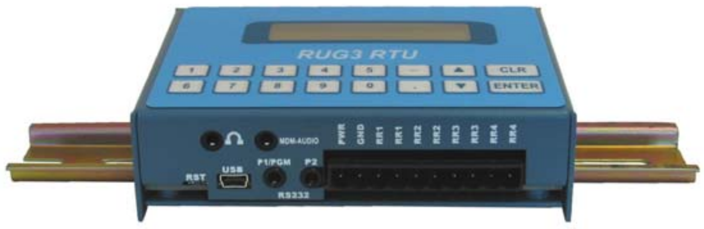


Figure 22 RUG3D Bottom Ports and Relay I/O Photo

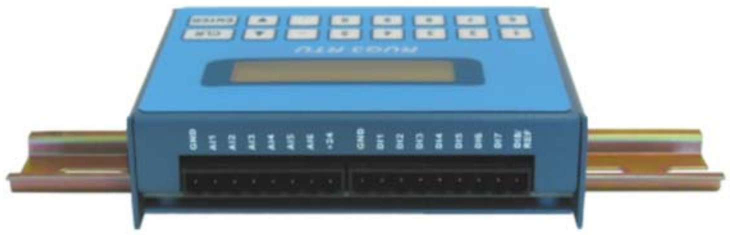


Figure 23 RUG3D Top I/O Photo

In addition to the above hardware, the RUG3 can be equipped with a 2 line by 16 character LCD and keyboard as shown in the photos. The following figure provides details of the RUG3 hardware.

Hardware

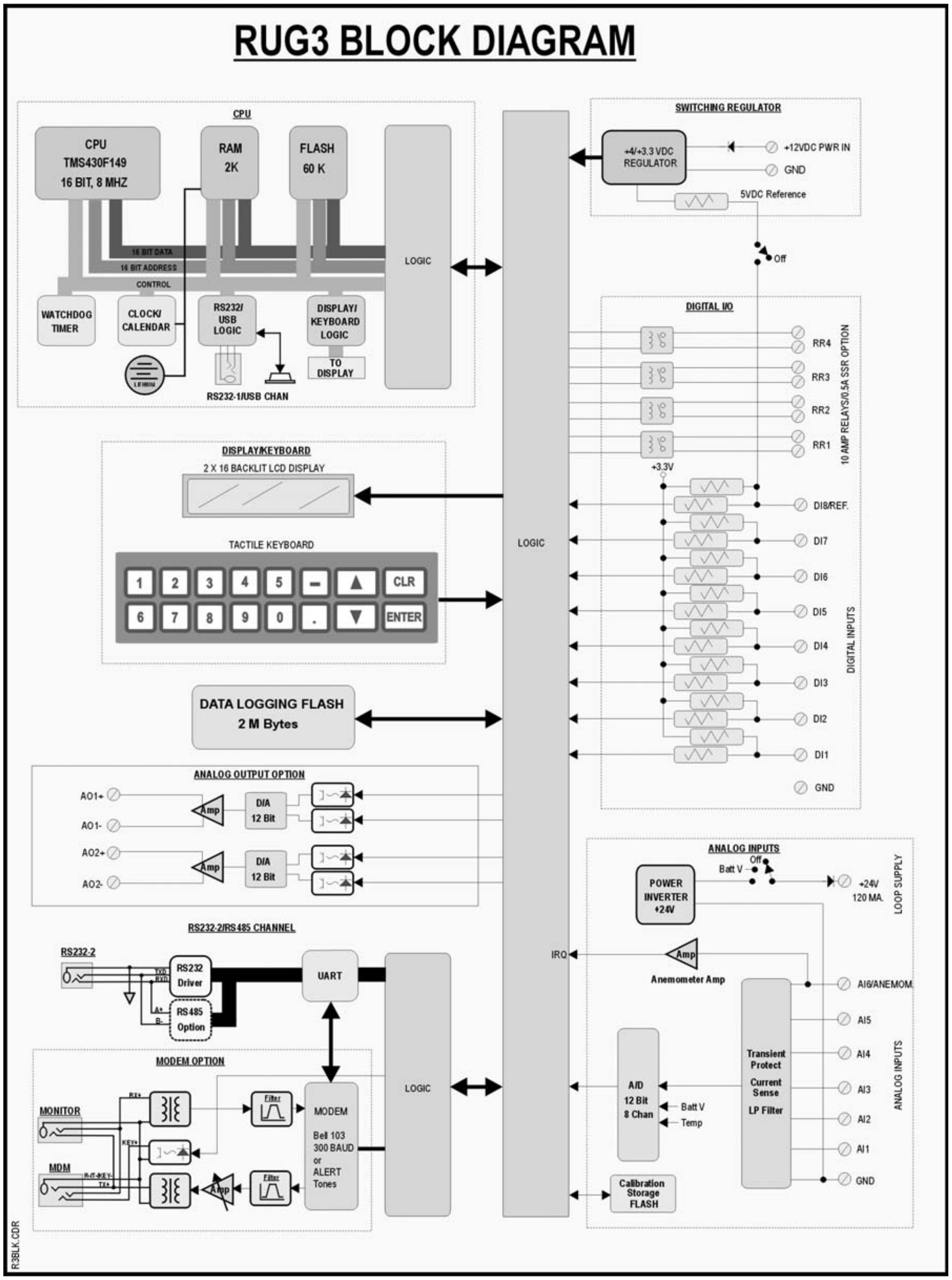


Figure 24 RUG3 Block Diagram

Mounting on DIN Rail

DIN rail mountable units have slots on the right and left sides, and springs installed on the back plate to affect secure mounting on standard 7X35 mm din rail. Assuming you are mounting the unit on a horizontally oriented DIN rail mounted on a vertical backpan, use the following procedure to install on the DIN rail:

- Place the RUG3 on the DIN rail with the bottom of the DIN rail inserted into the bottom of the RUG3's side slots.
- Push up firmly with your thumbs on the bottom face of the RUG3 (where the relay terminals are), while at the same time, pushing the top of the RUG3 toward the backpan with your fingers.
- When you feel the top of the RUG3 snap onto the top of the DIN rail, release the pressure with your thumbs and let the RUG3 slide downward slightly until the DIN rail's top flange is secure in the top of the RUG3's slot.
- Check to make sure that the DIN rail is secure in the slots on both sides of the RUG3.

Removing from DIN rail

To remove the RUG3 from a DIN rail, follow this procedure:

- Push up firmly with your thumbs on the bottom face of the RUG3 (where the relay terminals are), while at the same time, pulling the top of the RUG3 toward you with your fingers.
- When you feel the top of the RUG3 disengage from the DIN rail, let the RUG3 slide down slightly until the RUG3 comes free of the DIN rail's bottom flange.

Applying Power to the RUG3

RUG3 units should be powered from 13.5 VDC+/- 20%. All RUG3 units can be powered from 120 VAC using a standard 12 VDC wall transformer as shown in the figure below. For applications using DC power such as solar power applications, you can apply power to the power terminals as illustrated below. Refer to Table 1 in Chapter 2 for unit total power draw.

Hardware

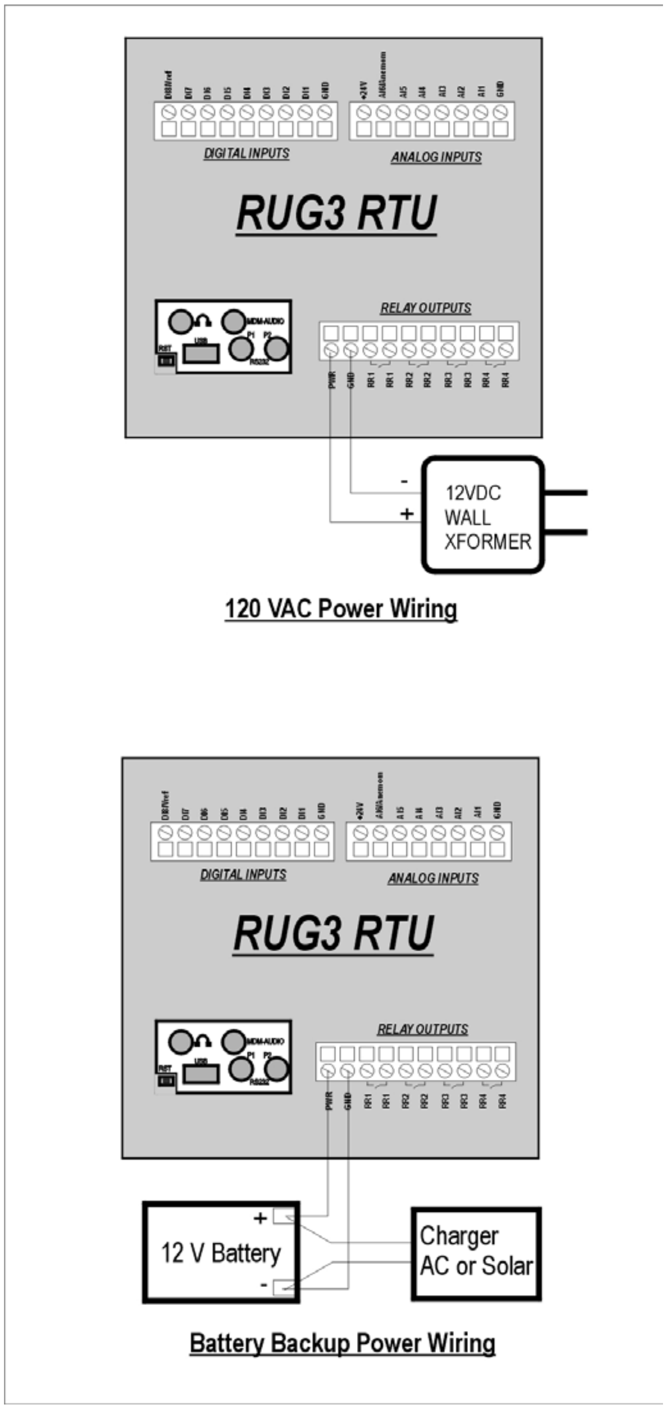


Figure 25 RUG3 Power Wiring Examples

Display/Keyboard Interface

The display is mounted on the circuit board using a 16 pin header and 2 pin backlight connection. The keyboard attaches to the circuit board using an 8 pin ribbon cable. If you must remove the circuit board from the enclosure, the ribbon cable will pull out from the connector on the board. To re-install, lay the keyboard face of the enclosure on a soft surface and place the circuit board face down on the four mounting tabs. Using needle nose pliers or forceps, grab the keyboard ribbon cable and gently force into the keyboard connector. Then install the back mounting plate with four screws.

Reset Button

A reset button is located to the left of the programming port on the bottom face of the unit. Pressing the reset button will stop the program and cause it to reboot. You will have to press the reset button repetitively as many as 15 times to cause the unit to abandon the program and revert to the system menu. To confirm that the program has really stopped, observe the menu on the serial port. If it does not include item #1 to start the program, then the program has been declared faulty and has stopped. The program will restart automatically within 60 seconds.

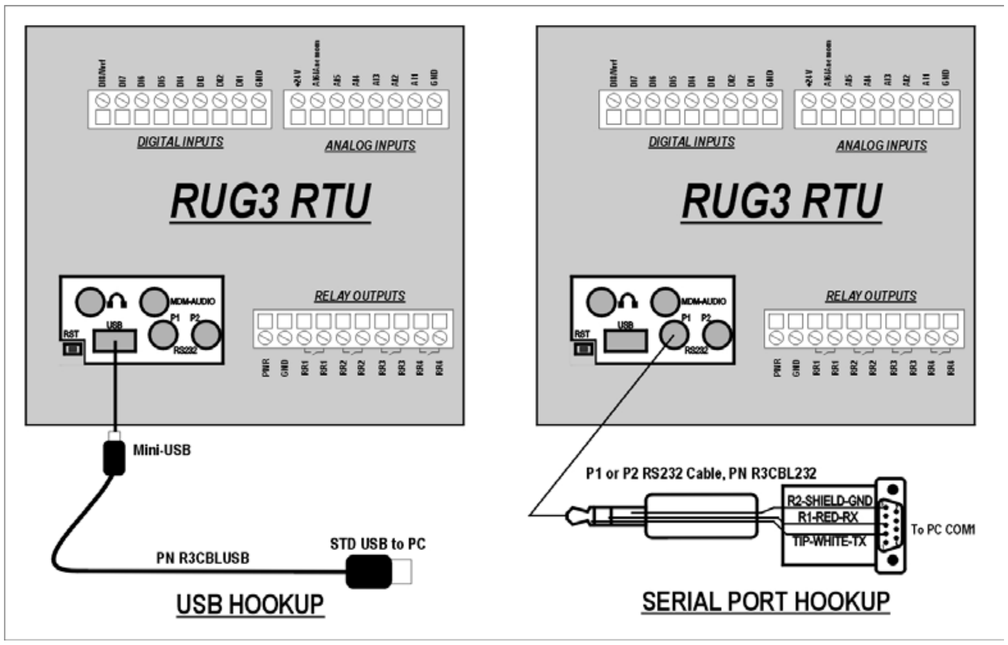


Figure 26 USB and Serial Port Hookup

RS232 Ports

All RUG3's are equipped with two RS232 ports. Port 1, labeled P1 on the back plate, is used for program loading. It is always set for 9600 baud, no parity, 8 bit word, 1 stop bit. The second serial port, labeled P2, is general purpose and has its baud rate and other parameters set by the ComSetup module. For either port, in order to interface to standard PC RS232 ports, you will need a modular to DB9 cable adapter (our part number R3CBL232).

Hardware

USB Port

All RUG3's are equipped with a USB 2.0 port. It enables communication between the PC and the RUG3 as if the PC were connected to RUG3 port 1. It is used primarily for program loading, but can perform all functions of serial port P1including general purpose serial access, watch window access, etc. To use the USB port, you will need a USB to USB-mini cable (our part number R3CBLUSB) and USB drivers for the PC. Install the cable between the RUG3 and your PC, then launch R3SETUP and change the COM port designation to the port used by the USB cable (probably COM4 or COM5). If your PC does not recognize the RUG3 device (CP2102) then you will need to install the USB drivers for the RUG3 as described below. Note that when the USB cable is installed, the RUG3 will disable port 1's RS232 port since both USB and RS232 port 1 use the same UART in the RUG3. **Note: on board revisions 10 and below, be sure to apply power to the RUG3 before plugging in the USB cable; and remove the USB cable before removing power.**

Installing USB Drivers

You can obtain the USB drivers from our web site: www.rugidcomputer.com. Go to the 'Downloads' page and click on 'RUG3 USB Drivers' to download to your PC. The download will appear as 'CP210X_Drivers' on your desktop; or the driver package will open automatically. Follow the instructions for installation on your computer. When installed, the software will redirect communications from the next available COM port to the USB port.

Flash Memory

The RUG3's flash memory does not require power to retain its contents indefinitely. It holds the RUG3 boot loader, operating system, analog calibration installed at the factory and user configuration file (program).

Changing the Battery

You should only need to replace the onboard lithium battery if the unit has been unpowered for an accumulated time of at least a year. To change the battery, remove the RUG3 board from its enclosure; then remove the battery from its holder. The battery holder is just above the reset button on the board. Install a fresh 3 volt CR2032 lithium battery in the holder with the positive terminal up. Re-install the RUG3 board into the steel case, if any. Power the unit up to confirm that handling the board has not corrupted any memory contents or time/date setting. If necessary, set the time/date. Power the unit down for 10 minutes and then reapply power. If the realtime clock shows the correct time, then the new battery is working correctly.

Removing the Board from its Case

To remove the RUG3 board from its case, lay the keyboard face of the enclosure on a soft surface and remove the four Philips screws holding the back plate in place. Remove the back plate and then firmly pull up on the board to remove it from the case. This will also pull out the keyboard ribbon cable from its connector but will not harm it.

<u>**Hardware**</u>

Installing the Board into the Case

To install the RUG3 board into its case, lay the keyboard face of the enclosure on a soft surface and place the circuit board face down on the four mounting tabs that correspond to the four holes in the corners of the board. Using needle nose pliers or forceps, grab the keyboard ribbon cable and firmly force it into the keyboard connector. Take care not to bend the keyboard ribbon cable sharply near the connector or you could crack one or more silver traces and render the keyboard inoperative. Then install the back plate with four screws.

Loop Supply

The RUG3 has a power inverter that boosts the unregulated 12 V bus to a regulated 24 VDC for powering analog loops. It is capable of delivering up to 120 ma. Its screw terminals are located on the analog input connector. Its most negative terminal is labeled GND; its most positive terminal is labeled +24. The loop supply can be turned on or off by controlling the loop supply control bit in the SysSetup module. It can also be attached to the unit's unregulated, nominal 12 VDC power supply by controlling another bit in the SysSetup module. The purpose of these controls is to enable you to use the loop supply to control power to external instruments. IMPORTANT…if power supply voltage to the unit falls below nominally 10 volts, the loop supply inverter that boosts voltage to 24 volts will be shut off to protect against the unit drawing excessive power in its attempt to maintain 24 VDC output. The loop supply will be re-engaged when input power supply voltage rises above 11 VDC.

Digital Inputs

The RUG3 provides eight digital inputs. Digital inputs are self-powered and are compatible with dry contacts or with logic inputs in the range of 0-3V to 0-12 VDC referenced to the RUG3 GND pins. Normal excitation voltage is 4 VDC. Trip threshold is nominally 1.5 V above ground. Each digital input channel can serve any of several functions depending upon the module you use to bring the digital input into the system as defined in the following table. Note that to read a shaft encoder requires two digital inputs per encoder.

Table 3 Digital Input Function Choices

APPLICATION TYPE	MODULE TO USE
Dry contact sensing	DigitalInputDC
Counting up to 110 counts per second	DiginCount
Measuring pulse durations	PulseDurationIn
Reading shaft encoder	ShaftEncoderInput

The figure below presents the proper method to connect contacts to the RUG3.

Hardware

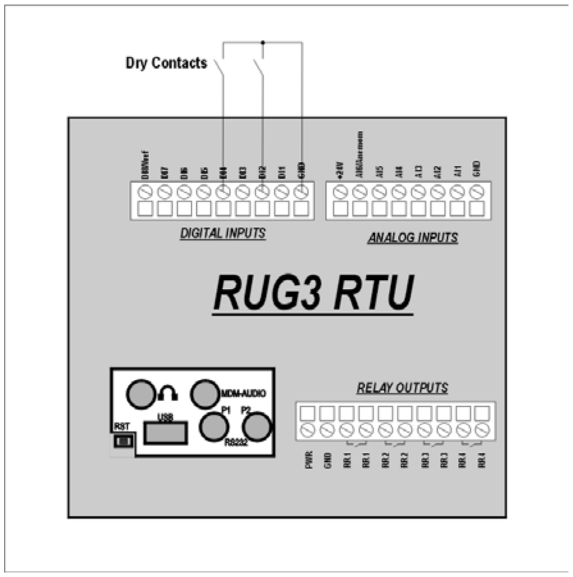


Figure 27 DI Hookup Example

Relay Outputs

The RUG3 provides 4 channels of 10 amp relay outputs that are compatible with both 120VAC and 30 VDC (MR option) or 4 channels of solid state relay outputs that are compatible with 48VDC/0.5 amp (SR option). Each is a Form A relay, completely isolated from any other connections on the RUG3. The relays provide 1500 volt isolation between the loads and the RUG3 bus, and have large coil to conductor spacing to minimize transient to coil coupling. Depending upon the control module you choose, the relays can be used in several different ways as defined in the following table:

Table 4 Digital Output Function Choices

RELAY OUTPUT FUNCTION	MODULE TO USE
On/OFF control of AC or DC load	DigitalOutput
Flashing alarm output	DigitalAlarmOutput
Pulse duration output	PulseDurationOut

Below is a simple hookup example.

Hardware

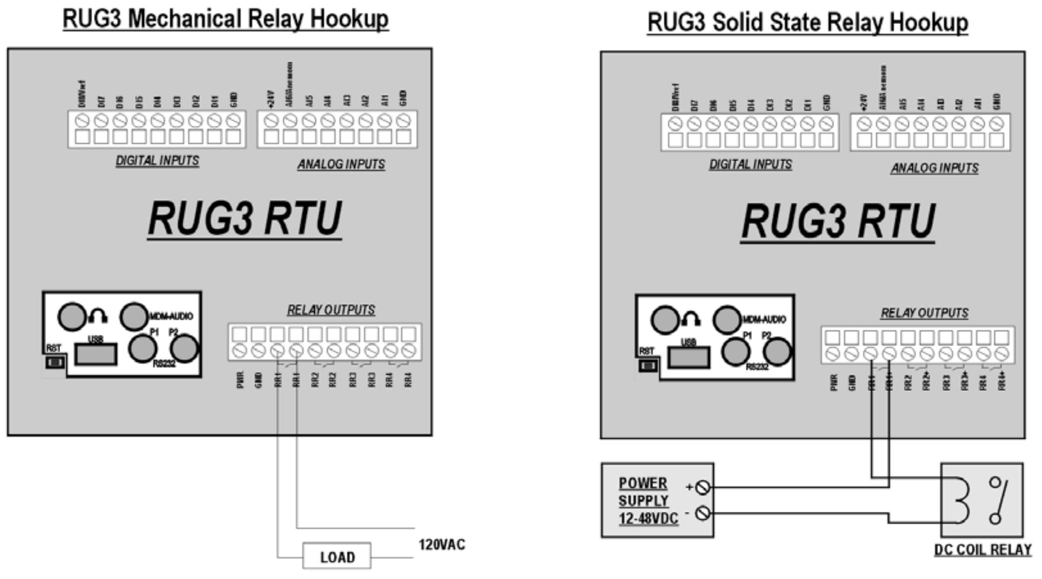


Figure 28 Relay Hookup Example

Analog Inputs

The RUG3 standard analog inputs use a single eight channel 12 bit A/D converter to convert analog voltages to digital form. Six A/D channels provide accurate analog measurements of external signals in the range of 0 to 5 volts or 4-20 ma. An **AnalogInput** software module enables you to select 0 to 5 volts or 4-20 ma operation for each channel under software control. It also includes offset, span, low pass filtering and range limiting entries. The remaining two channels measure onboard temperature and battery voltage. These measurements are available as outputs of the **SysSetup** software module. The board has onboard precision 221 ohm current sensing resistors that can be engaged by selecting 4-20 ma type channel in the **AnalogInput** software module to make the selected channel compatible with the 4-20 ma. instrument standard. Channel calibration for both 0-5V and 4-20 ma operation is held in onboard flash memory whose contents are set at the factory. The RUG3 uses the calibration constants to linearize and scale the channel values in the preprogrammed modules. The figure below presents the proper connection of a 4-20 ma. transducer. Additional transducers can be connected by connecting their positive terminals to the 24V positive terminal and their negative terminals to individual analog input terminals. Be sure to observe the loop supply's capacity of 120 ma total current, or a maximum of six analog 4-20 ma loops per board.

Hardware

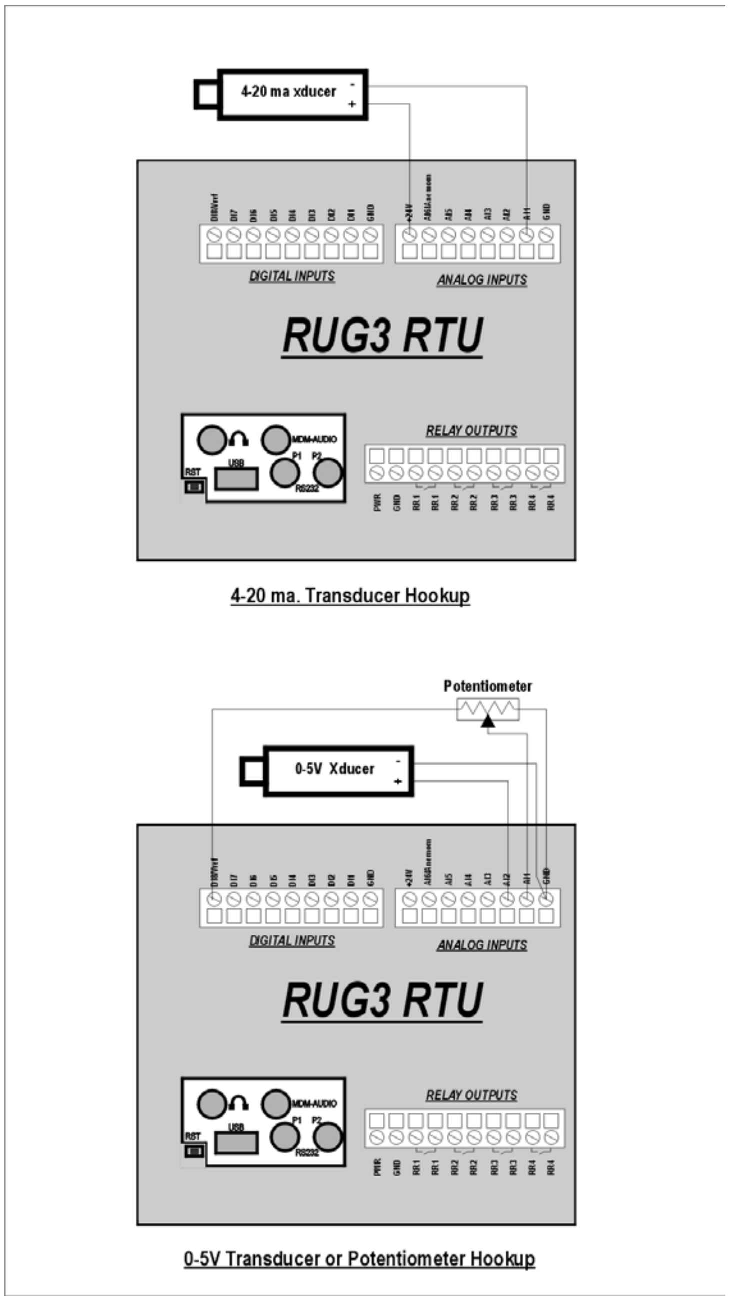


Figure 29 Transducer Hookup to Analog Inputs

Voltage type transducers and potentiometers can be connected also as shown above. For each transducer you must install a separate analog input software module and designate type of channel (4-20 ma or 0-5V) as well as engineering units scale factors, filter time constants, etc. You can mix 4-20 ma type transducers with voltage type transducers without limitation on a single RUG3 simply by choosing the proper connection (4-20 ma.or 0-5V) in each analog input software module.

Modem/RS232/RS485 Channel

For audio communications, the optional modem provides Bell 103, 300 baud capability compatible with 2-wire or 4-wire wireline channels, or audio radios. An onboard software-adjustable transmit channel amplifier makes the board able to communicate with leased lines or customer owned lines. An optoisolator provides transmitter keying control of audio radio transmit function. The optoisolator and 600 ohm transformers completely isolate the board from field circuits. The keying circuit employs the KEY terminal and the TX- terminal. For high speed communications, the RS232 port can be connected to radio modems, external high speed phone line modems, spread spectrum radios, or Ethernet to serial converters. The transmit amplitude, mode, baud rate, tone use, protocol, etc., are set using the **ComSetup** software module. Transmit amplitude is adjustable over the range of 0 to approximately 3 V p-p into 600 ohms using inputs of 0 to 255 on the **ComSetup** module. Modem baud rate is 300 baud only and must use what we refer to as the low tones mode where the RUG3 uses only the lower tone pair of the Bell 103 standard for both transmit and receive functions. This assures that all units can hear all other units. RS232 baud rates are user-definable over the range of 50 to 19,200 baud. The diagram below presents the modem and RS232 connections and necessary cables to connect to them. The headphone jack is compatible with common stereo headphones enabling you to listen to the transmit audio in one ear and the receive audio in the other.

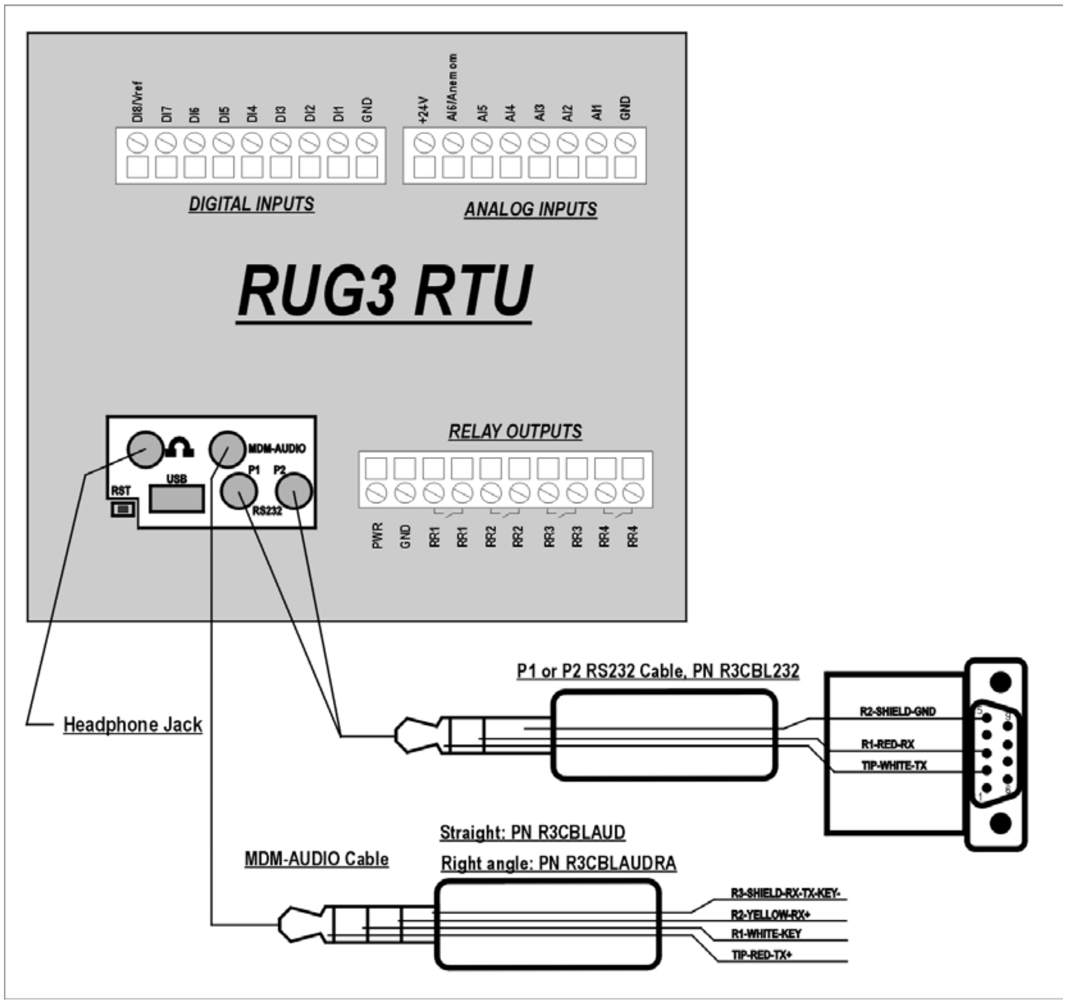


Figure 30 Modem/RS232 Channel Connections

Analog Outputs (Optional)

The RUG3 can be factory equipped with a two channel, 12 bit resolution analog output board. Each channel is optically isolated from all other signals. Each channel functions as if it were an unpowered two wire 4-20 ma. transducer. Conversion from engineering units to 4-20 ma. is performed by the **AnalogOutput** software module for each channel which uses calibration installed into flash memory during factory test. Connections to the analog outputs are made to screw terminals on a 4 pin removable screw header on the right side of the unit as illustrated below. Note that each analog output is loop powered and requires at least 8.5 volts across its output for accurate operation. Maximum allowed forward voltage across each output is 50 VDC. A blocking diode will prevent damage to the channel from application of reverse voltage of up to 200 VDC. The figure below presents an example of how to connect the two analog outputs to a pair of variable frequency drives (VFD's) using an external loop supply.

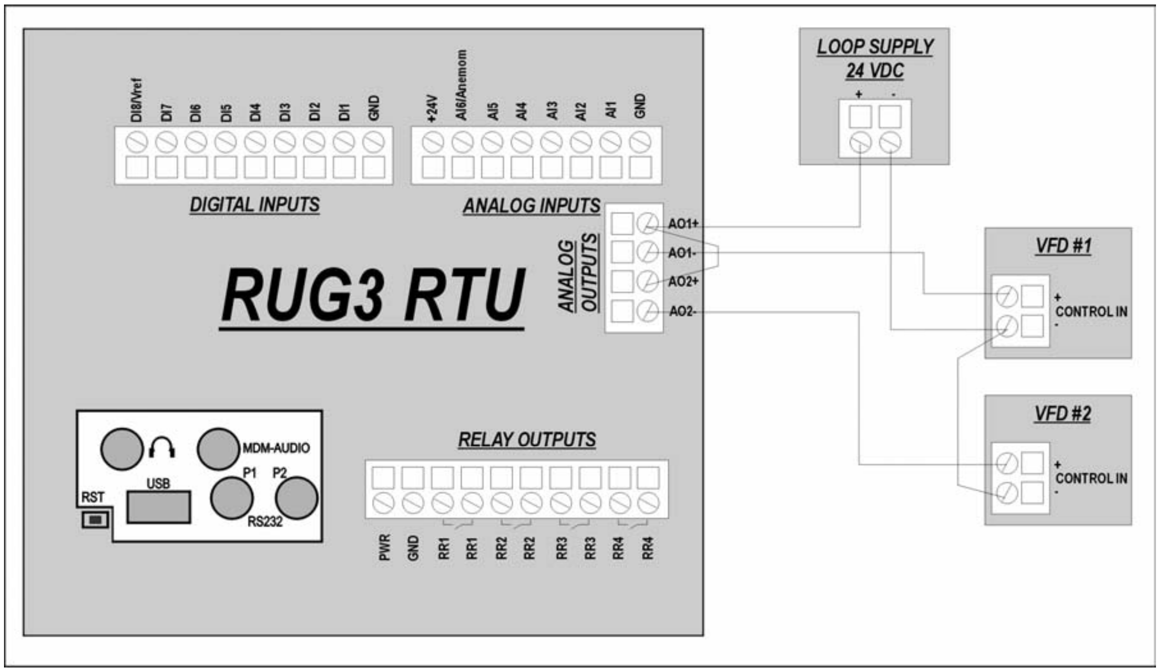


Figure 31 Analog Output Connections

Hardware

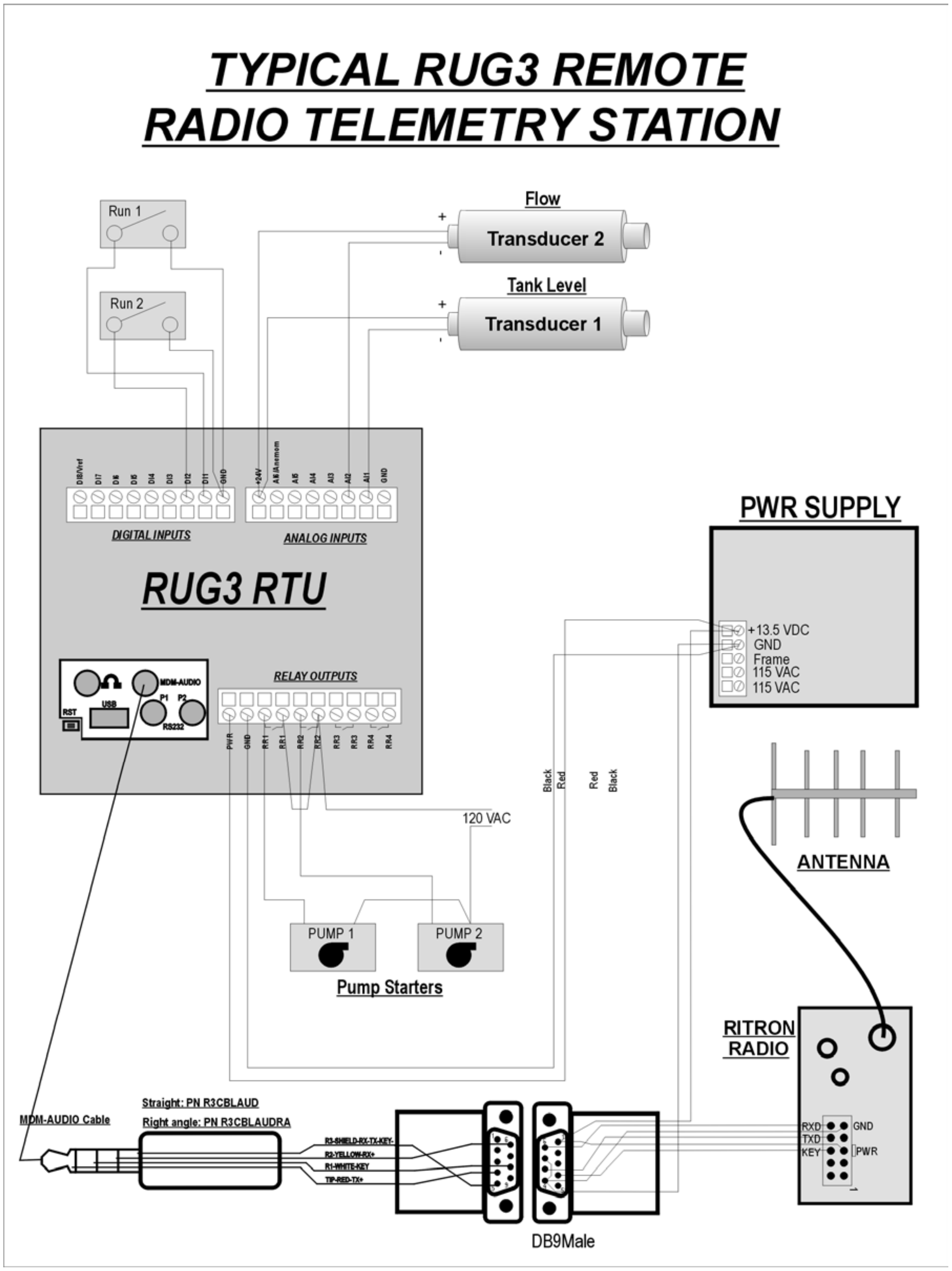


Figure 32 Typical RUG3 Remote Radio Application

Hardware

CHAPTER 4…USING RUG3 SUPPORT SOFTWARE

Introduction

Unlike earlier RUGID products that you program using the procedural BASIC language, in the case of the RUG3, you configure and interconnect preprogrammed modules using the R3SETUP program that runs under Windows. Preprogrammed modules are nothing more than software function blocks that perform specific functions based on inputs that you specify. After you complete your configuration, you save it and then press the 'Send To R3' button to send it to your RUG3 unit. The code for each module is sent to the RUG3 when you send the program , so as soon as the configuration is successfully loaded into the RUG3, it will begin executing modules as established by your configuration file. Modules will execute in alphabetical order, but since the program cycles about a 100 times per second, for all practical purposes, you can assume that all modules are executed simultaneously. Once execution starts, it can only be stopped by hitting the recessed 'Reset' key on the front of the RUG3's board next to the modular programming port, or by sending a new program to the RUG3.

Procedure for Setting Up a Project

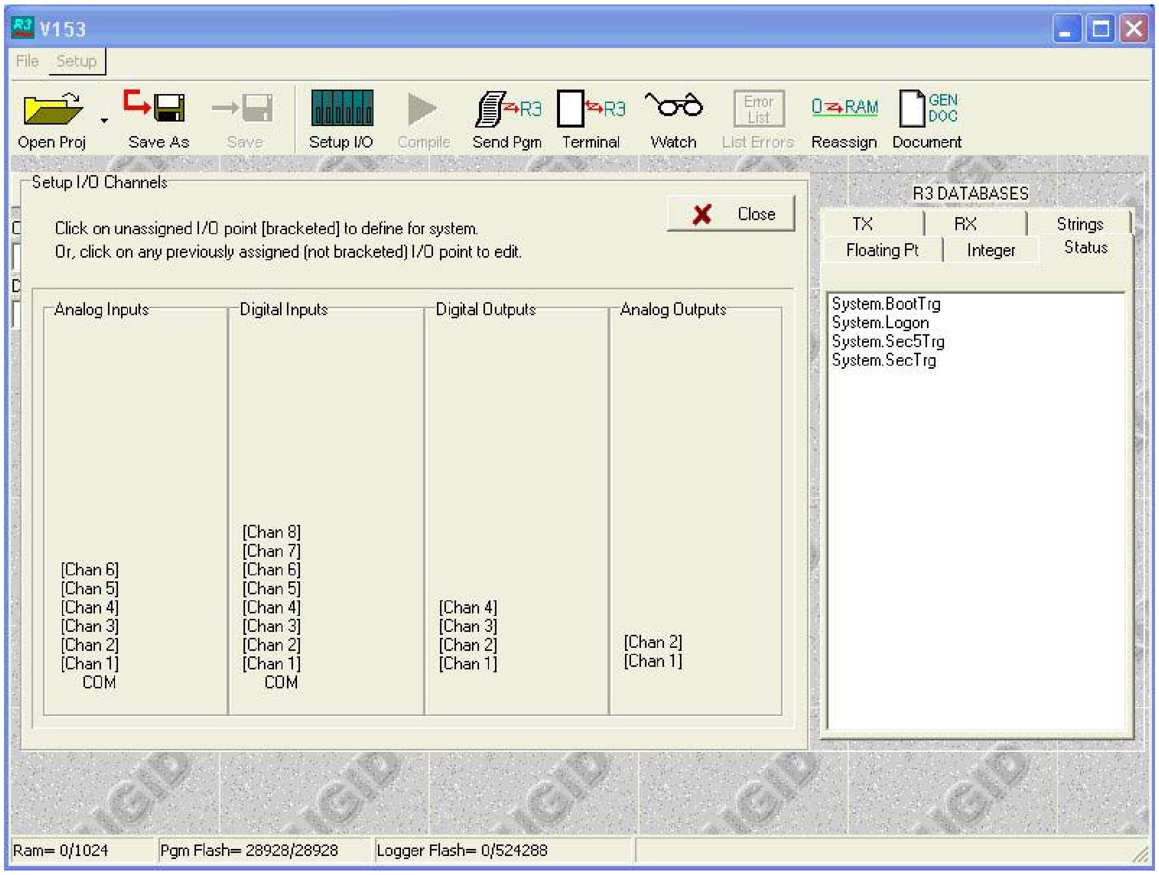
You can enter modules in any order you want, but we find it most convenient to follow the procedure below:

1) Configure I/O modules
2) Install setpoint modules
3) Set up receive telemetry arrays
4) Configure math calculations
5) Design control strategies
6) Configure statistics (totalizations, data logging, running times, minima, maxima, etc.)
7) Set up transmit telemetry arrays
8) Design displays and reports

This will generally assure that inputs to modules are established before they are needed. However, there will be instances where all inputs will not have been established before a module is defined. In that case, simply save the module without the input specified, and return to it later to finish it.

Starting R3SETUP Design Environment

To configure the RUG3, you must use the R3SETUP program included with your unit on request and available at no charge from RUGID's web site, www.rugidcomputer.com. Refer to the tutorial section, chapter 2, if you have not yet installed this software. Once installed, to start the software, either click on the R3 icon on your desktop; or click on the **START** button at the lower left of your Windows screen; select **PROGRAMS**; then select **R3SETUP** from the list of programs installed on your computer. The program should start and display the screen presented in the next figure. Note that when the program first launches it presents a panel labeled 'Setup I/O Channels', presuming that you will wish to specify I/O channels before anything else.



Click on unassigned I/O point [bracketed] to define for system.
Or, click on any previously assigned (not bracketed) I/O point to edit.

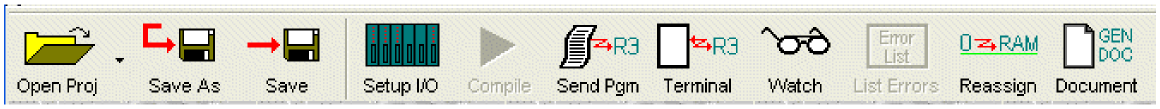
Analog Inputs	Digital Inputs	Digital Outputs	Analog Outputs
	[Chan 8]		
	[Chan 7]		
[Chan 6]	[Chan 6]		
[Chan 5]	[Chan 5]		
[Chan 4]	[Chan 4]	[Chan 4]	
[Chan 3]	[Chan 3]	[Chan 3]	
[Chan 2]	[Chan 2]	[Chan 2]	[Chan 2]
[Chan 1]	[Chan 1]	[Chan 1]	[Chan 1]
COM	COM		

System.BootTrg
System.Logon
System.Sec5Trg
System.SecTrg

Ram= 0/1024 | Pgm Flash= 28928/28928 | Logger Flash= 0/524288

Figure 33 R3SETUP Opening Panel

Tool Bar



The tool bar at the top of the screen gives you access to major compiler functions. Any that are inappropriate or unnecessary will be faded out and inoperative. Tool bar functions are listed below:

Open Project: Provides a list of the last 10 projects you accessed for quick selection. Also includes "New" to start a new project; and "Open" to open a dialog box giving you access to all projects.

SaveAs: Saves the current project with a new name/location.

Save: Saves the current project with existing name and location.

Setup I/O: Presents the I/O representation so you can access individual I/O points.

Compile: Causes the compiler to process your configuration, looking for errors.

SendPgm: Compiles your file, opens the terminal panel, and sends your file to an attached RUG3.

Terminal: Opens the terminal panel from which you can communicate with the RUG3, send a file, reload the RUG3 operating system, set the RUG3's realtime clock, etc.

Watch: Opens the watch window which you can use to observe database items as the RUG3 is running its program.

List Errors: Shows the results of the latest compilation.

Reassign: Clears RAM assignments made to modules, arrays, etc. so the compiler can reassign RAM. The compiler tries to avoid reassigning RAM used by modules so that setpoint values, totalizations, etc. are not corrupted when you change your configuration file. If the compiler cannot find enough RAM or if a RAM overlap is detected, it will prompt you to click the **Reassign** button.

Document: Causes the compiler to generate a descriptive ASCII text file of your project with the same name as your project except with a .Txt extension. It is stored in the **Projct** folder.

Loading an Existing Project

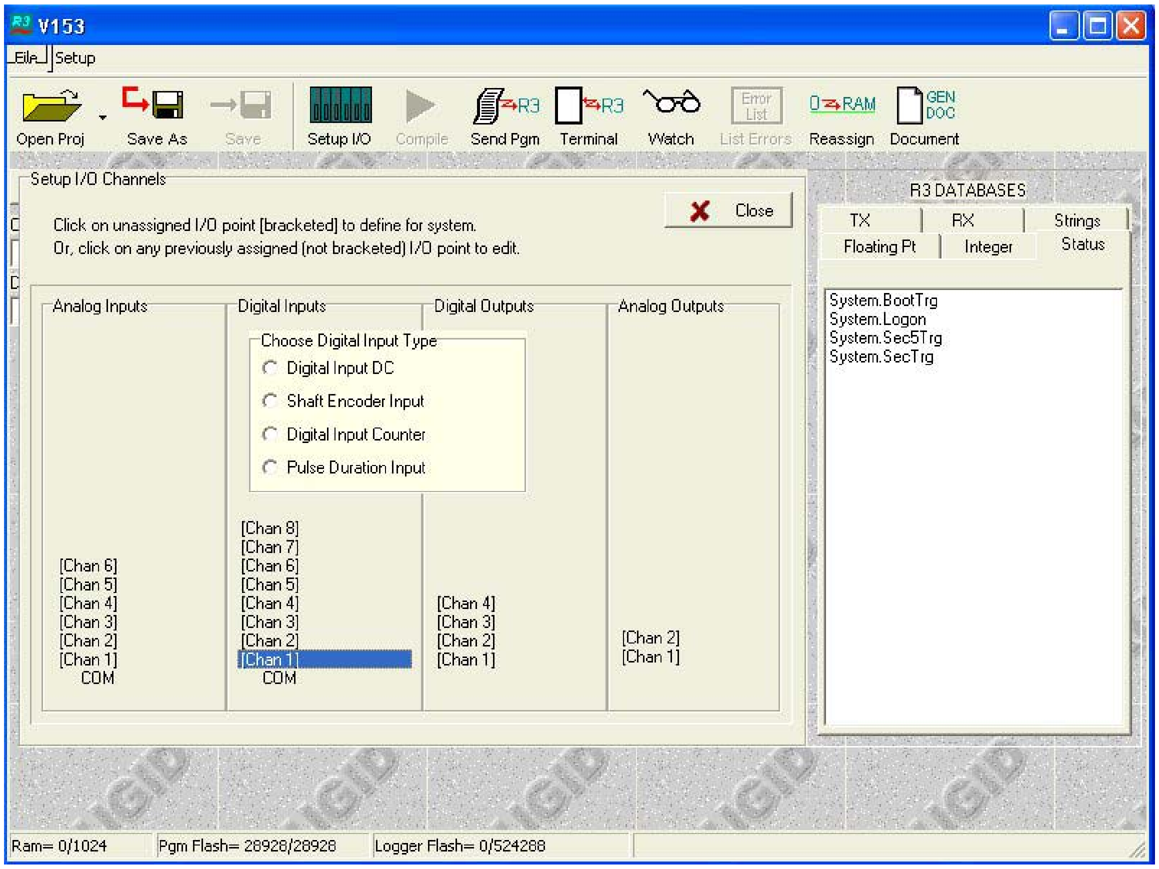
If you wish to load an existing project, click on the **Open Proj** button at the left end of the tool bar. That will open a file dialog box from which you can navigate to the folder you wish and select a file. If you click on the narrow vertical bar portion at the right of that button that has a small down arrow on it, the system will present you with a menu of the last 10 projects you worked on along with **Open** and **New** options. The **Open** item will open the file dialog; the **New** item will delete any project presently installed and prepare for designing a new project.

Saving a Project

To save your project, you can click the **Save** button to save with the same name and folder from which the file was originally read. If you have not already named the project, then click the **Save As** button to save the project with a new name.

Configuring I/O Modules

After you have decided on your I/O points, you should identify each I/O point that you intend to use and name it. You start that process by selecting an I/O point from the I/O panel. If the panel labeled 'Setup I/O Channels' is not visible in the center of your screen, click on the 'Setup I/O' button on the toolbar. To illustrate by example, as presented in the following figure we have selected the first digital input. The RUG3 has several ways in which to use digital inputs, so the panel presents us with those choices in a small sub-panel seen near the middle of the screen. (For digital inputs, the choices are: Digital Input DC, Shaft Encoder Input, Digital Input Counter, and Pulse Duration Input.)



Setup I/O Channels

Click on unassigned I/O point [bracketed] to define for system.
Or, click on any previously assigned (not bracketed) I/O point to edit.

Analog Inputs: [Chan 6] [Chan 5] [Chan 4] [Chan 3] [Chan 2] [Chan 1] COM

Digital Inputs: [Chan 8] [Chan 7] [Chan 6] [Chan 5] [Chan 4] [Chan 3] [Chan 2] [Chan 1] COM

Choose Digital Input Type
- Digital Input DC
- Shaft Encoder Input
- Digital Input Counter
- Pulse Duration Input

Digital Outputs: [Chan 4] [Chan 3] [Chan 2] [Chan 1]

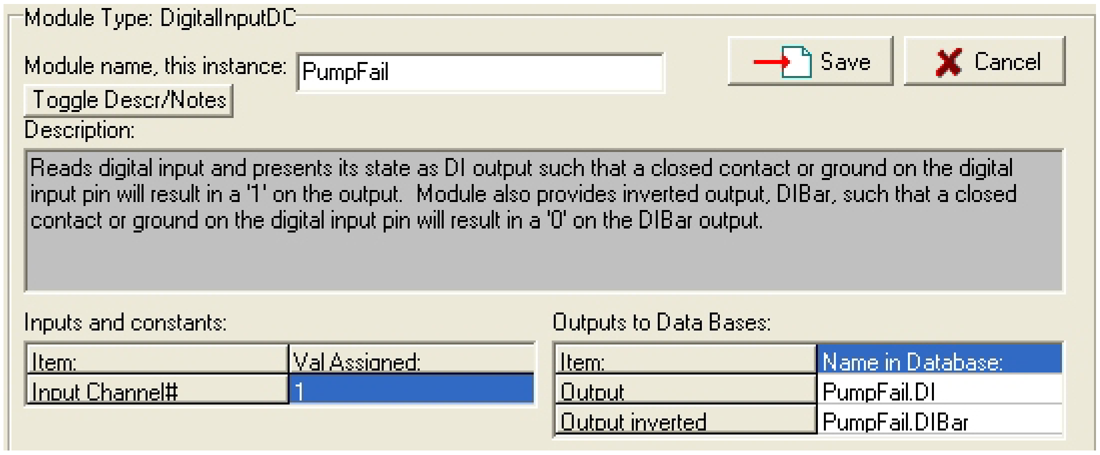
Analog Outputs: [Chan 2] [Chan 1]

System.BootTrg
System.Logon
System.Sec5Trg
System.SecTrg

Ram= 0/1024 Pgm Flash= 28928/28928 Logger Flash= 0/524288

Figure 34 Selecting I/O Type

Once you select a type of digital input, you will be presented with the module configuration screen, where you will name the module and set some or all of its input properties. The configuration screen for the DC type digital input is presented below.



Module Type: DigitalInputDC

Module name, this instance: PumpFail

Description:

Reads digital input and presents its state as DI output such that a closed contact or ground on the digital input pin will result in a '1' on the output. Module also provides inverted output, DIBar, such that a closed contact or ground on the digital input pin will result in a '0' on the DIBar output.

Inputs and constants:

Item:	Val Assigned:
Input Channel#	1

Outputs to Data Bases:

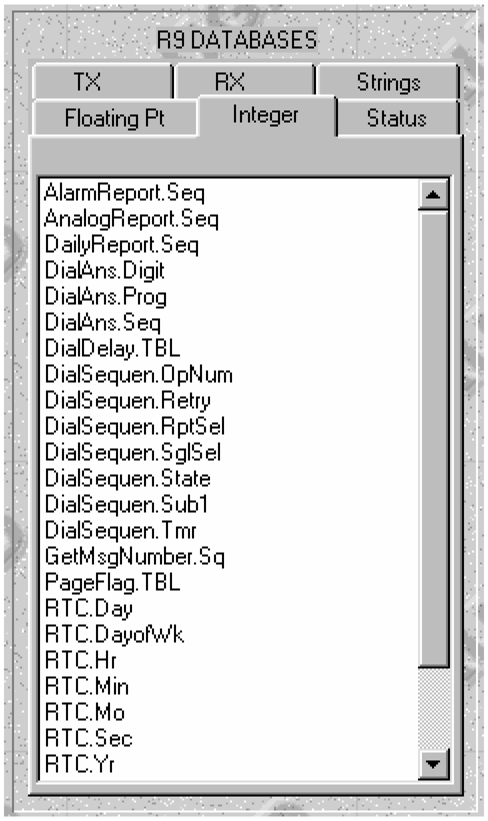
Item:	Name in Database:
Output	PumpFail.DI
Output inverted	PumpFail.DIBar

Figure 35 Digital Input Configuration Panel

Your cursor will rest in the module name edit box where you will type in your name for this module. As you do, you will notice that each of the module's outputs, on the right side of the panel, is given the name you type in followed by an extension that is different for each output, and suggests the output's function. The digital input module above, which has already been named **PumpFail**, has two outputs. The *.DI output basically follows the state of the external digital input. The *.DIBar output is the inverse of the actual digital input. For the digital input module presented above, the input channel number has already been filled in by the system, so you can click the **Save** button to save this module to the project. If you click **Cancel** instead, the module will be abandoned. Most modules have several input properties that you can fill in or leave blank if they are not needed. Once you save the module to the project, its outputs are saved to one or more of the databases using the name you gave the module followed by the individual output extensions. Once a module's output is installed in a database, it constitutes a signal that can be used as the input to any module, including the module that generates it. After you save this module, you can select another I/O point to configure until you have configured all you intend to use. If you wish to record your own notes regarding the function of a module in your project you can do so by first clicking the 'Toggle Descr/Notes' button to bring up the user notes panel in place of our description. You can then type in your own text which will be saved with the module within your project. As you save each module, its point on the I/O representation will be displayed with the name you gave the module. In some cases, particularly in the case of relay outputs, you will probably configure them before you have established the signals that will actually drive them, since they will probably be driven by the outputs of some control strategy. In those cases, simply configure the modules to the extent you can and leave blank any input properties that are not yet established in a database. You can return to them later. When you are done configuring I/O points on the cards, click the Close button to close the I/O panel. You can return to it at any time by clicking on the card cage button on the upper tool bar.

Data Bases

The compiler maintains six databases as illustrated below. They reside on the right side of the main project screen. The databases contain the project's signals, which are generated by modules, by ladder logic, or as the result of a reception on a telemetry channel. Basically, the databases constitute the repository of the outputs of the project. You can make visible any one database by clicking on its tab at the top. In the figure below, the integer database is visible and contains a number of signals, which are alphabetized for easy reference. When you are configuring a module, display, etc., and it needs a signal from a database, simply drag the signal from the database and drop it in the module's input cell. Note that you cannot drag items from the TX database since those entries are actually outputs of other modules…use the outputs of the same name where it resides in another database.

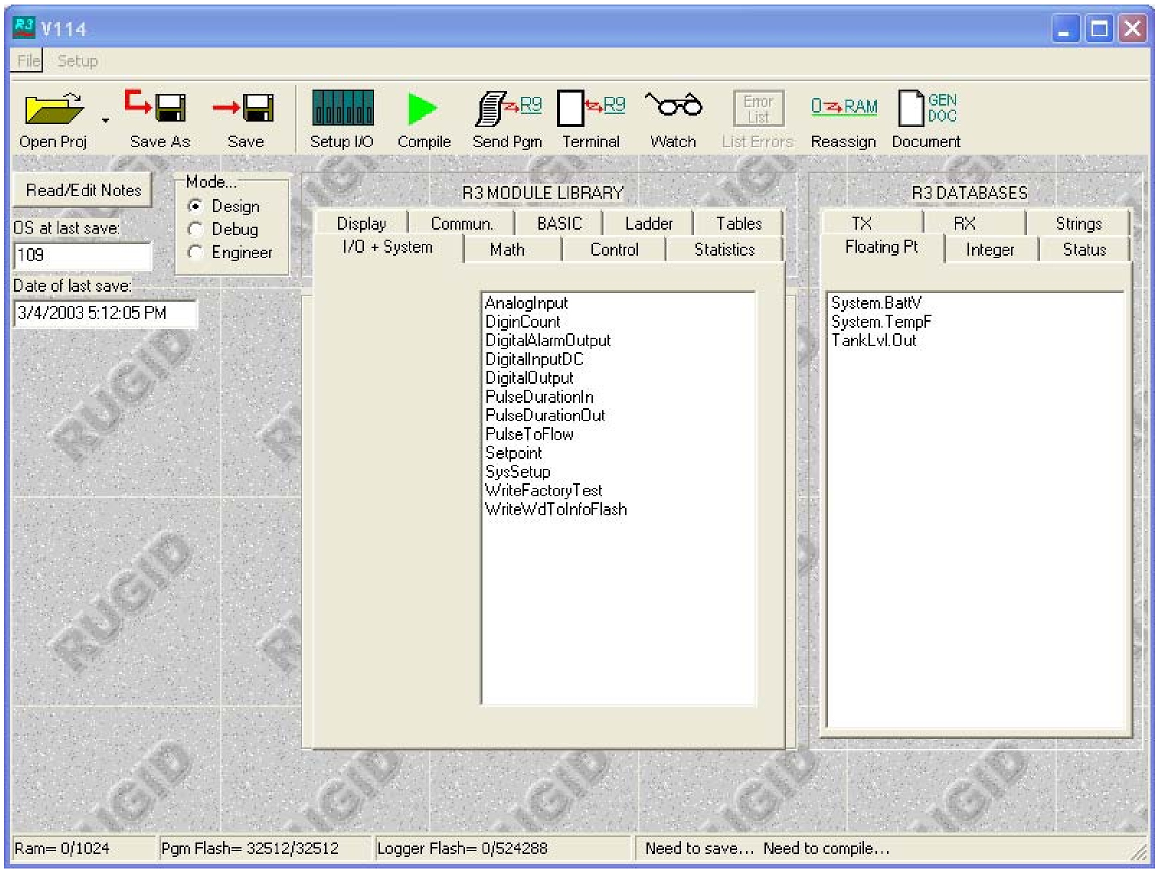


AlarmReport.Seq
AnalogReport.Seq
DailyReport.Seq
DialAns.Digit
DialAns.Prog
DialAns.Seq
DialDelay.TBL
DialSequen.OpNum
DialSequen.Retry
DialSequen.RptSel
DialSequen.SglSel
DialSequen.State
DialSequen.Sub1
DialSequen.Tmr
GetMsgNumber.Sq
PageFlag.TBL
RTC.Day
RTC.DayofWk
RTC.Hr
RTC.Min
RTC.Mo
RTC.Sec
RTC.Yr

Figure 36 Databases Showing Integer Database

Adding a Module to a Project

You can add a module to a project by clicking on one of the tabs in the 'R3 Module Library', to pull down the module list for that category. You would then click on one of the modules in the list to begin the process of including it in your project. The figure below presents the I/O category module list.



Mode: [✓ Design] Debug Engineer

OS at last save: 109

Date of last save: 3/4/2003 5:12:05 PM

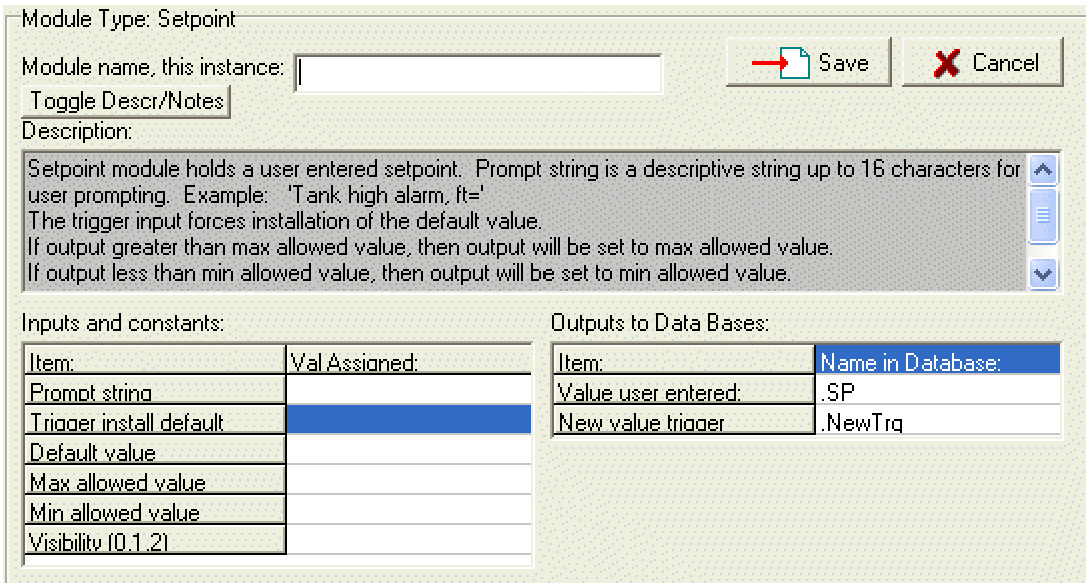
AnalogInput
DiginCount
DigitalAlarmOutput
DigitalInputDC
DigitalOutput
PulseDurationIn
PulseDurationOut
PulseToFlow
Setpoint
SysSetup
WriteFactoryTest
WriteWdToInfoFlash

System.BattV
System.TempF
TankLvl.Out

Ram= 0/1024 | Pgm Flash= 32512/32512 | Logger Flash= 0/524288 | Need to save... Need to compile...

Figure 37 Selecting Module From Module Library

Clicking on a module in the list will bring up the module configuration panel specific for the module you have selected (in this case, the setpoint module) as shown below:



Module Type: Setpoint

Module name, this instance:

Description:

Setpoint module holds a user entered setpoint. Prompt string is a descriptive string up to 16 characters for user prompting. Example: 'Tank high alarm, ft='
The trigger input forces installation of the default value.
If output greater than max allowed value, then output will be set to max allowed value.
If output less than min allowed value, then output will be set to min allowed value.

Inputs and constants:

Item:	Val Assigned:
Prompt string	
Trigger install default	
Default value	
Max allowed value	
Min allowed value	
Visibility (0,1,2)	

Outputs to Data Bases:

Item:	Name in Database:
Value user entered:	.SP
New value trigger	.NewTrg

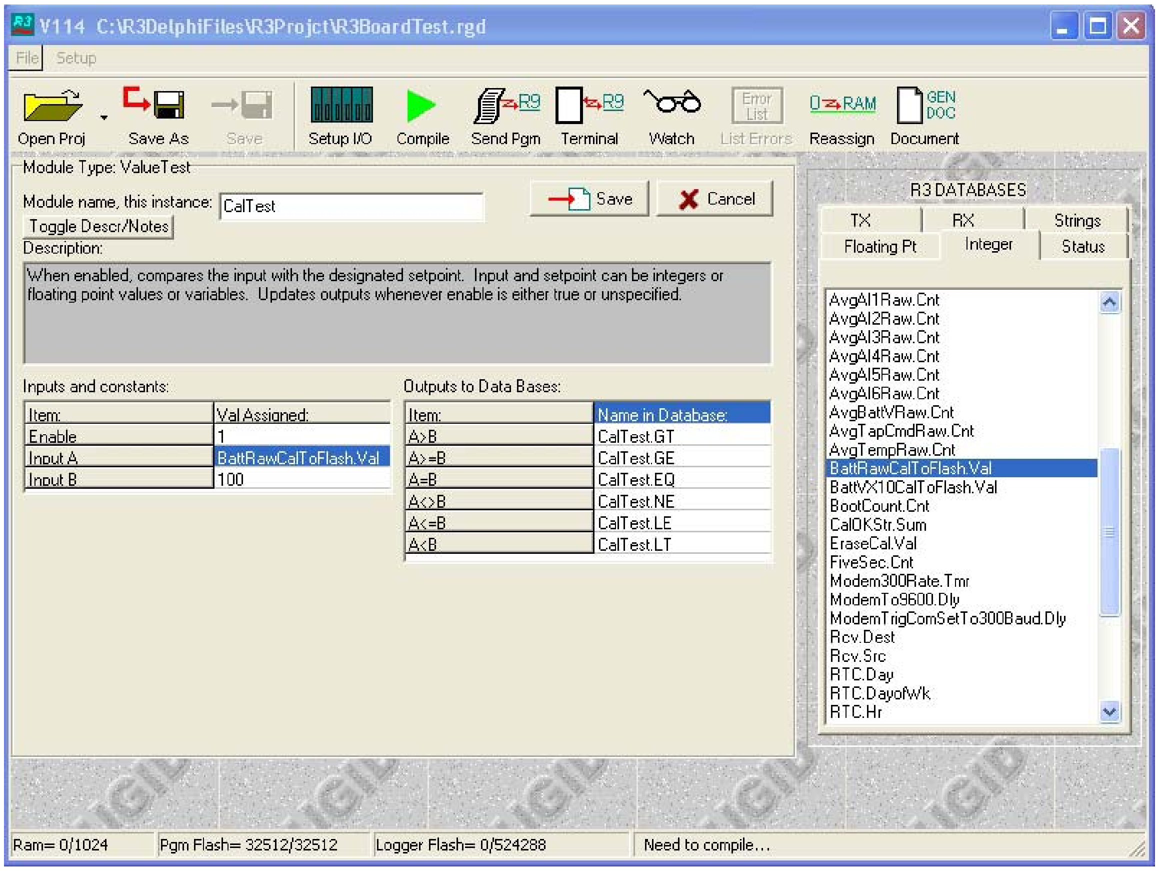
Figure 38 Setpoint Module Configuration Panel

Naming a Module

Once you have a module's editing page present as in the case above, it is important to enter a name for the module. The name must be less than 20 characters long and must be unique, or when you attempt to save the module, the R3SETUP system will complain that the name already exists. You should choose a name that suggests to you the function or signal being processed by the module, since that name will be the name given to all outputs of the module which then appear in the data bases for use by other modules. Also, the name is your reference for accessing the module later to make changes. For example, for analog inputs, you might choose names such as 'TankLvl', 'StreamFlow', 'Pump1Temp', etc. For relay outputs, such names as 'Pmp1CallRly', 'HiAlrmRly', or 'SandRly' suggest their functions. Also, names may contain blank characters and numerals for clarity.

Connecting Modules Together

You connect one module to another by connecting the output of the first module to an input of the second. Since all module outputs are placed in the data bases, visible to the right of the page above, the process of connecting an output to a module's input amounts to finding the signal you want in one of the databases, and then dragging it over to the module you are working on, and dropping it into one of the input cells. For example, in the screen shown below, the signal '**BattRawCalToFlash.Val'** is established as the input to the pictured module by dragging it from the integer database to the right and dropping it into the Input A property of the ValueTest module shown:



V114 C:\R3DelphiFiles\R3Projct\R3BoardTest.rgd

Module Type: ValueTest

Module name, this instance: CalTest

Description:

When enabled, compares the input with the designated setpoint. Input and setpoint can be integers or floating point values or variables. Updates outputs whenever enable is either true or unspecified.

Inputs and constants:

Item:	Val Assigned:
Enable	1
Input A	BattRawCalToFlash.Val
Input B	100

Outputs to Data Bases:

Item:	Name in Database:
A>B	CalTest.GT
A>=B	CalTest.GE
A=B	CalTest.EQ
A<>B	CalTest.NE
A<=B	CalTest.LE
A<B	CalTest.LT

AvgAI1Raw.Cnt
AvgAI2Raw.Cnt
AvgAI3Raw.Cnt
AvgAI4Raw.Cnt
AvgAI5Raw.Cnt
AvgAI6Raw.Cnt
AvgBattVRaw.Cnt
AvgTapCmdRaw.Cnt
AvgTempRaw.Cnt
BattRawCalToFlash.Val
BattVX10CalToFlash.Val
BootCount.Cnt
CalOKStr.Sum
EraseCal.Val
FiveSec.Cnt
Modem300Rate.Tmr
ModemTo9600.Dly
ModemTrigComSetTo300Baud.Dly
Rcv.Dest
Rcv.Src
RTC.Day
RTC.DayofWk
RTC.Hr

Ram= 0/1024 | Pgm Flash= 32512/32512 | Logger Flash= 0/524288 | Need to compile...

Figure 39 Dragging a Database Item Into a Cell

In general, the compiler will take care of converting the data base signal type (floating point, integer, status, etc.) to the type expected by the module into which it is dragged. However, four limitations exist:

-Inputs that specify channel number must be integer constants (you type them in).
-Inputs that call for the number of bytes in an output string must be integer constants (you type them in).
-Inputs that expect numbers or statuses must not be given entries from the string database. Inputs that expect strings will say so in their input prompts.
-Inputs that expect strings must not be given inputs dragged from any database other than the string database.

Module Inputs

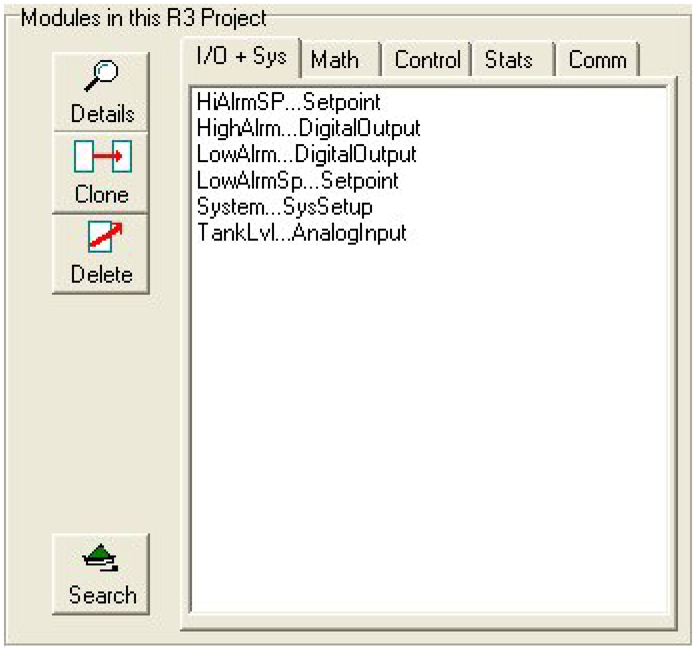
Almost all module inputs can accept either outputs from other modules or constants. Channel numbers for I/O type modules must be constants. These must be typed in or the compiler will complain. You should not leave inputs blank. If you fail to make an entry for a module input, the RUG3 will probably assume it is zero unless it is a status input that should default to a one for logic reasons. However, in some cases it can assume values that will be incorrect for your application. Therefore, you are encouraged to fill in all inputs with constants or values from the databases. You can drag any type of output to the inputs of modules that don't need constants, even if it doesn't make sense. The RUG3 will simply convert the input to the type needed and proceed with processing. The exception is that you cannot drag a string into an input that needs a numeric value; nor can you drag a numeric value into an input that expects a string. Most of the expected input types will be obvious from the prompt. If you are uncertain, consult the detailed module descriptions in chapter 5.

Triggers

Some modules generate triggers. A trigger is simply a status that is true for exactly one program scan. For example, the **TriggerEveryXMinute** module generates a trigger event each minute. The trigger output status from the module becomes true at the beginning of the full scan following the scan in which the conditions were such that the module generated the trigger. The status will remain true for the entire scan and then be returned to false at the end of the scan. Any module that uses that trigger will see it during that scan. You use triggers where one time actions are required. For example, displays require a trigger to cause the display to refresh. If you were to use a status generated by seconds=0, then once a minute the display will update several times during the zeroth second. If, instead, you use **TriggerEveryXMinute**, then the display will refresh exactly once per minute.

Listing Modules in the Project

In the center of the main project screen is a set of tabs labeled "Modules in This R3 Project". Clicking on one of these tabs will show you all the modules of the type in the tab title that you presently have installed in your project. For example, the figure below presents the list of all I/O and system modules in the project when you click the "I/O + Sys" tab. Notice that each module is shown with the name you assigned the module followed by the module type.



HiAlrmSP...Setpoint
HighAlrm...DigitalOutput
LowAlrm...DigitalOutput
LowAlrmSp...Setpoint
System...SysSetup
TankLvl...AnalogInput

Figure 40 List of I/O and System Modules Installed in Project

Modifying an Existing Module

If you wish to change a module after you have installed it in the project, you simply select it from one of the lists in the “Modules in This R3 Project” tabs, and click on the **Details** button. The module’s configuration panel will be presented wherein you can make your changes and then save to the project.

Copying (Cloning) a Module

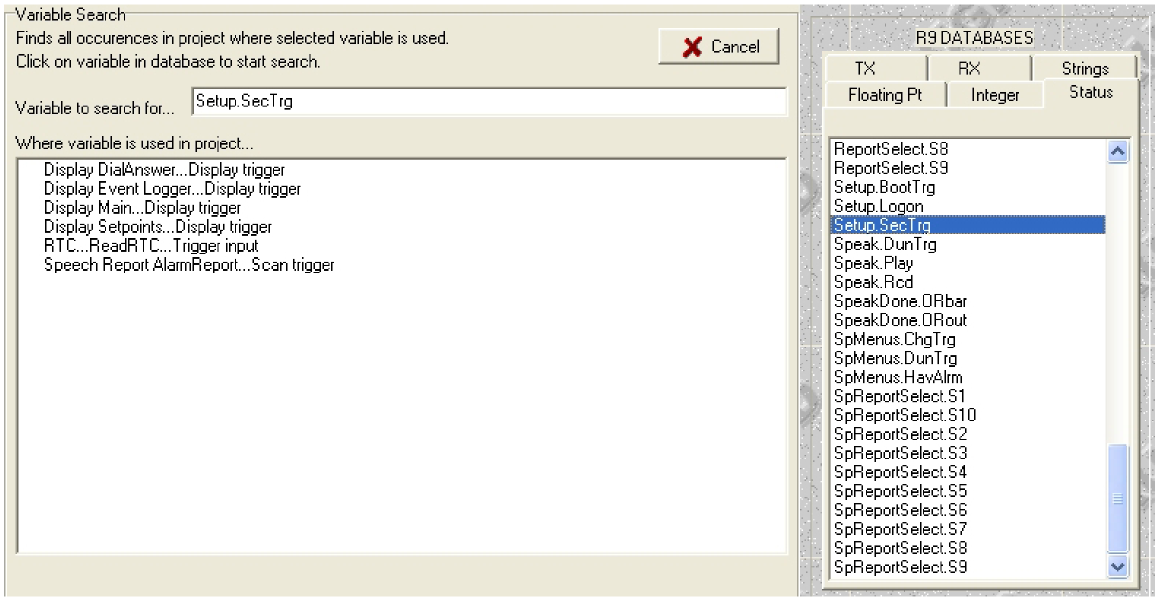
If you need a module of the same type and with similar inputs to one you have already designed, it sometimes saves time to clone an existing one. You do this by selecting the module you wish to clone from the “Modules in This R3 Project” tab and then click on the **Clone** button. You will have to give the module a new name and then change any input properties you wish. Then click on **Save** to install the new module.

Deleting a Module from the Project

To delete a module from a project, select it from the “Modules in this R3 Project” tab and then click on the **Delete** button. It will be deleted from the project and all its outputs will be deleted from the databases.

Searching for Where a Module’s Outputs are Used

Sometimes it is useful to know where a module’s outputs are used before altering or deleting the module. The ‘Search’ button will bring up the Variable Search panel. To search for all uses of a database item, simply click on the item. The search table will then list all locations where the module output is used, as illustrated below:



Variable Search

Finds all occurences in project where selected variable is used.

Click on variable in database to start search.

Variable to search for... Setup.SecTrg

Where variable is used in project...

```
Display DialAnswer...Display trigger
Display Event Logger...Display trigger
Display Main...Display trigger
Display Setpoints...Display trigger
RTC...ReadRTC...Trigger input
Speech Report AlarmReport...Scan trigger
```

```
ReportSelect.S8
ReportSelect.S9
Setup.BootTrg
Setup.Logon
Setup.SecTrg
Speak.DunTrg
Speak.Play
Speak.Rcd
SpeakDone.ORbar
SpeakDone.ORout
SpMenus.ChgTrg
SpMenus.DunTrg
SpMenus.HavAlrm
SpReportSelect.S1
SpReportSelect.S10
SpReportSelect.S2
SpReportSelect.S3
SpReportSelect.S4
SpReportSelect.S5
SpReportSelect.S6
SpReportSelect.S7
SpReportSelect.S8
SpReportSelect.S9
```

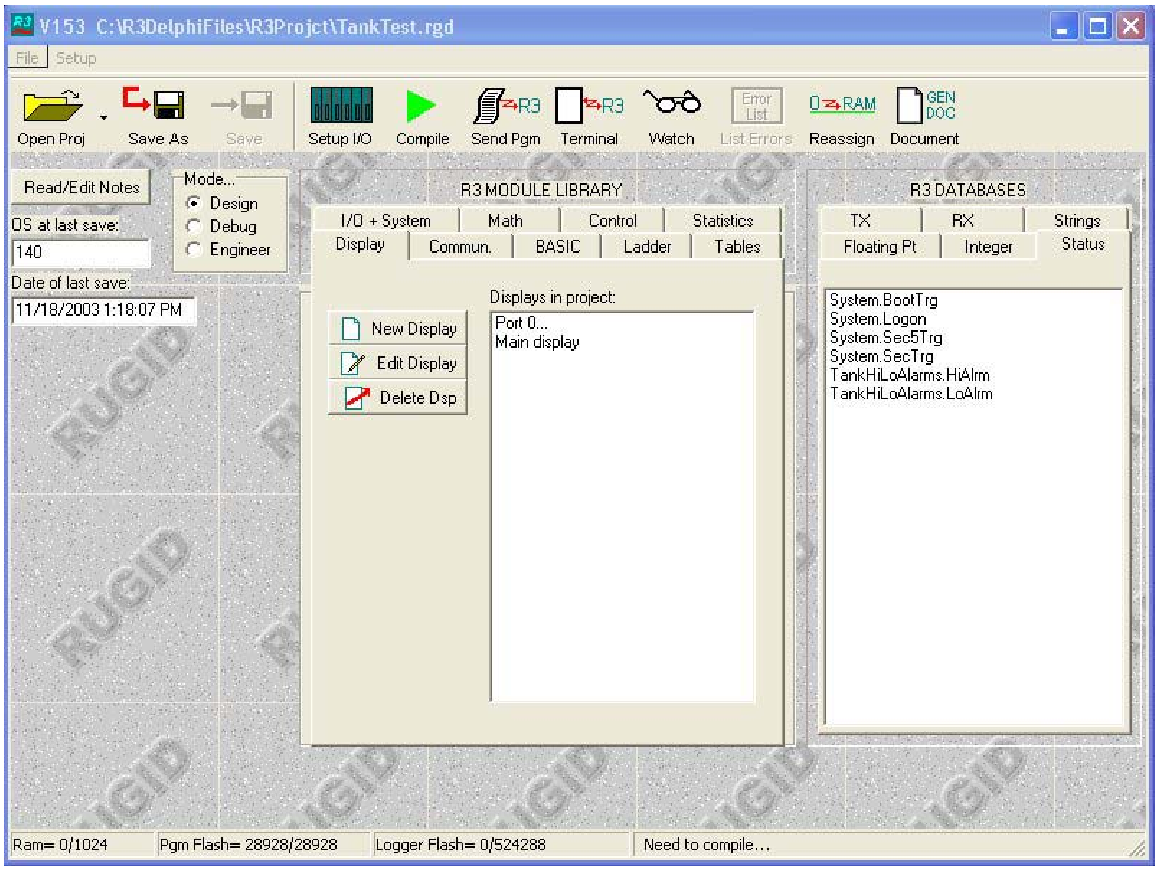
Figure 41 Searching for Output Usage

Returning to the Setup I/O Panel

To return to the I/O representation of your project, click on the Setup I/O button on the tool bar.

Configuring RUG3 Displays

Adding a display to the system will make an operator's job a lot easier than if the unit has no display since we can show him internal values and how setpoints are set. To add a display, first click on the **Display** tab in the **R3 MODULE LIBRARY** window. Your screen will change to this:



OS at last save: 140

Date of last save: 11/18/2003 1:18:07 PM

Mode: [✓ Design] Debug Engineer

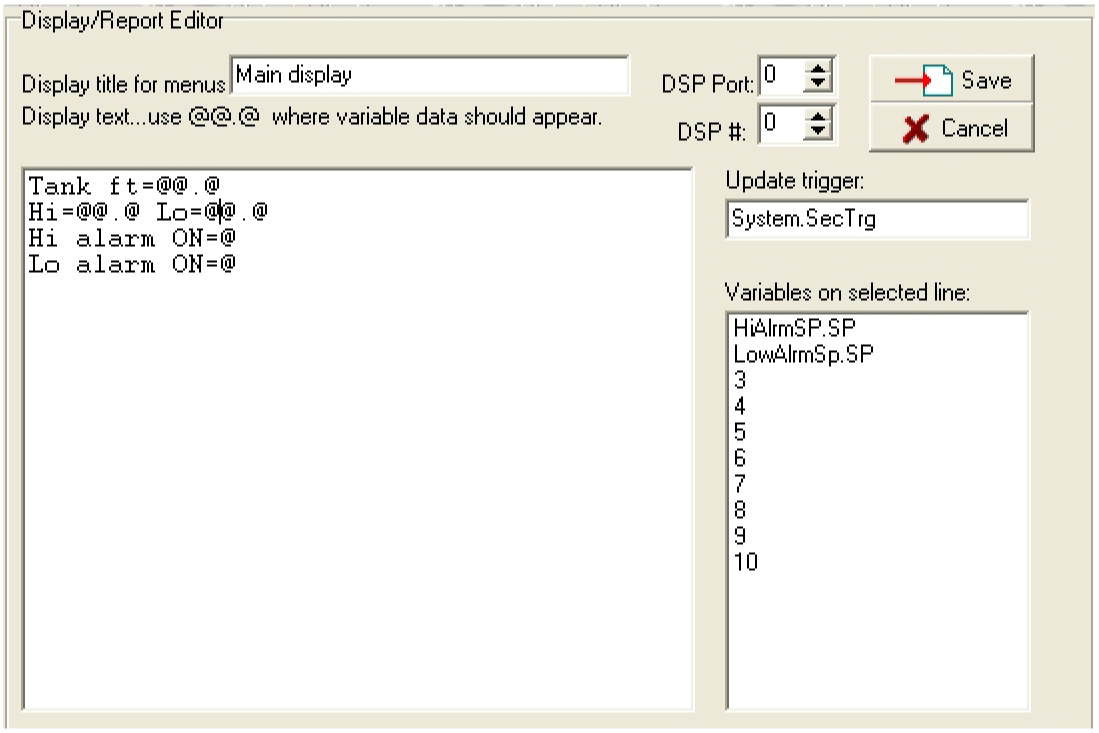
Displays in project:
Port 0...
Main display

System.BootTrg
System.Logon
System.Sec5Trg
System.SecTrg
TankHiLoAlarms.HiAlrm
TankHiLoAlarms.LoAlrm

Ram= 0/1024 | Pgm Flash= 28928/28928 | Logger Flash= 0/524288 | Need to compile...

Figure 42 The Display Tab

From here you can click on the 'New Display' button to create a new display; or, you can select an existing display and then click on either 'Edit Display' to edit the selected display, or you can click on 'Delete Dsp' to delete the display from the project. If you elect create a new display or edit an existing one, you will be presented with the following display editing panel (an existing display is shown):



Display title for menus: Main display

Display text...use @@.@ where variable data should appear.

DSP Port: 0

DSP #: 0

```
Tank ft=@@.@
Hi=@@.@ Lo=@@.@
Hi alarm ON=@
Lo alarm ON=@
```

Update trigger: System.SecTrg

Variables on selected line:

```
HiAlrmSP.SP
LowAlrmSp.SP
3
4
5
6
7
8
9
10
```

Figure 43 Configuring Display for LCD

On this panel, you must enter the display's name, port number and display number. Enter the display's name in the cell called 'Display title for menus'. This is used in menus to enable you to select a particular display to edit or delete; and for the operator to select a display when the unit is running. The port number identifies the port where this display will be observed. The following ports are valid for the RUG3:

Port 0: local LCD display
Port 1: unit programming RS232 port labeled P1
Port 2: unit modem/RS232 port, labeled P2 if RS232 or MDM-AUDIO if modem

The display number specifies where in the display list this display will be shown. Display number 0 will be at the top of any list, display number 1 next on the list, etc. Each display must have a unique number and there must be no gaps or duplicates in the display numbering or the compiler will complain. Note that each port will have its own display list. In most applications, you will design displays for the LCD port and none for the serial ports. However, you can design displays for serial ports that are designated in their ComSetup modules as ASCII mode. In that case, the operator can select displays to be presented on any of the ports independently. Displays presented on any port will not affect displays presented on other ports. The cell labeled 'Update Trigger' contains the trigger signal that governs when the display will be updated. Usually you will use the 'SysSetup.SecTrg' trigger from the status database to update the LCD once per second. For serial ports, you should use 'SysSetup.Sec5Trg' to update once each 5 seconds, since once per second updates for serial ports is usually too frequent.

The large window on the display panel is where you type in the text you want the display to show when accessed by the operator. Basically, what you type there is what the operator will see, except that you use '@@.@' symbol fields to designate where active data from the databases will be shown. To specify the particular database item to be shown on a line simply click on the line and then drag the item from a database that you wish to show on that line into the list labeled 'Variables on Selected Line:'. To illustrate, we'll design a display for the LCD below.

Designing a Display for the LCD

The RUG3's LCD display is limited to 2 lines by 16 characters. Also, it is designated as port 0 in the display addressing structure. Therefore, when you configure an LCD display, you must specify port 0, and keep your lines to 16 characters or fewer. You can place up to 100 lines on any display, but the RUG3 will only show two at a time. The operator accesses additional lines on the display by hitting the UP or DOWN arrow keys to traverse up or down the display by two lines per key press. You can have multiple displays designated for the LCD. The operator traverses to the next display by hitting the 'Enter' key. You will find it most convenient to design your display with related items on lines that will be displayed together. Notice in the display configuration page above, no line exceeds 16 characters, and the display port has been set to 0.

Notice the use of **@@.@** symbols. Basically, anywhere you want active data from a data base to appear on the LCD, you specify the location and format by using the **@@.@** characters. '@' symbols and decimal points are regarded as one field with the location of the decimal point specifying where the decimal point will be placed on the LCD screen. For each @@.@ field on a display line you must have a variable installed in the panel labeled 'Variables on Selected Line'. When the RUG3 program is running and the RUG3 is updating the display, if it encounters a '@@.@' field it will consult the 'Variables on Selected Line' list to tell it from where to obtain the data to be presented. The first '@@.@' field will be filled in with the value from the first item in the list; the second '@@.@' field will be filled in from the second item in the list and so on. In the example display above, the user's cursor is resting on the second line which contains two '@@.@' fields. For that line there are therefore two database items in the variable list.

Compiling the Project

At any time you can click the green **Compile** button to compile the project. The compiler will compile your project and give you a list of errors if any are found. The compiler will compile automatically whenever you command the compiler to send the project to the RUG3, and if you click on the watch window button. After a successful compilation the compiler will present the RAM and Flash memory required within the RUG3 in a small panel in the lower left corner of the R3SETUP form as illustrated below.

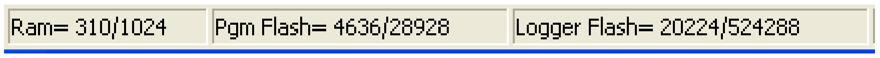

Ram= 310/1024 | Pgm Flash= 4636/28928 | Logger Flash= 20224/524288

Figure 44 Project Ram and Flash Utilization

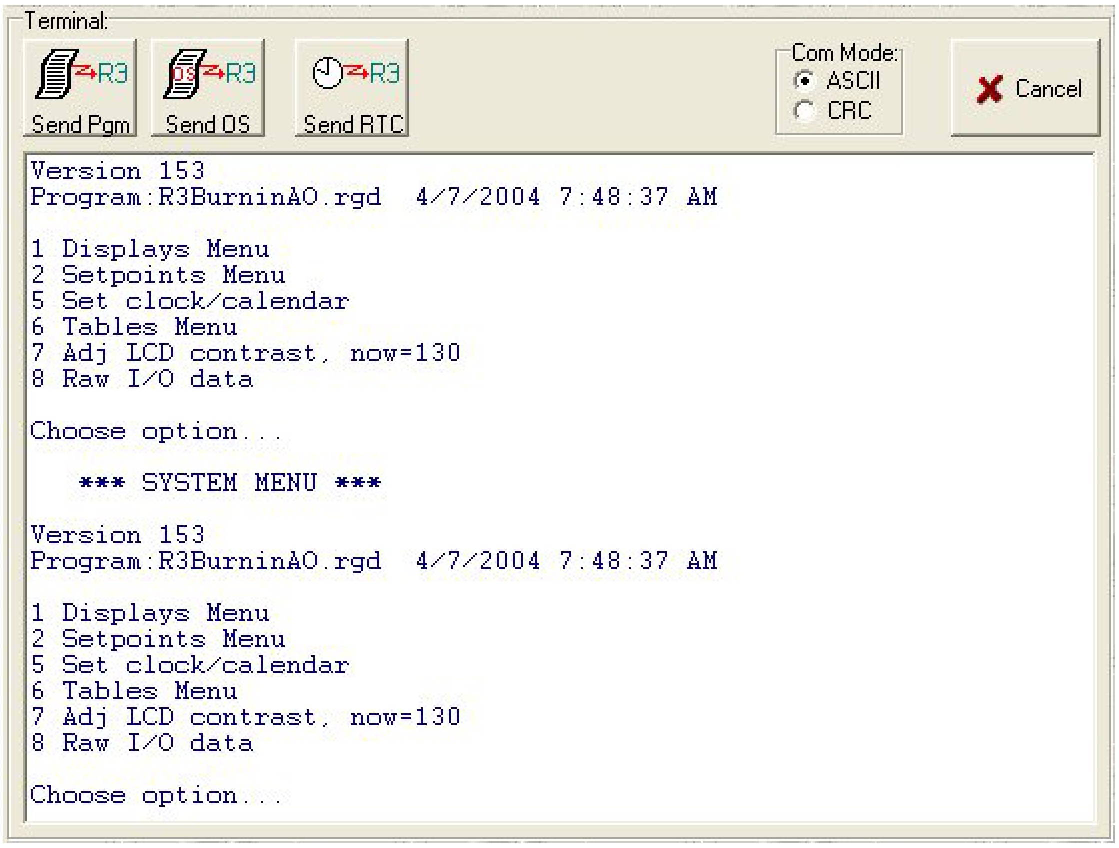
In each of these fields the leftmost number is the amount required by the project; the rightmost number is the unit's total capacity available for project use. RAM is the scratchpad area used by the modules as they execute. Pgm flash is the area where the configuration file is stored and is not changed as the program is running. Logger flash is used for data logging.

Sending the Program to the RUG3

After you have designed your project and it compiles successfully, you are ready to send it to the RUG3 for execution. You can do this either by clicking the **Send Pgm** button on the tool bar, or by clicking the **Send Pgm** button at the top of the terminal page. In either case, the compiler will compile the project and then open the terminal page and begin sending the program. If the compiler detects any errors, it will not send the program. Assuming no errors are found, the program is sent using the RUG3 CRC secured protocol to eliminate any transmission errors. If any transmission errors are detected, the repeat counter will be incremented and displayed. Once the program is successfully sent to the RUG3, the compiler will command the RUG3 to begin immediately running the program.

Opening the Terminal Page

The terminal page is a communications window that lets you send ASCII messages to the RUG3 and lets you observe ASCII responses from the RUG3. To open the terminal page, click the Terminal button on the tool bar. When you do so, R3SETUP will open a communications port on your PC and begin communicating with the RUG3. The page looks like the figure below:



Com Mode: [✓ ASCII] CRC

```
Version 153
Program:R3BurninAO.rgd  4/7/2004 7:48:37 AM

1 Displays Menu
2 Setpoints Menu
5 Set clock/calendar
6 Tables Menu
7 Adj LCD contrast, now=130
8 Raw I/O data

Choose option...

   *** SYSTEM MENU ***

Version 153
Program:R3BurninAO.rgd  4/7/2004 7:48:37 AM

1 Displays Menu
2 Setpoints Menu
5 Set clock/calendar
6 Tables Menu
7 Adj LCD contrast, now=130
8 Raw I/O data

Choose option...
```

Figure 45 Terminal Page Showing Normal RUG3 Boot Up Messages

Sending the Operating System to the RUG3

Since the RUG3's operating system (OS) is held in flash memory, it can be reloaded serially. Each revision of R3SETUP includes the latest RUG3 operating system. Clicking the **Send OS** button on the Terminal page will send the new OS to the RUG3. Loading the OS to the RUG3 will take several minutes. If it is interrupted, or if at any time the operating system integrity test on boot up fails, the RUG3 will issue a "Waiting for OS load" message and wait until you reload the OS. You should make sure that the operating system version present in the RUG3 matches the version of your R3SETUP. The RUG3's operating system revision is listed on the terminal page any time the RUG3 issues the "Welcome to RUGID Monitor" message. On the terminal page illustrated above, the OS version is given as 153. R3SETUP's revision is given in the upper left hand corner of the form as V153 for example.

Sending the PC's Realtime Clock to the RUG3

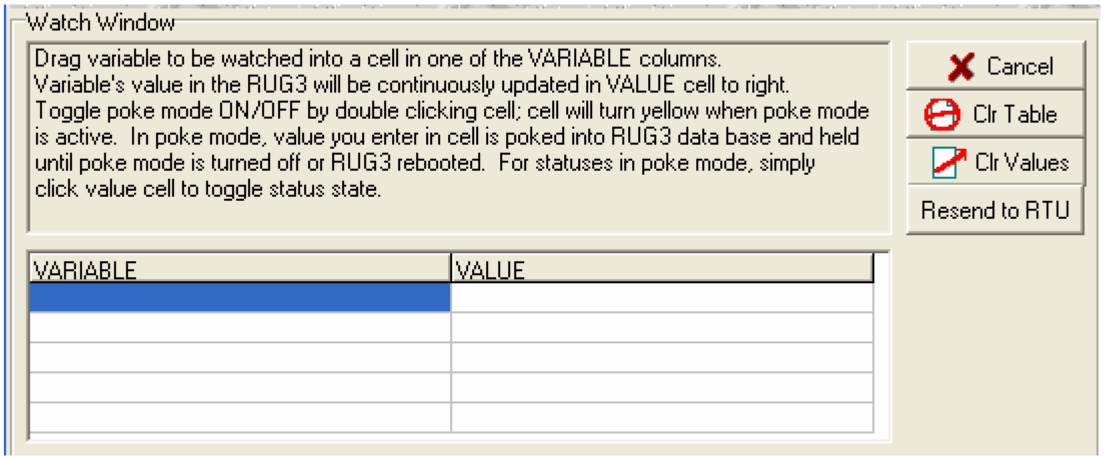
At any time, you can click the **Send RTC** button at the top of the terminal page, and R3SETUP will read your PC's clock/calendar and install it into the RUG3's realtime clock in CRC secured format. To return to ASCII mode, you will need to hit the CTRL K key 4 times afterward.

Stopping the RUG3's Program

You can stop the RUG3's program by typing Ctrl K three times. The RUG3 should send the 'Program halted!' message and wait for a command or program reload. After 60 seconds of no activity, the RUG3 will test the loaded program and restart it if no errors are detected.

Observing Program Execution

You can observe the values of data base items while the RUG3's program is running by hitting the **Watch** button on the tool bar. When you do so, the following watch window panel will appear:


Watch Window

Drag variable to be watched into a cell in one of the VARIABLE columns.
Variable's value in the RUG3 will be continuously updated in VALUE cell to right.
Toggle poke mode ON/OFF by double clicking cell; cell will turn yellow when poke mode is active. In poke mode, value you enter in cell is poked into RUG3 data base and held until poke mode is turned off or RUG3 rebooted. For statuses in poke mode, simply click value cell to toggle status state.

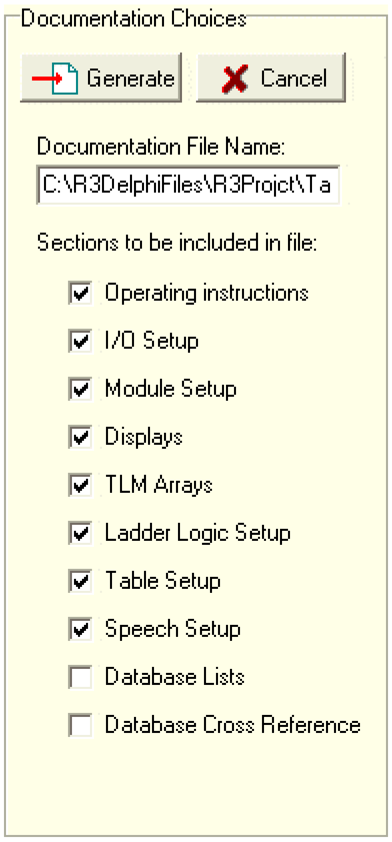
VARIABLE	VALUE

Figure 46 Watch Window

When you first see this window, all entries will be blank as shown above. To begin observing values from the RUG3, simply drag database items from your databases and drop them into the VARIABLE columns on the watch window. Within a few seconds the watch window should begin showing values present in the RUG3. If your RUG3 reboots for any reason or you reload its program, it will stop sending watch window updates until you re-drag the database items back into the watch window or until you click the 'Resend to RTU' button on the watch window panel to refresh the list in the RUG3.

Documenting Your Project

If you click the **Document** button on the tool bar, the compiler will present the following selection panel from which you can choose which portions of the automatic documentation generator you wish to engage to generate project documentation automatically.


Documentation Choices

Generate | Cancel

Documentation File Name: C:\R3DelphiFiles\R3Projct\Ta

Sections to be included in file:

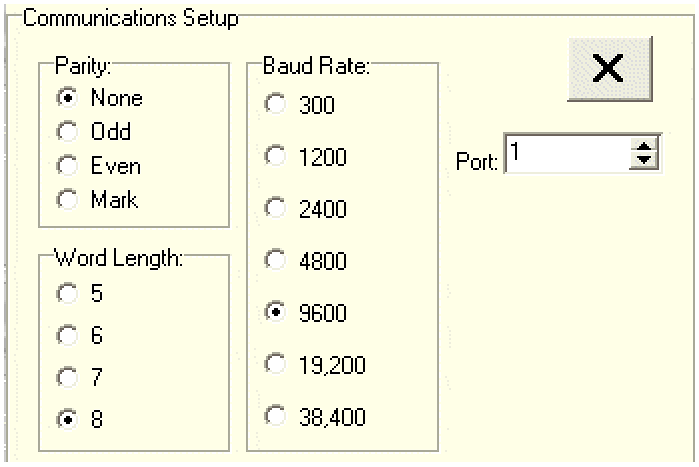
- [✓] Operating instructions
- [✓] I/O Setup
- [✓] Module Setup
- [✓] Displays
- [✓] TLM Arrays
- [✓] Ladder Logic Setup
- [✓] Table Setup
- [✓] Speech Setup
- [ ] Database Lists
- [ ] Database Cross Reference

Figure 47 Documentation Generator Choices

When you click the Generate button on this panel, the compiler will generate a text file detailing those items of your project that you selected. The file will be saved with the same name as your project except with the .Txt extension unless you elect to name the file otherwise in the edit box at the top of the panel.

Setting PC Communications Port Parameters

The menu just above the tool bar has a **Setup** menu item from which you can select **Com Port** to obtain the following panel.


Communications Setup

Parity: [✓ None] Odd Even Mark

Word Length: 5 6 7 [✓ 8]

Baud Rate: 300 1200 2400 4800 [✓ 9600] 19,200 38,400

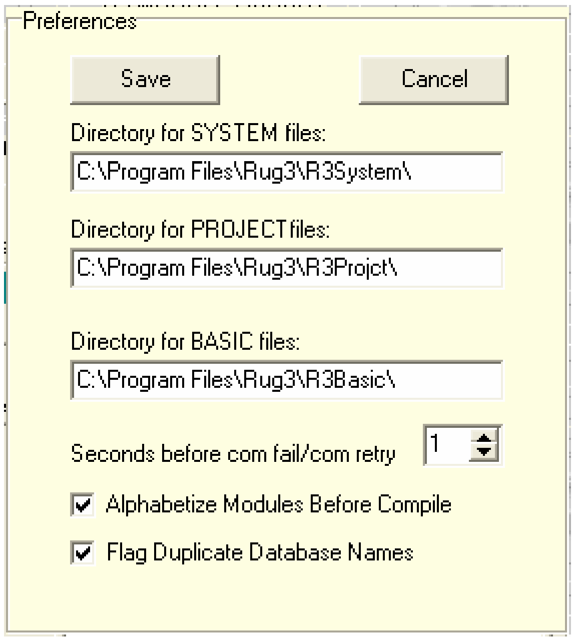
Port: 1

Figure 48 Com Port Setup Panel

The selections shown above are the defaults and are the settings we recommend. However, you can change them to suit your application. Changes you make will be saved in an initialization file and reinstalled each time R3SETUP is started. Note that if you wish to use the USB port, you will probably need to select a higher number com port such as port 4 or 5 depending upon how many USB connections are being redirected as com ports on your PC. Port range is 1 through 25.

Setting File Paths and Controlling Sorting

If you select Preferences from the Setup menu, you will be presented with the following panel.


Preferences

Save Cancel

Directory for SYSTEM files:
C:\Program Files\Rug3\R3System\

Directory for PROJECT files:
C:\Program Files\Rug3\R3Projct\

Directory for BASIC files:
C:\Program Files\Rug3\R3Basic\

Seconds before com fail/com retry: 1

[✓] Alphabetize Modules Before Compile

[✓] Flag Duplicate Database Names

Figure 49 Preferences Panel

Here you can change the paths to folders where R3SETUP stores your files. Simply type in the new paths you want and click the save button. You can also control whether the compiler sorts modules alphabetically before compiling. Other selections are the following:

- Seconds before com fail/com retry…Sets the time in seconds that the software will wait for a response from the RUG3 before declaring that the specific message has been missed and needs to be retried.
- Alphabetize modules before compile…If you leave alphabetization checked (the default), the compiler will sort before compiling, meaning that program execution in the RUG3 will be in alphabetical order of the names you have assigned the modules. Also, setpoints will be shown in alphabetical order. If you uncheck the alphabetization checkbox, modules will be executed in the RUG3 in the order that you installed them in your project; and setpoints will be presented in that order also. Generally, leave this checked (the default).
- Flag duplicate database names…If you uncheck the ‘Flag Duplicate Database Names’ checkbox, the compiler will ignore duplicate database names. If checked, duplicate database names and therefore, module names, will cause a compile error. Generally, leave this checked (the defaul

CHAPTER 5…SOFTWARE MODULES

Introduction

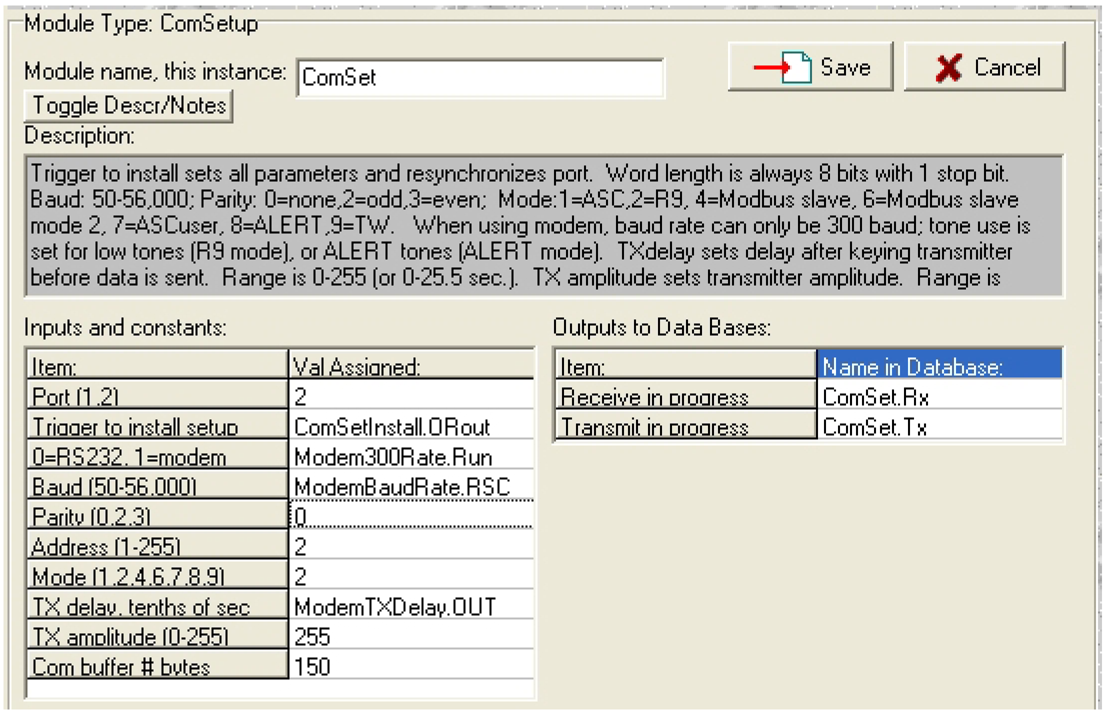
This chapter provides detailed descriptions of the preprogrammed software modules resident in the RUG3. With very few exceptions, you may use as many copies of each module as you need for your application; you must simply give each instance of a module's use a unique name. In addition to the individual modules, this section discusses the use of ladder logic.

Typical Module Setup

The figure below presents a typical module already set up for an application. This is the module editing panel that will appear any time you select a module from the module library. The inputs, description, and outputs will vary from module to module, but they will all have the elements essential for you to establish how the module is to behave in your application.

Naming a Module

Once you have a module's editing page present as in the case below, it is important to enter a name for the module. The name must be less than 20 characters and must be unique, or when you attempt to save the module, the R3SETUP system will complain that the name already exists. You should choose a name that suggests to you the function or signal being processed by the module, since that name will be the name given to all outputs of the module which then appear in the data bases for use by other modules. Also, the name is your reference for accessing the module later to make changes. For example, for analog inputs, you might choose names such as 'TankLvl', 'StreamFlow', 'Pump1Temp', etc. For relay outputs, such names as 'Pmp1CallRly', 'HiAlrmRly', or 'SandRly' suggest their functions. Also, names may contain blank characters and numerals for clarity. In the example module below, the name has been entered as "ComSet".



Module name, this instance: ComSet

Description:

Trigger to install sets all parameters and resynchronizes port. Word length is always 8 bits with 1 stop bit. Baud: 50-56,000; Parity: 0=none,2=odd,3=even; Mode:1=ASC,2=R9, 4=Modbus slave, 6=Modbus slave mode 2, 7=ASCuser, 8=ALERT,9=TW. When using modem, baud rate can only be 300 baud; tone use is set for low tones (R9 mode), or ALERT tones (ALERT mode). TXdelay sets delay after keying transmitter before data is sent. Range is 0-255 (or 0-25.5 sec.). TX amplitude sets transmitter amplitude. Range is

Inputs and constants:

Item:	Val Assigned:
Port (1,2)	2
Trigger to install setup	ComSetInstall.ORout
0=RS232, 1=modem	Modem300Rate.Run
Baud (50-56,000)	ModemBaudRate.RSC
Parity (0,2,3)	0
Address (1-255)	2
Mode (1,2,4,6,7,8,9)	2
TX delay, tenths of sec	ModemTXDelay.OUT
TX amplitude (0-255)	255
Com buffer # bytes	150

Outputs to Data Bases:

Item:	Name in Database:
Receive in progress	ComSet.Rx
Transmit in progress	ComSet.Tx

Figure 50 Module Editing Panel

Specifying Inputs

Most modules will have one or more inputs that you may specify. In the above panel, they appear in a list on the left side titled "Inputs and constants". You have three choices for each input: leave it blank, type in a value or string, or drag in an entry from one of the databases. In the above example panel, the first, fifth, sixth, seventh, ninth and tenth entries have had constants typed into them by the programmer, and the rest have had entries from the databases dragged into them. Here are the rules regarding specifying inputs:

- **Constants:** you may type in a constant in any input except in the cases of a few modules that require arrays from other modules, such as modules that act on data stored by data loggers and event loggers. You **must** type in a constant if the module description requires a constant. In the above example, the first input (Port #) must be entered as a constant to tell the compiler which port this module controls, and the last input (Com buffer # bytes) so the compiler can allocate RAM for the buffer. Generally, constants are only required to tell the compiler which channel number a module uses, or how much space to reserve for buffers and string outputs.
- **Blanks:** you may leave blank any input that is not needed in a calculation or other use. Generally, blank entries are regarded as 0.0 if they are terms, 1.0 if they are factors. It's safest to install an entry for all inputs.
- **Pointers:** when you drag an entry from a database into an input, you are really installing into the module a pointer to a place in the data base where the data resides. If a constant is not required as described above, then you can always drag in a pointer from a database. The only limitation is that if the module specifies a string input, then you must drag it in from the string database. Otherwise, you can drag a pointer from any of the other databases. Conversely, if the module's input does not specify a string input, then you must drag a variable from one of the other databases.
- **Data types:** in the module descriptions below, each input description indicates the data type expected (status, integer, floating point or string). Other than the case of strings described above, you do not need to be concerned about the data type. The compiler takes care of numeric

conversions among the numeric types (status, integer or floating point). Therefore, even though an input might indicate that it is expecting a floating point input, you are free to drag in, for example, an integer.

I/O Modules

AnalogInput

The **AnalogInput** module is used to add either a current type or voltage type analog input to a project. One of these modules is required for each analog input in your RTU that you wish to engage in your project. The module performs raw count to engineering units (EU) conversion, low pass filtering, and output range limiting. The EU at 4 ma or 0 volts entry sets the engineering units value that corresponds to a 4 ma. input from the field (if designated as a 4-20 ma type input), or the EU value that corresponds to 0.0 volts (if designated as 0-5 V type input). The EU at 20 ma or 5 v entry sets the engineering units value that corresponds to the high end of the instrument's span. The filter time constant sets the time in scans of the low pass filter. The higher the filter time constant, the slower the analog input will be to respond to changes in input value and the more stable it will be in a high noise environment. Values below 100 essentially provide no noise immunity. Typical useful filter constants range from 500 to 20,000. The engineering units output from this module will be available in the floating point data base.

Inputs That Must be Constants:
- Input channel #…which analog input channel on the board this module uses (1-6)

Other Inputs:
- 1=enable, 0=hold…status input that controls whether this module executes or skips
- Type 0=4-20, 1=0-5V…status that engages the channel sense resistor and selects the corresponding calibration set for the channel.
- EU at 4 ma or 0 V…engineering units value of output when input is either 4 ma or 0.0 volts
- EU at 20 ma or 5.0 V…engineering units value of output when input is either 20 ma or 5.0 volts
- Filter constant…low pass filter time constant in scans. The larger this number, the slower the input will be to respond to changes in its analog input signal. Zero or blank gives no filtering. For values in excess of 32767, use a decimal point, i.e., 56000.0
- High output limit EU…maximum output value in engineering units
- Low output limit EU…minimum output value in engineering units

Primary Outputs:
- EU output…Name.Out…floating point engineering units representation of this analog input
- Raw A/D output…Name.Raw…integer output of A/D converter (0-4095)

Expected Applications:
Use this module for analog inputs. Use setpoints for EU at 4 ma and EU at 20 ma if the operator needs to adjust calibration.

AnalogOutput

The **AnalogOutput** module is used to add a current type analog output to a project. One of these modules is required for each analog output in your RTU. The module performs engineering units (EU) to raw count conversion so that when you present an EU value to the module, the correct output current is emitted by the hardware. The EU at 4 ma entry sets the engineering units value that corresponds to a 4 ma. output. The EU at 20 ma entry sets the engineering units value that corresponds to a 20 ma. output. For example, if you use the analog output to control a VFD (variable frequency drive) and you wish to use an engineering units range of 0 to 100%, set EU at 4 ma to 0.0 and EU at 20 ma. To 100.0. When the enable update is true, the module will send a new value to the output hardware. IMPORTANT: do not set the 'enable update' input to '1' or the channel will attempt to send the analog output to the D/A converter every program scan. This will cause the channel to draw more than 4 ma. from the channel and thereby corrupt its linearity. Instead, use the System.SecTrg once per second trigger in the status database to cause the output to be updated once per second. If select=0 or blank, the module scales the floating point value (EU Source) by interpolation and extrapolation, and sends it to designated analog output port. This is how you should set it. When select=1 (used during factory calibration), the module sends the raw count value directly to analog output. Allowed values of raw count are 1 through 4095.

Inputs That Must be Constants:

- Channel # (1 or 2)…which analog input channel on the board this module uses (1-2)

Other Inputs:

- Enable update, 0=hold…status input that controls whether this module executes or skips. Use System.SecTrg here.
- EU source…floating point value that is to be converted to 4-20 ma. current at the output.
- EU at 4 ma…engineering units value that should produce 4 ma. output current
- EU at 20 ma…engineering units value that should produce 20 ma. output current
- Select, 0=EU, 1=raw…integer to specify that: Select=0, the module convert the EU value to output current; or Select=1, that the module transfer the raw value directly to the D/A converter.
- Raw count to send…raw value to send directly to D/A converter when Select=1

Primary Outputs:

- None

Expected Applications:

Use this module to control analog outputs for controlling VFD's, gates, valves, chart recorders, etc.. Use setpoints for EU at 4 ma and EU at 20 ma if the operator needs to adjust calibration.

Anemometer

The **Anemometer** module is used to read the anemometer input (alternate use of analog input #6) and compute wind speed from accumulated pulses. Result is: Windspeed=SpeedPerHz * PulsesPerSec + Offset. A new output is calculated once each 5 seconds using average pulses per second. If result falls below dropout, the result is set to 0.0. This module uses interrupts to count pulses at relatively fast rates. Design has been tested at 1200 Hz.

Inputs That Must be Constants:

- None

Other Inputs:
- Speed per Hz…floating point value from instrument data that specifies the wind speed engineering units that should result from 1 Hz input frequency
- Offset…floating point value that will be added to speed calculation to produce result (usually 0.0)
- Dropout…floating point value of the result below which the module will issue a result of 0.0

Primary Outputs:
- Windspeed…Name.Wind…floating point engineering units representation of wind speed

Expected Applications:
Use this module to read wind speed from anemometers with AC outputs.

DiginCount

This module is used to engage background pulse counting on a designated digital input. The input is sampled 256 times per second, enabling the input to capture square waves of up to 128 Hz, or pulses as narrow as 8 ms with a minimum separation of 8 ms. Counting range is –32768 through +32768. The counter counts up on each assertion of the digital input and stops when the counter hits its maximum value.

Inputs That Must be Constants:
- Channel #…which input channel (1-8) on the board this module uses

Other Inputs:
- Mode (0-2)… 0=rising edge, 1=falling edge, 2=both edges
- Trigger preset…trigger input that causes the preset count value input to become the present count
- Preset count value…integer count to install as the count value when preset triggered

Primary Outputs:
- Count output…Name.Count…integer count of pulses counted since last preset
- Present DI status…Name.DI…status of digital input being counted.

Outputs for Internal Use: None

Limitations:
Pulses or inter-pulse timing of less than 8 ms. will cause pulses to be missed.

Expected Applications:
Use this module alone for general purpose counting and totalizations, such as pump start counting. It's also useful for detecting momentary contact closures such as pushbuttons. This module is also required to perform counting for the **PulseToFlow** module. In that application, simply have both the **DiginCount** module and the **PulseToFlow** module reference the same digital input. This module works with all digital inputs.

DigitalAlarmOutput

This module should be used when you want a relay output to function as an alarm output, such as in most alarm annunciator panels. Its function is to flash its relay output every half second when its status input (alarm) turns on. When acknowledged, the relay output will go solid on if the alarm is still present or go off if the alarm has gone away. The module also has a lamp test input that will turn the relay output solid on as long as the lamp test input is true. Once the lamp test returns false, the alarm output will return to its state prior to the lamp test turning on.

Inputs That Must be Constants:
- Output relay #(1-4)…which output channel (1-4) on the board this module uses

Other Inputs:
- Status input…the status that constitutes the alarm this module is presenting.
- ACK input…trigger input used to acknowledge the alarm, i.e., make the output stop flashing on and off
- Lamp test/ACK input…status input that when on, will cause the relay output of this module to stay on until lamp test is turned off

Primary Outputs:
- Have unacked alarm…Name.UnAck…status output that will be true if an unacknowledged alarm is present.

Outputs for Internal Use: None

Limitations: None

Expected Applications:
This module implements standard alarm annunciation functionality. It works with all relay output modules.

DigitalInputDC

This module is used to bring DC type digital inputs into the system. One of these should be used for each DC type digital input to be sensed. When the digital input is on, this module's true output is 1; when the input is off, the module's true output is 0.

Inputs That Must be Constants:
- Input channel #…which input channel (1-8) on the board this module uses

Other Inputs: none

Primary Outputs:
- Output NO…Name.DI…true echo of external digital input state
- Output NC…Name.DIBar…inverted echo of external digital input state

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Use this module to sense contact closures.

<u>**Software Modules**</u>

DigitalOutput

Use one of these modules for each digital output to be controlled in your application, except for those used as alarm annunciators and needing the functionality of the alarm output module. This module accepts a status from the status data base and uses it to control a digital output.

Inputs That Must be Constants:

- Output channel #(1-4)…which output channel (1-4) on the board this module uses

Other Inputs:

- Input status that controls…status whose state is to be echoed by the relay controlled by this module

Primary Outputs: None

Outputs for Internal Use: None

Limitations: None

Expected Applications:

Use one of these modules for each relay output to be controlled by a single status.

GetUserValue

This module enables you to set up a process whereby the operator can enter values directly from a display using the local keyboard/LCD or serial port without having to access the setpoint list. When trigger=true, the module sends prompt string to the designated port, then captures the user's input value and saves it in the Val output when the user hits the ENTER key. If port 0 specified, prompt appears on top line of LCD and user input appears on second line. If high and low limits are specified, the user value will be ignored if outside that range. Login permission flag: 0=any user can input; 1=user must be logged in first.

Inputs That Must be Constants:

- Display port #...integer that specifies the port number where prompt to be sent and input value to be obtained (0-2).

Other Inputs:

- Prompt string to send…string to prompt user (max 16 chars on LCD)
- Trigger input…status trigger that will cause prompt to be issued on designated port.
- High limit…floating point value above which user's input will be ignored.
- Low limit…floating point value below which user's value will be ignored.
- Login permission flag…0=any user may input value, 1=user must be logged in first.

Primary Outputs:

- User value…Name.Val…floating point value last entered by user
- Have new input…Name.Trg…trigger status output indicating that a new value has been received.

Outputs for Internal Use:

- Internal sequencer…Name.Seq…internal sequencer
- Dsp type copy…Name.Dsp…copy of previous display setup.

Limitations: None

Expected Applications:
Use to provide convenient operator input method.

PulseDurationIn

The PulseDurationIn module is used to read the duration of pulses on digital inputs. Use one module for each digital input for which you wish to read pulse durations. Each time an input pulse ends, the module will output the pulse's duration in seconds with 4 ms resolution. It will also issue a trigger to indicate that a new pulse has been received. Additionally, if you have specified a cycle time and scale factor, the module will calculate the value EU=(K*(PULSE DURATION-MIN PULSE))/CYCLE TIME. You can use this to read pulses from pulse type flowmeters and convert to engineering units.

Inputs That Must be Constants:
- Channel #…which input channel (1-8) on the board this module uses

Other Inputs:
- Scale factor K…floating point input scale factor for duration to EU conversion equation
- Cycle time sec…floating point input setting the expected pulse repetition time
- Minimum pulse sec…floating point input setting the pulse width that will be regarded as zero value

Primary Outputs:
- Pulse duration sec…Name.Sec…floating point pulse duration
- EU value…Name.Value…floating point engineering units value after conversion
- Had pulse trigger…Name.Hadpulse…trigger status indicating that a new pulse was measured
- Input status…Name.Input…status echo of digital input

Outputs for Internal Use: None
- Missing pulse timer…Name.Tmr…timer for internal use to detect missing pulses

Limitations: None

Expected Applications:
Use this module to measure input pulse durations and to read outputs from pulse duration type instruments.

PulseDurationOut

The PulseDurationOut module is used to generate pulses on digital outputs. Use one module for each digital output for which you wish to generate pulse durations. Each time the module is triggered, the module will output the pulse's duration in seconds with 4 ms resolution. If a cycle time is specified, the module will repetitively issue a variable length pulse once each cycle. Ranges of pulses and cycle time are 0 to 262 seconds.

Inputs That Must be Constants:
- Channel #…which input channel (1-8) on the board this module uses

Other Inputs:
- Trigger pulse…trigger to start each pulse if cycle time blank
- Cycle time sec…floating point input setting the expected pulse repetition time

Primary Outputs: None

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Use this module to generate general purpose pulses of precise duration.

PulseToFlow

Module converts pulse rate on a designated digital input to flow rate and sends it to the floating point database. The calculation is performed at the end of each cycle. For each calculation, the module reads the number of pulses, and number of seconds since prior cycle end, then applies user's flow per pulse factor to calculate average flow rate per second during the cycle. Units of the output flow rate are related to user's flow per pulse entry. For example, if flow per pulse is in gallons per pulse, then flow rate would be gallons per second. To derive GPM, the output in this case must be multiplied by 60 (or the flow per pulse could be multiplied by 60). The cycle time is in seconds and should be long enough to accumulate multiple pulses in each cycle. This module requires that a **DiginCount** module be assigned to the same digital input to do the counting.

Inputs That Must be Constants:
- Channel #…which input channel (1-8) this module uses

Other Inputs:
- Cycle time, sec…integer time in seconds after which the module will read the count and calculate flow rate
- Volume per input pulse…floating point factor defining how many gallons are represented by each input pulse
- Preset total volume trig…status input to force preset value to total volume output
- Total volume preset value…floating point value to preset total volume when above trigger asserted
- Flow rate multiplier…floating point multiplier applied to flow rate after volume calculation
- Volume thresh to trigger…floating point setpoint that when exceeded by total flow will cause the Name.VolTrg trigger to be generated and the volume to be subtracted from total

Primary Outputs:
- Flow rate…Name.Flow…floating point output of flow rate calculation in user defined units
- Total volume…Name.Total…floating point total volume accumulated
- Volume>threshold trigger…Name.VolTrg…trigger status indicating that volume has exceeded threshold

Outputs for Internal Use:
- Old count…Name.Old…old count to compare against new count reading
- Temp volume…Name.Temp…floating point temporary total accumulator
- Internal timer…Name.Tmr…internal cycle timer, seconds
- Temp count…Name.Tcount…temporary sample counter

Limitations:
Trigger interval should be long enough for the counter to accumulate at least 10 pulses. Trigger interval must be less than 32767 seconds. Input pulse width must not be shorter than 8 milliseconds; and inter pulse interval must not be shorter than 8 milliseconds.

Expected Applications:
Use this module to provide flow rate from pulse type flow meters rather than using general purpose counters and time triggers because this module has a more accurate time base.

Setpoint

The setpoint module enables you to specify a setpoint to be added to the RUG3's setpoint list that an operator can use to enter values into the RTU's system to control operations. Access to setpoints in the RUG3 can be secured with an access code that must be entered correctly before access is granted. The access code is set in the SysSetup module. When you install a setpoint module, you must enter a setpoint string that is the prompt to the operator. Typical prompts are: 'High alarm ft=' and 'Pump 1 (0=off, 1=hand, 2=auto):'. These prompts will be presented on the RUG3's LCD or attached terminal in an alphabetized list with the current setpoint value shown as in the following example:

```
0 High alarm ft= 12.34
1 Pump 1 (0=off, 1=hand, 2=auto): 1
2 Static voltage=1234.56
3 Target level meters=13.8988
```

If you are using the local LCD, you must keep your prompts to 16 characters or fewer. Notice that the setpoint module has default value and default installation trigger inputs. Since the RUG3 uses compiled code, the values of entries in the data bases are not cleared automatically on program installation. Therefore, setpoints and other data base entries will initially have undefined values or will retain values that they had prior to the new program being loaded. The default value and trigger enable you to specify a value to be installed whenever the trigger is true, such as when a key is pressed (using the TrigOnKeyMany module).

Inputs That Must be Constants: None

Other Inputs:
- Prompt string…string the user will see in the RUG3's setpoint list
- Trigger install default…trigger status to cause default value to become the setpoint value.
- Default value…floating point value to be installed in the setpoint when the trigger to install default is true
- Max allowed value…floating point value that is the maximum to be output by the module
- Min allowed value…floating point value that is the minimum to be output by the module
- Visibility (0,1,2)…integer: 0=setpoint never shown; 1=shown if logged on; 2=shown always

Primary Outputs:
- Value user entered…Name.SP…present floating point value held by setpoint register.
- New value trigger…Name.NewTrg…status trigger asserted when a new value is entered into the setpoint, even if it is the same as the previous value.

Outputs for Internal Use: None

Limitations:
Value range is +/-10E+/-38.

Expected Applications:
Use the setpoint module to give the operator a way to input values to be used in control strategies. Values are saved as floating point, but operator entries can be either floating point or integer. You may use as many setpoints in your project as you need. Setpoints are presented to the operator up to 10 at a time in an alphabetized list when accessed on the serial ports; or singly when accessed using the local LCD and keyboard.

ShaftEncoderInput

The **ShaftEncoderInput** module reads a pair of digital inputs and interprets pulses detected as shaft encoder outputs. DI's are sampled 256 times per sec. When enabled, this module converts the accumulated count to engineering units (EU) for the designated channel. The following shaft encoder standards are supported with the channel interpretation shown:
Type 1: quadrature type...DI1=output A, DI2=output B.
Type 2: count/direction type...DI1=count, DI2=direction (0=down, 1=up)
Type 3: count up/count down type...DI1=up counter, DI2=down counter
Type 4: short pulse count up/count down type...DI1=up counter, DI2=down counter
DI pairs must be used together, i.e., DI1 and DI2, or DI3 and DI4, etc.
To establish EU calibration, enter the value presently being measured by the encoder into a setpoint that is routed to the 'Present value' input and then trigger the 'Preset trigger' input. The module will record the value and use it and the 'EU value per increment' to calculate the EU value thereafter. Or you can also enter the initial value using the GetUserValue module.

Inputs That Must be Constants:

- DI channel 1,3,5,7…which input channel (1,3,5,7) this module uses as the first channel of the pair

Other Inputs:

- Enable…when =1, causes the module to sample and convert its inputs
- EU value per increment…floating point value corresponding to a single input pulse.
- Preset trigger…status input to force preset value to be written to output
- Preset value…floating point value to preset output when above trigger asserted
  Type…integer specifying type of shaft encoder to decode. Type 1: quadrature type...DI1=output A, DI2=output B.
  Type 2: count/direction type...DI1=count, DI2=direction (0=down, 1=up)
  Type 3: count up/count down type...DI1=up counter, DI2=down counter
  Type 4: short pulse count up/count down type...DI1=up counter, DI2=down counter. Minimum pulse width 300 microseconds.

Primary Outputs:

- Output value…Name.Val…floating point output in user defined units
- Present low (Odd) DI state…Name.Low…status output of present DI state for odd channel
- Present high (Even) DI state…Name.Hi…status output of present DI state for even channel

Outputs for Internal Use:

- Output value at midrange…Name.Mid…value from preset that corresponds to engineering units at midrange of input counter

Limitations:

Range of pulse counters is -32768 to +32768.

Expected Applications:

Use this module to convert any of the listed shaft encoder types to engineering units output.

SysSetup

The **SysSetup** module is a catchall that is used to establish system parameters such as setpoint access security code, log-on time out interval, etc. This module is installed automatically each time you start a new project. Each project should have exactly one of these modules. The entries established by this module will take place after the first RUG3 program scan following boot up. The log-on timeout interval sets the time after which log-on will be disabled automatically following the last setpoint access. The backlight timeout sets the time interval after the last keystroke at which the LCD backlight will be shut off. Any new keystroke will re-energize the backlight. Leaving the backlight setting blank will cause the backlight to be energized continuously. System flags: bit 0 ON=suppress minus key menu function on LCD display (keeps user on realtime display).

Inputs That Must be Constants: None

Other Inputs:

- Programmer sec code…integer security code to grant access to log-on protected items such as setpoints and tables
- User logon security code…integer security code to grant access to log-on protected items such as setpoints and tables. Leaving cell blank allows access at any time.
- Logon timeout sec…integer duration in seconds that log-on protected items will be allowed access after log-on code received or after last setpoint entry.
- Backlight off/on/timeout…integer: 0=backlight OFF, 1=backlight ON, >1= duration in seconds that backlight will stay on after keystroke. Leaving cell blank will keep LCD backlight on continuously.
- LCD contrast …integer contrast setting control, range is 0-255. Should be about 130.
- Loop supply 0=off, 1=on…status to control loop supply on/off.
- Loop supply 0=12v, 1=24v…status to control loop supply voltage
- Reference 0=off, 1=on…status to control loop reference voltage (4 VDC) application to DI#8. Leave this off unless you need a reference voltage to send to potentiometers.
- DI count sample 4ms…integer to control digital input sampling for counters in increments of 4 ms. Range is 0-255. Use this to slow sampling in order to avoid counting contact bounces.
- System flags…integer flags in packed integer form, blank or zero has no effect: bit 0 (LSB): ON=suppress minus key menu function on LCD display (keeps user on realtime display).

Primary Outputs:

- Boot Trigger…Name.BootTrg…status trigger to signal program's first scan after boot up
- One second trigger…Name.SecTrg…status trigger issued once per second
- Logon status…Name.Logon…status 0=no one logged on; 1=operator logged on
- Batt voltage V…Name.BattV…floating point engineering units onboard battery voltage
- Temperature F…Name.TempF…floating point onboard temperature in degrees Fahrenheit.
- Five second trigger…Name.Sec5Trg…status trigger that occurs once each 5 seconds
- Scans per second…Name.Scans…integer program scans per second
- OS revision…Name.Rev…integer echo of operating system revision, e.g., 205=version 2.05.

Outputs for Internal Use: None

Limitations:

Security codes must be less than 7 digits. Log-on and backlight timeout durations must be less than 32,768 seconds.

Expected Applications:

Used in all projects to provide commonly used services.

Math Modules

BitsToNumeric

This module converts up to 16 status bits to their equivalent integer value. Input bits are binary weighted and added to produce the result. Bit 1 is the least significant bit with a value of 1; bit 2 has a value of 2, bit 3 has a value of 4, bit 4 has a value of 8, etc. For example, if status inputs are dragged into bits 1 through 4 and bits 1 and 3 happen to be turned on (1) then the output of the module will be 1 + 4 = 5.

Inputs That Must be Constants: None

Other Inputs:
- Bit 1(LSB) through Bit16(MSB)…status inputs representing a 16 bit integer word that this module is to convert to an integer value. Blank inputs are assumed to have zero value.

Primary Outputs:
- Output…Name.Out…integer output equivalent to binary weighted sum of status inputs.

Outputs for Internal Use: None

Limitations:
Output is unsigned integer in range of -32767 to 32768; bit 16 is regarded as a sign bit.

Expected Applications:
Module is used to convert status inputs to an equivalent integer value. Also used to convert any field of statuses received in a telemetry status word to an integer value.

Constant

The Constant module makes constant available to modules in both floating and integer form. It can also be used to convert an input variable to float and integer. Input value is rounded before conversion to integer.

.

Inputs That Must be Constants: None

Other Inputs:
- Value input…floating point value to be output as both floating point and integer.

Primary Outputs:
- Output float…Name.Float…floating version of input
- Output integer…Name.Int…integer version of input

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Used to provide simple constant for use by other modules in both floating point and integer form.

<u>**Software Modules**</u>

Cosine

When enabled, the **Cosine** module calculates Y=Cos(X) where X is any value in radians and result Y is in the range of -1.0 to +1.0.

Inputs That Must be Constants: None

Other Inputs:

- Enable…status input: 1=perform calculation, 0=hold output value
- X…floating point value in radians.

Primary Outputs:

- Y…Name.Out…floating point cosine output

Outputs for Internal Use: None

Limitations: None

Expected Applications:

Used in some volume and flow rate calculations.

FloatToInteger

The **FloatToInteger** module converts an input floating point value to a pair of integers. The resulting most significant integer and least significant integer are produced by dividing the input value by a user specified divisor and saving the quotient (most significant) and remainder (least significant). In this manner, a large number, such as a flow total or large event counter can be split into two 16 bit integers for installation into a transmit telemetry array to send to another site without loss of precision. For example, if a totalization results in a value of 12345678 and that is the input to this module; and the divisor is 10,000, then the most significant result integer would be 1234 and the least significant result integer would be 5678. Both of these numbers are below the limit of 32,768 that can be installed into a 16 bit telemetry word.

Inputs That Must be Constants: None

Other Inputs:

- Input value to convert…floating point or integer value to be split into two integers
- Divisor…floating point or integer divisor

Primary Outputs:

- MS part integer…Name.MS…integer quotient, most significant result
- LS part integer…Name.LS…integer remainder, least significant

Outputs for Internal Use: None

Limitations:

Expected Applications:

Module is used to split large numbers into less significant components for use in shorter word length registers such as telemetry arrays.

FlowAGA3

This module calculates gas flow through an orifice using standard AGA3 calculation:
When enabled, calculates:
Q=0.0005167*(Default factor)*Fg*Fpb*Ftb*Ftf*Fb. Result flow Q is cu meters/day.
Default factor=Ft*Y*Fpv*Fm*Fa*Fl and should be set = 1.00 for most applications. See documentation.
In above flow calculation:
Fg=sqrt(1/G) where G=specific gravity of gas.
Fpb=pressure base factor, =1.0055 for base pressure of 101kPa.
Ftb=temperature base factor, =1.00 for base temperature of 15.0 degC.
Ftf=flowing temperature factor, =sqrt(288.7 degK/(T+273.15 degK)).
Fb=base orifice factor from AGA3 tables.

Inputs That Must be Constants: None

Other Inputs:
- Enable…status input: 1=perform calculation, 0=hold output value
- Default factor…floating point default factor in calculation (normally 1.000)
- G gas specific gravity…floating point specific gravity used to calculate specific gravity factor
- Fpb pressure base factor…floating point pressure base factor (1.0055 for base pressure of 101kPa
- Ftb temperature base factor…floating point temp base factor (1.000 for base temp of 15.0 degC
- T temperature, deg C…floating point temperature from which Ftf is calculated
- Fb base orifice factor…floating point base orifice factor from AGA3 table.

Primary Outputs:
- Q Flow CuMPerDay…Name.flow…floating point flow result in cubic meters per day

Outputs for Internal Use: None

Limitations: None

Expected Applications:

FlowCipolletiRect

This module calculates the flow through an open rectangular channel with either Cipolletti weir, rectangular weir, or rectangular weir with end contractions. Units of the calculation result can be specified as one of four types: CFS, GPM, GPS and MGD. If the calculated flow falls below the specified low flow dropout, then the output of the module is clamped at 0.0.

Inputs That Must be Constants: None

Other Inputs:
- Enable…status input, 0=hold result, 1=calculate new result
- Desired flow units (1-4)…integer to select flow units of this calculation: 1=CFS, 2=GPM, 3=GPS, 4=MGD
- Low flow dropout…floating point engineering units flow value below which the flow will be regarded as 0.0.
- Fluid level, ft…floating point level in feet of fluid above bottom of weir.
- Crest length, ft…floating point width in feet of bottom of weir.
- Type, 1-3…integer type of weir, 1=Cipolletti, 2=rectangular to width of channel, 3=rectangular with end contractors

Primary Outputs:
- Flow in user units…Name.flow…floating point flow result in specified units

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Provides accurate flow calculation for selected weir.

FlowContainer

The **FlowContainer** module calculates flow based on the time it takes to fill or empty a tank of specified cross section and height. It has two modes of operation. 1) if there is a tank level transducer, the module calculates the flow as the tank is emptying or filling and any local pump is not running, based on the changing tank level. If there is no tank level transducer, the module calculates flow based on the time it takes to empty/fill the tank during pump off time, calculating volume from area and span between pump call and off setpoints. The module holds the flow constant during any local pumping.

Inputs That Must be Constants: None

Other Inputs:
- Tank level, ft…floating point tank level from tank transducer, if any. If no tank level transducer, leave blank.
- Tank area, ft…floating point tank cross sectional area, sq. ft.
- Pump running status…status indicating running status of any pump filling (pump up) or emptying (pump down) the tank.
- Delay sec after pump off…floating point delay time after pump off detected before calculation of flow rate to give time for tank level to settle.
- Tank level span, ft…floating point span between pump call and off setpoints for case where there is no tank level transducer.

Primary Outputs:
- Flow rate, CFS…Name.CFS…flow rate in CFS calculated from tank geometry and pump cycling time
- Flow rate, GPM…Name.GPM… flow rate in GPM calculated from tank geometry and pump cycling time

Outputs for Internal Use:
- Timer…Name.Tmr…integer internal pump cycling timer, 1.0 sec.
- Temp level…Name.Tmp…floating point level latch for rise/fall calculation
- Old pump run state…Name.Oldrun…status image of last pump run state

Limitations:

Expected Applications:
Module enables you to calculate flow into or out of a tank based on pump cycling and tank geometry when there is no flow meter to measure tank flow.

FlowConvert

This module converts flow in CFS to other units (CFS, GPM, GPS, MGD) and also applies low flow dropout to clamp flow to zero if the input flow value is below a user defined value.

Inputs That Must be Constants: None

Other Inputs:
- Enable…status input, 0=hold result, 1=calculate new result
- Flow input…floating point flow input value in CFS
- Desired flow units (1-4)…integer to select output flow units, 1=CFS, 2=GPM, 3=GPS, 4=MGD
- Low flow dropout user units…floating point threshold in output engineering units below which the output of the module will be set to 0.0

Primary Outputs:
- Flow in user units…Name.flow…floating point output flow in selected units

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Use to supply low flow dropout function, or flow units conversion function or both.

FlowHFlume

This module calculates the flow through an open rectangular H-type flume. Units of the calculation result can be specified as one of four types: CFS, GPM, GPS and MGD. If the calculated flow falls below the specified low flow dropout, then the output of the module is clamped at 0.0.

Inputs That Must be Constants: None

Other Inputs:
- Enable…status input, 0=hold result, 1=calculate new result
- Desired flow units (1-4)…integer to select flow units of this calculation: 1=CFS, 2=GPM, 3=GPS, 4=MGD
- Low flow dropout…floating point engineering units flow value below which the flow will be regarded as 0.0.
- Fluid level, ft…floating point level in feet of fluid above bottom of flume.
- Flume width, ft…floating point width in feet.

Primary Outputs:
- Flow in user units…Name.flow…floating point flow result in specified units

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Provides accurate flow calculation for selected flume.

FlowPalmerBowlus

This module calculates the flow through a Palmer-Bowlus flume. Units of the calculation result can be specified as one of four types: CFS, GPM, GPS and MGD. If the calculated flow falls below the specified low flow dropout, then the output of the module is clamped at 0.0.

Inputs That Must be Constants: None

Other Inputs:
- Enable…status input, 0=hold result, 1=calculate new result
- Desired flow units (1-4)…integer to select flow units of this calculation: 1=CFS, 2=GPM, 3=GPS, 4=MGD
- Low flow dropout…floating point engineering units flow value below which the flow will be regarded as 0.0.
- Fluid level, ft…floating point level in feet of fluid above bottom of flume.
- Flume width, in…floating point flume width in inches.

Primary Outputs:
- Flow in user units…Name.flow…floating point flow result in specified units

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Provides accurate flow calculation for selected flume.

FlowParshallLADWP

This module calculates the flow through a Parshall flume of specified widths. Units of the calculation result can be specified as one of four types: CFS, GPM, GPS and MGD. If the calculated flow falls below the specified low flow dropout, then the output of the module is clamped at 0.0. Allowed flume widths in ft are: 0.5, 0.75, 1.0, 2.0-8.0 (any value), 10, 12, 15, 20, 25, 30, 40, 50.

Inputs That Must be Constants: None

Other Inputs:
- Enable…status input, 0=hold result, 1=calculate new result
- Desired flow units (1-4)…integer to select flow units of this calculation: 1=CFS, 2=GPM, 3=GPS, 4=MGD
- Low flow dropout…floating point engineering units flow value below which the flow will be regarded as 0.0.
- Fluid level, ft…floating point level in feet of fluid above bottom of flume.
- Flume width, in…floating point flume width in inches.

Primary Outputs:
- Flow in user units…Name.flow…floating point flow result in specified units

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Provides accurate flow calculation for selected flume.

FlowQ=A*(H+B)**C

This module calculates the flow through an open channel whose flow can be calculated by the general purpose equation Q=A*(H+B)**C. In this equation A, B and C are constants related to the geometry of the channel and H is gauge height in feet. Q is flow in CFS.

Inputs That Must be Constants: None

Other Inputs:

- Enable…status input, 0=hold result, 1=calculate new result
- H…floating point gauge height in feet
- A, B, C…floating point values characterizing the channel. C can be non-integer.
- Low flow dropout…floating point engineering units flow value below which the flow will be regarded as 0.0.

Primary Outputs:

- Flow in user units…Name.flow…floating point flow result in specified units

Outputs for Internal Use: None

Limitations:
C must be greater than zero; or, if it is negative, the quantity (H+B) must be positive.

Expected Applications:
Provides accurate flow calculation for selected flume.

FlowTrapezFlume

This module calculates the flow through a Trapezoidal flume of one of three types: 1) large 60 degree, 2) 45 degree WSC, or 3) 45 degree SRCRC. Units of the calculation result can be specified as one of four types: CFS, GPM, GPS and MGD. If the calculated flow falls below the specified low flow dropout, then the output of the module is clamped at 0.0.

Inputs That Must be Constants: None

Other Inputs:

- Enable…status input, 0=hold result, 1=calculate new result
- Desired flow units (1-4)…integer to select flow units of this calculation: 1=CFS, 2=GPM, 3=GPS, 4=MGD
- Low flow dropout…floating point engineering units flow value below which the flow will be regarded as 0.0.
- Fluid level, ft…floating point level in feet of fluid above bottom of flume.
- Flume Type (1-3)...integer type specifier: 1=large 60 degree, 2=45 degree WDC, 3=45 degree SRCRC.

Primary Outputs:

- Flow in user units…Name.flow…floating point flow result in specified units

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Provides accurate flow calculation for selected flume.

Limit

The **Limit** module range tests and limits a value to the range set by the upper and lower setpoints specified, then outputs the range-limited result.

Inputs That Must be Constants: None

Other Inputs:
- Enable…status input, 0=hold result, 1=calculate new result
- Input variable X…floating point variable whose range is to be limited.
- Upper limit U…floating point upper limit X is not to exceed
- Lower limit L…floating point lower limit X is not to fall below

Primary Outputs:
Output Y…Name.LIM…Floating point result clamped to the range of Y=L<=X<=U.

Outputs for Internal Use: None

Limitations:

Expected Applications:
Used to range check floating point and integer variables. Note that the original variable X is unchanged.

LowPassFilter

The LowPassFilter module applies a low pass filter algorithm to slow the response of the output to changes in the input, then range tests and limits the value to the range set by the upper and lower setpoints specified, then outputs the range limited result. The longer the low pass filter time constant, the more slowly the output will follow the input and the more immune the output will be to transients on the input.

Inputs That Must be Constants: None

Other Inputs:
- Enable…status input, 0=hold result, 1=calculate new result
- Input value…floating point variable whose value is to be low pass filtered and whose range is to be limited.
- Filter time const sec…integer filter time constant, seconds…typical 5 to 30 seconds
- High output limit…floating point upper limit output is not to exceed
- Low output limit…floating point lower limit output is not to fall below

Primary Outputs:
Filtered output…Name.Out…Floating point low pass filtered result clamped between high and low limits

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Used to slow a signal's response to transients and clamp its value.

MaskInteger

This module logically AND's an input integer with a mask word to isolate selected bits. The mask must be entered in decimal. To mask (set to zero) all bits except certain ones, the mask must be the sum of the decimal equivalents of the bits you wish to let pass through. For example, to allow the input integer's bits 2, 3 an 4 to be passed to the output, a mask value of 14 would be required. The following table gives binary to hexadecimal to decimal equivalents for 16 bit words:

Table 5 Binary to Hexadecimal to Decimal Conversions

BINARY	HEXADECIMAL	DECIMAL
0000000000000001	$1	1
0000000000000010	$2	2
0000000000000100	$4	4
0000000000001000	$8	8
0000000000010000	$10	16
0000000000100000	$20	32
0000000001000000	$40	64
0000000010000000	$80	128
0000000100000000	$100	256
0000001000000000	$200	512
0000010000000000	$400	1024
0000100000000000	$800	2048
0001000000000000	$1000	4096
0010000000000000	$2000	8192
0100000000000000	$4000	16384
1000000000000000	$8000	32768

Inputs That Must be Constants: None

Other Inputs:
- Input integer…integer to be masked
- Mask (decimal)…integer mask to be ANDed with the input integer

Primary Outputs:
- Output integer…Name.MskInt…result of (Input Integer AND Mask).

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Used to isolate individual bits or multiple bit fields enabling use of portions of integers in control strategies.

NumericToBits

This module splits an integer value into its constituent bits, providing 16 single status outputs from an integer input.

Inputs That Must be Constants: None

Other Inputs:
- Enable…status input, 0=hold result, 1=calculate new result
- Integer to convert…Integer input to split into bits

Primary Outputs:
- Bit 1 LSB…Name.B1…status output echoing the least significant bit of the input integer
- Bit 2…Name.B2…status output echoing the next to the least significant bit of the input integer
- ….
- Bit16 MSB…Name.B16…status output echoing the most significant bit of the input integer

Outputs for Internal Use: None

Limitations: Only works for 16 bit integers or floating point values <65536

Expected Applications:
Used to unpack bits stored or transmitted as integers.

NumericToString

The numeric to string module provides a way whereby you can cause the RUG3 to format a floating point numeric value in specific ways, including controlling number of leading blanks, leading zeroes, trailing blanks, and trailing zeroes. The string output appears in the string database for use in other modules that require string inputs, such as the **SendStringtoPort** module. In the inputs to this module you specify the variable whose value is to be converted, and also the format. The format consists of a floating point number such as 7.3, to specify how many characters are to appear to the left and right of the decimal. The value 7.3 would specify 7 places to the left of the decimal point, and 3 places to the right of the decimal. You also specify how to fill the areas to the left and right of the numeric string result. Your choices are 0) no fill, 1) fill with blanks, or 2) fill with zeroes. For example, if the value to be converted is 1234.5 and you specified a format of 7.3, with zeroes to fill to the left, then the resulting string would be "0001234.500".

Inputs That Must be Constants:
- Max output characters…integer constant that defines how long the longest string will be produced by the module. Commonly set to 32.

Other Inputs:
- Enable…status input, 0=hold result, 1=calculate new result
- Input value to convert…floating point or integer value to convert to specified format.
- Format…floating point value such as 4.3 to specify number of characters of precision to be shown to the right and left of the decimal. The value 4.3 would specify that the output string would be formatted with four places to the left of the decimal and three places to the right of the decimal.
- Leading fill flag…integer: 0=no fill, 1=fill with blanks, 2=fill with 0's.

Primary Outputs:
- Output string…Name.Strg…string version of floating point input in specified format.

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Use to format numbers into specific formats for transmission to other devices such as other RTU's or output devices.

PackValues

The PackValues module will pack up to four values into a single 16 bit integer. This is useful for shrinking telemetry messages and logged data record sizes. For each input, the module adds an offset and then multiplies the input floating point or integer value by the multiplier, then packs the result into a single integer using the bits allocated for each input. Input #1 would be placed in the least significant bits of the output integer. Input #2 would be placed in the next most significant bits, etc. The following numbers of bits allocated to each input give the following potential ranges:

Allocate 1 bit=0-1	allocate 6 bits=0-63	allocate 11 bits=0-2047
Allocate 2 bits=0-3	allocate 7 bits=0-127	allocate 12 bits=0-4095
Allocate 3 bits=0-7	allocate 8 bits=0-255	allocate 13 bits=0-8191
Allocate 4 bits=0-15	allocate 9 bits=0-511	allocate 14 bits=0-16383
Allocate 5 bits=0-31	allocate 10 bits=0-1023	allocate 15 bits=0-32767

Inputs whose value is below zero will have all bits set to zero; inputs whose value exceeds the maximum value of its allocated bits will have all bits set to 1.

Inputs That Must be Constants: None

Other Inputs:
- Enable…status input, 0=hold result, 1=calculate new result
- Input #1-4 to convert…status, integer or floating point number to be packed into the output integer. The first blank entry marks the end of the input list.
- Offset #1-4…floating point offset that is added to the input before packing.
- Multiplier #1-4…floating point multiplier that multiplies the input+offset to create the final value to be packed.
- Bits allocated to #1-4…integer number of bits in range of 1 to 15 that specifies how many bits in final result are allocated for this measurement

Primary Outputs:
- Output integer…Name.Int…16 bit integer that is the packed result of all (up to 4) inputs.

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Use this module to pack values together to reduce telemetry size or to reduce logged record size.

PolynomialNthOrder

The polynomial module calculates up to a 12-th order polynomial result based upon the coefficients, the order designator and an input X. The polynomial equation is Y=a+b*X+c*X^2+d*X^3…+m*X^12.

Inputs That Must be Constants: None

Other Inputs:

- Enable…status input, 0=hold result, 1=calculate new result
- Input X…floating point input
- N…integer setting the order of the calculation
- a…m…floating point coefficients of the polynomial

Primary Outputs:

- Y…Name.Out…floating point result of the polynomial calculation

Outputs for Internal Use: None

Limitations: None

Expected Applications:

Used to specify a general purpose function not definable by other means, such as for linearizing transducers among other applications.

Sine

When enabled, the **Sine** module calculates Y=Sin(X) where X is any value in radians and result Y is in the range of -1.0 to +1.0.

Inputs That Must be Constants: None

Other Inputs:

- Enable…status input: 1=perform calculation, 0=hold output value
- X…floating point value in radians.

Primary Outputs:

- Y…Name.Out…floating point sine output

Outputs for Internal Use: None

Limitations: None

Expected Applications:

Used in some volume and flow rate calculations.

SuccessiveSampleFilter

When triggered, the **SuccessiveSampleFilter** module tests the last N samples collected. If all are within the deadband of each other, then the last sample is sent as output. Otherwise, the output is unaffected. Use to filter out intermittent erroneous values.

Inputs That Must be Constants: None

Other Inputs:

- Input value…floating point input
- Trigger save sample and test…status input, 0=hold result, 1=calculate new result
- N samples to test (2-10)…integer setting the number of samples to save and use to calculate result
- Deadband…floating point value that sets the range over which a new value and collected samples must be within before the new value will be sent to the output

Primary Outputs:

- Filtered output…Name.Out…floating point output. This will be the new sample if the new sample and all collected samples are within the deadband of each other.

Outputs for Internal Use:

- Index to samples…Name.Idx…integer index to collected samples
- Sample (1-10)…Name.S1-S10…floating point samples collected

Limitations: None

Expected Applications:

Use this module to sample floating point values and reject spikes in values.

SummingAccum

When triggered, the **SummingAccum** module adds a new input value to the accumulated value. Preset trigger installs the preset value into accumulator. Output trigger indicates when new value has been added. This is a form of totalizer that lets you control the sampling.

Inputs That Must be Constants: None

Other Inputs:

- Trigger add new value…status input, 0=hold result, 1=add input value to accumulator
- New value to accumulate…floating point input
- Trigger preset accumulator…status that will install a preset value into the accumulator
- Preset value…floating point value that is transferred to accumulator output when above trigger asserted

Primary Outputs:

- Accumulator value…Name.Accum…floating point output. This will be the sum of all values present when the trigger was asserted.
- Trigger have new value…Name.NewTrg…Status trigger asserted after a new value has been added to the accumulator.

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Use this module to totalize values at irregular intervals or when events occur.

Tangent

When enabled, the **Tangent** module calculates Y=Tan(X) where X is any value in radians and result Y can have any value in the floating point range.

Inputs That Must be Constants: None

Other Inputs:
- Enable…status input: 1=perform calculation, 0=hold output value
- X in radians…floating point value in radians.

Primary Outputs:
- Y…Name.Out…floating point tangent output

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Used in some volume and flow rate calculations.

TrigToNumeric

This module accepts up to 16 status or trigger inputs and outputs the numeric position in the list of the first one that is true. For example, if input number 5 is on and all others are off, then the module's output value would be 5. That value would remain latched in the output until another input turns on.

Inputs That Must be Constants: None

Other Inputs:
- Bit 1 LSB…status input which would give an output value of 1 if turned on
- Bit 2…status input which would give an output value of 2 if turned on
- …
- Bit16 MSB…status input which would give an output value of 16 if turned on

Primary Outputs:
- Output…Name.Out…integer output with value of lowest numbered input that is on; or last one that was on

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Used as an addressable latch to latch the last trigger to turn on.

Miscellaneous Math Modules

The math modules listed below take an input value X and apply the respective calculations on it to produce a floating point result Y. Disallowed calculations such as negative numbers to fractional powers, or divide by zero will return zero. The modules are:

Y=X^Z	(Power)
Y=sqrt(X)	(Square root)
Y=A*B	(Simple product)
Y=A*B*C*D*E*F*G*H*J	(Product)
Y=A*B+C*D+E*F+G*H	(Sum of products)
Y=A/B	(Simple quotient)
Y=A+B	(Simple sum)
Y=A+B*C/D-E	(Misc calculation)
Y=A+B*e^(X+C)	(Exponential)
Y=A+B*rand(1)	(Random number generator)
Y=A+B+C+D+E+F+G+H	(Sum of terms)
Y=A+B+C+D-E-F-G-H	(Sum of four terms-sum of four terms)
Y=A-B	(Simple difference)
Y=abs(X)	(Absolute value)
Y=log(X)	(Natural logarithm)
Y=log10(X)	(Common logarithm)
Y=M*X+B	(Rescaling with offset)

Inputs That Must be Constants: None

Other Inputs:
- Enable…status input, 0=hold result, 1=calculate new result
- All inputs are floating point. Integer or status inputs are allowed.

Primary Outputs:
- Y…Name.Out…floating point result.

Outputs for Internal Use: None

Limitations:
- Arguments of sqrt, log and log10 must be positive. Divisors must be nonzero. Any violations will result in zero result.

Expected Applications:
General purpose math.

UnpackToFloat

The UnPackToFloat module unpacks a designated portion of a 16 bit input integer into a floating point output. The module isolates the designated bits as a zero-based unsigned integer then multiplies that value by the multiplier, and finally adds the offset to produce the floating point output. LSbit to include designates the least significant bit in the input integer that is to be in the result field. Similarly, the MSbit to include designates the most significant bit in the input integer to be included in the result. For example, if the input integer has the following bits: (MS) 0001110111100101 (LS) and the LS bit to include is 2, and the MS bit to include is 5, then the field extracted would be 1001 or a value of 9. That would be multiplied by the multiplier and then the offset would be added to create the output result. You will need one of these modules to unpack each of the (up to 4) values packed by the PackValues module.

Inputs That Must be Constants: None

Other Inputs:
- Enable…status to enable the unpacking process. Blank or non-zero value enables unpacking.
- Input integer to unpack…integer containing one or more values to be unpacked.
- LS bit to include (0-15)…integer specifying which bit of the input integer constitutes the least significant bit of the value to be unpacked.
- MS bit to include (0-15)…integer specifying which bit of the input integer constitutes the most significant bit of the value to be unpacked.
- Multiplier…floating point multiplier that multiplies the bits extracted from the input integer.
- Offset…floating point offset that is added to the multiplied value to produce the final output value.

Primary Outputs:
- Floating point value…Name.FloatVal…Floating point result that is reconstruction of original packed value.

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Use this module to recover value packed into an integer by the PackValues module.

UnpackToInt

The UnPackToInt module unpacks a designated portion of a 16 bit input integer into an integer output. The module isolates the designated bits as a zero-based unsigned integer then multiplies that value by the multiplier, and finally adds the offset to produce the integer output. LSbit to include designates the least significant bit in the input integer that is to be in the result field. Similarly, the MSbit to include designates the most significant bit in the input integer to be included in the result. For example, if the input integer has the following bits: (MS) 0001110111100101 (LS) and the LS bit to include is 2, and the MS bit to include is 5, then the field extracted would be 1001 or a value of 9. You will need one of these modules to unpack each of the (up to 4) values packed by the PackValues module.

Inputs That Must be Constants: None

Other Inputs:
- Enable…status to enable the unpacking process. Blank or non-zero value enables unpacking.
- Input integer to unpack…integer containing one or more values to be unpacked.
- LS bit to include (0-15)…integer specifying which bit of the input integer constitutes the least significant bit of the value to be unpacked.
- MS bit to include (0-15)…integer specifying which bit of the input integer constitutes the most significant bit of the value to be unpacked.

Primary Outputs:
- Integer output…Name.IntVal…Integer result that is reconstruction of original packed value.

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Use this module to recover value packed into an integer by the PackValues module.

Control Modules

AlarmHiLo

The **AlrmHiLo** module compares a floating point value with high and low alarm setpoints and, if the value exceeds the high setpoint for the period of a specified delay, turns on its high alarm output. If the input value falls below the low alarm setpoint for the specified delay, then the low alarm output is turned on. If its delay is unspecified, then no delay will be applied. If the delay is specified and is set to zero, then the alarms will be disabled. Otherwise, the alarm must be true for the specified delay period before the output will turn on. If the alarm subsequently falls back within the setpoints, the status outputs will turn off without delay.

Inputs That Must be Constants: None

Other Inputs:
- Enable…status input, 0=hold result, 1=calculate new result
- Input level…floating point value to be compared against a setpoint.
- High alarm setpoint…floating point setpoint against which the input is compared to determine if a high alarm condition is present.
- Low alarm setpoint…floating point setpoint against which the input will be compared to determine if a low alarm condition is present.
- Delay, sec…integer period in seconds that the setpoint must be exceeded to declare an alarm. If the delay is zero, then the alarms are disabled.

Primary Outputs:
- High alarm output…Name.HiAlrm…status that becomes true if high alarm declared by comparison
- Low alarm output…Name.LoAlrm…status that becomes true if low alarm declared by comparison

Outputs for Internal Use:
- Delay timer…Name.Timer…integer countdown delay timer counting seconds

Limitations: None

Expected Applications:
Used to detect and alarm when a field value such as tank level, flow, temperature, etc. goes too high or low.

ANDgate

Performs the logical AND of the 8 status inputs. If all specified status inputs are ON (1 is on AND 2 is on AND 3 is on…) then the module's true status output (ANDout) will be on. If any of the status inputs is OFF, then the true status output will be off. Any unused status inputs, i.e., any that are left blank, will be ignored. The ANDbar output is the complement of the true output, i.e., it will be off when ANDout is on, and it will be on when ANDout is off.

Inputs That Must be Constants: None

Other Inputs:

- Enable…status input, 0=hold result, 1=calculate new result
- Input #1…status input to be ANDED with others
- Input #2…status input to be ANDED with others
- …
- Input #8…status input to be ANDED with others

Primary Outputs:

- AND output…Name.ANDout…status output that will be true only if all specified inputs are ON
- Inverted output…Name.ANDbar…status output that will be true if any specified input is OFF

Outputs for Internal Use: None

Limitations: None

Expected Applications:

Use the AND module to logically combine other statuses to detect when they are all on. Also use the AND module to invert a status (using the ANDbar output).

BackspinTimer

The BackspinTimer input status will be passed to the output status if the timer has timed out. Otherwise, the output will be unaffected unless a force input is asserted. Force ON status true will force the output ON independent of the timer. Force OFF status true will force the output OFF independent of the timer. When the output switches state, the timer will be restarted and output changes demanded by the main input status will be prohibited until the timer times out. Force timer restart true will restart the timer based on external event. Time delay has one second resolution and 32767 seconds range.

Inputs That Must be Constants: None

Other Inputs:

- Input status…status to be transferred to the output only after the timer has timed out.
- Time delay, sec…integer time delay in seconds.
- Force ON status…if true, forces the output on and restarts the timer.
- Force OFF status…if true, forces the output off and restarts the timer.
- Force timer restart…if true, forced the timer to restart its count down from the time delay input.

Primary Outputs:

- Output status…Name.Run…status that echoes the input status once the timer has timed out.

Outputs for Internal Use:
- Timer…Name.Tmr…integer down counter that counts down one count per second.

Limitations:
Count range is 0 to +32767 seconds.

Expected Applications:
Use to delay an output change until delay seconds after the previous change. This is useful to keep on output that controls say, a pump, from switching on/off too frequently.

ClearMemory

When trigger=true, the ClearMemory module erases all user memory (RAM), i.e., sets all RAM contents to zero. Note that this will erase all setpoints, logged values and totals. Program memory (FLASH) will be unchanged.

Inputs That Must be Constants: None

Other Inputs:
- Trigger erase RAM…status trigger input that when true will set all user memory to zero.

Primary Outputs: None

Outputs for Internal Use: None

Expected Applications:
Use this module to set all RAM to zero so that setpoints, totals and telemetry values upon a startup of a program will have known, zero values.

Counter

The **Counter** module is a general purpose up counter for counting events. It can also function as a sequencer. The counter increments its count at each OFF to ON transition of its status input. The counter is a signed 16 bit integer, so it can have negative as well as positive values. The module's preset trigger will cause the preset value to be installed as the new count when triggered.

Inputs That Must be Constants: None

Other Inputs:
- Enable…status input, 0=hold result, 1=calculate new result
- Status input…status whose low to high transitions are to be counted
- Preset trigger…status input which, when true, will cause the preset value to be installed as the present count
- Preset value…integer that will be installed as the count when the preset trigger is true.

Primary Outputs:
- Count…Name.Cnt…integer value presently held by counter

Outputs for Internal Use:
- Old input…Name.Old…copy of last status input to detect rising edge

Limitations:
Count range is –32768 to +32767.

Expected Applications:
Use for general event counting and sequencing.

CounterUpDNRollover

The **CounterUpDnRollover** module is a general purpose up/down counter for counting events and sequencing. It can also function as a sequencer. When enabled, the counter will increment its count at each OFF to ON transition of its count up trigger status input; and will decrement its count on each OFF to ON transition of its count down trigger status input. The counter is a signed 16 bit integer, so it can have negative as well as positive values. The module's four preset triggers will cause the associated preset value to be installed as the new count when triggered. If the counter hits its max or min count value, it will either stop counting (if mode=0 or blank) or roll over/back (if mode=1).

Inputs That Must be Constants: None

Other Inputs:
- Enable input…status that, when false, prohibits counting
- Count up trigger…input status that will increment the counter state on an OFF to ON transition
- Count down trigger…input status that will decrement the counter state on an OFF to ON transition
- Max count…integer highest state value allowed by counter
- Min count…integer lowest state value allowed by counter
- Mode…integer mode setting: 0=stop when hit min or max count, 1= rollover when hit min or max count
- Preset triggers(4)…status inputs which, when true, will cause the associated preset value to be installed as the present state
- Preset value…integer that will be installed as the state when the corresponding preset trigger is true.

Primary Outputs:
- Counter…Name.Count…integer value presently held by counter

Outputs for Internal Use:
- Copy of triggers…Name.Copy…copy of last status input to detect rising edges

Limitations:
Count range is –32768 to +32767.

Expected Applications:
Use for general event counting and sequencing.

Deadband

The **Deadband** module compares an input floating point value with a setpoint and a deadband value. If the input value is within the deadband of the setpoint, either above or below, then the module declares that the value is within the deadband. If not, then the module declares that the value is outside the deadband. If the value is above the (value + deadband), then the module will also declare that value is above the deadband. Also, if the value is below the (value-deadband), then the module will declare that the value is below the deadband.

Inputs That Must be Constants: None

Other Inputs:
- Enable…status input, 0=hold result, 1=calculate new result
- Input value…floating point value that is being tested.
- Setpoint…floating point value at center of deadband
- Deadband…floating point value that is added to and subtracted from the setpoint in making the deadband tests

Primary Outputs:
- Within deadband…Name.Inside…status output that will be true if the input value is closer to the setpoint than the deadband value.
- Outside deadband…Name.Outside…status output that will be true if the input value is either larger than the setpoint + deadband, or smaller than the setpoint – deadband.
- Above deadband…Name.Above…status output that will be true if the input value is larger than the setpoint + deadband.
- Below deadband…Name.Below…status output that will be true if the input value is smaller than the setpoint – deadband.

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Used to detect that an analog value is outside a desired value and needs to be adjusted.

DelayTimer

Accepts an input trigger, delays for a user specified delay period, and than issues a new trigger. Basically, this module implements a way to delay triggers. The module also provides a status output while the timer is running, so it can be used to generate a variable pulse width status. Timer resolution is 1.0 second. Timer range is 32767seconds.

Inputs That Must be Constants: None

Other Inputs:
- Preset trigger…trigger status input to begin delay and preset the delay timer to the delay value.
- Preset value sec…floating point input to set time delay

Primary Outputs:
- Timer running…Name.Run…status output indicating that the delay timer is running
- Timer done trigger…Name.Trig…status trigger output that goes true when timer is done
- Timer…Name.Tmr… integer countdown delay timer counting each second

Outputs for Internal Use: None

Limitations:None

Expected Applications:
Use this module to delay triggers as necessary to assure that other functions are ready before the trigger is ready. Also use it to generate variable pulse width pulses.

EORgate

Exclusive OR gate is used to detect when two status inputs match and when they mismatch. The exclusive OR gate's true output (EORout) turns ON if the two status inputs are different; and turns OFF if the two inputs are the same. The inverted output (EORbar) is the inverse of the true output. The EORgate can also be used to selectively invert or not invert a status by feeding the status to one of the EORgate inputs and using a second status to select whether the status is to be passed unchanged, or inverted. If the control status is OFF, then the first status is passed unchanged; if the control status is ON, then the first status is passed inverted.

Inputs That Must be Constants: None

Other Inputs:
- Input #1…status input to EOR gate
- Input #2…status input to EOR gate

Primary Outputs:
- EOR output…Name.EOR…exclusive OR of two status inputs
- Inverted output…Name.EORbar…exclusive NOR or two status inputs

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Use to test status mismatches and to selectively invert/not invert a status input.

EventFIFOQueue

The **EventFIFOQueue** module accepts event request statuses and queues them up in the order received, and then enables their corresponding outputs singly in the order received. An event is active as long as its status input is true. When its status input reverts to false, the module will turn that event's output off, advance the queue to the next event that was received and enable that event's status output. The module will also advance the queue and enable the next output when the override timer times out, or when the trigger to advance the queue is true. The purge queue trigger clears entire queue. If an event becomes false before advancing to the output, it will remain in the queue until it is the oldest in the queue, then will be active for one scan. If an event is in the queue already, a new copy will not be added to the queue.

This module is useful for sequencing equipment in an order based on the need for service rather than on any predetermined order. For example, a collection of filters could be backwashed in the order that their differential pressures indicate that they have become clogged. As each filter's differential pressure exceeds a threshold, its 'need to backwash' status would be set, which would be the event status input to this module. This module would then turn on an output bit for the filter that indicated it needed backwash first. At the end of that filter's backwash cycle, the 'need to backwash' bit would be turned off by other logic. This module would then turn off that filter's backwash bit and turn on the backwash bit for the next filter that indicated that it needed to be backwashed. This would continue until all filters that needed service had been backwashed.

Inputs That Must be Constants: None

Other Inputs:

- Enable…status input that when blank or true enables the module's functions; when false, disables the module.
- Override time delay sec…integer delay in seconds beyond which an active event will be abandoned and the queue will advance to the next event.
- Event 1-8 statuses…status inputs that indicate the need for an event to be added to the queue. As each status turns on, its event number (1-8) is added to the event queue to be processed.
- Purge queue trigger…status input that when 1 causes the module to clear the queue and turn off all event outputs.
- Advance queue trigger…status input that will force the existing event to be abandoned and the next one in the queue to become the active event output.

Primary Outputs:

- Status 1-8 outputs…Name.S1-S8…status outputs one of which will be true corresponding to the oldest event requested and present in the queue.

Outputs for Internal Use:

- Queued value 1-8…Name.V1-V8…actual queue of event numbers in the order received as status inputs.
- Override timer…Name.Tmr…integer timer with range of 0-32767 seconds to time out and override an event that would stall the sequencer by never reverting to false state.
- Image of inputs…Name.Img…integer packed image of input statuses to detect new events.

Limitations: None

Expected Applications:

Use to control multiple events that must be processed in the order received.

EventLogger

The EventLogger Module specifies up to 10 event statuses, each of which can trigger a log entry along with an associated information string and optional floating point value. Each event is logged with time tag, event string, optional value, ON/OFF state and index. Index, for use by DNP3, is the sum of the base index and the event # (1-10). Also, for DNP3, if no analog value specified then the status will be reported, otherwise, the analog value will be reported in DNP3 messages. The analog report code specifies, for R9 format log dumping, the handling of analog values:
0=no analog report, 1=report as 4-byte float, 2, 3=spares, 4=mult by 10,000, 5=mult by 1,000, 6=mult by 100, 7=mult by 10, 8=mult by 1, 9=mult by 0.1, 10=mult by 0.01, 11=mult by 0.001, 12=mult by 0.0001, 13=mult by 0.00001, 14,15=spares. States 4 thru 13 designate that the value be multiplied by the specified multiplier then be sent as a 2 byte signed integer.
Example specifying string: Tank level=@@.@@ feet, high alarm {ON/NORMAL} would result in the following logged message: "07/14/07 19:45:05 Tank level=19.26 feet, high alarm ON". Mode 0: log all ON events; mode 1: log all ON/OFFevents. The 'Which logger to use' entry must specify the '.Log' output of an EventLogSetup module. Multiple EventLogger modules (this module) can reference the same EventLogSetup module. Each event occupies 64 bytes of EEPROM. The Event X string entries can have a maximum length of 50 bytes. Max string length for display on the local LCD is 16 bytes.

You may have as many event loggers as you need. Each event logger must point to an event log as described below. The event log is established by an **EventLogSetup** module, which sets aside space for the logged events. See **EventLogSetup** below. You may have multiple **EventLogger** modules sending events to the same **EventLogSetup** module. You may also have multiple **EventLogSetup** modules, with each establishing independent event logs. For example, you could have one event log for alarms, and another for pump switching activity.

Specifying Which Logger to Use and Specifying Mode

The first input property of the **EventLogger**, titled "which logger to use", must point to an event log as established by an **EventLogSetup** module. You establish this connection by simply dragging into the first property the **EventLogSetup** module's ".Log" output from the integer data base. For example, if you have set up an **EventLogSetup** module with the name "AlarmList", then all you have to do is drag from the integer database the name "AlarmList.Log". This is actually a pointer to the beginning of the event log.

The second **EventLogger** input property specifies whether the logger is to log turn-on events only, or both turn-on and turn-off events. Leaving the input blank or setting it to zero will cause the module to log an event whenever a status turns on. Setting the input to one causes the module to log both on and off events.

Specifying Statuses and Messages

For each status you want to log, you must set up three properties: 1) you must drag in the status from the status data base; 2) you must supply a corresponding message or string to be logged; and 3) you may optionally supply a value to be logged as part of the message. When a status changes state, the logged message will consist of a time and date string, a user entered string identifying the point with the change of state, an optional analog value, and an ON or OFF indication. An example of the logged message format is the following:

10/27/1999 14:29:05 Richfield tank level=12.34 feet, high alarm ON.

The time and date string is supplied by the system and its format is fixed as MM/DD/YYYY HH:MM:SS. The user-entered string is entered in the same format as that of RUG9 display lines, with the exception of the trailing ON/OFF indication. To produce the above logged message, you would enter the message string as: "Richfield tank level=@@.@@ feet, high alarm ". The optional "@@.@@" field specifies the placement and format of an optional floating point value to be included in the message. With the message as in the example above, the system will append either ON or OFF indicating the new status state. If you append a true/false string field in braces to the end of the string, the logger will use your choices in place of "ON/OFF". For example, if you append "{FAIL/NORMAL}" to the message, then the system will use "FAIL" when the status turns on and "NORMAL" when the status turns off. The above message string becomes:

"Richfield tank level=@@.@@ feet, high alarm {FAIL/NORMAL}".

You simply specify your ON and OFF choices separated with a forward slash "/" character.

Inputs That Must be Constants: None

Inputs That Must be Pointers:
- Which logger to use…integer input must point to an event logger's Name.Log output in the integer database.

Other Inputs:
- Mode (0=On, 1=On/Off)…integer input specifies operating mode: 0=log ON events only, 1=log both ON and OFF events
- Enable…status that when true enables logging; when false inhibits logging.
- Base index…integer offset that is added to the input event number (1-10) and saved in the log for use by DNP3 and RUG9 format log dumping.
- DNP3 class…integer specifying inclusion in one of three DNP3 classes.
- Analog report code (0-15)…integer that specifies how the analog value is to be reported in RUG9-formatted reports: 0=no analog report, 1=report as 4-byte float, 2, 3=spares, 4=mult by 10,000, 5=mult by 1,000, 6=mult by 100, 7=mult by 10, 8=mult by 1, 9=mult by 0.1, 10=mult by 0.01, 11=mult by 0.001, 12=mult by 0.0001, 13=mult by 0.00001, 14,15=spares. States 4 thru 13 designate that the value be multiplied by the specified multiplier then be sent as a 2 byte signed integer.
- Event 1-10 status input…status input whose state change is to be logged. Drag these inputs from the status database.
- Event 1-10 string…string you type in that you want presented whenever the event log is to indicate that the above status changed state.
- Event 1-10 value…floating point or integer value to be imbedded in your message whenever the above status changes state and logs an event. For example, you might include tank level when you log a high tank alarm event.

Primary Outputs: None

Outputs for Internal Use:
- Image…Name.Img…image of input statuses used to detect when they change states.

Limitations: None

Expected Applications:
The **EventLogger** feeds events to event logs for later observation of status activity. Use to log alarms, pump switching activity, intrusions, communication activity, etc.

EventLogSetup

The EventLogSetup module sets up event log space. Each event uses 64 bytes of EEPROM including time tag, up to 50 byte string, analog value and ON/OFF flag. Number of events to log sets the database size, maximum is 32767 events. When the log is full, each new event overwrites oldest event. Each instance of this module creates a separate event log. Trigger to erase log will erase only this log. Use one or more EventLogger modules to specify statuses to trigger each log entry. Trigger to dump the log to port will send the entire log to the designated serial port starting with the newest log entry and continuing until all entries have been sent. For presentation to the local LCD, set LCD string # bytes to 18 and drag the Top and Bottom string outputs from the string database to a display designated for the LCD. Starting page number should be left blank...it will be filled in by the compiler. The logger ID number (0-255) identifies this logger when a request to perform a binary dump to the TLM channel occurs. Only the logger whose ID number is referenced in the poll will respond.

Inputs That Must be Constants:

- Number events to log…integer that establishes how many events this log is to hold.
- LCD string # bytes…integer that sets output string length…should be 18 for presentation to the local LCD

Other Inputs:

- Trigger erase log…trigger status input that, when true, will erase the log.
- Trigger dump log to port…trigger status input that will cause the module to dump the entire log to the port specified below.
- Port to dump to (1 or 2)…integer input that specifies to which port on a board the module is to send the log when triggered to dump the log.
- Starting page number…leave blank…compiler will fill in
- Logger ID number…integer in range of 0-255 that identifies this logger in binary dump using R9 protocol.

Primary Outputs:

- Log array…Name.Log…Integer output pointing to logged data. This output is the one that must be referenced by the **EventLogger** modules.
- Count of events logged…Name.Count…number of events presently in log.
- Top line to LCD…Name.Top…string with time tag to be sent to first line of local LCD if local events are to be shown.
- Bottom line of LCD…Name.Bottom…string with message identifying the specific event being presented.

Outputs for Internal Use:

- Start index…Name.StartIndex…integer index of oldest event in log.
- End Index…Name.EndIndex…integer index of next event to write in log.
- Dump index…Name.Idx…integer count of number of events that have been dumped.
- LCD index…Name.LcdIdx…integer index of event being shown on LCD.
- Sample count for TLM…Name.CntTLM…integer count of samples sent to TLM channel

Limitations:

- Maximum 32767 logged events per log.

Expected Applications:

Used to establish and supervise operation of each event log.

FlipFlop Module

The **FlipFlop** module provides a two state latch that can be set, cleared, or clocked. It provides the same functionality as a D-type logic flip flop. The set and clear inputs can be used to unconditionally set the flip flop state to 1 or 0 respectively. The clock input is used in conjunction with the D input. When the clock input transitions from 0 to 1, the state present on the D input will be transferred to the flip flop's output and held. The **FlipFlop** module provides both true (.Q) and inverted (.Qbar) outputs. Aside from the obvious latching function, the flip flop can be used to toggle between off and on states. For example, the flip flop can be used as an alternator simply by dragging its Qbar output into its D input. Then, each time its clock input is triggered, the flip flop will change state.

Inputs That Must be Constants: None

Other Inputs:

- Data input D…status input that will be latched by the flip flop each time the clock input is triggered.
- Clock input…status input that will cause the data input state to be latched and sent to the Q output whenever the clock input transitions from 0 to 1.
- Clear input (sets Q=0)…status input that when 1 causes the Q output to go to zero.
- Set input (sets Q=1)…status input that when 1 causes the Q output to go to one.

Primary Outputs:

- Output Q…Name.Q…status output that reflects the flip flop's latch state
- Inverted output…Name.Qbar…status output that is the inverse of the Q output

Outputs for Internal Use:

- Old clock input…Name.Old…image of last clock input to detect rising edges

Limitations: None

Expected Applications:

Use to latch statuses and to provide a toggle (alternating) function.

FlipFlopRS Module

The FlipFlopRS module provides a two state latch that can be set, or cleared. The set and clear inputs can be used to unconditionally set the flip flop state to 1 or 0 respectively. The dominance input sets the flip flop's state in the case that both set and clear inputs are true simultaneously. The **FlipFlop** module provides both true (.Q) and inverted (.Qbar) outputs.

Inputs That Must be Constants: None

Other Inputs:

- Set input S (sets Q=1)…status input that when 1 causes the Q output to go to one.
- Reset input R (sets Q=0)…status input that when 1 causes the Q output to go to zero.
- Dominance input…status that determines output if both set and clear inputs are true.

Primary Outputs:

- Output Q…Name.Q…status output that reflects the flip flop's latch state
- Inverted output…Name.Qbar…status output that is the inverse of the Q output

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Use to latch statuses.

HOA

The **HOA** module provides the functionality of a hardware hand/off/auto switch with a lockout function. When the AUTO input is ON, the HOA allows the CALL input to be passed to the CALL output. If the AUTO input is OFF, then the HAND/OFF input controls the output. In that case, HAND/OFF=0 turns the output off; HAND/OFF=1 turns the output on. If LOCKOUT=1, then the output will be kept off independently of the AUTO or HAND/OFF inputs. Use this module following the **LeadLag** sequencer to give control of individual outputs. Generally, the hand off and auto inputs would come from a telemetry receive array to give operators at the master site direct control of individual pumps.

Inputs That Must be Constants: None

Other Inputs:
- Call input…status input that will be passed to the call output when the auto input is on and the lockout input is off.
- Hand/Off input…status input that will control the output if the auto input is off and the lockout input is off. In that case, 0=off, 1=on.
- Auto input…status input that, when on and lockout is off, enables the call input to be passed to the call output. When auto is off and lockout is off, then the hand/off input controls the output.
- Lockout input…status input that when on will turn off the output independently of any other input.

Primary Outputs:
- Call output…Name.Call…status output generally routed to a relay to control a pump or other equipment.

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Use after a pump controller or **LeadLag** sequencer to give a way to intercept pump calls and enable manual control.

HOA2

The **HOA2** module provides the functionality of a hardware hand/off/auto switch with a lockout function. It's different from the **HOA** module above in that its HOA state is derived from a single integer rather than from a pair of statuses. A single numeric value, such as a setpoint or telemetry word, can control the position of the switch. When the lockout input is off, the state input controls the output as follows:
- State=0 (off): output is off
- State=1 (hand): output is on
- State=2 (auto): call input is passed to call output

If LOCKOUT=1, then the output will be kept off independently of the state or call inputs. Use this module following the **LeadLag** sequencer to give control of individual outputs. Generally, the state input would come from a telemetry receive array to give operators at the master site direct control of individual pumps.

Inputs That Must be Constants: None

Other Inputs:
- Call input…status input that will be passed to the call output when the auto input is on and the lockout input is off.
- State input…integer input: 0=off, 1=hand, 2=auto
- Lockout input…status input that when on will turn off the output independently of any other input.

Primary Outputs:
- Call output…Name.Call…status output generally routed to a relay to control a pump or other equipment.

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Use after a pump controller or **LeadLag** sequencer in order to intercept pump calls and enable manual control.

IndexedValueSave

When triggered, the IndexValueSave module will save the input value in the particular output designated by the index input. The index offset will be added to the index to determine the cell to save. If the sum of index + index offset is outside the range of 0 to 9, then no save will occur.

Inputs That Must be Constants: None

Other Inputs:
- Trigger to force save…status input that when true causes the input value to be saved in one of the outputs.
- Value to save…floating point value to be saved to an output when trigger is true.
- Index…integer that when added to the index offset specifies to which output to save.
- Index offset…integer that when added to the index specifies to which output to save.

Primary Outputs:
- Output0-9…Name.Out0-9…Floating point outputs that will hold input values.

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Use to store values such as totalizations at particular times.

Intrusion

The Intrusion module is used to implement timed security to detect unauthorized entry to a customer's site. It typically is used like this: a door switch is wired to a digital input whose output is routed to the intrusion status input of this module. Also, an acknowledgement status or trigger is routed to the ack status input of this module. When the door opens, the Intrusion module will begin timing. If the Ack status input becomes true before the timer times out then no alarm is generated. However, if the Ack status input is not true before the timer times out, then an alarm is declared. The time out is determined by the Ack delay setpoint. A typical value is 60 seconds. The alarm will be asserted until the auto reset delay has expired, at which time the alarm will be turned off. A typical value for the auto reset delay is 300 seconds. It needs to be long enough to inform by telemetry or audible alarm of the intrusion. The reset alarm status input true will remove the alarm. Finally, if the hold off status input is true, a timer will start for the period of the holdoff delay to disable the intrusion alarm for the holdoff period. This is used to disable the alarm while authorized personnel perform maintenance, etc.

Inputs That Must be Constants: None

Other Inputs:
- Intrusion status input…status input that when true starts an alarm cycle if not acknowledged.
- Ack status input…status input that must be asserted before the ack delay time expires.
- Hold off status input…status input that disables the alarm for the holdoff period.
- Ack delay setpoint sec…integer delay in seconds before intrusion declared if not acknowledged.
- Auto reset delay setpoint sec…integer delay in seconds to clear alarm after assertion
- Hold off delay sec…integer delay in seconds to disable alarm for maintenance
- Reset alarm status input…status input to silence alarm when true

Primary Outputs:
- Alarm output…Name.Alrm…status output that will be true if intruder detected.

Outputs for Internal Use:
- Sequencer state…Name.Seq…integer output for the internal sequencer
- State timer…Name.Tmr…integer timer used to time functions
- Old intrusion status…Name.Old…copy of last intrusion status

Limitations: None

Expected Applications:
Use to provide standard intrusion sequencing for detecting intruders.

LatchFloat

The **LatchFloat** module is used to latch or capture a floating point value when a trigger occurs. The value would be held until the next trigger event. This could be used, for example, to capture a totalizer value at a particular time such as midnight, for later transmission to a master site. Since the trigger input is really a level triggered function, if the trigger input is held high, then the floating point input would be continuously passed to the output until the trigger input goes false.

Inputs That Must be Constants: None

Other Inputs:

- Trigger…When true, causes the floating point input to be passed to the output. When false, leaves the output unchanged.
- Input to capture…Floating point input to be captured when the trigger input is true.

Primary Outputs:

- Latched value…Name.Latch…floating point output equal to the last input value present while the trigger input was true.
- Latch done…Name.Trg…status trigger output indicating that a value has been latched. Used to clear an input totalizer after its value is captured or to trigger some other function that needs to be accomplished after the value is latched.

Outputs for Internal Use: None

Limitations: None

Expected Applications:

Use to capture an analog value when a particular event occurs, such as trigger on time or alarm.

LatchInt

The **LatchInt** module is used to latch or capture an integer value when a trigger occurs. The value would be held until the next trigger event. This could be used, for example, to capture a totalizer counter value at a particular time, such as midnight for later transmission to a master site. Since the trigger input is really a level triggered function, if the trigger input is held high, then the floating point input would be continuously passed to the output until the trigger input goes false.

Inputs That Must be Constants: None

Other Inputs:

- Trigger…When true, causes the integer input to be passed to the output. When false, leaves the output unchanged.
- Input to capture…integer input to be captured when the trigger input is true.

Primary Outputs:

- Latched value…Name.Latch…integer output equal to the last input value present while the trigger input was true.
- Latch done…Name.Trg…status trigger output indicating that a value has been latched. Used to clear an input totalizer after its value is captured.

Outputs for Internal Use: None

Limitations: None

Expected Applications:

Use to capture a counter value when a particular event occurs, such as trigger on time or alarm.

LatchOnBitChange

This module monitors up to 16 status inputs and provides both a steady latched output and a trigger output when a change is detected in any one of the inputs. The module's steady output is reset whenever a true is present on the reset input. The module's mode input is used to select what types of changes are to be detected: 0=both on and off transitions, 1=off to on transitions, 2=on to off transitions. Use this module to trigger actions when changes in statuses occur, such as in quiescent RTU's that must transmit all changes.

Inputs That Must be Constants: None

Other Inputs:

- Reset input…status input that turns latched output off, preparing for the next event to latch
- Mode (0-2)…integer input that specifies: 0=trigger on any change, 1= trigger on off to on transitions, 2=trigger on off to on transitions.
- Input #1-#16…status inputs whose state changes are to be detected. Blank inputs are ignored.

Primary Outputs:

- Had change output…Name.Chg…status output latched on when a change is detected in any of 16 inputs.
- Trigger on change…Name.Trg…trigger status that turns on when a change is detected in any of 16 inputs. Goes off automatically on next scan.

Outputs for Internal Use: None

- Image…Name.Img…integer copy of last set of status inputs.

Limitations: None

Expected Applications:

Use to detect any change in one or more of 16 status inputs for quiescent systems to transmit on change.

LatchString

The **LatchString** module is used to latch or capture a string when a trigger occurs. The string would be held until the next trigger event. This could be used, for example, to capture a string at a particular time such as when a telemetry reception occurs, to show an operator when the last reception occurred. Since the trigger input is really a level triggered function, if the trigger input is held high, then the floating point input would be continuously passed to the output until the trigger input goes false.

Inputs That Must be Constants:

- Output string max chars…integer to specify the largest string to be captured. Tells the compiler how much RAM to allocate.

Other Inputs:

- Trigger…when true, causes the string input to be passed to the output. When false, leaves the output unchanged.
- String to capture…string input to be captured when the trigger input is true.

Primary Outputs:

- Latched string…Name.Latch…string output equal to the last input value present while the trigger input was true.
- Latch done…Name.Trg…status trigger output indicating that a value has been latched.

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Use to capture a string when a particular event occurs, such as trigger on time, pump call, or alarm.

LeadLagSeq4

The **LeadLagSeq4** module performs lead pump rotation of up to 4 pumps. It accepts up to 4 input pump calls, usually from pump up or pump down control modules. It also accepts up to 4 pump lockout statuses. It issues call outputs based upon the number of inputs called beginning with the designated lead pump. It will not call or switch off an output unless its call/backspin delay timer has timed out. The timer is started any time the module calls or turns off an output. The module's outputs can be connected directly to relay output modules; or they can be connected to HOA modules if HOA type interception is required between the sequencer and the actual output relays. If the 'Lead' input is specified, the pump designated to be lead will be called first and turned off last. If no lead pump is specified, then the module will determine the lead pump by rotating the next pump into the lead position each time a pump is called after all have been turned off. This means that if the condition never exists that all pump calls are off, the lead pump will never rotate. Note that the LeadLag sequencer determines how many pumps to call based on how many inputs have been turned on…it doesn't care which ones are on or in what order they turned on. Also, it determines how many pumps are in the lead/lag rotation by how many inputs are specified by you.

Inputs That Must be Constants: None

Other Inputs:
- Pump A-D call…status inputs from pump up or pump down controllers indicating the need to call one or more pumps.
- Pump 1-4 lockout…status inputs that will cause the corresponding output to be skipped in the calling rotation. Blank or zero enables pump, non-zero disables pump output.
- Lead pump designator…integer input specifying which pump is to be called first and shut off last. If lead designator is missing or set outside the range of number of pump inputs specified causes module to invoke lead pump rotation.
- Call/backspin delay, sec…floating point call/backspin delay. Sets time that must elapse after any output switching before another output will be allowed to switch on or off.

Primary Outputs:
- Pump 1-4 call…Name.P1…Name.P4…status outputs used to call output relays.

Outputs for Internal Use:
- Delay timer…Name.Tmr…timer to delay on/off switching.
- Present lead designator…Name.Lead…integer register to hold present lead pump designator.

Limitations: None

Expected Applications:
Use this module in combination with other modules to implement multi-pump controls with lead lag sequencing, lead pump rotation, and backspin delay.

LookupSwitch

The **LookupSwitch** module enables you to use a control index to select from one of 8 inputs/constants to be sent to its output. If the inputs are taken from a database, then the module constitutes a 8 channel multiplexer/selector switch. If the inputs are constants, then the module constitutes a table lookup. You can mix numeric entries from the databases with constants in the input set. If the control index is out of the range of 0 through 7, then the module outputs a value of 0.0.

Inputs That Must be Constants: None

Other Inputs:

- Control (0-7)…integer input to select one of the 8 analog inputs/table entries.
- Entry 0-7…floating point, integer or status database entries, or constants to be selected by the control index.

Primary Outputs:

- Output…Name.Look…floating point output selected from the 8 entry inputs by the control index.

Outputs for Internal Use: None

Limitations:

Control index out of range of 0-7 returns value of 0.0.

Expected Applications:

Use this module to perform channel selection and/or table lookups.

MismatchLatch

The alarm **MismatchLatch** module compares two statuses and turns on its output if the two statuses fail to match each other after a designated time delay. This is typically used to detect, for example, pump failure. If a pump is called and its run indication does not turn on within a certain time, then an alarm is issued. Similarly, if the pump turns off and the run indication does not indicate that the pump turned off, then an alarm is issued. In another application, if a pump is called and a flow switch does not indicate that flow exists after a time delay, then the pump is declared failed by the mismatch module. If the alarm condition is removed, then the alarm output will become false, but the latched alarm output will not return false until the reset latch input is true.

Inputs That Must be Constants: None

Other Inputs:

- Status input #1…status that must match status input #2…typically a pump call status.
- Status input #2…status that must match status input #1…typically a pump running status.
- Alarm delay, sec…integer period in seconds during which the two status must mismatch to declare an alarm. If the delay is zero, then the alarm is disabled.
- Alarm enable…status to enable/disable alarm…0=disable, 1=enable
- Reset latch…status that will return the latched alarm to false when reset input is true.

Primary Outputs:

- Alarm output…Name.Alrm…status indicating: 0=input statuses match, 1=statuses have mismatched for the dalay period.
- Latched alarm output…Name.LatchAlrm…status indicating that an alarm has occurred since the last reset latch event.

Outputs for Internal Use:

- Delay timer…Name.Timer…integer countdown delay timer counting seconds

Limitations: None

Expected Applications:

Use to detect pump failure or other control element failure. Latched output is used to generate pump lockout status.

OffDelay

The OffDelay module will produce a TRUE output whenever its main input is TRUE and will only produce a FALSE output if the input goes FALSE and stays FALSE for a specific delay time. When input becomes TRUE, output Q immediately becomes TRUE. Output Q will become FALSE delay seconds after input becomes FALSE and stays FALSE. Reset clears timer and output Q.

Inputs That Must be Constants: None

Other Inputs:

- Input status…status input that determines output state.
- Reset…status input that will immediately force the output false and clear timer.
- Time delay sec…integer number of seconds that the input must stay false before the output will be allowed to go false.

Primary Outputs:

- Output Q…Name.Q…status output that tracks input subject to delay.
- Inverted output…Name.QBar…status output inverse of Q output

Outputs for Internal Use: None

- Timer…Name.Tmr…integer timer for delay

Limitations: None

Expected Applications:

Use to eliminate intermittent OFF states from statuses.

ONDelay

The OnDelay module will produce a TRUE output whenever its main input is TRUE and stays TRUE for a specific delay time. When input TRUE, timer starts. If input stays TRUE, output Q will become TRUE after delay seconds. Output Q will become FALSE immediately after input becomes FALSE. Reset clears timer and Q

Inputs That Must be Constants: None

Other Inputs:
- Input status…status input that determines output state.
- Reset…status input that will immediately force the output false and clear timer.
- Time delay sec…integer number of seconds that the input must stay true before the output will be allowed to go true.

Primary Outputs:
- Output Q…Name.Q…status output that tracks input subject to delay.
- Inverted output…Name.QBar…status output inverse of Q output

Outputs for Internal Use: None
- Timer…Name.Tmr…integer timer for delay

Limitations: None

Expected Applications:
Use to eliminate intermittent ON states from statuses.

ORGate

The OR gate will accept up to 8 status inputs and issue a single output that is the logical OR of the inputs. That is, if any input is on, then the output is on. Only if all inputs are off will the output be off. Any inputs that you do not specify will be regarded as off. Both normal polarity (OR) and inverted polarity (NOR) outputs are provided.

Inputs That Must be Constants: None

Other Inputs:
- Input #1-8…status inputs to be ORed together .

Primary Outputs:
- OR output…Name.ORout…status output that is the OR of all inputs.
- OR inverted…Name.ORbar…status output that is the NOR of all inputs.

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Use for controls to provide an output if any one or more of the inputs is on.

ORGateLatch

The OR gate latch module will accept up to 8 status inputs and will turn on its output if any input is turned on. The output will stay on until the reset input is true. Any inputs that you do not specify will be regarded as off. The ORbar output is the logical inverse of the latched OR output.

Inputs That Must be Constants: None

Other Inputs:
- Reset input…status input that, when true will cause the output to go to zero.
- Input #1-8…status inputs to be ORed together. If any is on, the output will be on until reset.

Primary Outputs:
- OR output…Name.ORout…status latched output.
- OR inverted…Name.ORbar…inverse of the latched OR output.

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Use for controls to provide an output if any one or more of the inputs is on or has been on since the last reset.

PID

The RUG3 PID module implements a standard independent gain proportional/integral/differential (PID) linear control algorithm. It implements the following on each scan:

Output=Kp*E+Ki*∫(E)dt+Kd*d(PV)/dt +Bias (derivative of PV type)

Or

Output=Kp*E+Ki*∫(E)dt+Kd*d(PE)/dt +Bias (derivative of error type)

Where E=SP-PV, the error
SP=setpoint
PV=process variable
Kp=proportional gain
Ki=integral gain
Kd=derivative gain
Bias=output bias to force operation in linear range

It is used to provide linear control of a valve, heater, motor speed or other proportionally controllable device to achieve a desired operating condition. Basically, the PID algorithm is used to sense the difference between the process variable (tank level, flow rate, pressure, temperature, etc.) that you wish to control, and the setpoint to which you want to control the process variable; and then to adjust the output to make the process variable match the setpoint as closely as possible. You accomplish this by setting the proportional gain, integral gain and derivative gain to give the accuracy and response time needed in your application. In most water and wastewater applications, you will set the derivative gain to zero and make the remaining gains setpoints so they can be adjusted in the field to tune the algorithm. Rule of thumb: set proportional gain to a value equal to 1 to 3 times the ratio of output span divided by input span. Set integral gain to 10% of proportional gain. Set integrator error accumulator limit to 10 times output span. Set output bias to 50% of output span.

Proportional Control

If you set proportional gain nonzero and set the other gains to zero, you will get proportional control, which is

Output=Kp*(SP-PV) + Bias.

Let's say you that wish to control a water pressure with a variable speed drive using a PID, and that you try proportional only control. A good rule of thumb is to start with proportional gain set between 1.0 and 5.0, depending on how fast you wish the algorithm to control the output; and to set the bias to 50% of the required output span. Assume the output range is 0 to 100%, the setpoint is 60 PSI, we set the proportional gain to 2.0, and we set the bias to 50% of output range. If the measured pressure is 50 PSI, then the error is SP-PV=10, so the result is

Output=2.0*(60-50)+50=70.

Therefore, the controller is sending 70% speed command to the VFD. If the VFD cannot maintain 60 PSI at 70% speed, we will always have a residual error in pressure. This is a characteristic of proportional-only control…we will generally not be able to get the process to match our desired setpoint. One possible solution is to increase proportional gain, but we increase the danger of oscillation. Another solution is to engage the integral term in our PID equation. See below.

Proportional/Integral Control

The integral term in the PID equation is used to integrate the error as a function of time to eliminate the offset. In other words, the longer the error exists, the larger the correction the integral term makes to the output to bring the process variable closer to the setpoint. The algorithm does this by summing the error with each scan and using that sum times the integral gain to adjust the output. The main question is what should be the integral gain; and how large should the error sum be allowed to grow. A good starting rule of thumb is to set the integral gain to 10% of the proportional gain, and set the integrator error limit to 10 times the output span. Then, depending on the results, adjust the gains in small increments until stable control is achieved.

Proportional/Integral/Derivative Control

Derivative control is used to speed response to changes in process variable. It's use is beyond the scope of this manual.

Speed Control of This Module

Note that this module executes on each program scan when enabled. If you leave the enable input constantly on (i.e., set it to 1), the module will probably saturate its output all the way high or low too soon for practical use unless you use gains about 1/100th of normal. Instead, you should consider installing the Setup.SecTrg once per second trigger in the enable input. This will enable the calculation only once per second, synchronizing the calculation to an established timebase, thereby making its response time independent of the program's execution speed.

Inputs That Must be Constants: None

Other Inputs:

- Enable…status that when blank or true enables the algorithm to calculate; 0 disables the calculation. Consider using System.SecTrg here.
- Process variable PV…floating point variable to be controlled as nearly as possible to the setpoint value. This is your pressure, flow rate, tank level, temperature, etc. that you wish to control.
- Setpoint SP…floating point value to which you wish the process variable to be controlled. This is usually a user adjustable setpoint or received telemetry value, but could also be the output of another module.
- Proportional gain P…floating point value that sets the gain of the proportional term of the PID equation.
- Integral gain 1/sec I…floating point value that sets the gain of the integration term of the PID equation. Zero value disables integration action.

- Derivative gain sec D…floating point value that sets the gain of the derivative term of the PID equation. Zero value disables the derivative action.
- Action 0=rev, 1=fwd…status flag to select forward or reverse acting result. If action is set to forward (1), then if the process variable is greater than the setpoint, the output will rise. If the action is set to reverse (0), then if the process variable is greater than the setpoint, the output will fall.
- Deriv: 0=dPV/dt, 1=dE/dt…status input specifying whether the derivative term is to work from the derivative of the process variable, or from the error.
- Output bias…floating point value that sets the output value that will result if there is no error and there has been no accumulated error. Usually set to the middle of the normal output range.
- Output high limit…floating point value that sets the module's maximum output value.
- Output low limit…floating point value that sets the module's minimum output value.
- Integrator error limit…floating point value that clamps the maximum absolute value the integrator's error accumulator can assume.
- Output deadband…floating point value by which the absolute value of the new result must exceed the absolute value of the last result before the new result will become the output of the module.
- Trigger force output…status that when true forces the output value to force, below, to be jammed into the module's output
- Output value to force…floating point value to be forced into output when above trigger is true.
- Trigger clear integrator error…status that when true, clears the integrator error

Primary Outputs:

- Output…Name.Out…floating point output of the PID calculation. Use this value to control other modules or send to an analog output module to control equipment.

Outputs for Internal Use:

- Last error/PV for derivative…Name.Dold…floating point register that holds last scan error or PV value for use by derivative to calculate change from last scan.
- Integ error accumulator…Name.Ierr…floating point error accumulator used by integrator.

Limitations: None

Expected Applications:

Use this module to implement tunable control of linear devices such as valves, motors, heaters, etc.

Poke

The poke module enables you to jam a value into any register or module output in the system independently of other calculations or actions. This is commonly used to initialize accumulated values, iterative results, or received telemetry values. For example, you might poke a value into a receive telemetry register to simulate the reception of a value to examine actions take by modules that act upon that value, or to assure that unknown register contents do not cause potentially damaging control actions.

Inputs That Must be Constants: None

Inputs That Must be Pointers:

- Where to poke…name dragged from a database where you wish the value to be sent.

Other Inputs:

- Trigger…status trigger that will cause the value to be poked into the designated register
- Value to poke…floating point, integer, status or constant, to be poked to the named destination when triggered.

Primary Outputs: None

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Use to jam a number into a register for test or to initialize a register to known value.

PokeMany

The PokeMany module enables you to jam up to 10 values into any registers or module outputs in the system independently of other calculations or actions. This is commonly used to initialize accumulated values, iterative results, or received telemetry values. All specified values would be poked simultaneously. For example, you might poke a series of values into receive telemetry registers to simulate the reception of values in order to examine actions take by modules that act upon those values, or to assure that unknown register contents do not cause potentially damaging control actions.

Inputs That Must be Constants: None

Inputs That Must be Pointers:
- Where to poke 1-10…names dragged from a database where you wish the corresponding values to be sent.

Other Inputs:
- Trigger…status trigger that will cause the values to be poked into the designated registers
- Value to poke 1-10…floating point, integer, status or constant, other than string, to be poked to the named destinations when triggered.

Primary Outputs: None

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Use to jam numbers into registers for test or to initialize registers to known values.

PulseGen

The PulseGen module, when triggered, will generate a variable length pulse on its output. A trigger that occurs while the output is on will start a new delay and therefore, extend any pulse in progress. Output status pulses on with each low to high transition of the trigger input. Pulse duration in seconds is set by duration input with resolution of 1 sec. Range is 32767seconds. Pulse done trigger occurs for one scan at end of main pulse.

Inputs That Must be Constants: None

Other Inputs:
- Trigger input…Status input that causes pulse generation on each low to high transition
- Duration sec…Floating point delay in seconds that sets the pulse duration

Primary Outputs:
- Pulse output…Name.Pulse…status output that will stay true for a duration after each input trigger
- Pulse output inverted…Name.PulseBar…status output inverse of pulse output
- Pulse done trigger…Name.Trg…status trigger output indicating end of variable length pulse

Outputs for Internal Use: None
- Old input…Name.Old…copy of trigger input
- Pulse timer…Name.Tmr…delay timer to time output pulses

Limitations: None

Expected Applications:
Use this module to pulse valves open/closed with variable pulse lengths.

PulseGenFast

The PulseGenFast module, when triggered, will generate a variable length pulse on its output with finer resolution than the PulseGen module above. A trigger that occurs while the output is on will start a new delay and therefore, extend any pulse in progress. Output status pulses on with each low to high transition of the trigger input. Pulse duration in seconds is set by duration input with resolution of 0.1 sec. Range is 3276.7seconds. Pulse done trigger occurs for one scan at end of main pulse.

Inputs That Must be Constants: None

Other Inputs:
- Trigger input…Status input that causes pulse generation on each low to high transition
- Duration sec…Floating point delay in seconds that sets the pulse duration

Primary Outputs:
- Pulse output…Name.Pulse…status output that will stay true for a duration after each input trigger
- Pulse output inverted…Name.PulseBar…status output inverse of pulse output
- Pulse done trigger…Name.Trg…status trigger output indicating end of variable length pulse

Outputs for Internal Use: None
- Time counter…Name.Count…counter to time output pulses

Limitations: None

Expected Applications:
Use this module to pulse valves open/closed with variable pulse lengths.

PumpDnCtrl

The pump down controller will turn on its output whenever its input level exceeds its call setpoint level. When the input level falls below the module's off setpoint, the module will turn off its output. For correct operation, the call setpoint must be above the off setpoint. This module can be used to control a single pump or can be used in conjunction with one of the lead lag sequencers which will perform backspin timing and lead lag sequencing.

Inputs That Must be Constants: None

Other Inputs:

- Enable…status input, 0=hold result, 1=calculate new result
- Input level…floating point level whose value is to be controlled by the module's output. This would normally be the level of a tank, such as a sewage sump, that must be pumped down whenever its level exceeds a setpoint.
- Call setpoint…floating point setpoint above which the module will turn on its output.
- Off setpoint…floating point setpoint below which the module will turn off its output.

Primary Outputs:

- Call output…Name.Call…status output used to control a pump or used as the input to the lead lag sequencer.

Outputs for Internal Use: None

Limitations: None

Expected Applications:

Use this controller in pump down pump controls, such as in sewage lift station pumping.

PumpUpCtrl

The pump up controller will turn on its output whenever its input level is below its call setpoint level. When the input level exceeds the module's off setpoint, the module will turn off its output. For correct operation, the call setpoint must be below the off setpoint. This module can be used to control a single pump or can be used in conjunction with the lead lag sequencer which will perform backspin timing and lead lag sequencing.

Inputs That Must be Constants: None

Other Inputs:

- Enable…status input, 0=hold result, 1=calculate new result
- Input level…floating point level whose value is to be controlled by the module's output. This would normally be the level of a tank that must be pumped up whenever its level falls below a setpoint.
- Call setpoint…floating point setpoint below which the module will turn on its output.
- Off setpoint…floating point setpoint above which the module will turn off its output.

Primary Outputs:

- Call output…Name.Call…status output used to control a pump or used as the input to the lead lag sequencer.

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Use this controller in pump up pump controls, such as in storage tank pumping.

PumpUpDn

The pump up/down controller can be used to control a single pump in either the pump up or pump down mode. The up or down mode is set by a single input that can be a setpoint. The module includes call and off delays to avoid switching the pump too frequently. By incorporating a single module that can perform both pump up and pump down control, you can set up a unit whereby the operator can select the operating mode using a setpoint without having to have two configuration files and associated program loading for the two applications.

For pump up control: pump will be called when level falls below the call setpoint, and will be shut off if the level exceeds the off setpoint. Off setpoint must be greater than the call setpoint.

For pump down control: pump will be called when the level exceeds the call setpoint, and will be shut off if the level falls below the off setpoint. Call setpoint must be greater than the off setpoint.

Inputs That Must be Constants: None

Other Inputs:
- Enable…status input, 0=hold result, 1=calculate new result
- Level input…floating point tank or sump level whose value is compared with call and off setpoints to determine whether to call the pump or turn it off.
- Mode: 0=pump up, 1 down…integer choice of: 0=pump up, 1=pump down
- Call setpoint…floating point level to cause pump to switch on.
- Off setpoint…floating point level to cause pump to switch off.
- Call steady delay sec…integer time in seconds that pump call condition must remain true before the pump will be called.
- Off steady delay sec…integer time in seconds that pump off condition must remain true before the pump will be turned off.

Primary Outputs:
- Call output…Name.Call…status output that turns on when the pump should be called; turns off when the pump should be turned off.

Outputs for Internal Use: None
- Steady timer…Name.Tmr…integer timer to time call/off conditions to avoid too frequent switching.

Limitations: None

Expected Applications:
Use for all pump control applications to directly control pumps or to send pump calls to the lead lag sequencer to control pumps in combinations.

RateofChange

The rate of change module compares an input value with the value it last stored to detect when the change from one scan to the next has exceeded a user supplied rising rate setpoint; or has fallen below a user supplied falling rate setpoint. The module makes its calculations when triggered, so, for this to work in a practical application, the module should be triggered on a set time interval such as once per second, or once per minute, etc. For example, to detect when a reservoir is rising at a rate exceeding one foot per hour, you could trigger the module every 60 minutes and set the rising rate setpoint to 1.0. Alternatively, you could trigger the module once per minute and set the rising rate setpoint to 0.0167 feet, the change a one foot hourly rise would make in one minute.

Inputs That Must be Constants: None

Other Inputs:

- Trigger…status trigger input to cause the module to execute its calculation and save its results. This should be a fixed time base trigger for practical use.
- Input value…floating point value whose rate of rise or fall is to be tested.
- Rising alarm rate SP…floating point setpoint whose value must be exceeded by the difference between input level on successive scans to generate a high rising rate alarm.
- Falling alarm rate SP…floating point setpoint whose value must be exceeded in a negative direction by the difference between input level on successive scans to generate a high falling rate alarm. This setpoint must be negative for correct operation.

Primary Outputs:

- Change from last…Name.Change…floating point value of difference in input values between last two scans.
- High rising rate alarm…Name.RiseAlrm…status that will be true if difference in input values between last two scans exceeds rising alarm rate setpoint.
- High falling rate alarm…Name.FallAlrm…status that will be true if difference in input values between last two scans is below falling alarm rate setpoint.

Outputs for Internal Use: None

- Old value…Name.Old…floating point value of level at last scan

Limitations: None

Expected Applications:

Use with fixed time base to detect rate of change of an analog value and generate high/low rate alarms.

ReadRTC

This module is used to capture the realtime clock/calendar and split its values out into individual integers and strings. It reads the realtime clock/calendar each time it is triggered and latches the clock's values at that time. Therefore, it has two main uses: to capture the clock periodically, usually once per second, for display; and to capture the clock when an event occurs for time stamping the event. The module's outputs remain at their current values until the next trigger.

Inputs That Must be Constants:

- Output string chars…integer that sets how much RAM the compiler must set aside for the string outputs of the module. This should be set to 40 unless no string outputs are required. If set to zero, no strings will be generated and RAM will be conserved.

Other Inputs:

- Trigger input…status trigger input that forces the module to read the RTC/calendar when true. If left blank, the module will read the clock on each scan. For practical use, install the System.SecTrg here to trigger RTC reading each second.

Primary Outputs:

- Seconds…Name.Sec…integer output of the RTC seconds register. Range is 0 to 59.
- Minutes…Name.Min…integer output of the RTC minutes register. Range is 0 to 59.
- Hours…Name.Hr…integer output of the RTC hours register.
- Day of week…Name.DayofWk…integer representing day of week: 1=Sun…7=Sat
- Day…Name.Day…integer day of month. Range is 1 to 31
- Month…Name.Mo…integer month of year. Range is 1 to 12
- Year…Name.Yr…integer year. Range is 1950 to 2049
- Time/date string…Name.Str…string of time and date of format SUN 11/03/1999 14:39:08
- Time string…Name.Time…string of time of format 14:39:08
- Date string…Name.Date…string of date of format 11/03/1999
- Numeric RTC string…Name.NumStr…date/time string without day of week 11/30/2003 12:34:45

Outputs for Internal Use: None

Limitations: None

Expected Applications:

Use to capture clock for time tagging and for splitting RTC into individual items.

SequencerT2

SequencerT2 is a timed sequencer with a user defined time per state and four presets. When a preset trigger is received, the sequencer will assume the corresponding preset state and restart the timer. When the timer times out, the sequencer state will increment to the next state and restart the timer. If the enable is missing or set to 1, the sequencer will continue to count uninhibited. If the enable is set to 0, the sequencer state will not increment, but the timer will continue to count until it times out. Timer resolution is 1.0 second. Timer range is 32767seconds. The sequencer state count range is -32768 to +32767. To obtain a status for each state, you should use this sequencer with the **SequenOut** module, which provides 10 output statuses based on the state of the sequencer state. You may use as many **SequenOut** modules as you need to implement as large a sequencer as necessary.

Inputs That Must be Constants: None

Other Inputs:

- Enable counting…status input: 0=stop counting; 1 or blank=enable state counting
- Seconds per state…integer seconds between state transitions.
- Preset 1-4 triggers…status trigger inputs to force the state to a user defined state from which counting will continue.
- Preset 1-4 state…integer value the state counter is to assume when the corresponding preset trigger is received.

Primary Outputs:

- State…Name.State…integer value of the sequencer’s state counter.

Outputs for Internal Use: None

- Timer…Name.Tmr…integer time counter for state timing

Limitations: None

Expected Applications:
Use this sequencer for sequences that need a uniform time interval per state.

SeqTimedTrigger

The SeqTimedTrigger sequencer spends a user designated number of seconds in each state, then advances to the next state. Preset triggers force the sequencer to either of two specified states. Enable true enables time/state counting. Enable false halts time/state counting. Max state specifies the highest sequencer state. Mode specifies the action to be taken when the state output equals the max state: 0=stop, 1=roll back to state 1 and continue. Seconds in state inputs specify time to be spent in each state in tenths of a second with a range of 32767 tenths of a second each. Trigger outputs are asserted on entry to each state. Done trigger is asserted when state designated as max state is finished.

Inputs That Must be Constants: None

Other Inputs:
- Enable input…status input that will inhibit sequencing when zero, and will enable sequencing when blank or set to 1.
- Preset trigger N…status triggers that when true send the sequencer to the state specified by the preset N state inputs.
- Max state…integer that specifies the highest state that the sequencer counter can have.
- Mode 0,1…integer input that specifies action to be taken when sequencer hits its max count: 0=stop, 1=roll back to state 1 and continue.
- Tenth sec in state 1 to 10…integer tenths of a second the sequencer is to dwell in the corresponding state. Range is +3276.7 seconds. Resolution is 0.1second.

Primary Outputs:
- Sequencer count 1-10…Name.Count…integer count of sequencer state.
- State #1 to #10 trigger…Name.Trg1 to Name.Trg10…Trigger outputs that will be true for one scan when the sequencer transitions to the corresponding state.
- Done trigger…Name.TrgDun…trigger output that will be asserted when the sequencer finishes with the max state.

Outputs for Internal Use:
- Timer…Name.Tmr…integer time counter for state timing

Limitations: None

Expected Applications:
Use this sequencer for applications wherein you need a sequencer that needs a different time per state and a trigger to signal entry to each state.

SequencerTimed

The timed sequencer is a 7 stage sequencer that provides a time delay for each state and can be cascaded to give any length sequencer with a unique time per state. When reset, the sequencer assumes state 1; and the timer is started with the time at which it is to stay in state 1. When the timer times out, the sequencer advances to the next state and restarts the timer with the time setpoint for that state, and so forth. This continues until the sequencer hits state 8, where it stays until reset. The enable input enables or inhibits time counting but does not affect the outputs. To cascade these sequencers to obtain longer sequencers, connect the state 8 output of a low order sequencer to the enable input of the next higher sequencer.

Inputs That Must be Constants: None

Other Inputs:

- Reset input…status input that will reset the sequencer to state 1.
- Enable input…status input that will inhibit sequencing when zero, and will enable sequencing when blank or set to 1.
- Seconds in state 1 to 7…integer seconds sequencer is to dwell in the corresponding state. Range is +32767 seconds. Resolution is one second.

Primary Outputs:

- Sequencer count 1-8…Name.Count…integer count of sequencer state.
- State #1 to #8…Name.Seq1 to Name.Seq8…status output that will be true when the sequencer is in the corresponding state.

Outputs for Internal Use:

- State timer…Name.Sec…integer time counter for state timing

Limitations: None

Expected Applications:

Use this sequencer for applications wherein you need a sequencer that may need a different time per state.

SequencerUpDn

SequencerUpDn is a sequencer that can count up or count down depending on which of two triggers it receives. It has four presets. When a preset trigger is received, the sequencer will assume the corresponding preset state. If the enable is missing or set to 1, the sequencer will continue to count uninhibited. If the enable is set to 0, the sequencer state will not change when triggered. Presets work independently of the enable. The sequencer state count range is -32768 to +32767. To obtain a status for each state, you should use this sequencer with the **SequenOut** module, which provides 10 output statuses based on the state of the sequencer state. You may use as many **SequenOut** modules as you need to implement as large a sequencer as necessary.

Inputs That Must be Constants: None

Other Inputs:

- Enable counting…status input: 0=stop counting; 1 or blank=enable state counting
- Count up trigger…status trigger that will cause the state counter to increment.
- Count down trigger…status trigger that will cause the state counter to decrement.
- Preset 1-4 triggers…status trigger inputs to force the state to a user defined state from which counting will continue.

- Preset 1-4 state…integer value the state counter is to assume when the corresponding preset trigger is received.

Primary Outputs:
- State…Name.State…integer value of the sequencer's state counter.

Outputs for Internal Use: None
- Old count up…Name.OldUp…status image of last trigger input to detect rising edge.
- Old count down…Name.OldDn…status image of last trigger input to detect rising edge.

Limitations: None

Expected Applications:
Use this sequencer as a counter to detect the difference between counts of two pulse trains, or for general purpose sequencing.

SequenOut

The **SequenOut** module is a state decoder and sequencer output expander. Use it to read a sequencer's state and provide one status output for each state. The offset input enables multiple modules to be assigned to one sequencer to expand the sequencer's outputs to any number necessary. When the state minus the offset is in the range of 1 to 10, then one of the **SequenOut** outputs will turn on when the enable input turns on. For example, if the offset is set to zero, then the outputs will turn on in response to states 1 to 10. If the offset is set to 10, then the outputs will respond to sequencer states 11 to 20, and so forth. Therefore, to expand a sequencer's outputs to, say 30, set one SequenOut's offset to 0; set the next **SequenOut**'s offset to 10; and set the last **SequenOut**'s offset to 20. If the module's enable is false, then its outputs will all be off.

Inputs That Must be Constants: None

Other Inputs:
- Enable…status input that enables the outputs to turn on.
- State…integer input state that determines which output is to turn on.
- Offset…integer input that is subtracted from the input state to determine which output is to turn on.

Primary Outputs:
- Output 1-10…Name.S1 to Name.S10…status outputs, one of which will be on if the enable is on and the state-offset is in the range of 1 to 10.

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Used to expand any counter or sequencer to turn on a status corresponding to the counter state.

SetRTC

Use this module to set the realtime clock/calendar (RTC) from the program. The module provides a separate trigger for each of the seven elements of the clock/calendar. When triggered, the value installed takes effect immediately. Triggers can be issued simultaneously. This module is useful for setting the RTC from telemetry to keep all clocks in a telemetry system in synchronism.

Inputs That Must be Constants: None

Other Inputs:

- Trigger to install seconds…status trigger that will install the seconds value in the RTC.
- Seconds value to install…integer value to preset seconds in RTC. Range is 0 to 59.
- Trigger to install minutes…status trigger that will install the minutes value in the RTC.
- Minutes value to install…integer value to preset minutes in RTC. Range is 0 to 59.
- Trigger to install hours…status trigger that will install the hours value in the RTC.
- Hours value to install…integer value to preset hours in RTC. Range is 0 to 23.
- Trigger to install day of month…status trigger that will install the day of month value in the RTC.
- Day of month value to install…integer value to preset day of month in RTC. Range is 1 to 31.
- Trigger to install month…status trigger that will install the month value in the RTC.
- Month value to install…integer value to preset month in RTC. Range is 1 to 12.
- Trigger to install day of week…status trigger that will install the day of week value in the RTC.
- Day of week value to install…integer value to preset day of week in RTC. Range is 1 (Sun) to 7 (Sat).
- Trigger to install year…status trigger that will install the year value in the RTC.
- Year value to install…integer value to preset year in RTC. Range is 0 to 99.

Primary Outputs: None

Outputs for Internal Use: None

Limitations: None

Expected Applications:

Use to set RTC remotely.

SnapShotCount

When triggered, the SnapShotCount module reads the background counter on the designated DI channel and calculates counts since last trigger. The module does not alter the counter contents. Use this module to read pulse counts at desired time intervals without danger of losing counts by clearing the counter after reading. When using this module, you must have a DICounter or ShaftEncoder assigned to the channel and you must not clear or preset the counter. Range is 0-32767..

Inputs That Must be Constants: None

Other Inputs:

- Trigger read count…trigger status that, when true, causes the module to read the designated counter and subtract the previous count from it to produce the output count.
- DI channel (1-8)…integer designates which digital input channel whose counter is to be read.
- Preset value…integer that will be installed as the count when the preset trigger is true.

Primary Outputs:

- Count since last trigger…Name.Cnt…integer difference between last sample and present count.

- Trigger have new cnt…Name.Trg…trigger status that will be true the next scan after a new sample has been read.

Outputs for Internal Use:

- Old count…Name.Old…copy of last count read from background counter

Limitations:

Count range is 0 to +32767.

Expected Applications:

Use to capture a count at a specified time interval or between events without affecting the DI counter.

StringSwitch

The string switch uses an index to select one of up to 11 string inputs to be sent to the output. It transfers the selected string when it detects a change in its input index. This module is mostly used to convert a number (the index) to a string for display on the LCD. For example, if the state of a hand/off/auto switch is represented by an integer with a value of 0=off, 1=hand and 2=auto, it is more informative to an operator to see a word such as “auto” than to see the value “2”. To accomplish this, drag the index into the StringSwitch’s input index entry; and then enter the strings: “OFF” into the string zero cell; “ON“ into the string 1 cell; and “AUTO” into the string 2 cell. Then drag the output of the string switch into the display’s variable list where the HOA state is being displayed.

Inputs That Must be Constants:

- Number bytes/string…integer identifying space to be set up for this module’s output. This value should be greater than the number of characters in the longest input’s string.

Other Inputs:

- Input index (string to select)…integer in the range of 0 to 10 that selects a string from the input string list.
- String 0 to 10…string inputs from which one is to be chosen by the index to become the output of this module.

Primary Outputs:

- Output string…Name.String…string output selected from the string input list by the index.

Outputs for Internal Use:

- Old index…Name.Old…copy of previous index to detect changes.

Limitations: None

Expected Applications:

Used to provide a more pleasing display of alarm conditions, sequencer states, etc. than simple numbers.

StringSwitchByBits

The StringSwitchByBits module is similar to the string switch above. It uses an index calculated from the binary sum of 4 bits to select one of up to 16 string inputs to be sent to the output. It transfers the selected string when it detects a change in its input index. This module is mostly used to select strings based upon status bits. For example, if a status represents CLOSED when zero and OPEN when one, then it could be dragged into the bit (val=1) input of this module and then the strings 'CLOSED' and 'OPEN' typed into the first two string inputs respectively. Then the module's string output would give 'CLOSED' or 'OPEN' depending on the input state.

Inputs That Must be Constants:

- Number bytes/string…integer identifying space to be set up for this module's output. This value should be greater than the number of characters in the longest input's string.

Other Inputs:

- Enable…status input, 0=hold result, 1=calculate new result
- Bit (val=1…8)…statuses that select a string from the input string list.
- String 0 to 15…string inputs from which one is to be chosen by the index to become the output of this module.

Primary Outputs:

- Output string…Name.String…string output selected from the string input list by the index.
- Byte sum…Name.Sum…integer sum of status inputs (0-15)

Outputs for Internal Use: None

Limitations: None

Expected Applications:

Used to provide a more pleasing display of alarm conditions, sequencer states, etc. than simple statuses.

StringSwitchPriority Module

When triggered, the **StringSwitchPriority** module reads input bits until it finds one that is ON, in which case the module outputs that bit's corresponding string. If no bit is ON in the list, it emits default ALLOFF string. If mode=0 or blank, shows only first ON bit found (highest priority) even if more than one bit is on. If mode=1 and more than one bit is on, module will advance to next true bit's string each time enable is asserted.

The main intention of this module is to enable you to show any of a series of statuses on a single line of the display by showing the highest priority one that is true (mode=0); or showing any that are on, say once per second, by triggering the module once per second (mode=1).

Inputs That Must be Constants:

- Max string bytes…integer that tells compiler maximum output string length.

Other Inputs:

- Trigger new test…status input that causes module to scan inputs and issue new output string.
- Mode (0,1)…status input: 0= show only first ON bit found (highest priority); 1= advance to next true bit's string each time enable is asserted.
- All off string…string that will be output if all status inputs are off.
- Bit 1-10…status inputs that indicate need to output corresponding string.
- String 1-10…strings that will be emitted when corresponding bit is true

Primary Outputs:
- Output string…Name.Str…string output of highest priority or next bit that is true.
- Have true bit status…Name.HavBit…status output indicating that at least one input status is on.

Outputs for Internal Use:
- Cycle counter…Name.Cnt…integer of last bit that tested true if mode=1.

Limitations: None

Expected Applications:
Use to condense list of statuses or alarms to occupy only one line on display with highest priority status shown; or showing status strings in cyclic method.

SyncValues

When triggered, SyncValues scans two input values and compares them with associated output values. If either value has changed, the module sets the value output to the new value and installs the new value back into the older input. Any change in A values will cause the change in A trigger output to be asserted. Any change in B values will cause the change in B trigger output to be asserted.

Inputs That Must be Constants: None

Other Inputs:
- A input…Floating point input that is compared with the Value output.
- B input…Floating point input that is compared with the Value output.

Primary Outputs:
- Value…Name.Val…floating point output that holds the most recent value present on the corresponding inputs.
- A changed trigger…Name.AChng…status trigger indicating that a new value has been installed from the input A input.
- B changed trigger…Name.BChng…status trigger indicating that a new value has been installed from the input B input.
-

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Use this module to hold the latest value entered by an operator or received from the telemetry channel. For example, the input A entry could come from a local setpoint; input B entry could come from the telemetry receive array. Whichever was entered last would be captured in the output values.

SyncToRTC Module

When enabled, the **SyncToRTC** module issues triggers on the specified interval, synchronized to realtime clock (RTC). Enter one of the three input intervals and leave the others zero or blank. This module will issue a trigger after expiration of the specified interval aligned to the interval's zero event, and zero of more subordinate times. For example, if you specify a 5 minute sample interval and leave the seconds and hours entries blank, then the module will issue triggers at 0,5,10,15,20... minutes of the hour, at the zero second of each 5 minute tick.

Inputs That Must be Constants: None

Other Inputs:

- Enable…status input that when true or blank enables triggers to be generated. Zero disables trigger outputs.
- Seconds interval (1-30)…integer in range of 1 to 30 seconds that sets the seconds interval of the RTC at which triggers will be generated in each minute. E.g., if you set this to 13, then a trigger will be generated when the RTC second hits 0,13,26,and 39 seconds.
- Minutes interval (1-30)… integer in range of 1 to 30 minutes that sets the minutes interval of the RTC at which triggers will be generated in each hour. E.g., if you set this to 15, then a trigger will be generated when the RTC minute hits 0,15,30,and 45 minutes.
- Hours interval (1-12)…integer in range of 1 to 12 hours that sets the hours interval of the RTC at which triggers will be generated each day. E.g., if you set this to 6, then the trigger will be generated when the RTC hour hits 0,6,12, and 18 hours.

Primary Outputs:

- Trigger output…Name.Trg…status trigger that will be generated on the specified uniform interval.

Outputs for Internal Use:

- Temp time…Name.Temp…value of last read RTC item.

Limitations: None

Expected Applications:

Use to generate uniform triggers synchronized to the RTC.

Toggle Module

The **Toggle** module provides a two state latch or flip flop that can be set, cleared, or toggled. The set and clear inputs can be used to unconditionally set the flip flop state to 1 or 0 respectively. When the clock input transitions from 0 to 1, the flip flop will transition to the opposite state, which will become its new output. The module provides both true (.Q) and inverted (.Qbar) outputs.

Inputs That Must be Constants: None

Other Inputs:

- Enable input…status input that prohibits toggling when false.
- Clock input…status input that will cause the state to be toggled and sent to the Q output whenever the clock input transitions from 0 to 1.
- Clear input (sets Q=0)…status input that when 1 causes the Q output to go to zero.
- Set input (sets Q=1)…status input that when 1 causes the Q output to go to one.

Primary Outputs:
- Output Q…Name.Q…status output that reflects the flip flop's latch state
- Inverted output…Name.Qbar…status output that is the inverse of the Q output

Outputs for Internal Use:
- Old clock input…Name.Old…image of last clock input to detect rising edges

Limitations: None

Expected Applications:
Use to give users a simple toggle on/off control from keystrokes.

TriggerDelay

The trigger delay module accepts an input trigger, times out for a user specified number of seconds, then reissues the trigger. This module enables you to delay actions until other events have completed that may be initiated by the same trigger. The delay timer has 1.0 second resolution and range of 32767 seconds.

Inputs That Must be Constants: None

Other Inputs:
- Trigger input…status trigger input to initiate timing.
- Delay sec…floating point number of seconds that the input trigger is to be delayed before the output trigger is issued.

Primary Outputs:
- Trigger output…Name.Trg…status trigger output issued after a delay from the input trigger.
- Status armed…Name.Arm…status output indicating that the module is timing a trigger

Outputs for Internal Use: None
- Delay timer…Name.Dly…integer timer for delay

Limitations: None

Expected Applications:
Use to sequence functions triggered by the same event.

TriggerEveryXSecond

This module issues a trigger event at specified intervals synchronized with the real time clock seconds register. The module will count seconds until the designated number of seconds has passed. Then it will issue a trigger and reset the seconds counter. The counter is a count down counter that can be preset to force it to synchronize with other events. The preset forces the X value to be installed in the count down seconds counter. If no preset trigger is received, the module will issue triggers indefinitely on the interval specified. This module is typically used to cause events to happen every X seconds; for example, to force a poll every 10 seconds.

Inputs That Must be Constants: None

Other Inputs:
- X interval seconds…integer number of seconds between successive trigger outputs.

Primary Outputs:
- Trigger output…Name.Trg…status trigger output that will be issued every X seconds.

Outputs for Internal Use:
- Seconds counter…Name.Cnt…integer counter of seconds before issuing trigger.

Limitations: None

Expected Applications:
Used to trigger events in synchronism with the RTC seconds register.

TriggerGen

This module is used to issue a trigger on the rising and falling edges of a status input. Since status inputs can stay on continuously, and some modules execute events when a status input is true, this module enables you to cause the event to be executed only when the status turns on and/or off, rather than repetitively as long as the status is on.

Inputs That Must be Constants: None

Other Inputs:
- Status input…status whose rising or falling edge is to cause at trigger to be issued.

Primary Outputs:
- Output trigger…Name.Trg…status trigger output that will remain true for the one scan following the scan during which the input status transitions from off to on.
- Output falling edge trigger…Name.FallTrg…status trigger on high to low transition of input.

Outputs for Internal Use:
- Old input…Name.Old…status copy of last status input for rising edge detection.

Limitations: None

Expected Applications:
Use to convert a steady status to a trigger.

TriggerOnChange

This module issues a trigger output whenever an input value differs by more than a user specified amount (deadband) from the value it had recorded at the previous trigger. This module can be used to watch for changes in analog values in order to trigger a poll when a change needs to be reported.

Inputs That Must be Constants: None

Other Inputs:
- Enable…status input, 0=hold result, 1=calculate new result
- Input value…floating point value that is being monitored for a change.

- Deadband…floating point amount by which the input value must change before a change trigger is issued.

Primary Outputs:

- Output trigger…Name.Trg…status trigger output indicating that the input value has changed by more than the deadband.

Outputs for Internal Use:

- Old value…Name.Old…floating point value captured at previous trigger.

Limitations: None

Expected Applications:

Use this module to detect changes in analog values to trigger reporting or other events.

TriggerOnKeyMany

The **TriggerOnKeyMany** module issues a trigger whenever the user hits a designated key while a particular display is being presented either on the LCD or on some other port. You can designate the port number where the keystroke must be entered in addition to the designated display number. If no display is specified, then the designated keystrokes will be accepted on any display on that port. The port number must be entered as an integer in the range of 0 through 2. Port 0 is the LCD display, port 1 is the programming serial port, and port 2 is the modem/RS232 port. .

This module enables you to setup a project so that an operator can press a key to cause some action to take place. For example, say you want to enable the operator to acknowledge alarms from display number 3 on the LCD. If you set up the TrigOnKeyMany module to issue a trigger when display 3 is presented on port 0, and you use the output of, say key 2 from the TriggerOnKeyMany module as the input of all alarm output modules, then when the operator is looking at that display on the LCD and he presses key 2, all alarm outputs will cease flashing. Of course, you should include a prompt on display number 3 such as “Key 2…acknowledge alarms” to prompt the operator.

If you want the **TriggerOnKeyMany** module to only respond when an operator is logged on, drag the System.Logon output from the status database and drop it into the enable input of this module.

Inputs That Must be Constants: None

Other Inputs:

- Enable…status input, 0=disabled, 1=enable output
- Display #…integer number of the display from which the keystroke is to be accepted. If left blank, then the keystroke will be accepted from any display.
- Port #…integer port number of the port from which the keystroke is to be accepted.

Primary Outputs:

- Trigger on key 0-9…Name.Key0…Key9…status trigger issued immediately after the user pressed the designated key on the designated port.

Outputs for Internal Use: None

Limitations: None

Expected Applications:

Module enables operator to cause changes in control actions, acknowledge alarms, etc.

TriggerOnRTC

The **TriggerOnRTC** module enables you to trigger an event or action based on specific comparisons against the realtime clock/calendar (RTC). It does this by giving you a cell for each of the seven RTC numeric items into which you can install a number that the RTC must match in order for the module to generate its output trigger. Items you leave blank are regarded as always comparing true. In addition, once the module detects a true comparison and generates its trigger, it will inhibit further true outputs until the comparison has returned to false for at least one scan. For example, say you wish to generate a trigger to cause an event, such as a daily report, at 10 minutes after 6 AM each day. You would set up the inputs as follows:

RTC ITEM	VALUE
Seconds	
Minutes	10
Hours	6
Day	
Month	
Day of week	
Year	

With this setup, the comparison will be true only when the RTC minute value is 10, and the hour is 6. All other values are regarded as always true. Notice that the comparison would be true for the entire minute at 6:10 AM, so the module disarms itself until a false comparison, which will occur at 6:11 AM. The next true comparison won't happen until 6:10 AM the following day.

Inputs That Must be Constants: None

Other Inputs:
- Seconds …integer seconds to compare. Range is 0 to 59.
- Minutes …integer minutes to compare. Range is 0 to 59.
- Hours …integer hours to compare. Range is 0 to 23.
- Day of month …integer day of month to compare. Range is 1 to 31.
- Month …integer month to compare. Range is 1 to 12.
- Day of week …integer day of week to compare. Range is 1 (Sun) to 7 (Sat).
- Year …integer year to compare. Range is 0 to 99.

Primary Outputs:
- Trigger event output…Name.Trigger…trigger true when all 7 RTC items match.

Outputs for Internal Use:
- True compare…Name.Old…flag to disarm until after next false comparison

Limitations: None

Expected Applications:
Use to trigger RTC-based functions.

TriggerOnSpecialKeys

The **TriggerOnSpecialKeys** module Issues a trigger event for use by other modules whenever a special keystroke occurs while the designated user display (not a menu) is presented on the LCD. Special keys supported are the clear (C), up arrow (U), down arrow (D) and enter (E) keys. Leave

Display # blank if you want triggers from all displays. To require user to be logged on, drag logon status from system module into the Enable input of this module.

Inputs That Must be Constants: None

Other Inputs:
- Enable…status input, 0=disabled, 1=enable output
- Display #…integer number of the display from which the keystroke is to be accepted. If left blank, then the keystroke will be accepted from any display.

Primary Outputs:
- Trigger on key Enter key…Name.Enter…status trigger issued immediately after the user presses the 'Enter' key.
- Trigger on key Clear key…Name.Clr…status trigger issued immediately after the user presses the 'Clear' key.
- Trigger on key Up arrow key…Name.Up…status trigger issued immediately after the user presses the 'Up arrow' key.
- Trigger on key Down arrow key…Name.Down…status trigger issued immediately after the user presses the 'Down arrow' key.

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Use this module to augment the TrigOnKeyMany module to enable detection of the non-numeric keys.

TrigOnBitThenClr

The **TrigOnBitThenClr** module will issue a trigger when its input status bit becomes true, and then it will clear the input status. It is most useful for detecting a flag sent in from another unit or from a modbus SCADA master so that the action triggered by the flag will be taken only one time. By clearing the source flag after its detection, the trigger will only be acted upon once unless the SCADA source sends it again.

Inputs That Must be Constants: None

Other Inputs:
- Status input…status that when true will cause the module output to become true for one scan. After detecting the true input, the module will clear the status input.

Primary Outputs:
- Output trigger…Name.Trg…status trigger output generated by true status input.

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Use to assure that only one action will be taken on received telemetry status.

TrigOnChangeMany

This module issues a trigger output whenever an input value differs by more than a user specified amount (deadband) from the value it had recorded at the previous trigger. It works on as many as 8 inputs, provides an individual change trigger on the first 8 inputs, and provides a composite trigger that indicates if a change has occurred on any of the 8 inputs. This module can be used to watch for changes in analog values in order to trigger a poll when a change needs to be reported.

Inputs That Must be Constants: None

Other Inputs:

- Value 1-8…floating point values that are being monitored for a change.
- Deadband 1-8…floating point amounts by which the input values must change before a change trigger is issued.

Primary Outputs:

- Value 1-8 changed…Name.Trg1-8…status trigger outputs indicating that the corresponding input values have changed by more than the associated deadband.
- Any value changed…Name.TrgAny…status trigger output indicating that a change has been detected on one or more of the 8 inputs.

Outputs for Internal Use:

- Value 1-8 copy…Name.Val1-8…floating point copy of value captured at previous trigger.

Limitations: None

Expected Applications:

Use this module to detect changes in analog values to trigger reporting or other events.

ValueTest

The ValueTest module enables you to compare two floating point or integer values and obtain any combination of the possible results for use in controls, alarm generation, etc. The comparison is:

Result= Input A – Input B

The output statuses will remain true or false for as long as the condition persists, or for as long as the enable is false.

Inputs That Must be Constants: None

Other Inputs:

- Input A…floating point value from which input B is subtracted for the result.
- Input B…floating point value to be subtracted from input A for the result.
- Enable…status input to enable the calculation. Outputs will be held unchanged if enable is false.

Primary Outputs:

- A>B…Name.GT…status output true if input A is greater than B.
- A>=B…Name.GE…status output true if input A is greater than or equal to input B.
- A=B…Name.EQ…status output true if input A is exactly equal to input B.
- A<>B…Name.NE…status output true if input A is not exactly equal to input B.
- A<=B…Name.LE…status output true if input A is less than or equal to input B.

- A<B…Name.LT…status output true if input A is less than input B.

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Use for general purpose integer or floating point value comparisons.

ValueTestTrig

The ValueTestTrig module, when enabled compares the input A with input B. The function input specifies the comparison to be made: 0: A>B, 1: A>=B, 2: A=B, 3: A<>B, 4: A<=B, 5: A<B. When the comparison produces a true result, the True output status will be 1; if the comparison produces a false result, the True status output will be 0. When the comparison goes from false to true, the trigger output will be generated.

Inputs That Must be Constants: None

Other Inputs:
- Enable…status input that enables the comparison. Outputs will be held unchanged if enable is false.
- Input A…floating point value from which input B is subtracted for the result.
- Input B…floating point value to be subtracted from input A for the result.
- Function…integer that specifies the comparison to be made: 0: A>B, 1: A>=B, 2: A=B, 3: A<>B, 4: A<=B, 5: A<B

Primary Outputs:
- Status output…Name.True…status result from input comparison. The specific comparison is specified by the function input.
- Trigger output…Name.Trig…trigger status output that will be asserted when the status output transitions from false to true.

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Use to provide a single comparison and provide both a status result and a trigger result. The advantage of this module over the ValueTest module above is that avoids filling up the status database with unnecessary variables.

Statistics Modules

AvgValue

The **AvgValue** module is used to maintain an ongoing average of its input over time. It does this by keeping an accumulation of input samples and dividing by the number of samples it has summed. Its reset input will latch the present average and then zero the sum and sample counter to start a new average. In a typical application, you might use this module to keep a daily average that you reset at midnight, so the latched average shows the average over the prior 24 hours, and the present average shows the average since midnight.

Inputs That Must be Constants: None

Other Inputs:
- Enable…status input, 0=hold result, 1=calculate new result
- Input to average…floating point value whose average this module is to calculate.
- Reset…status trigger input that will cause the module to latch the present average and start a new average.

Primary Outputs:
- Average value…Name.Avg…floating point value representing the present average since the last module reset.
- Latched average…Name.Avg…floating point value equal to the average that existed just before the last module reset.

Outputs for Internal Use:
- Sample count…Name.Cnt…integer count of samples taken since last module reset.

Limitations: None

LogMany

The LogMany module is used to log multiple items at the same time. With each sample it stores one record, consisting of a time tag and up to 20 values, to the onboard serial EEPROM which has 2 Mbytes capacity. The time tag and each floating point or integer sample consume 4 bytes of storage each. Time tag resolution is 4 milliseconds. Status inputs to be logged are packed all into a single 4 byte field. When the log is full, if mode=0, then logging stops; if mode=1 then logging continues with the new sample overwriting the oldest stored sample. Starting page number should be left blank; it will be assigned by the compiler. Logger ID number (0-7) identifies this logger for the DumpLogTLM module to respond to telemetry polls for logged data. The Sample count for TLM output contains the count of samples taken since the last telemetry dump of the log. When a program with the LogMany module is compiled, the logger flash occupancy is presented at the bottom of the R3Setup screen along with the logger capacity. If the logger memory requirements exceed the capacity, R3Setup will present an error message upon compilation.

Inputs That Must be Constants:
- Number of records in DB…integer specifying how many time tag/data records are to be stored.

Inputs That Must be Pointers: None

Inputs Used Internally:

- Starting page number…integer assigned by compiler to allocate space on EEPROM. This should be left blank.

Other Inputs:

- Trigger sample…status trigger that will cause a sample to be taken when true.
- Enable logging…status input that will disable sampling if false.
- Clear log/reset…status input that when true will erase entire log.
- Input sample #1-20…floating point inputs to be logged. Logging of a record stops if blank input encountered.
- Starting page number…integer that should be left blank…it will be entered by the compiler.
- Mode (0,1)…0=stop logging when log full; 1=overwrite oldest sample if log full.
- Logger ID number…integer (0-7) that identifies this logger in this unit for the DumpLogTLM module.

Primary Outputs:

- Number of records logged…Name.Log…integer indicating how many records have been logged
- First index…Name.First…integer index pointing to oldest record in database.
- Next index…Name.Next…integer index pointing to next record in database to be written.
- Number of records saved…Name.Num…integer record count presently stored.

Outputs for Internal Use: None

Limitations: None

Expected Applications:

Use for simultaneous storage of multiple values with the same time tag.

MaxValue

The MaxValue module is used to latch the maximum value that an input has attained since the module was last reset. Upon reset, the output value is set to match the input value. If at any time the input value exceeds the output value, then the output value is set to the input value.

Inputs That Must be Constants: None

Other Inputs:

- Enable…status input, 0=hold result, 1=calculate new result
- Input…floating point input whose maximum value is to be captured.
- Reset…status trigger input that will reset the latch to the present value.

Primary Outputs:

- Maximum value…Name.Max…floating point output equal to the maximum value the input has attained since the last reset.

Outputs for Internal Use: None

Limitations: None

Expected Applications:

Use to capture maximum reservoir level, flow rate, temperature, etc.

MinValue

The MinValue module is used to latch the minimum value that an input has attained since the module was last reset. Upon reset, the output value is set to match the input value. If at any time the input value falls below the output value, then the output value is set to the input value.

Inputs That Must be Constants: None

Other Inputs:

- Enable…status input, 0=hold result, 1=calculate new result
- Input…floating point input whose minimum value is to be captured.
- Reset…status trigger input that will reset the latch to the present value.

Primary Outputs:

- Minimum value…Name.Min…floating point output equal to the minimum value the input has attained since the last reset.

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Use to capture minimum reservoir level, flow rate, temperature, etc.

TotalizeEvent

This module is designed to totalize a value, such as a flow, during an event and to latch the total at the end of the event. The event start and end are designated either by a status that goes true to start the event and false to end the event; or by the measured analog value exceeding a threshold to start the event, ending the event when the value falls below the threshold. During the event, the module totalizes the analog value and ceases totalizing when the event ends. Then it presents the total as an output value, along with the event duration in seconds, and start and end times as strings. Some rescaling math is included in this module. The value compared against the threshold is the input analog value times multiplier M1. The value totalized is input value * M1 * M2. Multipliers M1 and M2 can be used to rescale the measured value to alternate units. If no rescaling is required, values M1 and M2 can be left blank. This module can be used to calculate pump efficiency by using a pump running contact to trigger the event, and then dividing the total flow by the event duration to obtain flow per unit time.

Inputs That Must be Constants:

- String length chars…integer maximum string size of time tags. Use a value of 40 here.

Other Inputs:

- Input to totalize…floating point value to be totalized.
- Multiplier 1…floating point value M1 that is multiplied by the analog input before threshold comparison.
- Multiplier 2…floating point value M2 that is multiplied by analog input and M1 before totalization.
- Event threshold/dropout…floating point value that must be exceeded by the analog input * M1 to begin an event. When analog input * M1 falls below this threshold, the event is terminated.
- Reset trigger…status input that will zero the totalizer and arm for another event.
- Force event status input…status input whose OFF to ON transition will start an event; and whose ON to OFF transition will terminate an event.
- Enable…status input that will enable operations when true and disable operations when false.

Primary Outputs:
- Latched total…Name.Total…floating point total of (input * M1 * M2) latched at end of event.
- Latched event sec…Name.Sec…integer duration of event in seconds, latched at end of event.
- Latched start time…Name.Start…string time tag of start of event
- Latched end time…Name.End…string time tag of end of event.
- Event in progress status…Name.Event…status output that will be true during an event and return to false at the end of the event.
- Event done trigger…Name.Done…status trigger output that will be true for one scan at the end of the event, indicating that new values have been latched into module's latched outputs.
- Present total…floating point running total of (input * M1 * M2) during an event in progress.

Outputs for Internal Use:
- Event timer…Name.Time…integer timer to accumulate event time
- Accumulator…Name.Accum…floating point sample accumulator
- Sample counter…Name.Cnt…integer counter of samples taken so far during event
- Force event image…Name.Img…status image of force status input to detect changes
- Temp string…Name.TempStr…string latch of beginning time tag

Limitations: None

Expected Applications:
Use to gather statistics of short term events and calculate pump efficiency, energy consumption, etc.

TotalizeFlow

This module will totalize flow whenever the flow input exceeds the low flow dropout. It will perform a fine flow accumulation, taking a sample each scan for one minute, and then add that total to its flow accumulator at the end of each minute. You can select from four input flow engineering units (CFS, GPM, GPS, MGD) and have the totalizer provide output total in one of five total flow engineering units (Cuft, gallons, Kgal, Mgal, acre ft). A preset input and trigger are provided so the operator can preset the flow to a desired value, or zero it as necessary.

Inputs That Must be Constants: None

Other Inputs:
- Flow input…floating point input that is to be totalized whenever it exceeds the low flow dropout threshold.
- Units of flow input (1-4)…integer selection of input engineering units: 1=CFS, 2=GPM, 3=GPS, 4=MGD
- Dropout…floating point value in the same units as the flow input below which the flow will not be totalized.
- Units of total (1-5)…integer selection of output engineering units: 1=Cuft., 2=gallons, 3=Kgal, 4=Mgal, 5=acre ft.
- Preset trigger…status trigger input that when true will transfer the preset input value to the total output
- Preset input value…floating point input in the same units as the total output that will be loaded into the total output to become the new beginning total.
- Enable…status input that must be true to enable totalizing. False disables totalizing.

Primary Outputs:
- Total flow…Name.Total…floating point total flow output in output engineering units.

Outputs for Internal Use:
- Minute holder…Name.Min…integer minute value to detect new RTC minute.
- Sample counter…Name.cnt…integer count of samples taken this minute, maximum 65535 counts.
- Sample accumulator…Name.Accum…floating point flow sample accumulator this minute.

Limitations: None

Expected Applications:
Use for general purpose flow totalization.

TotalizeTime

This module will totalize time whenever an input status is ON. It will perform a fine time accumulation each minute, and then add that total to its time accumulator at the end of each minute. Output is time in hours. A preset input and trigger are provided so the operator can preset the time total to a desired value, or zero it as necessary.

Inputs That Must be Constants: None

Other Inputs:
- Status input…status input that, when true, indicates totalization is to proceed, and, when false, indicates totalizion is to be suspended. This is usually a status that indicates when a piece of equipment is running.
- Preset trigger…status trigger input that when true will transfer the preset input value to the total output
- Preset input value…floating point input that will be loaded into the total output to become the new beginning total.

Primary Outputs:
- Total hours…Name.Total…floating point total running time output in hours

Outputs for Internal Use:
- Minute holder…Name.Min…integer minute value to detect new minute.
- Sample counter…Name.cnt…integer count of samples taken this minute.
- Sample accumulator…Name.Accum…floating point time accumulator this minute.

Limitations: None

Expected Applications:
Use for general purpose equipment run time totalization.

Communications Modules

ComSetup

The communications setup module is used to establish the protocol, baud rate, tone use, etc. of each serial communication port in the RUG3. At least one **ComSetup** module must be supplied for each serial port installed in a unit, and each must be triggered before the port can be used to send or receive data. The exception is port 1, the programming RS232 port, designated P1, which defaults to 9600,N,8,1, ASCII on boot up. Port number input must be a constant. All other inputs can be either constants or variables except the "Trigger to install setup" input, which must be a trigger input, such as the SysSetup module's System.BootTrg output, which would cause the **ComSetup** module's parameters to be installed right after boot up. Note that you may have as many **ComSetup** modules as you wish for each port; the last one to be triggered will govern the behavior of the port.

Setting Communication Protocol

The **ComSetup** module establishes the communications protocol, or mode, the serial port is to use. These are the available choices:

- 1…ASCII In this mode, the port uses the ASCII character set to send and receive human-readable text. This mode is used when the port is to provide formatted displays to be read by a human operator.
- 2…RUG9 protocol This protocol is the preferred protocol to be used in RUG3 to RUG3/5/9 communications. It can employ messages up to 254 characters, or 120 integers, or 60 floating point values per message. It can also provide automatic store and forwarding involving up to three intermediary stations.
- 3…RUG6 protocol. This protocol matches that used in the RUG6,7 and 8 units. It should be used if the system consists of a mix of RUG6,7, or 8 units that must interact with a RUG3. This protocol enables a RUG3 to be added to a RUG6 system without requiring modification to the RUG6 system.
- 4…MB slave protocol. This is the common Modbus RTU protocol (as opposed to Modbus ASCII protocol). This, or the MB slave2 protocol, should be used when communicating with a SCADA master. In this implementation, the RUG3 maintains the transmit and receive arrays as separate; i.e., commands to read from the RUG3 will obtain data from the RUG3's transmit array, and commands to send data to the RUG3 will send data into the RUG3's receive array.
- 5…MB master protocol This protocol enables the RUG9 to poll Modbus slave RTU's and collect data from them. In this mode, you specify the message type, slave address, starting register and number of registers to be transferred, and the RUG9 assembles the correct message and transmits it, then collects the reply. This should be used to poll non-RUGID RTU's that use the Modbus RTU slave protocol.
- 6…MB slave2 protocol. This protocol is almost the same as mode 4 above. The difference is that polls to read from RUG3 holding registers will obtain their values from the RUG3's receive array rather than the RUG3's transmit array. This more closely corresponds to the standard Modbus operation than does mode 4 above since it enables the SCADA master to read back setpoints and control bits it has written to the RUG3.
- 7…ASC user. In this case, the RUG3's operating system will not act upon characters received, but will instead leave them in the input buffer for parsing by other modules. This mode should be used for communications with instruments that use ASCII to send and receive data.
- 8…ALERT. This protocol uses slightly different modem tones and enables the RUG3 to function as an ALERT transmitter for reporting remote environmental data.
- 9…True Wireless…same as RUG9 protocol with different CRC security
- 11…Modbus TCP slave. Same as mode 4 above except TCP compatible.
- 12…Modbus TCP slave mode 2. Same as mode 6 above except TCP compatible.
- 13…Modbus TCP master. Same as Mode 5 above except TCP compatible.

Inputs That Must be Constants:
- Port # (1,2)…integer designating which port on the board this module references (1=pgm serial port, 2=modem/RS232/RS485 port).
- Com buffer # bytes…integer that defines buffer size allocated to this port.

Other Inputs:
- Trigger to install setup…status trigger input. This input TRUE causes this module's setup to be installed in the designated port. This should be done at boot up and/or periodically, say once per hour.
- 0=232, 1=Mdm, 2=485…integer to specify hardware mode of channel…RS232, 300 baud internal modem, or RS485. RS485 is optional.
- Baud (50-56,000)…integer specifying any baud rate from 50 to 56,000 baud. You are not limited to standard baud rates. The actual baud rate is derived from a clock and calculation as BAUD=115,200/[(integer)(115,200/baud designator)]. When the modem is in use, the system will set the modem for FSK low tones operation for all baud rates 300. Recommended: 300 baud for modem use, 9600 baud for RS232 use.
- Parity…integer specifying parity choice: 0=none, 1=odd, 2=even, 3=mark, 4=space. Recommended: 0=none.
- Address (1-65535)…integer address for this port on the network. The address establishes this unit's address in a system of RUGID units. No two units in a system can have the same address, except that any number of units can have the same address as long as only one is allowed to transmit. The others would be listen-only in that case. Address range 1-255 forces use of one byte address in msg header. For R9 protocol, if address>255, unit uses 2-byte address with range of 1-65535.
- Mode (1-13)…integer to select communication protocol. See above.
- TX delay tenths of sec…integer to set delay between the time the RUG3 keys its modem and radio, and the time it actually sends data. This is necessary to enable receiving radios and modems to acquire the signal. Recommended: for phone line applications, set this to 15; for radio applications, set it to 15.
- TX amplitude (0-255)…integer to set output transmitter amplitude. 0=no output, 255=max output.
- Com flags…integer in form of packed bits: bit 0 (LSB)=spare, bit 1=suppress trailing nulls, bit 2=suppress reply if #ReplyRegs=0. Usually leave blank.

Primary Outputs:
- Receive in progress…Name.Rx…status output will be true while data being received.
- Transmit in progress…Name.Tx…status output will be true while data being transmitted.

Outputs for Internal Use: None

Limitations: None

Expected Applications:
ComSetup must be used to set necessary parameters for each serial port in the system.

ComWatch

When triggered, the module will read the designated serial port buffer and send an ASCII version of the contents to the other serial port in hex-ASCII format. When triggered to erase the port buffer, the module will write hex 0's to the designated port's buffer. This module is intended to enable you to read buffer contents in order to debug serial port problems by examining the contents of the buffers to see if the received/transmitted strings are correct or to see if the port is working at all. The output format is the following:

```
54 65 73 74 20 73 74 Test st
72 69 6E 67 20 23 31 ring #1
2E 65 73 74 20 73 74 ,Test s
73 74 72 69 6E 67 20 tring #
32 0D 0A 00 00 00 00 2......
00 00 00 00 00 00 00 .......
00 00 00 00 00 00 00 .......
00 00 00 00 00 00 00 .......
00 00 00 00 00 00 00 .......
00 00 00 00 00 00 00 .......
00 00 00 00 00 00 00 .......
00 00 00 00 00 00 00 .......
```

Inputs That Must be Constants:
Port#…integer port number (1-2) to be read or erased.

Other Inputs:
Trigger read port buffer…status trigger that when true will cause the unit to read the designated serial port and send the contents to the alternate serial port.
Trigger erase port buffer…status trigger that when true will cause the unit to write all zeros into the designated buffer.

Primary Outputs: None

Outputs for Internal Use:
- Progress index…Name.Prog…integer for tracking writing of output to buffer.

Limitations: None

Expected Applications:
Use to debug serial port performance.

CycleDisplay

The **CycleDisplay** module is a sequencer that enables the programmer to specify a range of displays to be presented automatically with controllable dwell time on the displays. When enabled, controls cycling through user defined displays. Port range is 0 (LCD), 1 (programming port) and 2 (modem/RS232 port). Sequencer will cycle from min through max display numbers then restart with min display. Dwell sec specifies how long unit will dwell on each display. Holdoff time specifies how long the module will wait before resuming cycling after the user has accessed one of the menus on the designated port.

Inputs That Must be Constants:

- Port#…integer port number (0-2).

Other Inputs:

- Enable…status input: 0=disable cycling, 1=enable cycling.
- Min display #...integer specifying lowest numbered display to start the cycle
- Max display #...integer specifying highest numbered display in the cycle
- Dwell sec per display…integer seconds a display is to be shown before moving to the next display in the cycle
- Holdoff time sec…integer seconds that cycling is to be disabled after a user keystroke.

Primary Outputs:

- Present display number…Name.Dsp…integer number of display presently being displayed.
- Present menu sequence…Name.Menu…integer indicating which menu presently being displayed

Outputs for Internal Use:

- Timer…Name.Tmr…integer timer used for all timing.

Limitations: None

Expected Applications:

Use to automatically cycle user displays.

DumpLogTLM

DumpLogTLM module prepares responses to polls requesting dumps of logged data using the R9 CRC-secured format. This module works with either LogMany or EventLogSetup modules. This module will decode the dump request received by telemetry, find the logger using the logger ID contained in the message, prepare the response, and then trigger transmission of the reply.

Inputs That Must be Constants:

- Port (1 or 2)…integer indicating port to use (1-2).

Primary Outputs: None

Outputs for Internal Use:

- Received control field…Name.Ctrl…integer control field of received request.
- Cmd 1st index to send…Name.Cmd1st…integer identifying first logged item to send.
- Cmd num indices to snd…Name.CmdNum…integer specifying number of logged items (records) to send.
- Format word 1-5…Name.F1-5…integer with packed bits specifying format of analogs to be dumped.
- Next index to send…Name.Next…integer index to send in next message.
- Number indices to go…Name.Num…integer indicating how many records are left to send.
- Last index sent…Name.Last…integer first index sent in last message.
- Last number to go…Name.LNum…integer number of records to go as of last message sent.

Limitations: None

Expected Applications:

Use to send handle requests for dumping of logged data using R9 protocol.

DumpLogToPort

This module is used to read a data log, format it to ASCII, and dump it to the designated port. To use this module, you must identify the data logger to use, and then trigger the dump. When triggered, the module sends logged data to designated port in ASCII form. Header fields, if defined, precede the first time tag in the dump. The dump starts with first sample and ends when the specified number of records have been dumped or at end of file. Each line consists of date field followed by one or more data fields. A new line starts when next date code is found. One delimiter char separates data entries on each line (default comma). Special fields in the header, notably the period character, must be entered using <123> format, where the value between <> characters specifies the decimal equivalent of an ASCII character. Header characters that MUST use <> fields are: period=<46>, left carat=<60>, right carat=<62>, and vertical bar=<124>. The special format field identifies dump format: 0 or blank=ASCII format with delimiters; 1=WISKI format. Mode specifies the direction through the log: 0=start with oldest record, 1=start with newest record. First field to dump on line specifies which of up to 20 inputs logged in each record is to be sent first after the time tag. # of fields to dump per line sets how many logged inputs are to be dumped per line. The module avoids overrunning the transmit buffer by looking at buffer vacant capacity before sending the next sample. When done, the module will issue a trigger that you can use to clear the log, if necessary, or issue a prompt, etc. The dumped data format is the following:

```
11/15/2006 13:14:36,1.15,1.56,2.78,4.76,4.77,4.79,5.1,5,5.01,5.12
11/15/2006 13:14:45,5.15,5.25,5.26,5.78,4.95,4.34,4.02,3.78,3.41
```

A time/date tag found in the data will cause the module to issue a carriage return/line feed to start a new line. Therefore, each line will begin with a time tag, taken from the logged data file, followed by a series of data samples converted to numeric strings and formatted as specified in this module. The delimiter character between each field can be any ASCII character; comma is the default. Using the comma character makes the file comma-delimited, so it is compatible with Microsoft's Excel spreadsheet along with others.

Inputs That Must be Constants:

- Send to Port #…integer indicating port to use (1-2).

Inputs That Must be Pointers:

- Logger holding data…integer pointer to data logger's data. It is the logger's Name.Log output from the floating point database.

Other Inputs:

- Trigger dump…status trigger input that initiates data dumping sequence.
- Delim char (44=coma)…integer defining which character is to be used to separate each sample or time tag in the file. The following are commonly used characters as delimiters: 44=comma, 20=blank, 124=pipe
- Chars right of decimal…integer in range of 0 to 7 specifying how many characters to the right of the decimal are to be included in conversion to ASCII before dumping each sample. Since samples are stored as floating point numbers, full precision is available in the data log.
- Special format…integer indicating any special formatting. Blank or zero gives standard ASCII. 1=LADWP WISKI format.
- Mode…integer, 0=dump oldest to newest samples, 1= dump newest to oldest samples.
- # records to dump…integer specifying how many records to dump.
- First field to dump on line…integer specifying which field in record (1-20) is to be the first dumped on each line.
- # fields to dump per line…integer specifying number of fields in record to be dumped in each line.
- Header string #1-11…strings to be dumped at the beginning of each dump.

Primary Outputs:
- End of line trigger…Name.LineTrg…trigger issued at end of each dumped line.
- End of log trigger…Name.LogTrg…trigger issued at the completion of each data dumping event.

Outputs for Internal Use:
- Sequencer…Name.Seq…integer internal state sequencer
- Next index…Name.Nxt…integer internal index counter
- Count…Name.Count…integer count of records sent.

Limitations: None

Expected Applications:
Use to send formatted data log to serial port.

GetStrFromPort

This module captures a string from a port, up to a CRLF and makes the string available at its output. Its primary use is to capture strings so they can be later parsed by the **ParseStr** module. As each string is captured, when the module encounters a carriage return/line feed character pair, it will issue a trigger to signal subsequent modules that a new string has been received. Note that the port with which this module interfaces must be set for mode 7, ASCII User. This mode prevents background software from responding to user entries such as responses to menu prompts.

Inputs That Must be Constants:
- Output string max chars…integer that defines how much RAM is to be allocated for this module's string output. This number must be larger than the largest string this module is to receive, or else some data will be lost from the string.
- Port #…integer in range of 1 to 2 specifying with which port on the board this module is to work.

Other Inputs:
- Trigger to clear buffer…status trigger input to reset input buffer. Generally used to resynchronize to input signal stream.

Primary Outputs:
- Output string…Name.Str…string having characters captured from port so far. Will be complete string when trigger below is issued.
- New string trigger…Name.NewTrg…status trigger output true when complete string has been received.
- New string length bytes…Name.Length…integer indicating length of newly output string in bytes.
- New character trigger…Name.CharTrig…trigger indicating that a new character has arrived in buffer.
- Characters in buffer…Name.Charcnt…integer indicating number of characters in buffer.

Outputs for Internal Use:
- Temporary string accum…Name.StrTemp…string being built from received characters.
- Temp string index…Name.IdxTemp…integer index into temporary string

Limitations: None

Expected Applications:
Use this module to accept characters from a port for use by other modules.

ParseString

The ParseString module reads characters from input string and parses fields into individual strings and floating point outputs. If a header string is specified, then parsing to outputs will only occur if the incoming string begins with a header that exactly matches the header string. If no header string is specified, then all strings will be parsed. If a delimiter character is specified, then the parser will assume that fields are separated by one delimiter character. If the delimiter is designated as the blank (<32>) character, then multiple blank characters will be regarded as a single delimiter. If the delimiter is set to zero, and an implied field length is specified, then the parser will not look for delimiter characters but will instead assume that each field is exactly the length specified by the implied field length. The start index offset specifies starting field #. For example, a start index of 5 means that the parser will only start parsing and saving outputs after the 5th field.

Inputs That Must be Constants:

- Output string max chars…integer setting the size of each output string.

Inputs for Internal Use:

- Set by compiler…integer set by compiler specifying type of parser.

Other Inputs:

- Trigger to convert…input status that enables module operation and conversion of input string. This is usually the new string trigger output from the GetStringFromPort module.
- String to parse…input string to parse into individual strings and, if possible, floating point values.
- Delim char 44=comma…integer decimal equivalent of ASCII character that designates start of next field.
- Start index offset…integer number of fields to skip before starting to parse outputs.
- Header string to find…string to find before starting to parse.
- Implied field length X…integer specifying field length for strings that have no delimiters.

Primary Outputs:

- Parsing done…Name.DunTrg…status trigger output indicating that a message was parsed.
- Field 1-10 string…Name.F1Str…string outputs after parsing
- Field 1-10 numeric…Name.F1Val…value of individual parsed strings.

Outputs for Internal Use: None

Limitations: None

Expected Applications:

Used to pull values and other fields from serial strings.

ParseStringToFloat

The ParseStringToFloat module reads characters from input string and parses fields into individual floating point outputs. If a header string is specified, then parsing to outputs will only occur if the incoming string begins with a header that exactly matches the header string. If no header string is specified, then all strings will be parsed. If a delimiter character is specified, then the parser will assume that fields are separated by one delimiter character. If the delimiter is designated as the blank (<32>) character, then multiple blank characters will be regarded as a single delimiter. If the delimiter is set to zero, and an implied field length is specified, then the parser will not look for delimiter characters but will instead assume that each field is exactly the length specified by the implied field length. The start index offset specifies starting field #. For example, a start index of 5 means that the parser will only start parsing and saving outputs after the 5th field.

Inputs That Must be Constants:

- Output string max chars…integer setting the size of each output string. Unused in this module…set to zero.

Inputs for Internal Use:

- Set by compiler…integer set by compiler specifying type of parser…leave blank.

Other Inputs:

- Trigger to convert…input status that enables module operation and conversion of input string.
- String to parse…input string to parse into individual floating point values.
- Delim char 44=comma…integer decimal equivalent of ASCII character that designates start of next field.
- Start index offset…integer number of fields to skip before starting to parse outputs.
- Header string to find…string to find before starting to parse
- Implied field length X…integer specifying field length for strings that have no delimiters.

Primary Outputs:

- Parsing done…Name.DunTrg…status trigger output indicating that a message was parsed.
- Field 1-10 value…Name.F1Val…value of individual parsed strings.

Outputs for Internal Use: None

Limitations: None

Expected Applications:

Use to pull floating point values from serial strings.

ParseStringToInt

The ParseStringToInt module reads characters from input string and parses fields into individual integer outputs. If a header string is specified, then parsing to outputs will only occur if the incoming string begins with a header that exactly matches the header string. If no header string is specified, then all strings will be parsed. If a delimiter character is specified, then the parser will assume that fields are separated by one delimiter character. If the delimiter is designated as the blank (<32>) character, then multiple blank characters will be regarded as a single delimiter. If the delimiter is set to zero, and an implied field length is specified, then the parser will not look for delimiter characters but will instead assume that each field is exactly the length specified by the implied field length. The start index offset specifies starting field #. For example, a start index of 5 means that the parser will only start parsing and saving outputs after the 5th field.

Inputs That Must be Constants:

- Output string max chars…integer setting the size of each output string. Unused in this module…set to zero.

Inputs for Internal Use:

- Set by compiler…integer set by compiler specifying type of parser…leave blank.

Other Inputs:

- Trigger to convert…input status that enables module operation and conversion of input string.
- String to parse…input string to parse into individual integer values.
- Delim char 44=comma…integer decimal equivalent of ASCII character that designates start of next field.
- Start index offset…integer number of fields to skip before starting to parse outputs.
- Header string to find…string to find before starting to parse.
- Implied field length X…integer specifying field length for strings that have no delimiters.

Primary Outputs:

- Parsing done…Name.DunTrg…status trigger output indicating that a message was parsed.
- Field 1-10 value…Name.F1Val…value of individual parsed strings.

Outputs for Internal Use: None

Limitations: None

Expected Applications:

Use to pull integer values from serial strings.

ParseStringToStatus

The ParseStringToStatus module reads characters from input string and parses fields into individual status outputs. If a header string is specified, then parsing to outputs will only occur if the incoming string begins with a header that exactly matches the header string. If no header string is specified, then all strings will be parsed. If a delimiter character is specified, then the parser will assume that fields are separated by one delimiter character. If the delimiter is designated as the blank (<32>) character, then multiple blank characters will be regarded as a single delimiter. If the delimiter is set to zero, and an implied field length is specified, then the parser will not look for delimiter characters but will instead assume that each field is exactly the length specified by the implied field length. The start index offset specifies starting field #. For example, a start index of 5 means that the parser will only start parsing and saving outputs after the 5th field.

Inputs That Must be Constants:

- Output string max chars…integer setting the size of each output string. Unused in this module…set to zero.

Inputs for Internal Use:

- Set by compiler…integer set by compiler specifying type of parser…leave blank.

Other Inputs:

- Trigger to convert…input status that enables module operation and conversion of input string.
- String to parse…input string to parse into individual status values.
- Delim char 44=comma…integer decimal equivalent of ASCII character that designates start of next field.
- Start index offset…integer number of fields to skip before starting to parse outputs.
- Header string to find…string to find before starting to parse.
- Implied field length X…integer specifying field length for strings that have no delimiters.

Primary Outputs:

- Parsing done…Name.DunTrg…status trigger output indicating that a message was parsed.
- Field 1-10 value…Name.F1Val…status value of individual parsed strings.

Outputs for Internal Use: None

Limitations: None

Expected Applications:

Use to pull status values from serial strings.

Poll

The **Poll** module is used to prepare and issue a poll message directed to a specific station address. The poll will be sent on the modem/RS232 port using the protocol established by the communication setup module above. Destination address, specific words to be transmitted, and forwarding path are established by this module. A poll will be issued each time this module's trigger input transitions from false to true. Transitions that occur while the prior poll is still being transmitted will initiate a new transmission and corrupt both transmissions. Once the poll trigger is received by this module, it will access the transmit array for the port and destination station specified and prepare the outgoing poll, using words taken from the transmit array, which it places in the port's transmit buffer, ending with CRC security. It then enables transmit interrupts to start the output timing and transmission process. When the message has been transmitted, the interrupt software will turn off transmit interrupts and enable receive interrupts to capture any reply. For systems involving only two stations, this module alone, properly triggered, is sufficient to do polling. For larger systems, where round robin polling is required, it is recommended that you use the **SequenPoll** module to control this module.

Setting Word Selections

Word settings establish which words in the specified transmit array are to be sent in the outgoing poll; and how many words the reply message is to bring back from the destination station. Each word is a 16 bit integer. In the case of the RUG3, a floating point value takes two words.

In the case of the RUG9 protocol, the word selections work as you would expect. Specifically, the "First word to send" specifies the first word in the transmit array to be sent in the message. The "Number of words to send" specifies how many words from the transmit array will be sent. These are also true for specifying which words the reply is to contain.

Setting Forwarding Path

The forwarding addresses enable you to specify up to three intermediary stations through which the poll will be passed to get to a final destination. If you specify one or more forwarding addresses, the message will be sent from the polling unit, to forwarding address 1, to forwarding address 2, to forwarding address 3, then to the final destination. The destination's reply will be returned back through the same path in reverse order. Any forwarding address left blank, or set to a negative address will be skipped in the forwarding path. If no forwarding is required, you can leave the forwarding addresses blank.

Inputs That Must be Constants: None

Other Inputs:

- Trigger input…status that, when going from false to true, causes the module to issue a poll.
- Destination station address…integer address in range of 1 through 255 designating destination station.
- First word to send…integer that, for RUG9 protocol, specifies first word in transmit array to send in poll.
- Number of words to send…integer that, for RUG9 protocol, specifies number of words to transmit from transmit array. Max value for RUG3 is 125.
- First word to reply…integer that, for RUG9 protocol, specifies first word in transmit array at destination to send in reply. Range is 0 through 65535.
- Number of words to reply…integer that, for RUG9 protocol, specifies number of words to transmit from destination's transmit array. Max value for RUG3 is 125.
- Forward station #1, #2, #3…integer in range of 1 through 255 that specifies addresses of stations who are to forward the message to the next station in the path.

Primary Outputs: None

Outputs for Internal Use: None

Limitations:

Expected Applications:
Use to perform details of polling. Usually used in conjunction with **SequenPoll** module.

PollModbus

The PollModbus module is used to issue polls to Modbus slave devices using the Modbus RTU protocol. The poll will be sent on the designated port using the protocol established by the **ComSetup** module above. Note that the protocol can be either Modbus RTU for serial communications directly with slave devices including other RUG5/9 units; or Modbus TCP for communications with slaves over the ethernet. Destination slave address, message type and specific words to be transmitted are established by this module. A poll will be issued each time this module's trigger input transitions from false to true. Transitions that occur while the prior poll is still being transmitted will initiate a new transmission and corrupt both transmissions. Once the poll trigger is received by this module, it will access the transmit array for the port and destination station specified and prepare the outgoing poll, using words taken from the transmit array, which it places in the port's transmit buffer, ending with CRC security. It then enables transmit interrupts to start the output timing and transmission process. After the message has been transmitted, the interrupt software will turn off transmit interrupts and enable receive interrupts to capture any reply. This module works for Modbus polling only. For RUG6 or RUG9 polling, use the Poll module. For systems involving only two stations, this module alone, properly triggered, is sufficient to do polling. For larger systems, where round robin polling is required, it is recommended that you use the **SequenPoll** module to control this module.

Address offset range +/- 32767. MB item# + offset=R9 item#. For example: to read MB register 5039 and have it appear in Rugid register 10, first slave item to send/rcv should be 5039 and offset should be -5029. To write register 9456 with data in Rugid register 3, first slave item to send/rcv should be 9456 and offset should be -9453.

Message Types Supported

The Modbus protocol uses a separate function code for each type of message. The **PollModbus** module supports the following function codes/message types:

Table 6 Modbus Function Codes Supported

Function Code	Use	Items to Send/Rcv Refers to…
1	Read multiple coils	Statuses
2	Read multiple status inputs	Statuses
3	Read holding registers	Words
4	Read input registers	Words
5	Force coil	Status
6	Preset register	Word
15	Force multiple coils	Statuses
16	Preset multiple registers	Words

When you use the **PollModbus** module, you must specify which one of the message types above is to be used in the poll. You can change the message type as necessary to support the polling you wish to do, or you can have a different **PollModbus** module for each message type, and just trigger the one to issue the poll. The easiest way is to use a table to control inputs to this module, and just sequence through the table.

Inputs That Must be Constants:
- Port #…integer in range of 1 to 2 specifying with which port this module is to work.

Other Inputs:

- Trigger to send…status that, when going from false to true, causes the module to issue a poll.
- Slave address…integer address in range of 1 through 255 designating destination station
- First item to send/rcv…integer that specifies first word in transmit array to send in poll. Range is 0 through 32767.
- Number of items to send/rcv…integer that specifies number of words to transmit from transmit array. Max value is 125.
- Message type…integer specifying message type as identified in table above.
- Register/coil item # offset…integer that is added to the coil or register address before transmission of polls referencing coils. Range is +/- 32767.

Primary Outputs: None

Outputs for Internal Use: None

Limitations: None

Expected Applications:

Used to perform details of polling.

QuiescentController

The quiescent controller provides the sequencing, logic and time delays to implement a report by exception communications controller for a remote site. Its primary job is to monitor 16 statuses and latch when any one or more change state. If a change is detected and the time delay has expired, the module will assert its trigger output to cause a poll module to poll immediately. If the timer has not expired, the latch is held until the timer times out, at which time the trigger is asserted, and the timer restarted. When the time delay is restarted, it is initiated with the sum of a fixed part and a random part, to minimize the possibility of synchronously blocking another quiescent station repetitively. In addition, if the module fails to obtain a status that indicates that the poll had a reply, it will count such failures and, when the count exceeds a blockage threshold, will double, then triple, etc., the time delay, to adjust to the level of activity of the communication channel. When the module receives a reply, it steps down the blockage factor until normal delays are restored. In this manner, the module will respond to excessive call blockages by slowing its calling rate to minimize the possibility of the channel becoming clogged in times of high activity.

Inputs That Must be Constants: None

Other Inputs:

- RCV/Reset latch…status input that, when true, indicates that a message was received; i.e., a reply to the last poll. This is usually obtained from the TrigOnRcv module.
- Mode (0-2)…integer specifying the sense of status changes to trigger transmission: 0=any change, 1=off to on, 2=on to off.
- A fixed delay sec…integer fixed component of time delay where delay before next transmission allowed=A+B*rand().
- B random delay sec…integer random component of time delay where delay before next transmission allowed=A+B*rand().
- Max retries (0=forever)…integer number of times the module will retry a transmission before resetting its change latch. Value of zero or blank will enable unlimited retries.
- Blockage thresh (0=none)…integer number of successive failed responses at which the time delay will be stepped out by the blockage factor to slow transmissions.

- Trigger now #1 and #2…trigger to assert output trigger without delay and restart delay. This is implemented to provide convenient manual transmission triggering.
- Input #1 through input #16…status inputs whose change is to cause transmission

Primary Outputs:
- Trigger output…Name.Trg…trigger output to poll module to cause poll to be issued.
- Latched changes…Name.Latch…status output indicating that a change has been detected, but the module is waiting to transmit, or the module has transmitted with no reply, so is waiting to retry.

Outputs for Internal Use:
- Image…Name.Img…integer copy of last set of status inputs to watch for change.
- Timer…Name.Tmr…integer delay timer
- Retry counter…Name.Retries…integer retry counter
- Blockage counter…Name.Bcnt…integer counter of messages that have not received a reply (cleared on reply).
- Blockage delay factor…Name.Bfactor…integer blockage step out factor to slow transmissions.
- Trigger end of retries…Name.RetryDone…trigger status indicating last retry has been tried.

Limitations: None

Expected Applications:
Use to control polling in remote sites that must report by exception.

SendAlertData

The **SendAlertData** module formats data taken from its inputs into the standard ALERT 4-byte format and sends the data out the modem port using ALERT tones. When triggered, sends ID/data combination pairs using the ALERT format. ID range is 0-8191, data value range is 0-2047. Each pair results in a 4 byte ALERT standard message being sent to port 2. Up to 10 ID/data pairs can be defined to constitute one transmission. Module will install data in buffer for transmission until it encounters a blank ID or hits the end of the input list.

Inputs That Must be Constants: None

Other Inputs:
- Trigger input…status that when true initiates transmission immediately
- ID #1-10…integer ID numbers for up to 10 measurements being sent, range is 0-8191.
- Data #1-10…floating point data to be sent along with above ID’s. Range is 0-2047.

Primary Outputs: None

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Use to format any data in RUG3 into ALERT format for acceptance by ALERT base stations.

SendStrtoPort

This module is used to create a string from set of input strings and floating point values. The output string is assembled from the input values and strings in the order they appear in the module's input list. The output string can have its component elements separated by a user specified delimiter character. Each of the module's 20 inputs can be either a string or a floating point value. String inputs are assembled into the output string unchanged. Numeric inputs are converted to strings with user defined precision before being assembled into the output string. If a port number is specified, the string is sent to the designated port for transmission, in addition to being emitted from this module as a string. Fields embedded in string inputs of the form <nnn> will be converted to the ASCII equivalent character before being sent to a port. E.g., the field 'ABCD<63>EFG' would result in the string 'ABCD?EFG' being sent. Individual inputs can be strings of up to 40 characters.

Inputs That Must be Constants:

- Output string max chars…integer that defines the RAM to be allocated for the output string. Should be longer than the longest string anticipated to be created by this module. If set to zero, the output string will be suppressed.
- Port #…integer in range of 1 to 2 specifying with which port on the board this module is to work. If left blank, the module will simply create the string as an output, but will not send it out a port.

Other Inputs:

- Trigger input…status trigger input that, when true, will cause the module to perform a new string conversion.
- Mode flags...integer specifying prepending options: 0=no prepending, 1=prepend leading zeroes, 2=prepend leading blanks.
- Delim char (44=comma)…integer form of the desired delimiter character entered as the decimal equivalent of an ASCII character. The following are commonly used characters as delimiters: 44=comma, 20=blank, 124=pipe. Leave blank if no delimiter is desired.
- Float format…floating point value to designate desired floating point format: [chars left of decimal].[chars right of decimal]. For example, the value 5.3 would make the module format numeric inputs with up to 5 characters to the left of the decimal and three characters to the right of the decimal.
- End char (10=CRLF)…integer form of the desired terminating character. A value of 10 gives the CRLF character pair. Leave blank if no terminating character is desired.
- Input 1 through 20…either numeric or string inputs to be placed in order in the output string. Blank inputs are ignored.

Primary Outputs:

- Output string…Name.Str…string produced by this module. This is the aggregate of all the inputs converted as necessary to string form.
- String ready…Name.RdyTrg…status trigger output indicating that a new string is present on the output.
- String sent/buffer empty…Name.SntTrg…status trigger output indicating that the port has finished transmitting the string.

Outputs for Internal Use:

- Flag…Name.Flg…status output, internal general purpose flag
- Trigger image…Name.Img…status image of trigger input to watch for rising edge

Limitations: None

Expected Applications:

Use this module to prepare strings to send users, or to send to instruments that need ASCII string inputs.

SequenPoll

The **SequenPoll** module provides the logic, sequencing and timing necessary to implement round robin polling, as a master would do in a purely polled telemetry system. With an extended time delay after reply, it can also control infrequent polling used by the master in a quiescent system. The module's main task is to sequence through a series of remote site addresses, polling one after the other in the order specified by its state counter. In a typical application, the module's state counter output would be the address of the polled station.

One State of a Typical Polling Cycle

To illustrate the operation of this module, here is a typical polling cycle:

- Advance the module state to the next state. If the state exceeds the max state input, preset the state to the min state input value. If station on-line flag indicates off-line, keep advancing.
- Issue the com install trigger. This triggers a **ComSetup** module to initialize the serial port, if necessary.
- Issue a poll trigger. This sends the poll message. Initialize the wait timer with the wait time to receive input value.
- Waiting…If a reception detected, clear the com fail counter and start the wait after reception timer. If wait timer times out without reception, increment the com fail counter. If com fail counter exceeds com fail threshold, set com fail flag output.
- If done waiting, issue write trigger.
- Go to next cycle.

Inputs That Must be Constants: None

Other Inputs:

- Trig on Rcv input…status indicating that a reception has resulted from the last poll. This usually is obtained from a **TriggerOnRcv** module.
- Enable…status input that, if blank or true, enables polling. If set to zero, polling is halted.
- Preset trigger…status trigger input to force the sequencer to a known state.
- Preset state…integer input of the state to assume when preset trigger is asserted.
- Station on-line flag…status input: 0=skip this station, 1=poll this station
- Max state…integer setting the highest state for this module's sequencer. After the sequencer hits this state, it will transition to the min state, below.
- Min state…integer setting the lowest or beginning state of the module's sequencer.
- Poll delay after rcv sec…integer number of seconds the module is to wait after a reception before the next poll
- Wait time for rcv sec…integer number of seconds the module is to wait for a reception before declaring that the destination station has failed to reply.
- Station com fail count in…integer com fail entry…future.
- Poll count before com fail…integer number of polls in succession that fail to evoke a reply before the module declares that communications with this station has failed…future.

Primary Outputs:

- Sequencer state…Name.State…integer state of main sequencer. This can be sent to the **Poll** module as the station address.
- Poll trigger…Name.PolTrg…status trigger output sent to the **Poll** module to issue one poll.
- Write table trigger…Name.Wtrig…status trigger output…future.
- Com install trigger…Name.InstTrig…status trigger output used to trigger installation of the serial port parameters (triggers **ComSetup** module).
- Com fail count out…Name.Comcnt…integer count of how many unsuccessful polls have been issued since last successful poll…future.
- Com fail flag…Name.Fail…status: 0=com ok, 1=com failed…future

Outputs for Internal Use:

- Subsequence counter…Name.Sub…integer state counter used to control module sequencing for each poll.
- Timer…Name.Tmr…timer for internal sequencing, waiting for reception, etc.

Limitations: None

Expected Applications:

Use in all but the simplest applications to control round robin poll sequencing.

StringMid

When triggered, reads characters from the input string and emits a substring beginning with 1st character # specified and ending when it has emitted the specified number of characters or has hit the end of the string. Len output is length in characters of incoming string. Leftmost character of incoming string is character # 0. You MUST enter a constant for the output string max character length larger than your largest anticipated output string. To obtain leftmost N characters, set 1st char to emit to 0, and number of characters to emit to N. To obtain rightmost characters starting with character N, set 1st char to emit to N, and number of chars to emit to a number greater than anticipated string length (such as 1000).

Inputs That Must be Constants:

- Output string max chars…integer that defines the RAM to be allocated for the output string. Should be longer than the longest string anticipated to be created by this module. If set to zero, the output string will be suppressed.

Other Inputs:

- Trigger to convert…status input that when true will cause the module to read the input string and output the specified output string.
- 1st char # to emit…integer that specifies with which input character the output string is to begin. If set to zero, the output string will begin with the first character of the input string.
- Number of chars to emit…integer that specifies how many input characters are to be emitted. If set to a larger number than the length of the input string, then all characters to the end of the string will be emitted.

Primary Outputs:

- Conversion done…Name.DunTrg…status trigger indicating that a new conversion has been completed
- Length of input string…Name.Len…integer indicating number of characters in input string.
- Substring…Name.SubStr…string output of portion of input string.
- Length of output string…Name.LenOut…integer indicating length of substring extracted from input string.
- Ascii of first output char…Name.Asc…integer ASCII equivalent of first output character.
-

Outputs for Internal Use: None

Limitations: None

Expected Applications:

Use to extract a portion of an incoming string.

TriggerOnRCV

The trigger on receive module will issue a trigger each time a formatted message is received with correct CRC on the designated port and having the addresses specified. This module is useful for triggering a poll immediately after a reception, for collecting communications statistics, for notifying the operator of a reception, or for general purpose debugging of communications performance. The module's outputs are valid immediately after the reception. Trigger event output is true for one scan after the reception. Source and destination address outputs show the source and destination of any message received on the designated port with valid CRC, and remain valid until overwritten by another reception. They are not dependent upon the source and destination address input specifications of this module.

Inputs That Must be Constants:

- Port to RCV…integer in range of 1 to 2 specifying with which port on the board this module is to work.

Other Inputs:

- Address of source…integer address of source station from which transmission is to cause a trigger. Leaving this entry blank will cause the module to issue a trigger on any transmission received by the destination address.
- Address of destination…integer address of message destination to which a transmission is to cause a trigger. Leaving this entry blank will cause the module to issue a trigger on any reception from the source address.

Primary Outputs:

- Trigger event output…Name.Trigger…status trigger that becomes true when a message is received on the port and board specified, and having the source and destination addresses specified.
- RCVed source address…Name.Src…integer address of last received message on the specified board and port.
- RCVed destination addr…Name.Dest…integer address of last received message on the specified board and port.

Outputs for Internal Use: None

Limitations: None

Expected Applications:

Use to show evidence of reception.

<u>Software Modules</u>

CHAPTER 6…DISPLAYS, LADDER

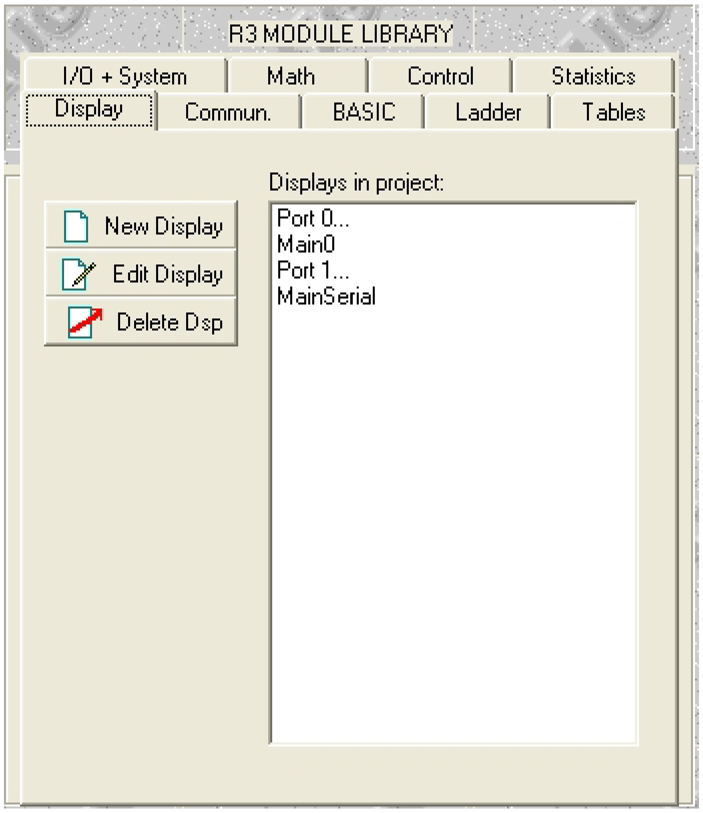
Display Definition

Displays for the RUG3 are defined a screen or page at a time by typing them in the Display/Report Editor page within the R3SETUP program. Displays can be defined for either of the serial ports (port 1 and port 2) and for the local LCD display (port 0). Displays are independent for each port; i.e., displays designated for one port are not accessible on another port and do not effect any display being shown on another port. Along with the display text, you will enter a name for the display that will become a menu entry when run on the RUG3 so an operator can access the display. You will also define to which port the display applies. Finally, where you wish the RUG3 to present variable data such as flow rate, tank level, etc. on the display or report, you will drag in the name of the measurement to a table that holds references to variable data for each line of the display. In the following sections, we will describe the process of defining displays.

Selecting a Display to Edit

To select a display to edit or to define a new display, first select the ‘Display’ tab in the R3 Module Library. You should see the display tab expand to look as shown below. At this point, if you wish to define a new display, you would press the ‘New Display’ button. To edit an existing display, first select one of the named display titles in the ‘Displays in Project’ window and then press the ‘Edit Display’ button. If you wish to delete a display from the project, select the display name from the list and then press the ‘Delete Display’ button. Once you press a button to edit or define a display, the system will present the display definition page illustrated below.

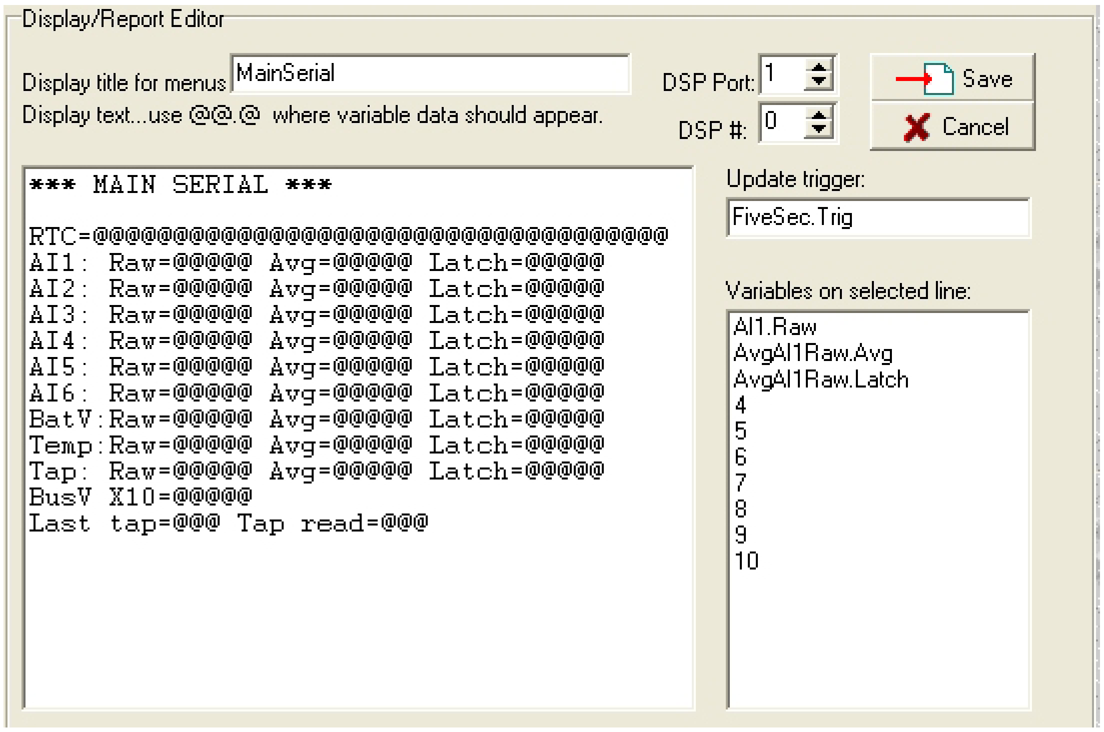
Displays, Ladder



Displays in project:

```
Port 0...
Main0
Port 1...
MainSerial
```

Figure 51 Display Selection Panel



Display title for menus: MainSerial

Display text...use @@.@ where variable data should appear.

DSP Port: 1

DSP #: 0

Update trigger: FiveSec.Trig

```
*** MAIN SERIAL ***

RTC=@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
AI1: Raw=@@@@@ Avg=@@@@@ Latch=@@@@@
AI2: Raw=@@@@@ Avg=@@@@@ Latch=@@@@@
AI3: Raw=@@@@@ Avg=@@@@@ Latch=@@@@@
AI4: Raw=@@@@@ Avg=@@@@@ Latch=@@@@@
AI5: Raw=@@@@@ Avg=@@@@@ Latch=@@@@@
AI6: Raw=@@@@@ Avg=@@@@@ Latch=@@@@@
BatV:Raw=@@@@@ Avg=@@@@@ Latch=@@@@@
Temp:Raw=@@@@@ Avg=@@@@@ Latch=@@@@@
Tap: Raw=@@@@@ Avg=@@@@@ Latch=@@@@@
BusV X10=@@@@@
Last tap=@@@ Tap read=@@@
```

Variables on selected line:

```
AI1.Raw
AvgAI1Raw.Avg
AvgAI1Raw.Latch
4
5
6
7
8
9
10
```

Figure 52 Display/Report Definition Panel

Naming the Display

The display will always be referenced by a name you give it in the 'Display title for menus' field. In the example above, the display name is 'MainSerial'. The name 'MainSerial' will appear in the RUG3's display selection menu for an operator to use to reference this display. It will also appear in the display list when you click the display tab in the R3SETUP module library to enable you to return to the display to edit it. When you enter the screen shown above, the cursor will rest in the display name field. For new displays, you should be sure to enter a unique name for each display before attempting to save the display.

Numbering the Display

To the right of the display name is the display number and port number assigned to this display. The display number sets the order in which the display names will appear in any lists or menus on the designated port. Display zero will be the first shown on any list, display 1 will be next, etc. You can change the display numbers at any time to control the order in which displays are accessed by operators as they scan down through them using the RUG3 [ENTER] key. If you have displays with duplicate numbers or the list of displays is missing a number, the compiler will complain and will not allow the program to load.

Setting Display Port Assignment

The port number specifies which port in the RUG3 the display will use. In the RUG3, there is a separate display list for each port for which any display has been defined. An operator can access displays on any port independently of the displays being shown on any other port. The table below specifies the port numbers assigned to ports in the RUG3. Note that displays will not be sent to any port not in ASCII mode. The mode is set using a ComSetup module. Port 1, the programming port defaults to 9600,N,8,1, ASCII mode.

Table 7 RUG3 Port Assignments

Port 0	LCD Display
Port 1	Programming serial port
Port 2	Modem/RS232 port

Setting Display Trigger

The display trigger enables you to specify the update interval or event trigger to update each display. You must specify a trigger or the display will never update after being initially accessed by an operator. The trigger can be any status from the status database. To specify a status to act as a display update trigger, simply drag the entry from the status database and drop it into the Update Trigger field. In the above example, 'FiveSec.Trig' is the trigger. It causes the display to update once each five seconds. It's best to use triggers here rather than statuses that may be true constantly because as long as a status is true, it would cause the display to be rewritten every scan, wasting time on unnecessary display updates. Another example would be triggering displays that show received telemetry data. Here, it would be good to use the '.Trig' output of a 'TrigOnRcv' module, since then the display would update immediately after the reception of telemetry data and only then. For displays directed to serial ports, its best to use a trigger less frequently than once per second because the display will scroll up before the operator can read it. We find that the 5 second update, as in the example above, works well.

Entering the Display Text

The large window to the left of the trigger window is the area where you type the text you wish to display on the RUG3's LCD, or to transmit to a serial port. Once you position your cursor into that window, you may type whatever you wish to display. You may move your cursor using the mouse and delete or insert text at will. There are no limitations on what you type except that the '@' symbol is used to specify the position and format of active data fields as described below. In the case of sending displays to serial ports for display on monitors, you may want to enter more data than the display window can show. Typically, a monitor will be able to show 80 characters by 25 lines, and a printed page can show 80 characters by 66 lines. This is not a problem. Simply type in longer lines as necessary and use the arrow keys to position left and right along the line. Also, you can enter up to 100 lines in a single RUG3 display.

Specifying Variable Data Fields

Wherever you wish the RUG3 to show dynamic variable data such as tank level, pump starts, flow rate, etc., you must use '@' symbols to indicate the location and format of the area where you wish the data to be presented. For example, if you wish for the RUG3 to show an integer as a three digit field on a line, you would type '@@@' where the value is to appear on the line. If you wish to show a floating point value with 5 places to the left of the decimal and 3 places to the right, you would type '@@@@@.@@@' where that value is to appear on the line. When the compiler encounters '@' symbols as it scans each line, it reserves exactly the space you specified for variable data to appear, and will format the measurement to occupy the space you have specified, left justified. After you have formatted each line, you must specify a variable to be presented in each '@@@@' type field you have entered on the line. You do that in the small window to the right of the large display window that contains your text. It is labeled 'Variables on selected line' and contains 10 rows where you can drag and drop up to 10 variables from the data bases that you wish to be displayed in the '@'-fields on the presently selected line. In the example above, the cursor rests on line 4, which is "AI1: Raw=@@@@@ Avg=@@@@@ Latch=@@@@@". This line has three variable fields, one for AI1's raw count ('@@@@@'), one for AI1's average value ('@@@@@') and one for AI1's latched value ('@@@@@'). Therefore, it needs three variables specified in the 'Variables on selected line' window. As you can see, the three variables that were dragged in for this line are 'AI1.Raw', 'AvgAI1Raw.Avg' and 'AvgAI1Raw.Latch'. The remaining entries 4 through 10 in the variables list simply have numeric place holders. As the compiler scans each display line left to right, it uses variables from the variables window from the top down. You do not have to worry about the choice of floating point, integer, status and string variable types. The compiler will format the referenced variable data to display properly in the '@@@@' format specification. Variables or strings whose lengths are too long to display in the designated space will be presented as '#####' fields, indicating overflow. Fields of '@' characters for which you have not assigned variables will be displayed with '----' characters to differentiate those fields from over range fields.

Special @ Fields

The '@' character is used to specify location and format of variable value presentation, and to specify insertion of special characters and alternate display methods such as trends. The table below defines all the special uses of the '@' fields:

Table 8 List of '@' Field Uses

Field in Display	Meaning	Where Applicable
@@@@.@@@	Show value with that format	LCD, serial port
@F	Insert form feed	Serial port

Saving the Display

After you have finished formatting a display, click the 'Save' button to save it. If you do not wish to save your changes, click the 'Cancel' button to abandon it.

Ladder Logic

The RUG3 contains a complete ladder logic facility whereby simulated contacts and relay coils can be interconnected to produce custom logic. The contacts can be any statuses in the status database; and the coils constitute new statuses that are placed into the status database when defined. Contacts can be regarded as normally open or normally closed. Each coil has a call time delay and separate off time delay. You specify the ladder logic by basically drawing its schematic. To do that, you click on the 'Ladder' tab in the R3 Module Library. The following design page would then appear.

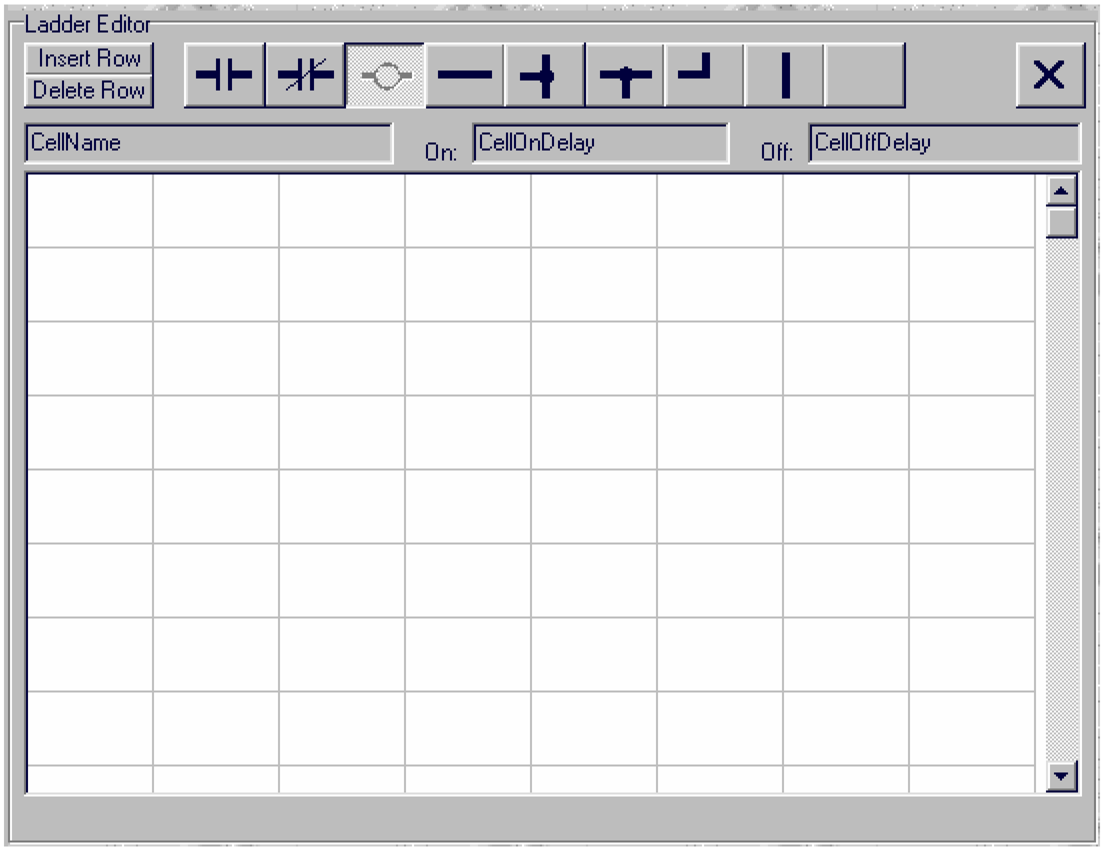
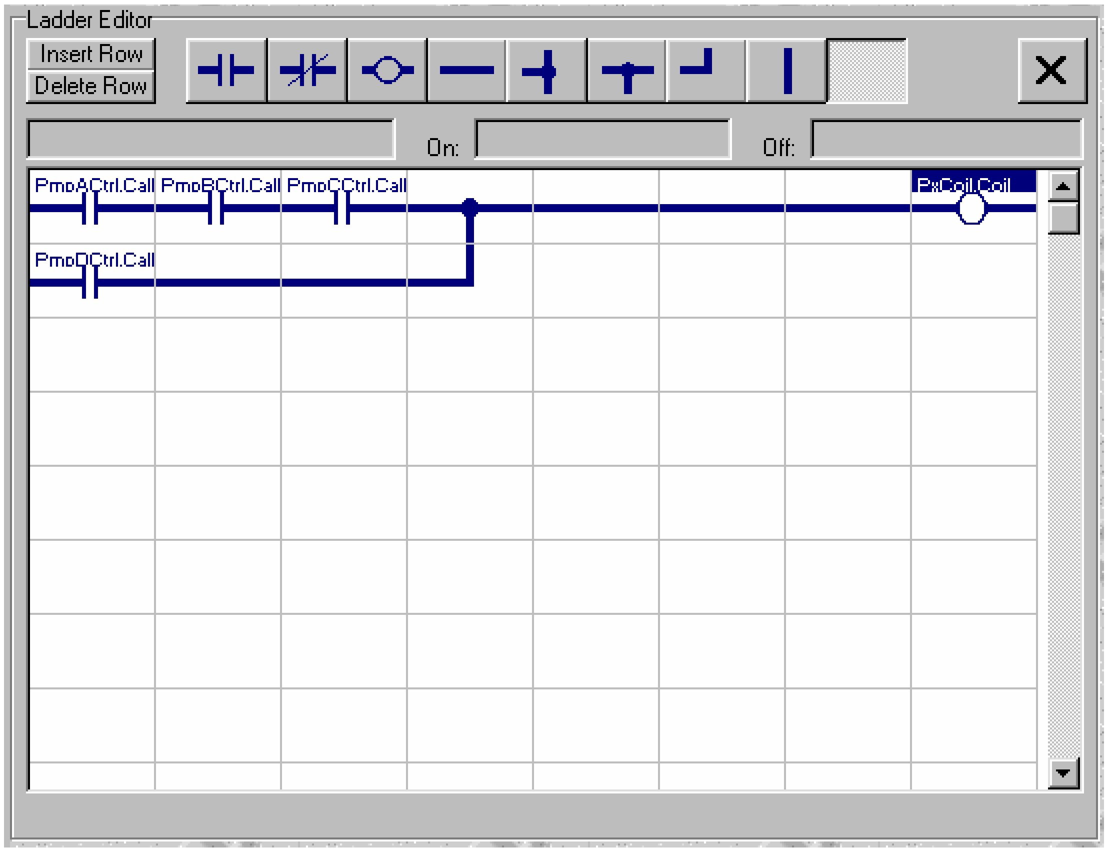


Figure 53 Ladder Editor Initial Panel

Specifying Ladder Logic Schematics

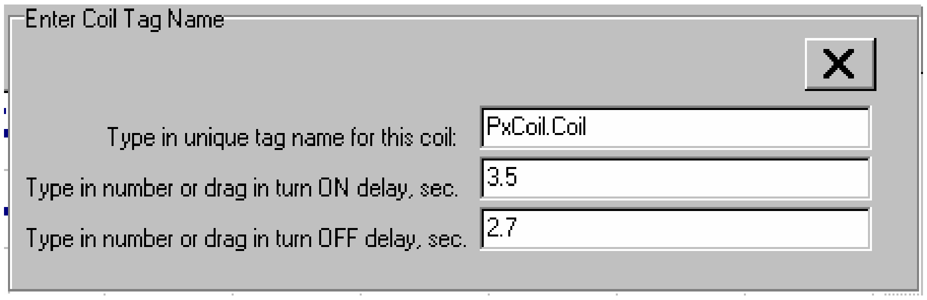
Above, a blank schematic is shown. At the top of the schematic grid are nine buttons with schematic icons on them. Once you click on any of these buttons, it becomes the selected button. Thereafter, wherever you click in the schematic page, the selected schematic item will appear. Each row, or 'rung' of the ladder can have eight entries, the rightmost of which must either be blank or have a coil in it. Contacts on the same row are 'anded' together; contacts connected vertically are 'ored' together. For example, say we wish to turn on a coil whenever PmpActrl.Call and PmpBCtrl.Call and PmpCCtrl.Call are all on, or whenever PmpDCtrl.Call is on. To do this we would need the following ladder logic schematic which we produce by selecting schematic icons and dropping them onto the schematic page. For each normally open or normally closed contact that we include, we must drag a status point from the status database and drop it onto the contact.



Here, our schematic created a coil that we named ‘PxCoil.Coil’, which now appears in the status database. It can be used by any module including any rungs of our ladder diagram. Note that it is not automatically connected to any physical relay. To have this coil control a relay, we would have to connect it as the control input of a digital output relay module.

Specifying Coil Name and Delays

When we drop a coil at the right end of our schematic we are presented with a screen where we can specify the name of the coil (we used PxCoil and the system added the ‘.coil’ appendage), and the ON and OFF delays as shown below.



Enter Coil Tag Name

Type in unique tag name for this coil: PxCoil.Coil

Type in number or drag in turn ON delay, sec. 3.5

Type in number or drag in turn OFF delay, sec. 2.7

Figure 54 Coil Name and Delay Specification Page

Generally, you will leave the delays at zero, but you can have both call and off delays with ranges of 32767 seconds and resolutions of 1 second. After we name the coil and set its delays, if any, the coil is added to the status database. Note that you may use constants, as in the example above, or variables such as setpoints from a database for the time delays.

CHAPTER 7…COMMUNICATIONS

Introduction

The RUG3 provides a complete set of protocols to securely retrieve field data, send control data to remote sites, and interface with virtually all SCADA software packages. RUG3 communications hardware and software are compatible with both radio and phone line channels; serial RS232 channels; leased line, customer-owned lines; and virtually all types of radios including audio types, radios with built in modems, and spread spectrum radios. R3SETUP enables you to define telemetry formats that you need to accomplish fast, efficient communications. All protocols except ASCII include CRC-16 security, so you can be assured that if a message is accepted by the RUG3, the data contained in the message is intact; any messages with errors are rejected entirely. This chapter discusses telemetry design methodology, communications module setup, array setup, and other aspects of communication design.

Overall Communications Design Methodology

This section is intended to assist in selecting the correct modules, communications hardware, and array setups. Basically, the job of designing a communication system is to first establish what data are to be sent between the various stations at what speed. Then decide the channel characteristics and, thereby the necessary RUG3 hardware. Finally, choose the software modules and array setups that will accomplish the required communications over the channel.

Communications

RUG9 protocol Basics

Normally, for unit to unit communications, you will use the RUG9 protocol. It is presented in detail later in this chapter, but some explanation is necessary before we proceed with the following sections. The RUG9 protocol uses a variable length message that includes data within both the initiating message (the poll) and the reply message. The messages contain the addresses of both the initiating station and the destination. Each station in a system must have a unique address in the range of 1 through 65535. Address 0 is reserved for broadcast use. Typically, an initiating station will send a poll containing data intended for the destination station. If the destination station receives the poll intact, it will always reply to confirm that it received the message. If the initiating station fails to obtain a response to its poll, it will assume the poll did not get through, even though it can happen that the poll was received intact but the reply was corrupted. Usually, if enough polls in succession fail to result in successful reply receptions, the initiating station will declare a communications failure. Only the addressed destination station is allowed to reply to a poll. However, other stations can listen in and extract data from both the poll and the reply. Data included in any transmitted message are obtained from the station's transmit array and inserted in the outgoing message, whether it is a poll or a reply. Data received by a station will appear in its receive array if the received message contained no errors. If the received message had an error, no data will be accepted from it.

System Types: Polled, Report by Exception (Quiescent) and Peer to Peer

There are three main variants of system communications topology; and the RUG3 supports them all as well as various combinations. Which type you design into your system depends on the number of RTU's in your system, the need for them to communicate among themselves, and the system reporting speed requirements. The three types are:

- Polled system…system where the master site polls each RTU in turn, and RTU's do not need to communicate directly with each other. Each RTU must wait its turn to report data.
- Report by exception (quiescent) system…system wherein RTU's decide when to report to the master based on changes in analog values, alarms, etc. Each RTU reports as soon as a change is detected.
- Peer to peer…report by exception system wherein RTU's communicate directly with each other as well as with a master. Can function without a master site.

Of the above systems, the polled system is the simplest. In it, the master polls RTU's in a round robin fashion continuously. Polling is easy to set up, and the RTU's simply have to reply to the polls. The polled system is also the slowest to report changes. This is because that since the master polls all RTU's in turn, an alarm that occurs just after an RTU has been polled must wait for the rest of the RTU's to be polled before the alarm is reported. Our rule of thumb is that systems of more than 10 to 15 RTU's should consider a quiescent system.

With a quiescent system, RTU's report only when they detect a change. In this case, the communication channel is nominally quiet except for the occasional report. Since each RTU can report as soon as it detects a change, alarms are reported immediately. The quiescent system usually includes occasional polling by the master site to detect communications failures. Of course, two RTU's can decide to report simultaneously. In that case neither message gets through to the master so the RTU's must repeat their polls. For this reason, the RUG3 quiescent polling controller includes a random time delay on repeat messages to avoid synchronous blocking.

Finally, the peer to peer system is a variant of the quiescent system that includes direct RTU to RTU communications. In this case, the master site is not necessary to normal system operation; but is required to install changes to setpoints from a SCADA system, as well as to provide visibility of system operation, and detection of communications failure using infrequent polling.

The figure below illustrates the three types discussed.

Communications

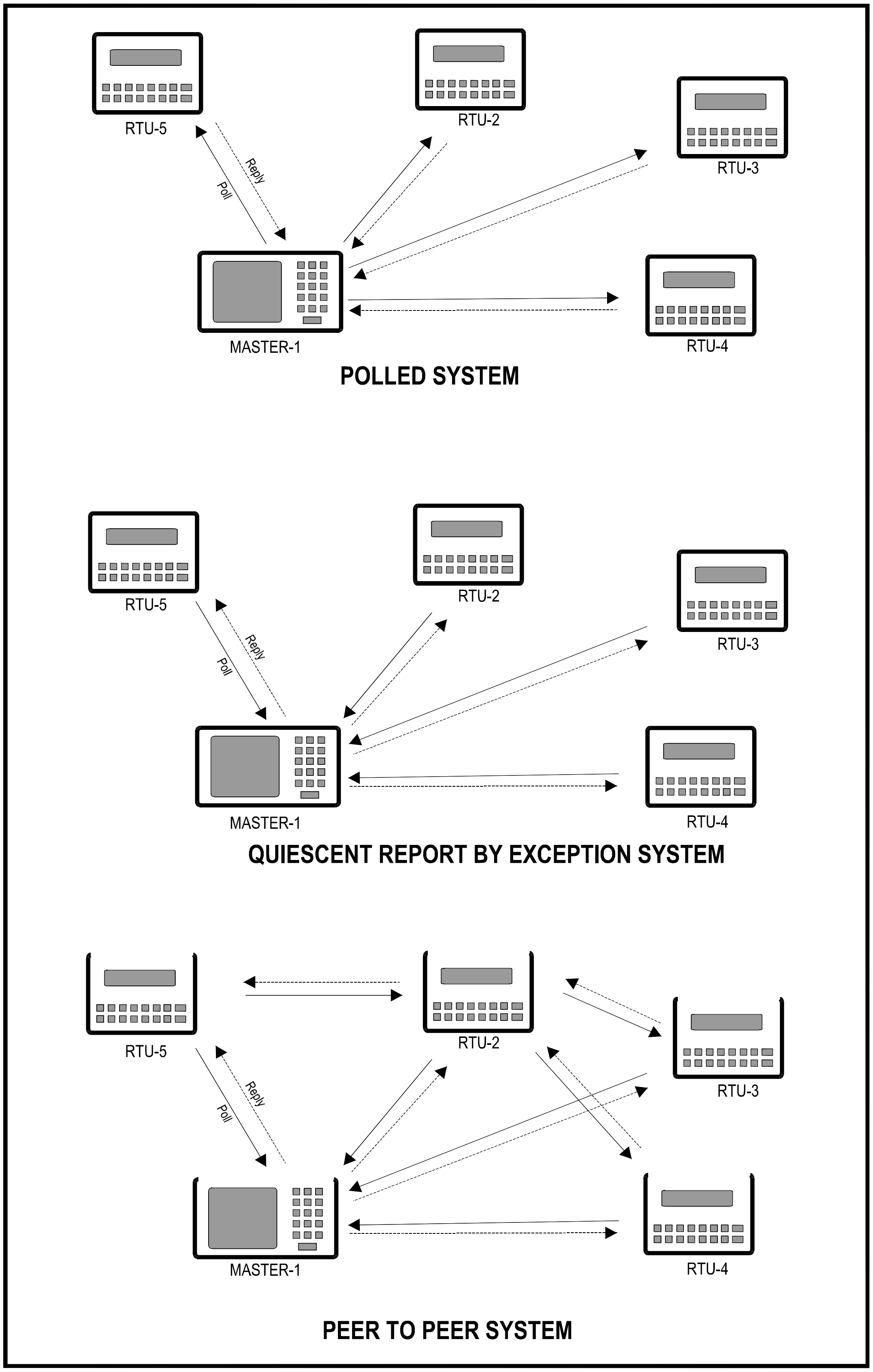


Figure 55 Illustration of Three Main Communication System Types

Communications

Establishing Data Transfer Requirements and System Type

To establish the system's data transfer requirements, you must first list all signals that need to be sent from all sites to any other sites. This is called the **Tag List**. Usually, a system will require that field data and statuses be sent to a master site, and master setpoints and control bits be sent to the remote sites. For each site you must list data to be transmitted from the site to the master; and then list data to be received at the remote site from the master. Assume that status bits are packed into 16 bit words (i.e., 16 or fewer statuses per word), and that each analog value requires one 16 bit word. Then add up how many words must be sent from the remote site to the master, and how many must be sent from the master to each remote. From that compilation, we can estimate the communications times and assign data to your transmit and receive arrays. For example, suppose we are designing a simple system consisting of a single tank site, a single pumping site and a master site. The tank level must be sent to the pumping station so it can decide when to turn on and turn off its pump. Assume the master controls all polling, and that the sites communicate only with the master and not with each other directly. Therefore, a polled system will apply. The following table illustrates the data that must be passed between each site and the master:

Table 9 Sample Telemetry Tag List

Word	Master to Tank Site	Tank Site to Master	Master to Pump Site	Pump Site to Master
1	• Spare statuses	• High alarm status • Low alarm status • Power fail status	• Spare statuses	• Pump call status • Pump run status • Pump fail status • Power fail status
2	Tank low alarm SP	Tank level	Pump HOA command	Pump total starts
3	Tank high alarm SP	Battery voltage	Pump call SP	Pump total run time
4			Pump off SP	Battery voltage
5			Tank level	Flow
6				Total flow
Total Wds	3	3	5	6

The above table constitutes a tag list for this simple system. We are interested in estimating the time the master would take to transmit and receive from both stations, i.e., the polling cycle time. That time can be estimated from the following equation:

Time per each leg of message = acquisition time + 20* (4 + number of data words)/(baud rate)

If we assume a radio system with 300 baud data rate, and a transmit delay for acquisition of 0.75 sec., then the cycle time of the above system would be:

- Master to tank site: 1.22 sec.
- Tank site to master: 1.22 sec.
- Master to pump site: 1.35 sec.
- Pump site to master: 1.42 sec.
- <u>Total cycle time: 5.21 sec.</u>

As long as the system is small, say less than 10 remotes, the polled approach will probably work. However, for larger systems, the polled system becomes slow. To illustrate, assume the system consists of 50 remotes sending and receiving 25 data words each message. At 300 baud, the polling time, using the above equation becomes:

Time per each leg of message = .75 + 20 * (4 + 25) / 300 = 2.68 seconds

For 50 remotes with 2 legs each, the cycle time becomes 2.68 * 100 = 268 seconds.

To provide faster reporting, consider using the report by exception, or quiescent system. With that approach, alarms and other changes will be reported in less than a second.

Store and Forward

If you are implementing a radio system in mountainous terrain, you may have some stations not in line of site with one or more stations with which they need to talk. If the station that is out of sight is accessible from another RTU, then that intermediary RTU can be used to relay messages using a technique called store and forward. Basically, store and forward simply means that a station will store a message, test it, and then relay it forward if it sees that it is not itself the destination, and its address is next in the message's imbedded forwarding path. As an example, the figure below indicates that RTU-3 is not visible to the master site. This figure is the same as the figure above, except that links between the master and RTU-3 have been omitted because they are blocked by terrain. In that case, polls from the master can be routed through RTU-2. In the RUG3 communications implementation, the initiating station specifies the path the message is to take to its destination. That path is specified in the message so the reply can return along the same path. You specify the path in the **Poll** module using up to three forwarding entries. Therefore, in addition to the source and destination addresses, you can specify up to three intermediary stations the message can use to relay out to a distant site.

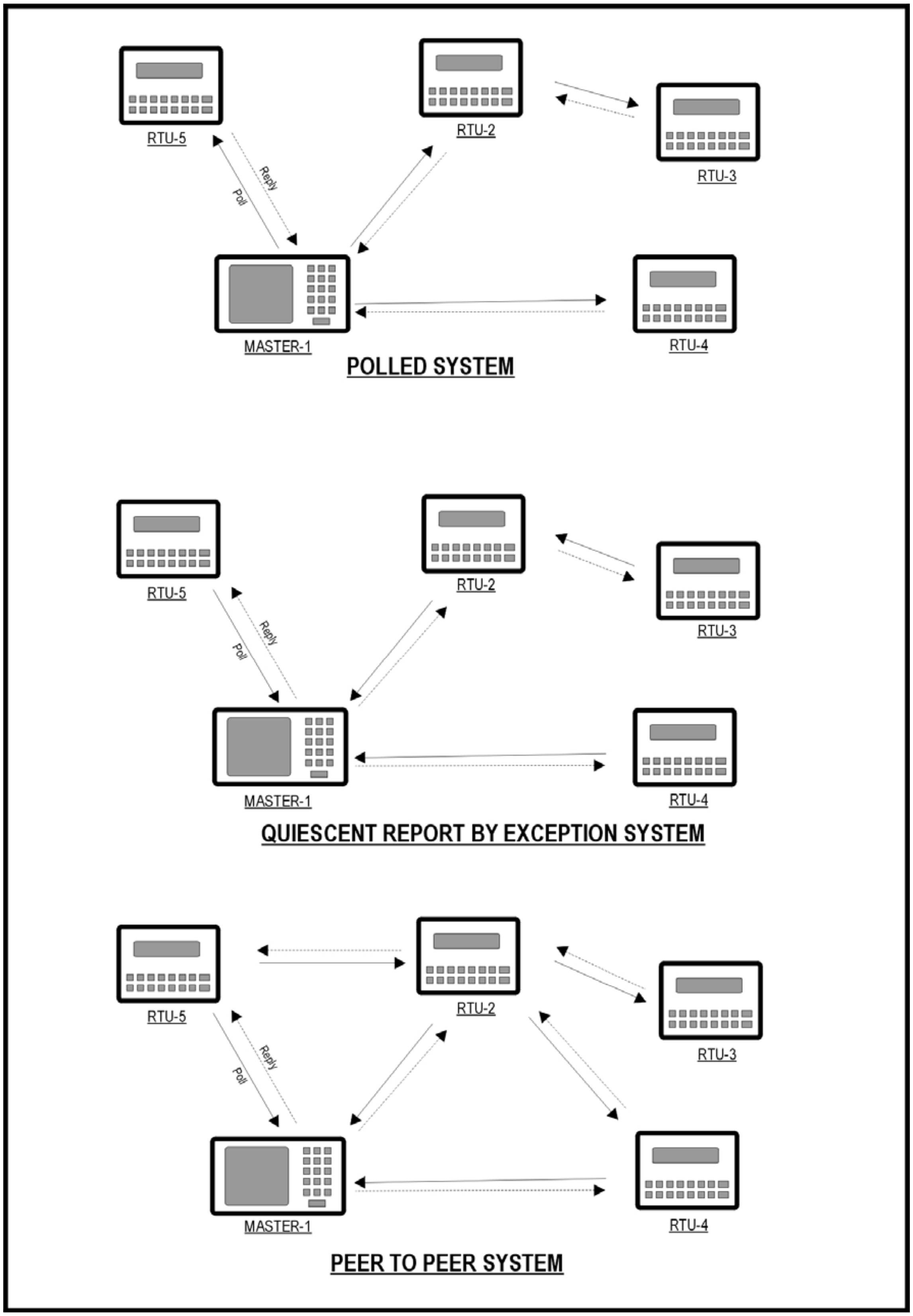


Figure 56 System Types Using Store and Forwarding to RTU-3

Protocols Supported

The RUG3 operating system provides complete communications services for its two serial ports. All communications are secured with appended CRC16 codes except for ASCII, which uses no security. The RUG3 supports the following protocols. A port's protocol is specified in the unit's ComSetup module for that port.

Table 10 Protocols Supported

PROTOCOL	WHEN TO USE
ASCII	Use for ASCII com when you want RUG3 menu services supported
RUG9	Use to communicate with other RUG3's, RUG5's and RUG9's
RUG6	Use to communicate with older RUG6's, RUG7's and RUG8's
MODBUS MASTER	Use to poll other slave units
MODBUS RTU SLAVE	Use as slave to Modbus master when Rcv and Tx arrays are to be kept separate
MODBUS RTU SLAVE 2	Use as slave to Modbus master when Rcv array to be used as read/write holding registers as well as I/O receive registers
ASCII 2	Use for ASCII com when you don't want RUG3 menu services interfering
ALERT	Use to be compatible with the ALERT weather/flood warning system
True Wireless	Use in proprietary industrial applications
Modbus TCP slave	Use to implement Modbus slave for Ethernet communications
Modbus TCP slave 2	Use as slave to Modbus master for Ethernet communications when Rcv array to be used as read/write holding registers as well as I/O receive registers
Modbus TCP master	Use to poll Modbus slave devices over Ethernet

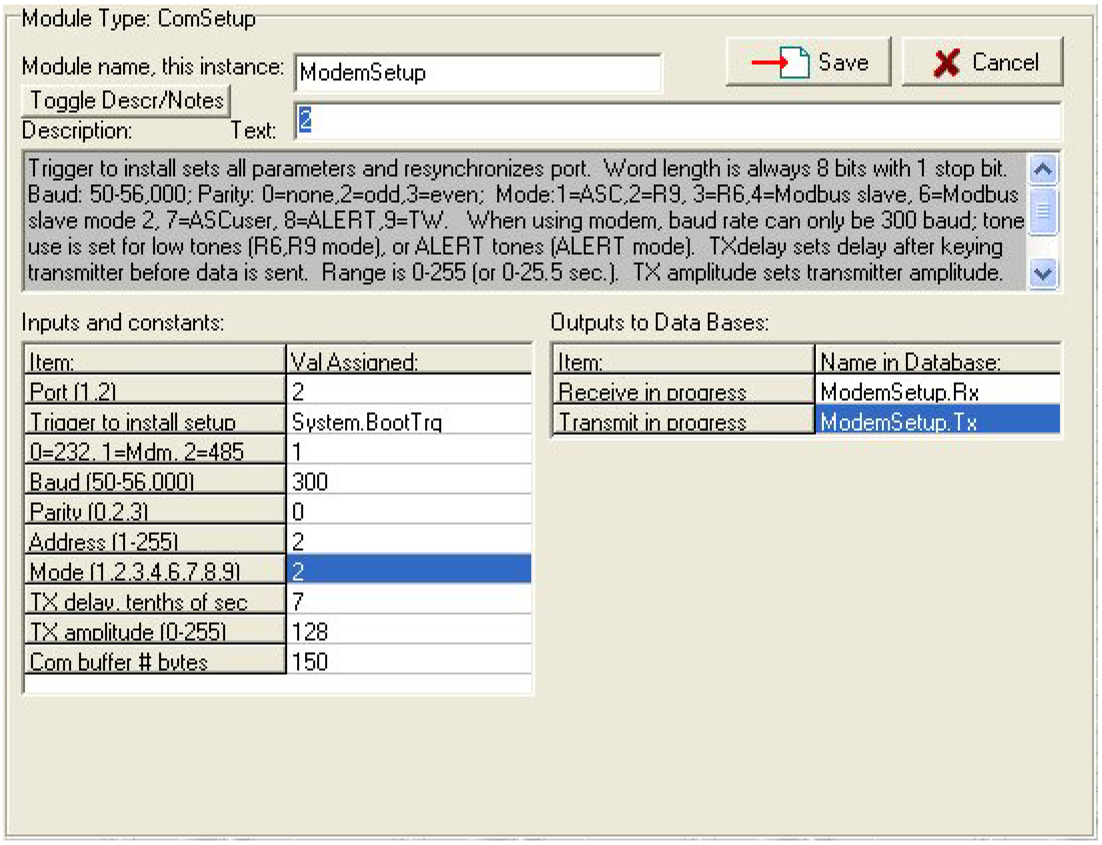
All messages except for ASCII and ALERT employ a poll and reply dialogue whereby one station transmits a poll and the addressed station replies. The RUG3 can be either the polling station or the replying station, or both. It can therefore be used in polled systems or quiescent report by exception systems. The RUG3 also supports store and forward operation when using RUG9 or RUG6 protocols. In those cases, the forwarding path is set by the polling station with all succeeding communications based upon the forwarding path imbedded in the message. The RUG9 protocol supports up to 3 intermediary stations.

Setting Up Ports and Communications Arrays

Port Setup

Before any port will function normally, it must be initialized with baud rate, buffer size, pointers, etc. You do this with a **ComSetup** module as illustrated below. The setup shown is typical for radio applications. It would be installed right after boot up since the installation trigger is taken from a "System.bootTrg" module. In the case of units doing polling using the **SequenPoll** module, the **SequenPoll** module installs the **ComSetup** module just before each poll. Most commonly, you would use constants for inputs as in the example. However, you can use the outputs of other modules, such as setpoints, as inputs to this module except for the port designator, which must be a constant. For example, you could use a setpoint module as the baud rate input so the operator could adjust the channel baud rate. You may have as many **ComSetup** modules as you wish in your project. Whichever one executes last for a given port governs the port's setup. Note that since each port in a unit can have a different setup, each port can have a different address; or the ports can have the same address…they are treated independently by the RUG3 operating system.

Communications


Module Type: ComSetup

Module name, this instance: ModemSetup

Description: Text: 2

Trigger to install sets all parameters and resynchronizes port. Word length is always 8 bits with 1 stop bit. Baud: 50-56,000; Parity: 0=none,2=odd,3=even; Mode:1=ASC,2=R9, 3=R6,4=Modbus slave, 6=Modbus slave mode 2, 7=ASCuser, 8=ALERT,9=TW. When using modem, baud rate can only be 300 baud; tone use is set for low tones (R6,R9 mode), or ALERT tones (ALERT mode). TXdelay sets delay after keying transmitter before data is sent. Range is 0-255 (or 0-25.5 sec.). TX amplitude sets transmitter amplitude.

Inputs and constants:

Item:	Val Assigned:
Port (1,2)	2
Trigger to install setup	System.BootTrg
0=232, 1=Mdm, 2=485	1
Baud (50-56,000)	300
Parity (0,2,3)	0
Address (1-255)	2
Mode (1,2,3,4,6,7,8,9)	2
TX delay, tenths of sec	7
TX amplitude (0-255)	128
Com buffer # bytes	150

Outputs to Data Bases:

Item:	Name in Database:
Receive in progress	ModemSetup.Rx
Transmit in progress	ModemSetup.Tx

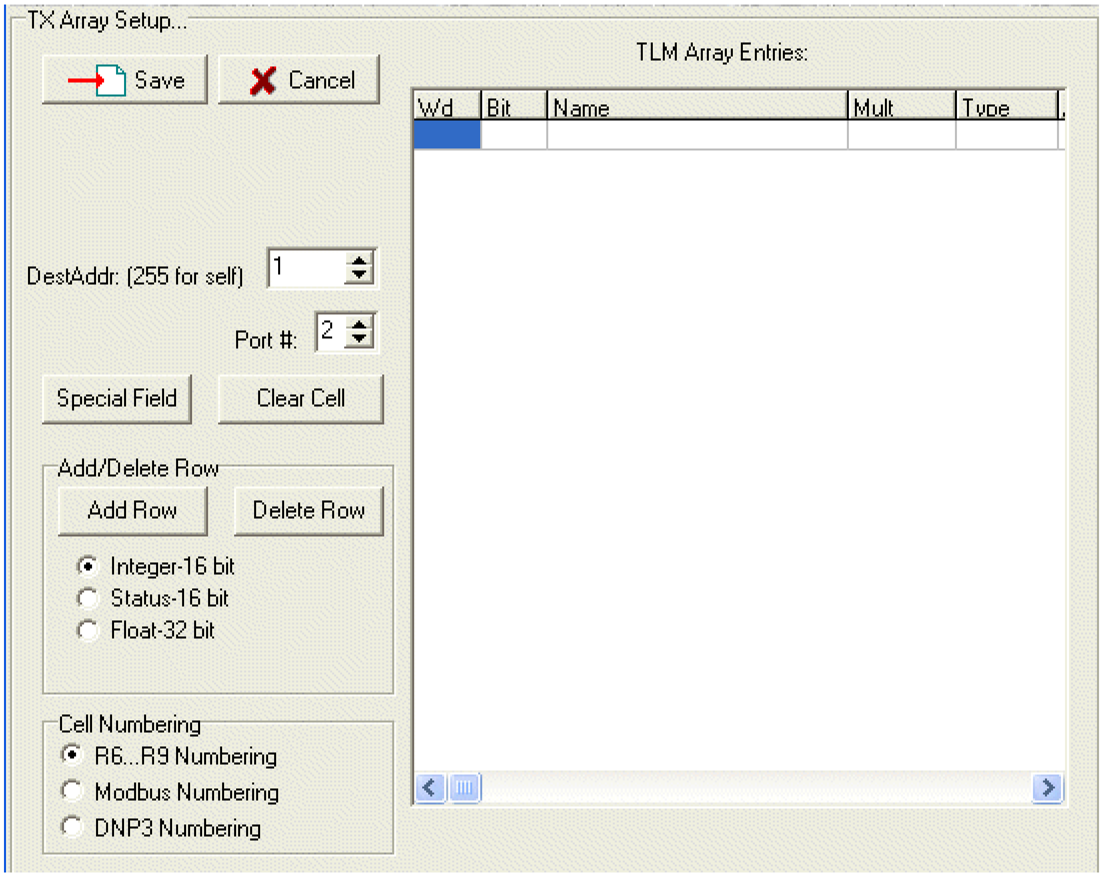
Figure 57 Typical Modem Setup for Radio Applications

Array Setup

While the **ComSetup** module described above establishes the port communications parameters, you still need to establish what data is to be transmitted and how received data are to be interpreted. To do this, you must select the communications tab in the R3 module library, and then select either “RX Array Setup” or “TX Array Setup”. Note that you must set up one transmit array and one receive array for each station with which this RTU is to communicate. If you wish for this RTU to capture data as it is being transferred from one RTU to another, and neither has the same address as this one, then you only need to set up the receive array in this RTU, using the source and destination addresses of the two RTUs whose data you wish to capture.

Transmit Array Setup

After you click the “New TX Array” button from the communications module tab, you will be presented with the following screen, which will enable you to establish the transmit format for one destination address on one port. The first thing you should do after this screen is presented is set the port number and destination address that you wish to define. You do that by clicking the up or down arrows in the spin edit boxes for port and destination address. If you have previously defined a format for a selected combination of port and destination address, then it will be presented in the large window in the center of the screen for you to modify. Otherwise, a blank format will be presented. In the example screen below, the format is being defined for transmitting to station address 1 on port 2.



DestAddr: (255 for self) 1

Port #: 2

Add/Delete Row: (● Integer-16 bit) Status-16 bit, Float-32 bit

Cell Numbering: (● R6...R9 Numbering) Modbus Numbering, DNP3 Numbering

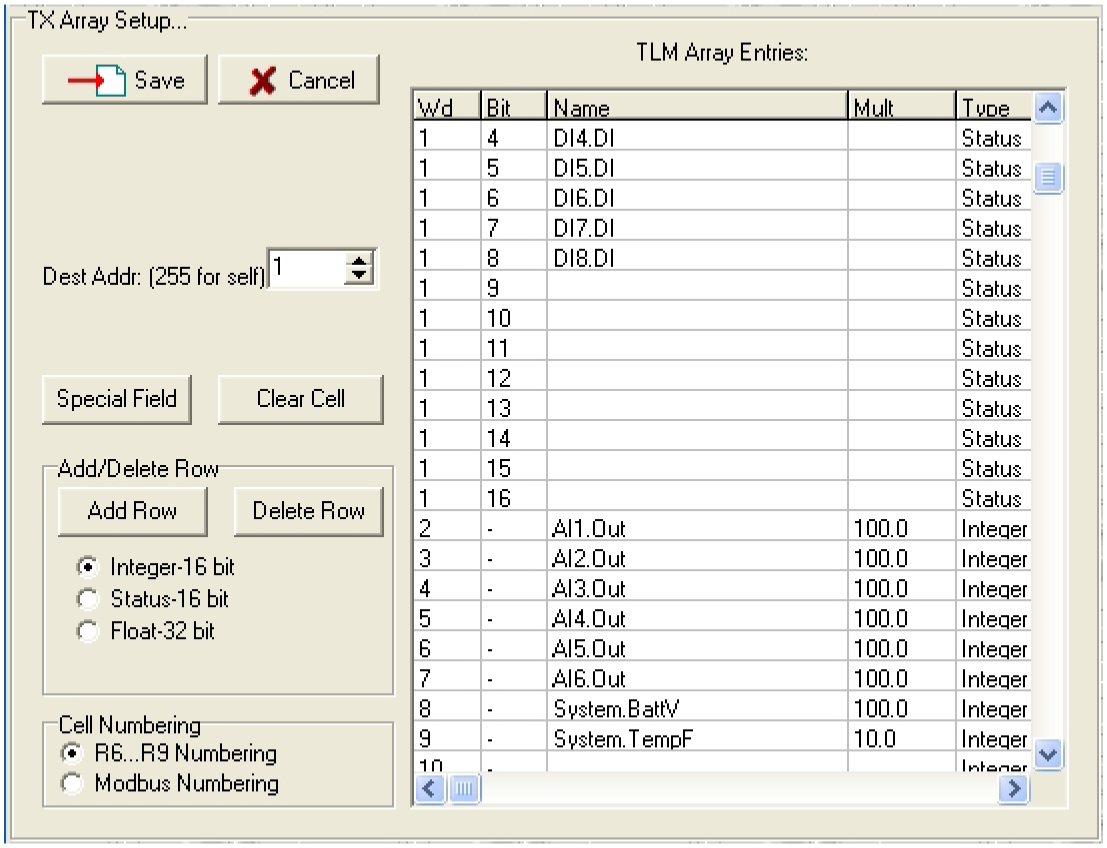
Figure 58 Initial TX Array Setup Screen

Adding and Deleting Rows

You will need a row for each signal you wish to place in the telemetry format. To add a row to the format in the large window, select the type of entry you wish to add (integer-16 bit, status-16 bit or float-32 bit) by clicking one of the radio buttons in the “Add/Delete Row” box in the screen’s lower left hand corner. Then, each time you click the “Add Row” button, a new row of the type you selected will be added to the format, just above the selected row in the large window. One row will be added for each click except for the case of adding a status word, in which case 16 rows will be added, one for each status bit in a 16 bit word. For each row you add to the format, the system will show you the word number, bit number (if status), and type (integer, status…). It will also provide space for you to specify the name of measurement to be sent and its multiplier, if any. To delete a row, simply select the row in the large format window and click on the “Delete Row” button.

Specifying Measurement to Send

You design the transmit telemetry format by dragging the name of the measurement from one of the data bases, and dropping it into the "Name" cell of the word you wish it to occupy in the formatted message. You can drag and drop from any of the databases except the string or TX databases. As shown in the screen presented below, the variable "AI6.Out" was dragged from the floating point database and will be sent in the seventh word of the format as a 16 bit integer after being multiplied by 100. The multiplier is used to enable floating point values to be passed as integers while preserving places to the right of the decimal. Notice that in this example, we have dragged a floating point variable from the floating point data base and dropped it into an integer cell. When it is time to transmit this format, the RUG3 operating system will fetch the floating point tank level measurement from the floating point data base, multiply it by the specified multiplier (100.0 in this case), convert to integer, range limit to -32767 to +32768 and save in the output buffer for transmission. Other variables are dropped into other words and specified to be multiplied by other factors. If you do not need to multiply a value to preserve fractional parts, you can leave the multiplier blank or set it to 1.0.



TLM Array Entries:

Dest Addr: (255 for self) 1

[✓ Integer-16 bit] Status-16 bit Float-32 bit

[✓ R6...R9 Numbering] Modbus Numbering

Wd	Bit	Name	Mult	Type
1	4	DI4.DI		Status
1	5	DI5.DI		Status
1	6	DI6.DI		Status
1	7	DI7.DI		Status
1	8	DI8.DI		Status
1	9			Status
1	10			Status
1	11			Status
1	12			Status
1	13			Status
1	14			Status
1	15			Status
1	16			Status
2	-	AI1.Out	100.0	Integer
3	-	AI2.Out	100.0	Integer
4	-	AI3.Out	100.0	Integer
5	-	AI4.Out	100.0	Integer
6	-	AI5.Out	100.0	Integer
7	-	AI6.Out	100.0	Integer
8	-	System.BattV	100.0	Integer
9	-	System.TempF	10.0	Integer
10	-			Integer

Figure 59 Example TX Array with Variables Installed

SPECIFYING A MULTIPLIER

The multiplier is used to change the range of a floating point variable so that it can be sent as a 16 bit integer. For example, a tank level of 0 to 20.0 feet, if sent without a multiplier, would only give one foot resolution at reception by the receiving station. To preserve better resolution, it is desirable to multiply the measurement by a factor of 10 or 100 before transmission, then divide it by that same factor on reception. For example, if our tank level is 14.5678 feet and we multiply by 100 on transmission and by 0.01 on reception, then our received level would be 14.56 feet, preserving 1/100 foot resolution. You must be careful that the measurement after multiplication does not go outside the range of -32767 to +32768. Otherwise your measurement will be range limited. The alternative is to send the measurement as a floating point number, which the RUG3 supports, but floating point requires 4 bytes per measurement in the transmitted format as opposed to 2 bytes when sent as an integer.

To specify a multiplier, click on the multiplier cell on the row you are specifying. You will be presented with a panel of multipliers in the range of 0.0001 through 10000.0, as shown below. When you click on one, it will become the multiplier for the measurement. Any measurement with no multiplier will assume a multiplier of 1.0.

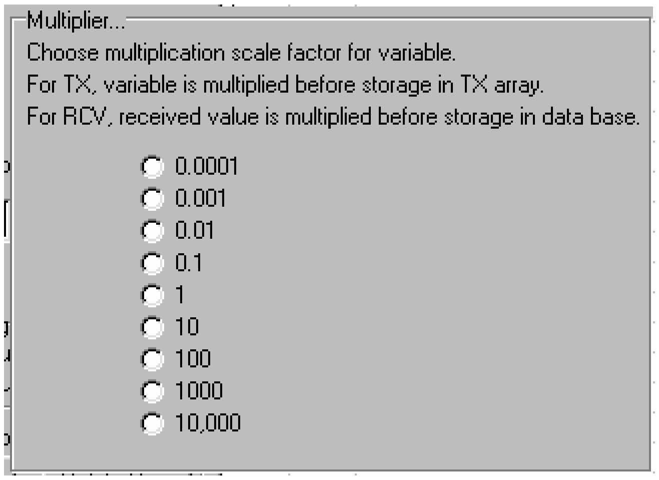

Multiplier...
Choose multiplication scale factor for variable.
For TX, variable is multiplied before storage in TX array.
For RCV, received value is multiplied before storage in data base.

Figure 60 Panel to Select Telemetry Multiplier

Saving Format

After you have finished editing the format, simply click the “SAVE” button to save it to the project. If you wish to abandon your changes, click “CLOSE”.

Receive Format Setup

After you click the “New RX Array” button from the communications module tab, you will be presented with the following screen, which will enable you to establish the receive format for one source and destination address on one port. This format is almost identical to the transmit array setup panel above. The first thing you should do after this screen is presented is to set the source address and destination address that you wish to define. You do that by clicking the up or down arrows in the spin edit boxes for source and destination address. If you have previously defined a format for a selected combination of addresses, then it will be presented in the large window in the center of the screen for you to modify. Otherwise, a blank format will be presented. In the example screen below, the format is being defined for receiving from station address 1. Note that the destination address is specified as 255. This means that the destination address is set to the address of this port, no matter what it may be. So, if a ComSetup module sets this port’s address to 15, then this port will receive all messages from address 1 to address 15, and ignore all others.

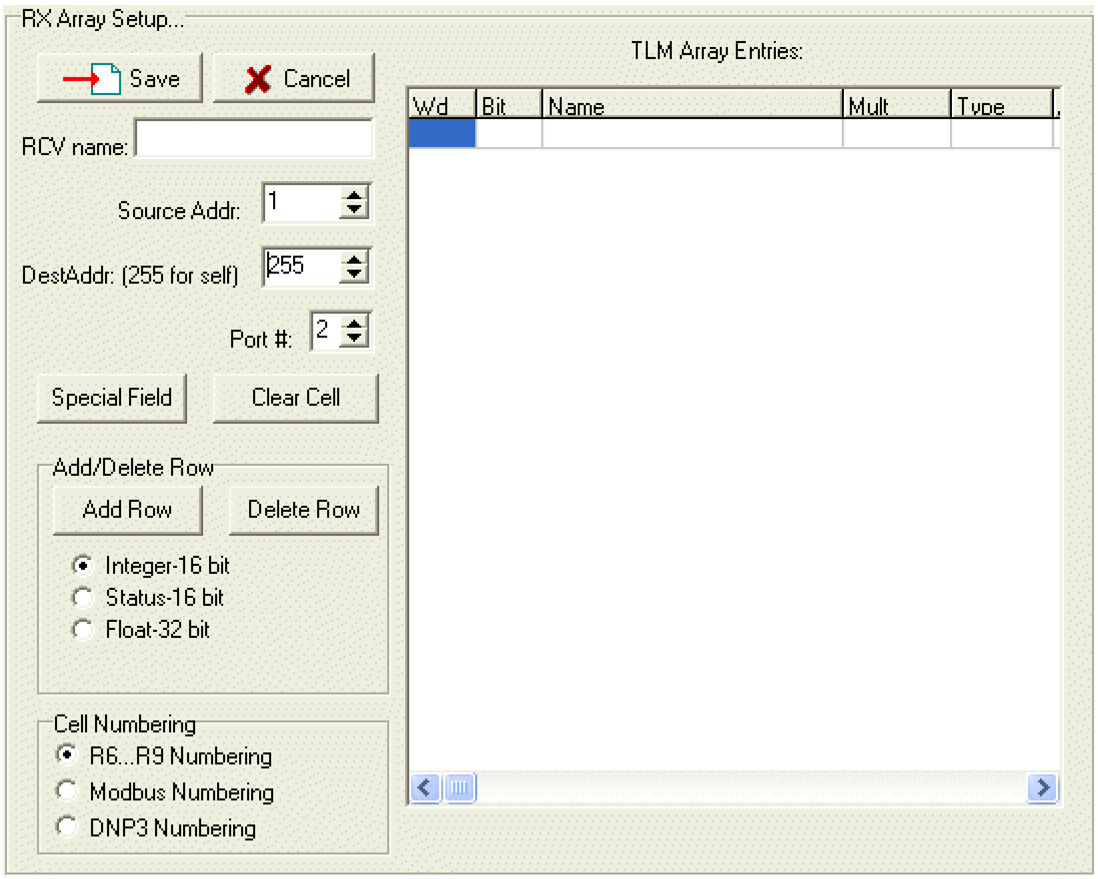


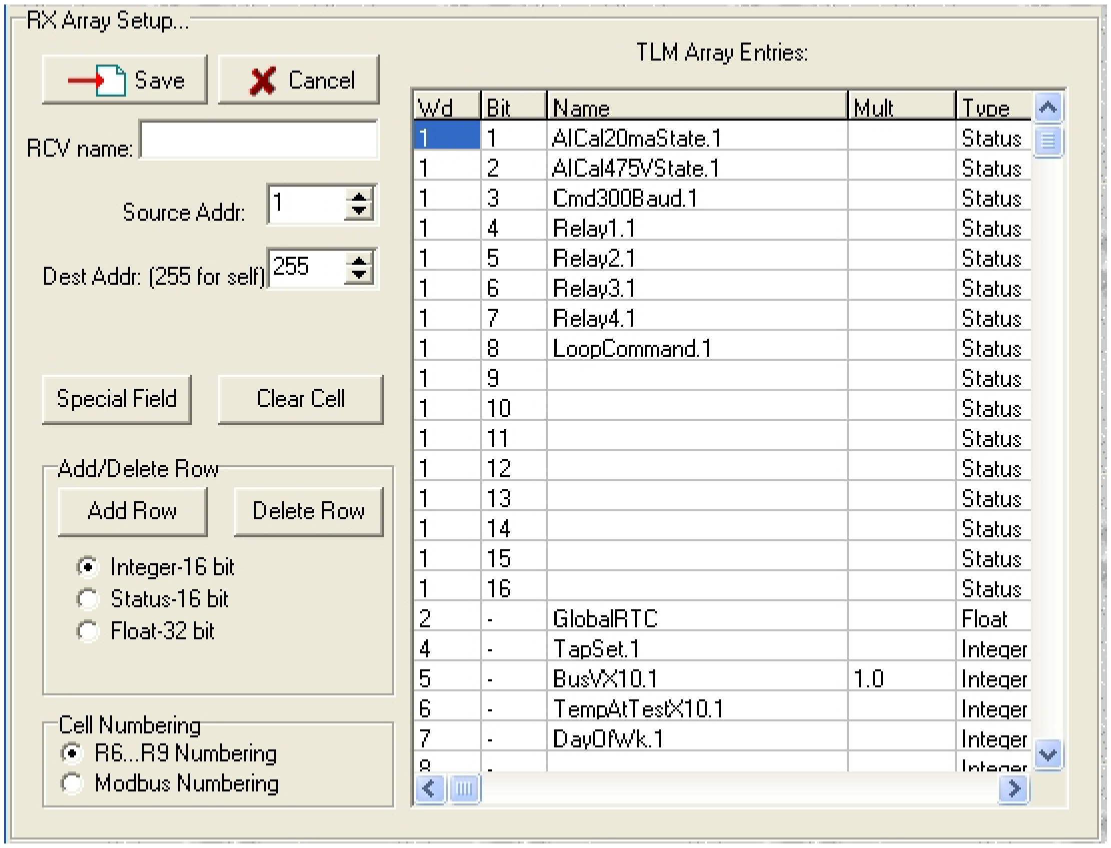
Figure 61 Initial Receive Array Setup Panel

Adding and Deleting Rows

You will need a row for each signal you wish to place in the telemetry format. To add a row to the format in the large window, select the type of entry you wish to add (integer-16 bit, status-16 bit or float-32 bit) by clicking one of the radio buttons in the “Add/Delete Row” box in the screen’s lower left hand corner. Then, each time you click the “Add Row” button, a new row of the type you selected will be added to the format, just above the selected row in the large window. One row will be added for each click except for the case of adding a status word, in which 16 rows will be added, one for each status bit in a 16 bit word. For each row you add to the format, the system will show you the word number, bit number (if status), and type (integer, status…). It will also provide space for you to specify the name of measurement to be sent and its multiplier, if any. To delete a row, simply select the row in the large format window and click on the “Delete Row” button.

Naming Received Data Fields

The large window in the center of the screen above contains the received data format. By selecting integer vs status vs floating point entries as you expand the format, you have already specified the type and data base location of each measurement or status in the receive format. All that is left is to place a name in each row and specify a multiplier, if any. You must type a name into the edit box for each name cell in the format, since receive array entries constitute inputs to the project, and must be named uniquely. After you click on a name cell, any name in it will be transferred to the "RCV Name" edit box in the upper left of the example screen illustrated above. At that location you can edit the name or replace it. When you click on another name cell, the edit box entry you just entered will be transferred into the format's name field on the cell you had formerly selected. In addition, the compiler will append the source station's address to the name for convenience and to help give each name a unique field. In the example screen below, the variable "TapSet" was typed into the name edit box for the line 4 of the format. It is saved for future reference as "TapSet.1" so that you can easily see that the variable originated at site address 1. Variables in the RX database can be used as the inputs to any other modules.



RCV name:

Source Addr: 1

Dest Addr: (255 for self) 255

(● Integer-16 bit) ○ Status-16 bit ○ Float-32 bit

(● R6...R9 Numbering) ○ Modbus Numbering

Wd	Bit	Name	Mult	Type
1	1	AICal20maState.1		Status
1	2	AICal475VState.1		Status
1	3	Cmd300Baud.1		Status
1	4	Relay1.1		Status
1	5	Relay2.1		Status
1	6	Relay3.1		Status
1	7	Relay4.1		Status
1	8	LoopCommand.1		Status
1	9			Status
1	10			Status
1	11			Status
1	12			Status
1	13			Status
1	14			Status
1	15			Status
1	16			Status
2	-	GlobalRTC		Float
4	-	TapSet.1		Integer
5	-	BusVX10.1	1.0	Integer
6	-	TempAtTestX10.1		Integer
7	-	DayOfWk.1		Integer
8	-			Integer

Figure 62 Receive Array Signal Naming

Saving Receive Format

After you have finished editing the format, simply click the "Save" button to save it to the project. If you wish to abandon your changes, click "Cancel". Once you save the receive format, all the receive names you entered will be entered into the RX database with an appendix equal to the station address from where the data is being received.

Special Fields…GlobalRTC

To support special functions of realtime clock synchronization, the communication system provides special field designators to be installed in transmit and receive arrays as defined below:

- GlobalRTC…designates that a 4 byte word contain a packed version of the realtime clock/calendar, consisting of seconds, minutes, hours, day of month, month and year. Does NOT contain day of week (mon, tue..). When transmitting, the system will read the realtime clock/calendar, pack it into 32 bits and insert it into the format. When receiving, the system will unpack the 32 bit word and jam the values into the local realtime clock/calendar.

To use these, you must first install a 32 bit floating point cell in the telemetry format for each. Then click on the telemetry word's name field to indicate where you want the special field to be installed. Finally, click the 'Special Field' button and select the type of special field to install from the radio list. You can place as many of these special fields as you wish in your telemetry formats. In the figure above, a GlobalRTC field is installed in line 2, designating that this unit is to receive a realtime clock update in that field from another site. Note that you must have identical formats in both transmit arrays and corresponding receive arrays for this to work properly.

How to Synchronize Realtime Clock/Calendars

Using the GlobalRTC telemetry entry you can easily keep RTU realtime clocks synchronized with the master clock. Simply place the GlobalRTC entry in a 4 byte floating point entry in the transmit array at the master site; and place one also in the RTU's receive array at the same telemetry word. Then, when the master transmits data to the remote, the remote will receive and automatically install a copy of the master's realtime clock/calendar entries. Note that the remote's clock will be as much as a second behind the master's clock due to the time difference between the master's initiation of transmission and the remote's acceptance of the message.

Communications Formats

RUG9 Formats

The RUG9 formats support operating system loading, flash erasing, configuration file loading; status, integer and floating data reporting; operating status reporting; store and forward, and more. Each message is in binary format with a variable length header, a data field, and CRC-16 security. The header field is presented below.

Communications

Table 11 RUG3 MESSAGE HEADER

Byte #	0	1	2			3	4,5,6	7
Function	Sync	# Bytes in Message Including CRC	Message Type:			Source Station Addr	Variable Length Forward Path 0-3Bytes	Dest. Station Addr
			Bits 7-4	Bits 3,2	Bits 1,0			
Subfunct.	$C8=init $C9=reply		0=TLM data 1=reserved 2=TLM data w/ext addr 3=special commands 4=log data w/ ext addr 5=reserved 6=load RTC 7=OS load 8=get status 9=pgm load	# addr in fwd path	Place in fwd path			

In the above table, if extended addresses are used, addresses are two bytes in MS, LS order. In the following formats, all messages include the above header, so assume the block labeled “HEADER” means to include the format above.

TLM Data Transfer Format

The following format is used to transfer data from a source station to a destination station and requests that the destination station reply with data. Header length assumes no store and forwarding.

Table 12 RUG3 DATA TRANSFER POLL FORMAT

Byte #	0-4	5,6	7,8	9,10	11-N	N+1,N+2
Function	HEADER	First TX word MS,LS in this message	First RX word to be in reply, MS,LS	# RX words requested, 0-255	TX data sent to destination	CRC-16

Table 13 RUG3 DATA TRANSFER REPLY FORMAT

Byte #	0-4	5,6	7-N	N+1,N+2
Function	HEADER	First RX word in reply, MS,LS	RX data sent from destination to source, MS,LS	CRC-16

Special Command Format

The special command is used to control flash access, arm flash erasure, etc.

Table 14 RUG3 SPECIAL COMMAND FORMAT

Byte #	0-4	5		6	7
Function	HEADER	COMMAND		DATA	CRC-16
Bits		MS Nibble	LS Nibble		
Subfunction		0=reserved 1=Halt pgm/ arm flash access 2=Start OS/ close flash 3=Start user pgm/ close flash 4=Erase pgm 5=erase OS 6=xfer to boot blk code/arm flash	Data	Last sector to erase	

Table 15 RUG3 STATUS REPLY FORMAT

Byte #	0-4	5	6,7	8	9, 10
Function	HEADER	STATUS: Bit 0: 1=halted Bit 1: 1=config error Bit 2: 1=OS error Bit 3: 1=Sector armed	OS version, LS,MS	Board revision	CRC-16

Send Flash Load Format

This format is used to load the flash with user configuration data. Hex ASCII representation uses vertical bar delimiters separating fields of hex numbers (absolute addresses and pointers) as ASCII except that constants and string constants are straight ASCII.

Table 16 RUG3 FLASH LOAD FORMAT

Byte #	0-4	5-8	9-N	N+1,N+2
Function	HEADER	Absolute starting address in hex ASCII	One line of load in hex ASCII	CRC-16

Table 17 Flash Load Reply Format

Byte #	0-4	5	6,7
Function	HEADER	STATUS: Bit 0: 1=halted Bit 1: 1=config error Bit 2: 1=OS error Bit 3: 1=Sector armed	CRC-16

Logger Dump Request Format

This format is used to request a dump of EventLogger or LogMany logger contents. Following reception of this request, the RUG3 will reply with the specified items up to the length of the communications buffer or 255 bytes maximum message length including header and CRC. Note that there are two dump request formats. The first initiates the dump and specifies all aspects of the dump source and format including which logger is to be dumped, time resolution of the time tags, first index to be sent, number of indices to be sent, synchronizing time stamp, and the format of all analogs to be sent. The second dump request is much shorter for efficiency and specifies whether the last dump is to be repeated (in case it was not received by the requester) or the next series of log indices is to be sent. End of log is signaled by a bit in the dump reply control field. If both the first log index to send and number of indices to send are zero, then the dump will commence with the most recent sample and continue until all samples taken since the last dump have been sent. The count of samples since the last dump is maintained by each logger and is cleared automatically at the end of each dump.

Table 18 RUG3 LOGGER INITIAL DUMP REQUEST FORMAT

Byte #	0-6	7		8,9	10,11	12-15	16-N	N+1, N+2
Funct	HDR	Control field		First log index to send	Number of indices to send	Time stamp	Analog formats	CRC -16
		MS 5 bits: B7:0=Logmany 1=Eventlog B6:0=1sec tags 1=ms tags B5,4,3: 0=init request 1=send next 2=repeat last 3=restart dump with existing setup 4=clr logger's 'new sample' counter 5-7=spares	LS 3 bits: Which logger (Logger ID=0-7) in RUG3 to respond	MS, LS, first log index to send counting from most recent logged item	MS, LS, number of indices to send	UNIX format time stamp, 1 sec. res. to init RTC. If zero, no effect on RTC	4-bits/analog: 0=skip value 1=4-byte float 2,3=spares Below: mult analog by value then send as signed int 4=*10,000 5=*1000 6=*100 7=*10 8=*1 9=*0.1 10=*0.01 11=*0.001 12=*0.0001 13=*0.00001 14,15=spares	

Subsequent dump requests do not need full definition of all parameters, only a command to send the next set of samples, repeat the last set of samples, or restart the dump from the start. That format is presented below.

Table 19 RUG3 LOGGER SUBSEQUENT DUMP REQUEST FORMAT

Byte #	0-6	7		8,9
Function	HEADER	Control field		CRC-16
		MS 5 bits: B7:0=Logmany 1=Eventlog B6:0=1sec tags 1=ms tags B5,4,3: X=init request 1=send next 2=repeat last 3=restart dump with existing setup 4=clr logger's 'new sample' counter 5-7=spares	LS 3 bits: Which logger (Logger ID=0-7) in RUG3 to respond	

Logger Dump Format

This format is a response to the above dump requests and is used to dump logger contents. Contents of either the event loggers or the log many modules can be dumped using these formats. Which type of logger is to be dumped is contained in the dump request above. Following reception of the request, the RUG3 will reply with logged data or events as defined in the table below. The first byte after the standard 7-byte header is a one byte dump control byte that indicates whether the message contains the last samples from the logger and, in the case of the LogMany module, whether the analog values are followed with a 16 bit status field containing the packed states of all logged statuses for those records.

Table 20 RUG3 LOGGER DUMP FORMAT (applies to both LogMany and Event Logger)

Byte #	0-6	7	8-N	N+1,N+2
Function	HEADER	Dump control	Logged event/data item(s)	CRC-16
		For EventLog: B7: 1=End of log B6-0: spares For LogMany: B7: 1=End of log B6: 1=16 bit status word follows last analog B5-0: spares	One or more records from EventLogger or LogMany module	

Each logged item in bytes 8-N above consists of a self-contained variable length binary message consisting of a one byte preamble followed by a 2, 4, or 6 byte time tag. Formats of these items are presented below and are different depending upon whether the requested dump is from a LogMany module or an Event Logger module. After the time tag are one or more analog values. The preamble identifies whether the time tag is absolute or relative to the last absolute tag sent; contains the event status; and specifies the event index or number of analogs following the time tag. The first item in a message will always contain an absolute time tag of 4 byte length if the request above specifies 1 second time resolution; or 6 byte length if the control field specifies ms (millisecond) time resolution. Subsequent items will use relative time tags of the same resolution unless the delta time between the new item and the last absolute time tag exceeds a two byte range. In that case, a new absolute time tag will be sent with the item; and all subsequent items in that response will be relative to it. Time tags present the number of seconds or milliseconds since January 1, 1970. Time tags in any message are presented MS byte first followed by intermediary bytes and ending with the LS byte.

The table below presents the data format for a single event from an EventLog module. The most significant preamble bit is a spare. The next most significant bit specifies whether the item's time tag is absolute or relative to the last absolute time tag sent. The next most significant bit holds the event status (0 or 1). The least significant 5 bits specify which channel caused the event to be recorded. Integer values in any message are presented MS byte first followed by the LS byte. Floating point values are presented with exponent first followed by the mantissa MS byte, intermediary byte, and LS byte last.

Table 21 RUG3 LOGGED EVENT ITEM FORMAT (format for each event/data item in table above)

Byte #	0		1-N	N+1…N+4
Function	PREAMBLE, 1 byte		Time Tag (2, 4 or 6 bytes)	Analog Value
	B7: Spare B6: 0=abs time tag 1=rel time tag B5: event status (0,1)	Bits 0-4: Channel Index from event logger module, 0-31	Absolute 6-byte (ms resolution) Absolute 4-byte (1 sec resolution) Relative 2-byte (ms resolution) Relative 2-byte (1 sec resolution) Byte order: MS…LS	2-byte signed integer, or 4-byte float (Format specified by initial request) Byte order: Integer: MS,LS Float: Exp,MS…LS

Similar to the table above, the table below presents the data format for a single record from the LogMany module. The first preamble byte defines the format for that event of all data following the preamble. The most significant preamble bit is a spare. The next most significant bit defines the time tag format. Remaining bits are spares.

Table 22 RUG3 LOGMANY ITEM FORMAT

Byte #	0		1-N	N+1,…N+X
Function	PREAMBLE		Time Tag (2, 4 or 6 bytes)	Analog Value(s)
	B7: Spare B6: 0=abs time tag 1=rel time tag B5: Spare	B0-B4: Spares	Absolute 6-byte (ms resolution) Absolute 4-byte (1 sec resolution) Relative 2-byte (ms resolution) Relative 2-byte (1 sec resolution) Byte order: MS…LS	2-byte signed integers, or 4-byte floats, (Format specified by initial request) Byte order: Integer: MS,LS Float: Exp,MS…LS

Logger Dump Request/Response Examples

Below is an example of requesting all records from a LogMany module that have been logged since the last dump. The LogMany has been recording a single 16 bit integer per record. The request specifies 1 second time tags followed by the single integer value per record. The logger had saved 12 records since the last dump.

INITIAL DUMP REQUEST:

$C8	Header sync byte
$13	Message length byte (19 bytes incl. CRC)
$40	Message type (request logger dump using extended addresses)
$00	Source address MS
$01	Source address LS (Source address is $0001=1)
$02	Destination address MS
$05	Destination address LS (Destination address is $0205=517)
$03	Control field specifies LogMany with ID=3, 1 sec time tags
$00	First log index to send MS
$00	First log index to send LS (start at most recent sample in log)
$00	Number of indices to send MS
$00	Number of indices to send LS, value of zero specifies to send all samples since last dump
$00	Time stamp MS, value of zero specifies to not alter realtime clock
$00	Time stamp
$00	Time stamp
$00	Time stamp LS
$08	Analog to be dumped as integer with multiplier of 1 (i.e., unaltered)
$F1	CRC byte 1
$A8	CRC byte 2

RESPONSE:

$C9 4B 40 00 01 02 05	Reply, 59 byte length, source address 517, destination 1
$00	Dump control, not end of log, no status word follows analogs
$00 48 66 5D D5 00 0B	1st item: abs time tag=June 28, 2008 15:50:45, integer value=11
$40 00 02 00 0A	2nd item: rel time tag=2 seconds before above, integer value=10
$40 00 04 00 09	3rd item: rel time tag=4 seconds before above, integer value=9
$40 00 06 00 08	4th item: rel time tag=6 seconds before above, integer value=8
$40 00 08 00 07	5th item: rel time tag=8 seconds before above, integer value=7
$40 00 0A 00 06	6th item: rel time tag=10 seconds before above, integer value=6
$40 00 0C 00 05	7th item: rel time tag=12 seconds before above, integer value=5
$40 00 0E 00 04	8th item: rel time tag=14 seconds before above, integer value=4
$00 48 63 BA A4 00 03	9th item: abs time tag=June 26, 2008 15:49:56, integer value=3
$42	CRC byte 1
$AE	CRC byte 2

SUBSEQUENT DUMP REQUEST TO SEND NEXT BATCH OF SAMPLES:

$C8	Header sync byte
$0A	Message length byte (10 bytes incl. CRC)
$40	Message type (request logger dump using extended addresses)
$00	Source address MS
$01	Source address LS (Source address is $0001=1)
$02	Destination address MS
$05	Destination address LS (Destination address is $0205=517)
$0B	Control field specifies LogMany with ID=3, 1 sec time tags
$8D	CRC byte 1
$38	CRC byte 2

RESPONSE:

$C9 1B 40 00 01 02 05	Reply, 59 byte length, source address 517, destination 1
$80	Dump control, end of log, no status word follows analogs
$00 48 63 BA A2 00 02	1st item: abs time tag= June 26, 2008 15:49:56, integer value=2
$40 00 02 00 01	2nd item: rel time tag=2 seconds before above, integer value=1
$40 00 04 00 00	3rd item: rel time tag=4 seconds before above, integer value=0
$91	CRC byte 1
$16	CRC byte 2

If an additional request were to be sent to dump the next batch of samples, identical to the subsequent dump request above, or if a request to clear the 'new sample' counter is sent, the RUG3 would reply with an 'empty log' response:

$C9 0A 40 00 01 02 05	Reply, 10 byte length, source address 517, destination 1
$80	Log empty bit is set
$8D 38	CRC bytes

Below is an example of a valid dump request. It requests a dump of all log contents of a LogMany module with ID=3 from the most recent index to the end of log. The logger is assumed to have recorded 10 analog values per record. The request specifies 1 second time tags to be followed by analog values number 1,2,3,4 and 7 of each record. Logged values 5, 6, 8, 9, and 10 of each record are not dumped. Here would be a valid request and response for this dump (all bytes shown as hexadecimal):

INITIAL DUMP REQUEST:

$C8	Header sync byte
$17	Message length byte (23 bytes incl. CRC)
$40	Message type (request logger dump using extended addresses)
$00	Source address MS
$01	Source address LS (Source address is $0001=1)
$02	Destination address MS
$05	Destination address LS (Destination address is $0205=517)
$03	Control field specifies LogMany with ID=3, 1 sec time tags
$00	First log index to send MS
$00	First log index to send LS (start at most recent sample in log)
$01	Number of indices to send MS
$80	Number of indices to send LS ($0180=384)
$48	Time stamp MS…June 17, 2008 at 10:16:21 (AM)
$57	Time stamp
$8E	Time stamp
$F5	Time stamp LS
$18	Analog #1 as integer (LS nibble), analog #2 as float (MS nibble)
$76	Analog #3 as integer (float *100.0), analog #4 as integer (float *10.0)
$00	Skip analog #5 and #6
$05	Analog #7 as integer (float *1000.0), skip analog #8
$00	Skip analog #9 and #10
$C7	CRC byte 1
$B9	CRC byte 2

RESPONSE:

$C9 56 40 00 01 02 05	Reply, 86 byte length, source address 517, destination 1
$00	Dump control, not end of log, no status word follows, logger ID=3
$85	First item: LogMany, abs time tag, no status word, 5 analogs follow
$48 5A 6C 35	Absolute time tag: June 19, 2008, 14:24:53
$00 0E 42 8B 18 96 05 D1 02 B7 3A 2D	14, 69.5, 1489, 695, 14893
$C5 00 01	Second item, 1 second prior to first item above
$00 0E 42 8B 18 96 05 D1 02 B7 3A 2D	14, 69.5, 1489, 695, 14893

$C5 00 04	Third item, 4 seconds prior to first item above
$00 0E 42 8B 18 96 05 D1 02 B7 3A 2D	14, 69.5, 1489, 695, 14893
$C5 00 07	Fourth item, 7 seconds prior to first item above
$00 0E 42 8B 18 96 05 D1 02 B7 3A 2C	14, 69.5, 1489, 695, 14892
$C5 00 0A	Fifth item, 10 seconds prior to first item above
$00 0E 42 8B 18 96 05 D1 02 B7 3A 2C	14, 69.5, 1489, 695, 14892
$XX XX	CRC

Note: the values contained in the response are: analog #1 (14.892 as integer), analog #2 (69.5 as float), analog #3 (14.892 *100 as integer), Analog #4 (69.5*10 as integer), and analog #5 (14.893*1000 as integer).

CHAPTER 8…SAMPLE APPLICATIONS

Introduction

This chapter presents a number of application examples to illustrate how to apply the RUG3, configure the software modules, set up displays, use ladder logic, etc. All examples assume the use of standard RUG3 board configurations with displays (RUG3D, RUG3P). In addition, we have used constants for most module input properties in these examples. Since virtually all input properties can be taken from the databases, we could quite easily have used setpoints for many of the input properties that could otherwise be set with setpoints. For example, we have used the constant "5" for the analog input filter time constant in all examples. This means that the input value "5" would be obtained directly from the flash memory and would be very secure against inadvertent change due to transients. If you change that to a setpoint, the user would be able to adjust the analog input filtering, and would become responsible for getting it right. In these examples, we will not discuss all details of each application; only those details we wish to illustrate with the example.

AP NOTE #1: Stand Alone Tank Level Monitor, TankTest.rgd

The following application uses a RUG3D to monitor and display a tank level, and issue high and low alarms using a pair of relays. It also presents the tank level and alarms on its LCD. High and low alarm setpoints are entered locally. The following diagram presents the necessary wiring. Configuration file for this application is TankTest.rgd.

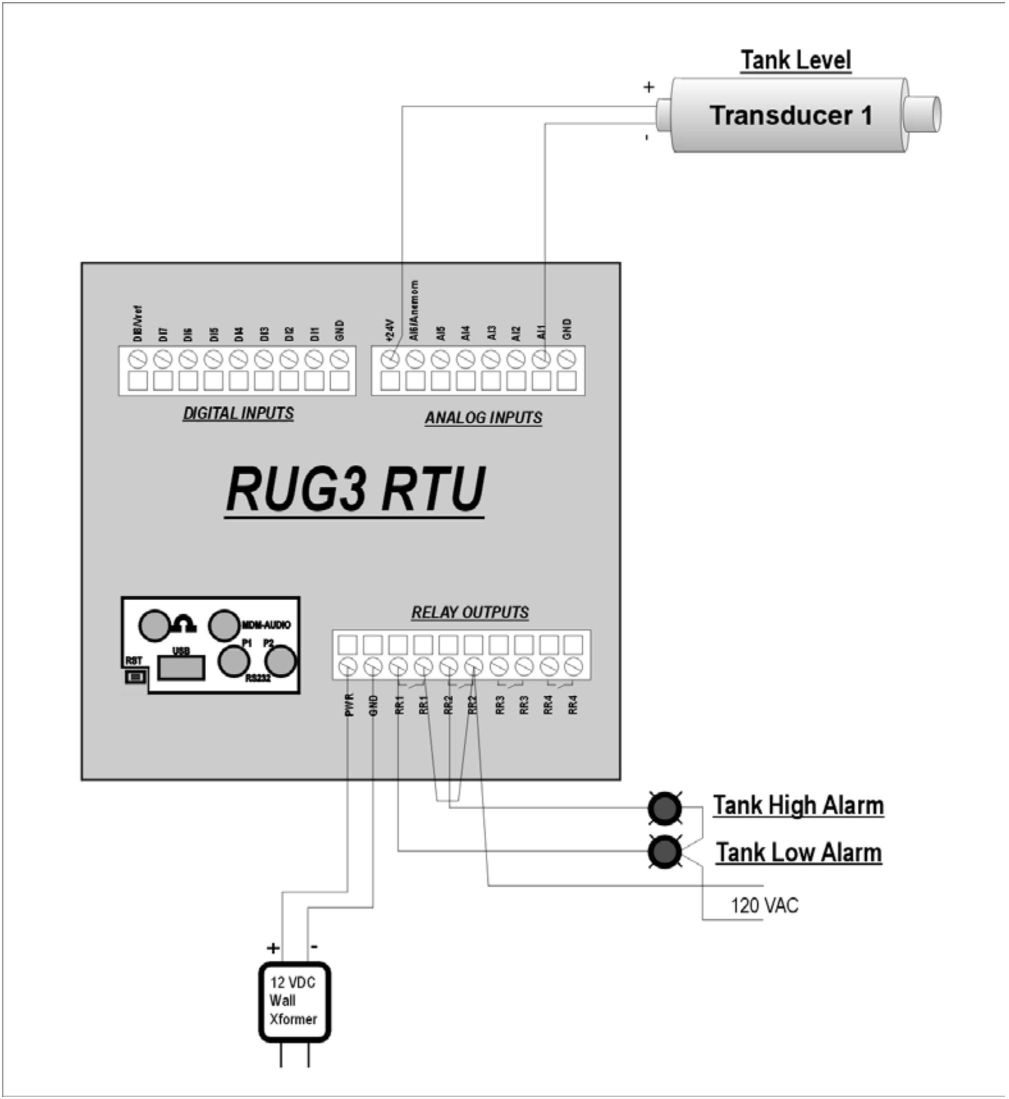
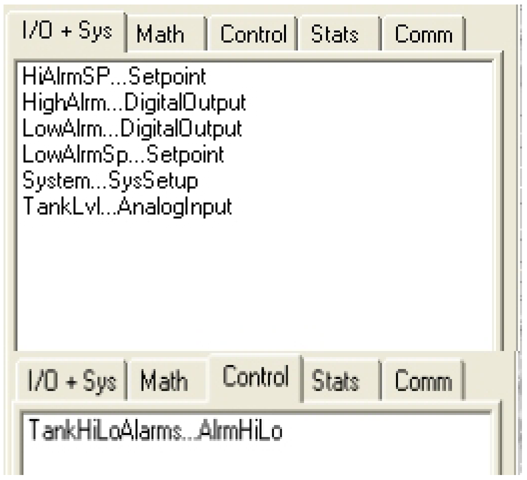


Figure 63 Tank Level Monitor Application

The following modules are used in this example:

Sample Applications



HiAlrmSP...Setpoint
HighAlrm...DigitalOutput
LowAlrm...DigitalOutput
LowAlrmSp...Setpoint
System...SysSetup
TankLvl...AnalogInput

TankHiLoAlarms...AlrmHiLo

Figure 64 Modules in R3DTank1 Application

Here is a list of what each module does:
HighAlrm…Digital Output and **LoAlrm…Digital Output**:
These modules are used to connect the alarm statuses for low alarm and high alarm, to the first two relays on the relay output board. If we had used AlarmOutput modules instead of Digital Output modules here, the outputs would have flashed on and off automatically when their corresponding alarms were declared.

System…SysSetup:
This module sets the display backlight blanking interval, the logoff interval, and sets up the logon security code for setpoint access.

TankHiLoAlarms…AlrmHiLo:
This module compares the high and low alarm setpoints with the tank level and issues output statuses when alarm conditions exist. They delay the declaration of an alarm by 7 seconds using timers.

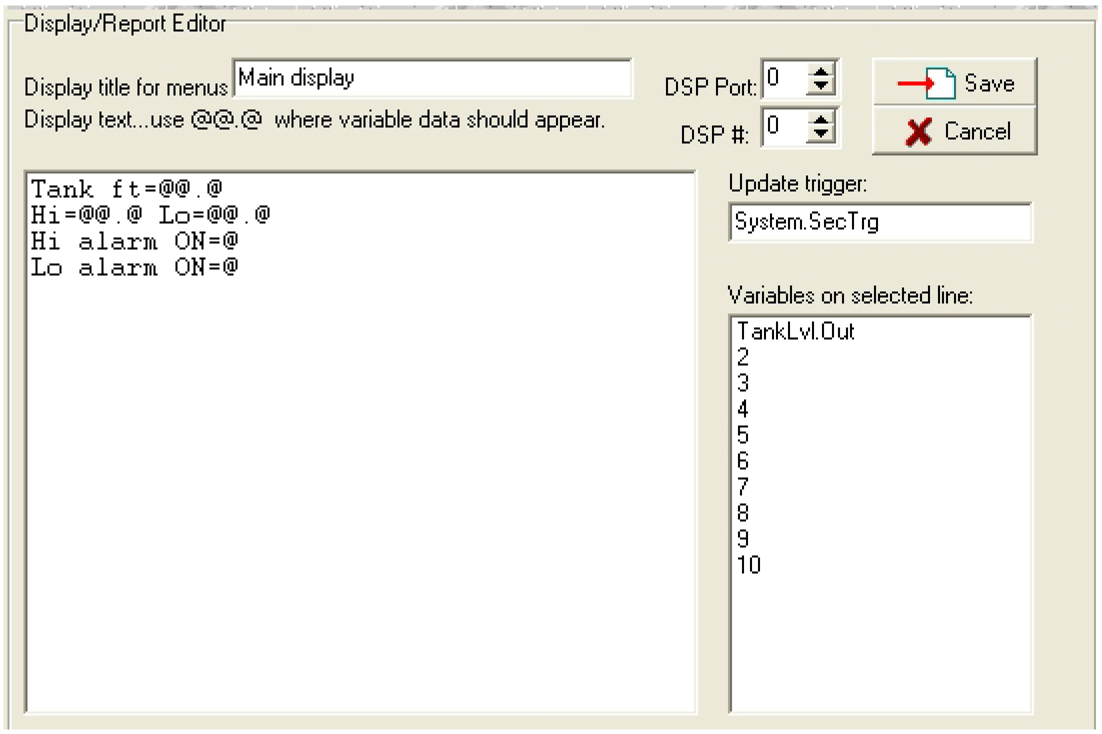
HiAlrmSp…Setpoint and **LowAlrmSp…Setpoint**:
This is where the high and low alarm setpoints are created so they become part of the setpoint list. Their values are sent to the floating point data base for use by the alarm detection modules above and for display.

TankLvl…Analog Input 4-20:
This module reads the channel 1 A/D converter, low pass filters it, converts its value to engineering units in the range of 0 to 20 feet, and sends the result to the floating point data base where it is available for display and use by alarm modules. The module assumes the input is a 4-20 ma. signal.

Main Display Setup

We'll examine the main display, which is presented below.

Sample Applications



Display title for menus: Main display

Display text...use @@.@ where variable data should appear.

DSP Port: 0

DSP #: 0

```
Tank ft=@@.@
Hi=@@.@ Lo=@@.@
Hi alarm ON=@
Lo alarm ON=@
```

Update trigger: System.SecTrg

Variables on selected line:

TankLvl.Out
2
3
4
5
6
7
8
9
10

Figure 65 R3DTank1 Main Display Setup

Notice that the display title for menus is given as "Main display". This means that this display will be referred to as "Main display" in the display list box of the R3Setup program, and in the RUG3 display menu when the RUG3 is running. The display port choice of zero means that this display will be presented on the RUG3's LCD. Display number zero means that this display will always be listed first in menus. The update trigger of "System.SecTrg" was dragged into the trigger list box from the status data base. When running on the RUG3, that trigger will cause the display to rewrite once per second. Finally, notice that the cursor is resting on the top line, the line that presents the tank level in the large display editing window. At the same time, the ten item list at the right side of the setup screen above begins with the variable TankLvl.Out. That variable name was dragged from the floating point data base into the "Variables on selected line" list box while the cursor was resting as shown. Its placement at the top of the list establishes that when the RUG3 hits the "@@.@" field on the top line of the display, it will go get the tank level and place it on the LCD in place of the "@@.@" field.

AP NOTE #2: Stand Alone Two Pump Controller, TANK2PUMPS.rgd

This example implements a controller for two pumps to maintain a tank's level using a local tank level measurement. It is based on the example file "Tank2Pumps.rgd". In it, we wish to illustrate use of the following modules:

- **LeadLagSeq4**…lead lag sequencer for rotating the lead pump and including backspin delay
- **PumpUpDnCtrl**…pump up/down controller
- **MismatchLatch**…used to detect pump failure
- **StringSwitchByBits**…used to present meaningful messages to LCD

The figure below presents the wiring for this example:

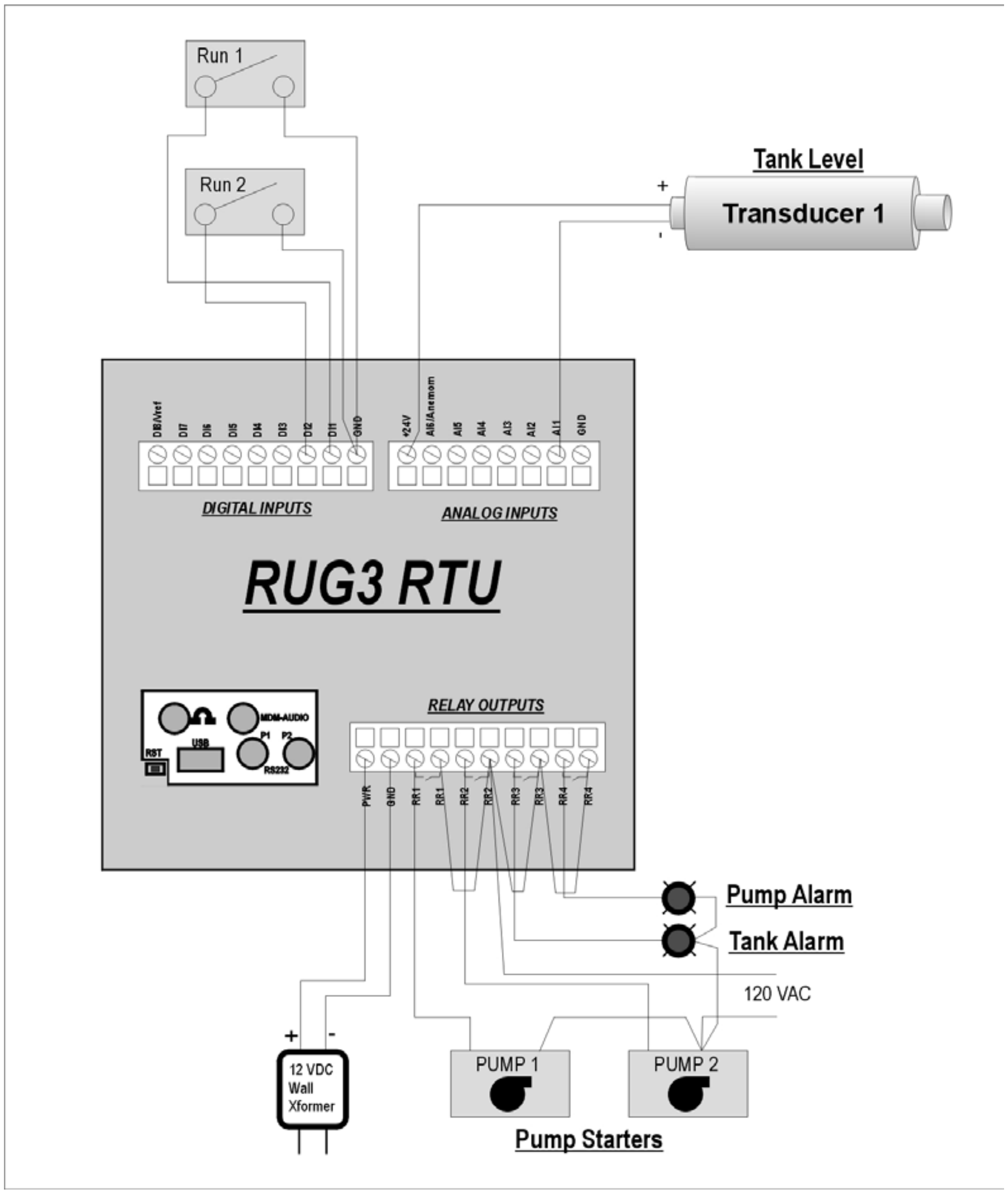


Figure 66 Two Pump Stand Alone Controller

General Operation

The pump control function is accomplished by two **PumpUpDnCtrl** modules, each of which constantly compares the tank level with CALL and OFF setpoints to determine if its pump needs to be called. As the tank's level falls, pump A will first be called, then Pump B, etc. Outputs from these two controllers are routed to a **LeadLagSeq4** sequencer. Its job is to call as many of its outputs as its inputs demand, beginning with the lead pump. Since we haven't specified a lead pump, it rotates the lead pump to equalize wear on the pumps. It also delays pump switching so that a specified delay occurs between successive pump switch actions. Sequencer outputs are routed to two relays that would be used to call pump starters. This logic is presented in the following diagram. The **MismatchLatch** modules are used to detect pump failures by comparing the pump call signals with pump run digital inputs. If the run indication is not present within a user determined time delay, then the pump is declared failed, and an alarm output is latched. The **MismatchLatch** module's latched output it used to lock out the corresponding pump. Aspects of the use of these modules in this application are discussed below.

Sample Applications

In a practical application, you might wish to interject **HOA2** modules between the **LeadLagSeq** module’s outputs and the relay output modules. You could then use setpoints to control the HOA states so the operator could declare each pump’s OFF, HAND or AUTO mode by changing its setpoint.

Note that using PumpUpDn modules enables you to switch this control strategy from pump up to pump down with a simple setpoint change..

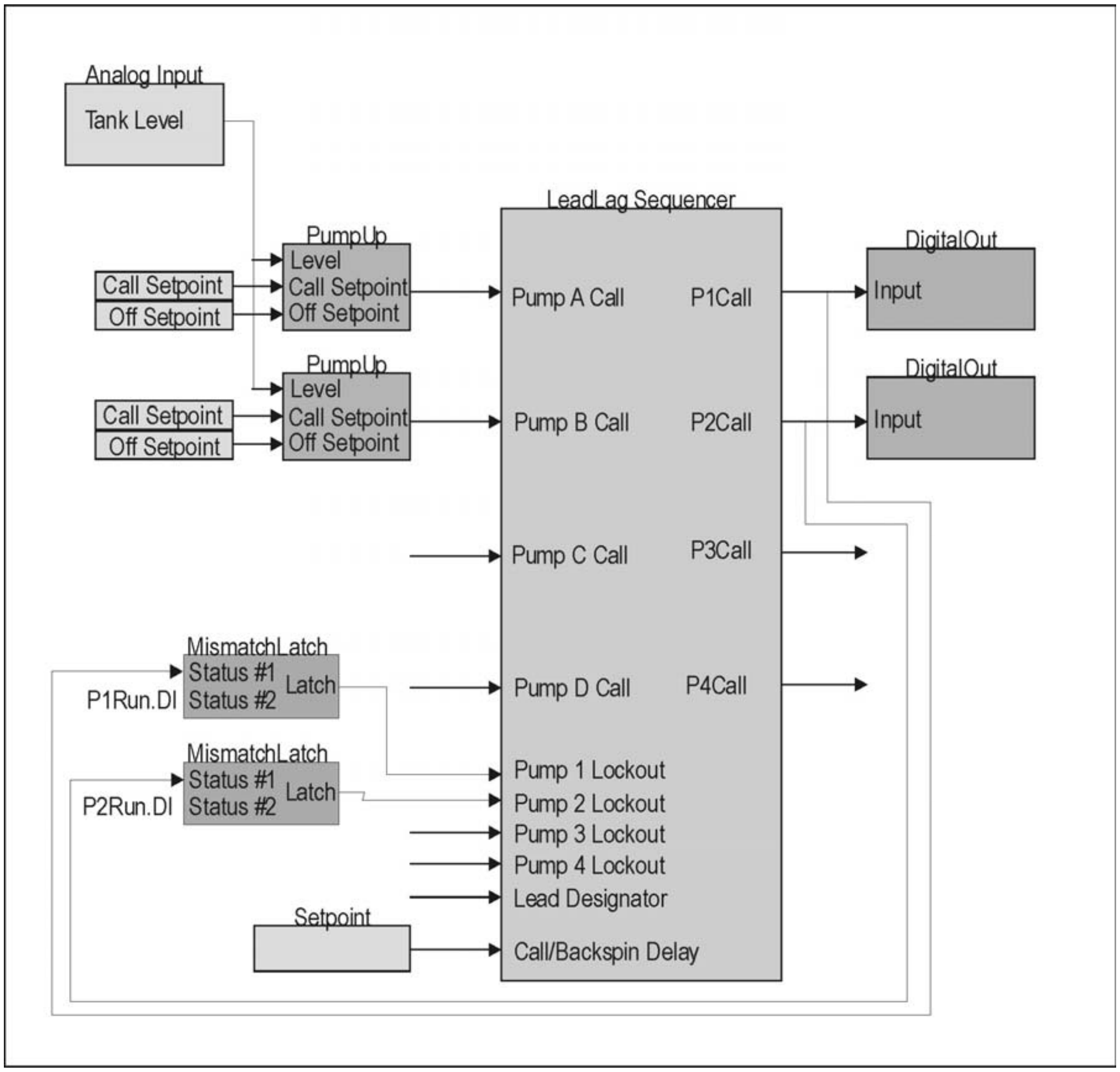


Figure 67 Four Pump Control Logic Diagram

LeadLagSeq4 Module

In this application, the **LeadLagSeq4** module receives pump call demands from the **PumpUpDn** modules. When it receives its first call, it will immediately call the lead pump. In this application, the lead pump designator has been left blank, so the sequencer will increment the lead designator each time the first call is received when all pumps are off. You could install a setpoint and connect it to the lead designator to enable the operator to designate a lead pump in the range of 1 to 4. To allow lead pump rotation, he would simply designate a lead pump of zero. This module also performs ON and OFF delay timing to assure that no two pumps will switch on or off at the same time. To control more or fewer pumps, simply install more or fewer pump up controllers and connect them to the **LeadLagSeq4** module’s call inputs. The sequencer determines how many pumps to control based on how many call inputs have been configured.

Sample Applications

PumpUpDn Module

The two **PumpUpDn** modules in this example perform the comparisons between tank level and user-entered setpoints. Here, we are using them in pump up mode (mode=0), where they function to maintain a tank above a certain level by calling for a pump if the tank's level falls below a user entered pump call setpoint. During pumping, if the tank rises above the pump off setpoint, then the module turns off the pump call. Using a setpoint (UpDownChoice.SP in this example) the operator can change the mode from Mode=0 to Mode=1, causing the **PumpUpDn** controller to become a pump down module. In this mode, it acts to call a pump if the tank level rises above a certain level, then pumps it down and turns off the pump once the level falls below the pump off setpoint.

MismatchLatch Module

Two **MismatchLatch** modules are used to watch for a pump's call output to match its run indication. If they are not the same (both=0 or both=1) within a user determined time delay, then the module declares an alarm and latches the alarm. The latched alarm is used in this application to lockout the corresponding pump until an operator presses a reset button on the RUG3's keyboard. The latched alarms are also ORed together to produce a pump fail indication that controls relay #4 that could be used to energize a lamp or horn.

StringSwitchByBits Module

It's easy to present statuses on a display as either '0' or '1', but it is clearer to plant operators if we use words like CALL, RUN, FAIL, OK, etc. The **StringSwitchByBits** module is used in this example to do just that. Each module accepts up to 4 statuses and uses them to select one of 16 strings to present as the module's output. We use the string outputs in place of '0' or '1' on the display.

AP NOTE #3: Telemetering Tank Site, TANKTESTTLM

In this application note, we will take the stand alone tank site example above, and add telemetry to it. This application is based on the example file "TankTestTlm.rgd". In it, we wish to illustrate use of the following modules:

- **ComSetup**…establishes communications channel parameters for the modem
- **Rx and Tx arrays**…arrays that define communications formats
- **TriggerOnRCV**…issues trigger when the unit receives a message from a particular station

The figure below presents the wiring diagram for this example:

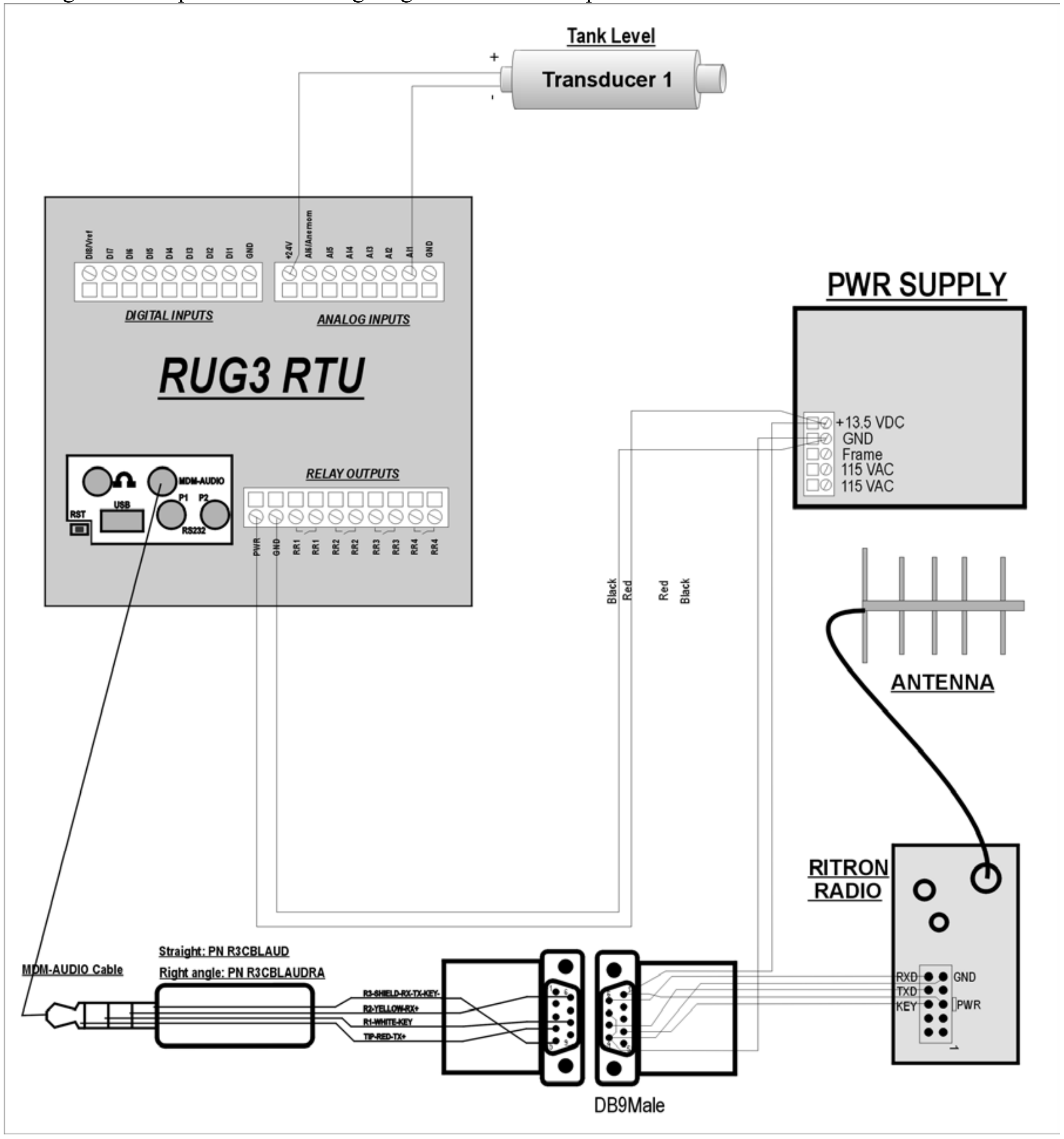


Figure 68 Radio Telemetry Tank Monitor

General Operation

This program reads an analog input value, the tank level, and compares it with high and low alarm setpoints it has received from the master site via the receive telemetry (RX) array. It also keeps the tank level in the outgoing transmit (TX) array for sending to the master when the master polls. It generates high and low tank level alarms to send to the master. It also generates low battery voltage alarm, and measures battery voltage and on board temperature to send to the master.

This uses the modem's four wire channel for communications with the master, with transmit delay set compatible with radios. Therefore, it can be used for either audio radio or lease phone line applications. Station address is adjustable using a local setpoint, so the program can be used at any address in a system. However, before the unit will communicate, the address setpoint will need to be set correctly after first installation.

Communications Setup

The modules used to setup communications are the **ComSetup** module, which initializes the modem, the RX and TX arrays, which hold received data and data to be transmitted, respectively, and the **TriggerOnRCV** module, which detects when a message has been received. These modules are discussed below:

ComSetup

ComSetup module input properties for this application are presented below:

Inputs:

- Port (1,2)…integer port designator, set to 2 designating the modem port.
- Trigger to install setup…status trigger input. This input TRUE causes this module's setup to be installed in the designated port. Uses output of **OrGate** module causing installation on either boot up or installation of a new unit address. Also input to the OR gate is a once per 10 minutes trigger that reinstalls the ComSetup module parameters each 10 minutes as a safeguard against the modem port losing a critical parameter and becoming non-communicative indefinitely.
- 0=RS232, 1=modem…integer to establish connection between the UART and other hardware on the modem. Set to mode 1 to connect to audio radios or to 4-wire leased line phone systems.
- Baud (50-19,200)…integer specifying any baud rate from 50 to 19,200 baud. Set to 300 baud for radio or phone line use. The RUG3 modem is limited to 300 baud to keep the design extremely low power. Higher baud rates would require an external modem or use of a radio with internal modem.
- Parity (0,2,3)…integer specifying parity choice: 0=none, 2=odd, 3=even. Set to 0=none.
- Address (1-254)…integer address for this port on the network. Set to the output of a setpoint module.
- Mode (1,2,3,4,6,7,8,9)…integer to select communication protocol. Set to 2, RUG9 protocol.
- TX delay tenths of sec…integer to set delay between the time the RUG3 keys its modem and radio, and the time it actually sends data. This is necessary to enable receiving radios and modems to acquire the signal. Set to 7, for radio or phone line application.
- TX amplitude(0-255)…transmit amplitude, where 0-255 spans 0 to approximately 4 V peak to peak. Set to 127.
- Com buffer #bytes…size of buffer allocated for receiving and transmitting. Must be set to something greater than the longest anticipated received or transmitted message. Set to 150 bytes, which should allow messages of up to 65 integer registers.

Detecting and Displaying Receptions

The **TriggerOnRcv** module detects receptions. In order to ascertain that the communications system is working, it is useful to know if this station is successfully receiving from the master. For a reception to be accepted, it must be received with the correct cyclic redundancy check (CRC). If the CRC check fails, no data is accepted from the message, no reply is transmitted, and the **TriggerOnRcv** module will not issue a trigger indicating that a successful reception occurred. **TriggerOnRcv** module inputs are set as follows:

Inputs:

- Port to receive…integer port specifier, set to 2, the modem port.
- Address of source…integer address of source station from which transmission is to cause a trigger. Leaving this entry blank will cause the module to issue a trigger on any transmission directed to the destination address. Set to 1, the master's address.
- Address of destination…integer address of message destination to which a transmission is to cause a trigger. Leaving this entry blank will cause the module to issue a trigger on any reception from the source address. Set to 255 specifying whatever address this unit has.

Primary Outputs:

- Trigger event output…Rcv.Trigger…status trigger that becomes true when a message is received on the port and board specified, and having the source and destination addresses specified.
- RCVed source address…Rcv.Src…integer address of last received message on the specified board and port, even if it was not destined for this unit.
- RCVed destination addr…Rcv.Dest…integer address of last received message on the specified board and port, even if it was not destined for this unit.

The **TriggerOnRcv** module's trigger output is used to preset a counter that counts seconds, so the counter will indicate the number of seconds since the last reception.

Handling Address Setting by Setpoint

In this application, we enable the operator to set the address simply by setting a value in a setpoint. That in itself is easy; we simply install a setpoint module and use its output as the address input to the **ComSetup** module and as the destination address for the **TriggerOnRcv** module above. But, we must trigger the **ComSetup** module to install the address before it will have effect in actual communications. That involves the using a **TrigOnChange** module that watches for changes in the setpoint and issues a trigger when it detects a change. That trigger is used to re-install the **ComSetup** module.

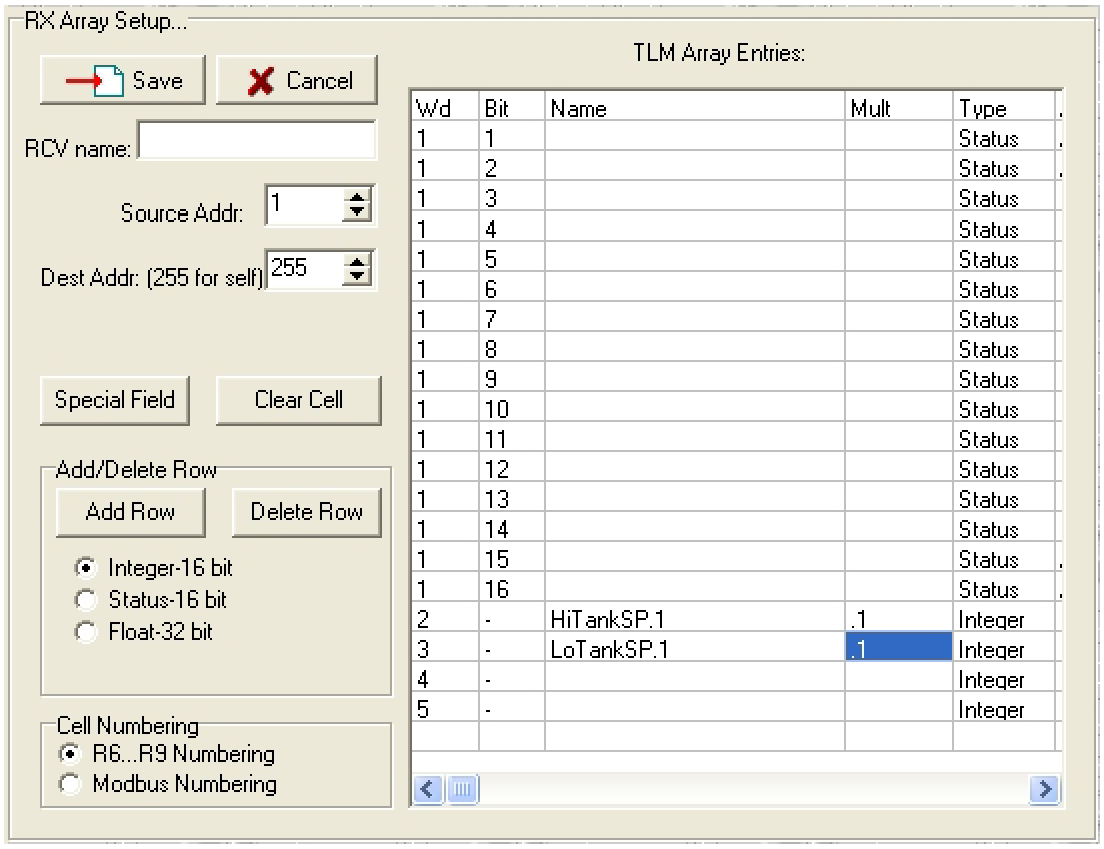
Setting Up Telemetry Arrays

To establish the contents of the messages flowing between the master and this remote, we must configure the telemetry arrays for both incoming data and transmitted data.

Receive Array

The figure below presents the RX array setup, i.e., the array where data from the master will arrive.

Sample Applications



RCV name:

Source Addr: 1

Dest Addr: (255 for self) 255

[✓ Integer-16 bit] Status-16 bit Float-32 bit

[✓ R6...R9 Numbering] Modbus Numbering

Wd	Bit	Name	Mult	Type
1	1			Status
1	2			Status
1	3			Status
1	4			Status
1	5			Status
1	6			Status
1	7			Status
1	8			Status
1	9			Status
1	10			Status
1	11			Status
1	12			Status
1	13			Status
1	14			Status
1	15			Status
1	16			Status
2	-	HiTankSP.1	.1	Integer
3	-	LoTankSP.1	.1	Integer
4	-			Integer
5	-			Integer

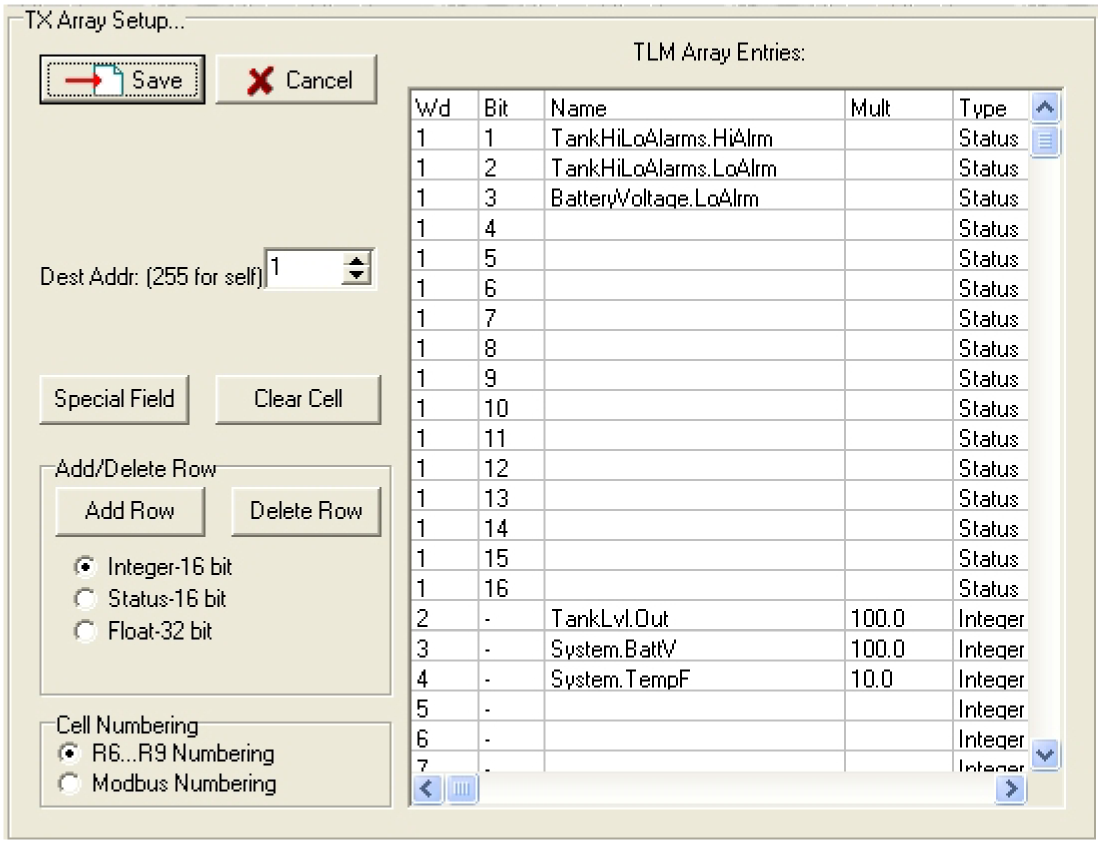
Figure 69 Receive Array Setup

Notice that this array references source address=1 and destination address=255. Source address=1 means that this array will accept data from address number 1, the master site. Destination address=255 means that this array is to accept messages destined for us, no matter what our address is set to. Notice that there are only two entries in the array: TankHiAlrmSP.1 and TankLoAlrmSP.1. These names were typed into the cells as the names of signals from the master. The ".1" appendage was added by the compiler to designate that these variables are arriving from site number 1. The multiplier of 0.1 for each of these variables, and the designation of integer, indicate that the system is to read each of these as an integer from the receive buffer and multiply by 0.1 before storing the result in the receive database. At the master, before transmission, each setpoint is multiplied by 10.0 before being sent to this site. The multiplier preserves one place to the right of the decimal point, yet allows the value to be sent as an integer, which is more efficient to transfer (2 bytes) than a 4 byte floating point value.

Transmit Array

The figure below presents the transmit array setup, i.e., the array that establishes the contents of this station's reply to master polls.

Sample Applications



Dest Addr: (255 for self) 1

[✓ Integer-16 bit] Status-16 bit Float-32 bit

[✓ R6...R9 Numbering] Modbus Numbering

TLM Array Entries:

Wd	Bit	Name	Mult	Type
1	1	TankHiLoAlarms.HiAlrm		Status
1	2	TankHiLoAlarms.LoAlrm		Status
1	3	BatteryVoltage.LoAlrm		Status
1	4			Status
1	5			Status
1	6			Status
1	7			Status
1	8			Status
1	9			Status
1	10			Status
1	11			Status
1	12			Status
1	13			Status
1	14			Status
1	15			Status
1	16			Status
2	-	TankLvl.Out	100.0	Integer
3	-	System.BattV	100.0	Integer
4	-	System.TempF	10.0	Integer
5	-			Integer
6	-			Integer

Figure 70 Transmit Array Setup

Here, there is no source address, since we are the source. The destination address is set to 1, the master's address. If we wished to send to another unit, we would have to create a different array for that address. The entries in this array consist of three statuses (the alarms) and three floating point values that are designated to be multiplied by a factor (100.0 or 10.0), and then be sent as integers. The statuses are dragged from the status database, and the three floating point values are dragged from the floating point database. Note that the range of value that can be sent without a multiplier is –32767 to +32768. If we use a multiplier, such as 100.0 as in the case of the tank level above, then the tank level that we can report is limited to the range of –327.67 to +327.68…normally not a limitation for practical tanks.

AP NOTE #4: Communications Setup, General

In our experience, it is preferable to define the transmit and receive arrays in the remotes before defining them in the master site, since control strategies and I/O assignments will largely dictate the specific data that must be passed between the RTU's and the master. After that, then define the master arrays to match the remote arrays. To illustrate the relationship between the arrays, the following diagram shows how the tank level originates in the tank site and is passed among other arrays to get to its destinations. Notice that the tank level is multiplied by 100.0 every time it is transmitted and by 0.01 whenever it is received to preserve two decimal places. Site to site transfers are done using the communications channel, whereas, within a unit such as the master site, transfers are done by dragging the measurement from one array to another.

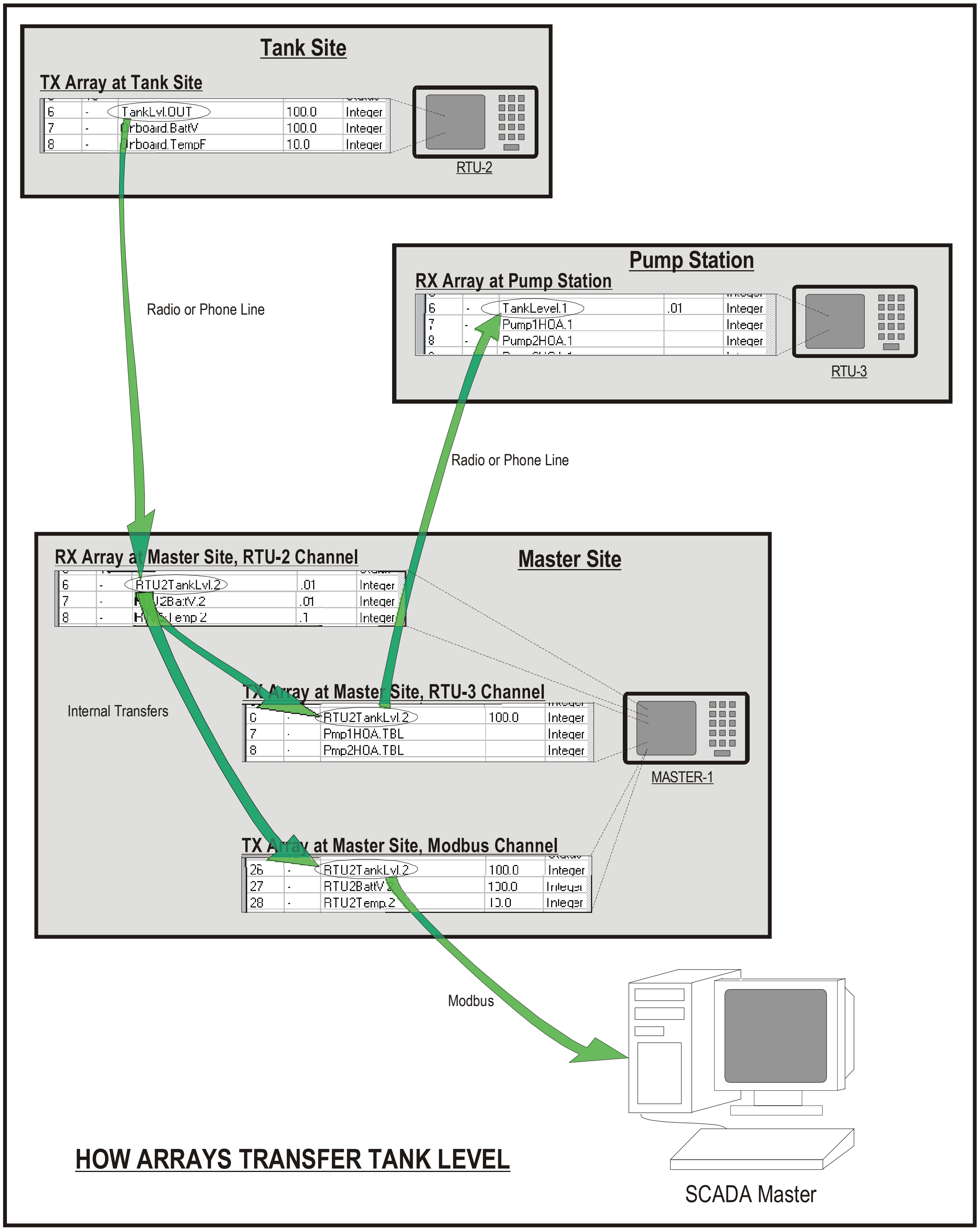

Tank Site
TX Array at Tank Site
6 - TankLvl.OUT 100.0 Integer
7 - Onboard.BattV 100.0 Integer
8 - Onboard.TempF 10.0 Integer
RTU-2
Pump Station
RX Array at Pump Station
6 - TankLevel.1 .01 Integer
7 - Pump1HOA.1 Integer
8 - Pump2HOA.1 Integer
RTU-3
Radio or Phone Line
Radio or Phone Line
RX Array at Master Site, RTU-2 Channel
6 - RTU2TankLvl.2 .01 Integer
7 - RTU2BattV.2 .01 Integer
8 - RTU2Temp.2 .1 Integer
Master Site
Internal Transfers
TX Array at Master Site, RTU-3 Channel
6 - RTU2TankLvl.2 100.0 Integer
7 - Pmp1HOA.TBL Integer
8 - Pmp2HOA.TBL Integer
MASTER-1
TX Array at Master Site, Modbus Channel
26 - RTU2TankLvl.2 100.0 Integer
27 - RTU2BattV.2 100.0 Integer
28 - RTU2Temp.2 10.0 Integer
Modbus
HOW ARRAYS TRANSFER TANK LEVEL
SCADA Master

Figure 71 Communications Setup, Passing Tank Level Among Sites

AP NOTE #5: Ethernet Hookup

Introduction

The RUG3 can easily be connected to an Ethernet network using one of the commonly available serial to Ethernet adapters such as the Digi One Real Port that we tested. We derived the installation procedure below from the manufacturer's installation and configuration guidelines for the device.

Digi One RealPort RUGID Configuration

RealPort hardware installation:

Connect the RealPort Ethernet interface to the LAN
Connect the RealPort serial connection to the RUGID using a "**NULL Modem** cable or adapter.
Apply power to the RealPort

Software Installation (Configure RealPort IP Address):

Install **Digi Port Authority – Remote** software
Insert Digi "Access Resource" CD
Select your operating system from the drop down menu
Select **Digi One** from the Hardware menu
Select **Digi Port Authority – Remote** from the Software menu
Install Software
Run the DPA – Remote software

2. If DPA-Remote is not set for ADDP, choose ADDP as the Discovery Protocol.
3. Choose Discover.
A list of Digi devices appears. Systems with IP addresses of 0.0.0.0 need IP addresses.
4. Select a device from the list and then choose Configure.
5. Supply an IP address, subnet mask and default gateway and then choose OK.
DPA-Remote configures the IP address, subnet mask and default gateway.
If the **DPA – Remote** software was unable to configure an IP address
Consult chapter 2 in the documentation included on the CD – Rom
\documentation\92000305_B.pdf

Configuring the Digi One RealPort:

Open Internet Explorer and enter the IP address of the RealPort
At the Login prompt enter "root" (default)
At the password prompt enter "dbps" (default)
From the menu on the left select "**Configure**" "**Port**" **Port1**"
Port1 should have the following parameters
Device Type = RealPort
Terminal Type = vt100
Flow control = none
Baud Rate = 9600
Data Bits = 8
Stop Bits = 1
Parity = None
Disable SocketID, Disable AutoConnect
Click the **Submit** button

Sample Applications

Software Installation (RealPort device driver):

Install **RealPort** software
Insert Digi "Access Resource" CD
Select your operating system from the drop down menu
Select **Digi One** from the Hardware menu
Select **RealPort** from the Software menu
Install Software
Follow the onscreen directions
You will need the previously configured IP address of the Digi One device to complete the installation
Record the COM port that is used by the RealPort driver

Accessing the RUGID remotely:

The PC software that "talks" to the RUGID through the Ethernet must be configured to use the same COM port as the RealPort driver.
The PC software must also be configured as follows
9600 baud
no flow control
8 data bits
1 stop bit
no parity
This is the default for R3setup

AP NOTE #6: ALERT Transmitter

The RUG3's extremely low power (<3ma.), even with its modem able to receive, makes the RUG3 ideal for remote solar powered applications. The ALERT community uses a compact serial protocol that is now installed into the RUG3, lending the RUG3 to ALERT applications. This application presents a simple way to report a tank/reservoir level and tipper bucket rain level. Other measurements could easily be added using spare RUG3 I/O. The program is 'TedR3DAlert!.rgd'. (Submitted by Ted Roper of Roper Associates.) The following figure presents a typical wiring diagram:

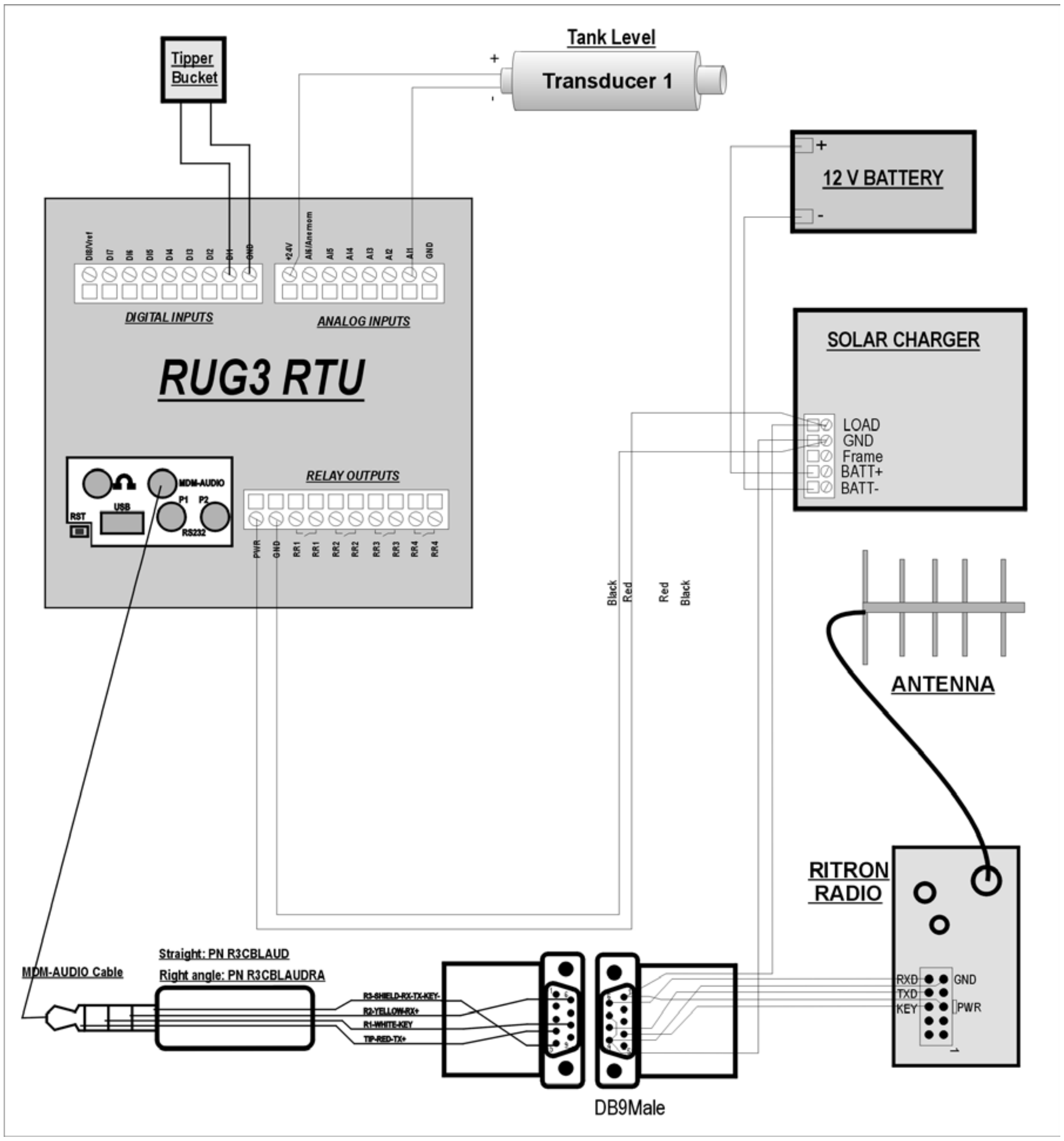
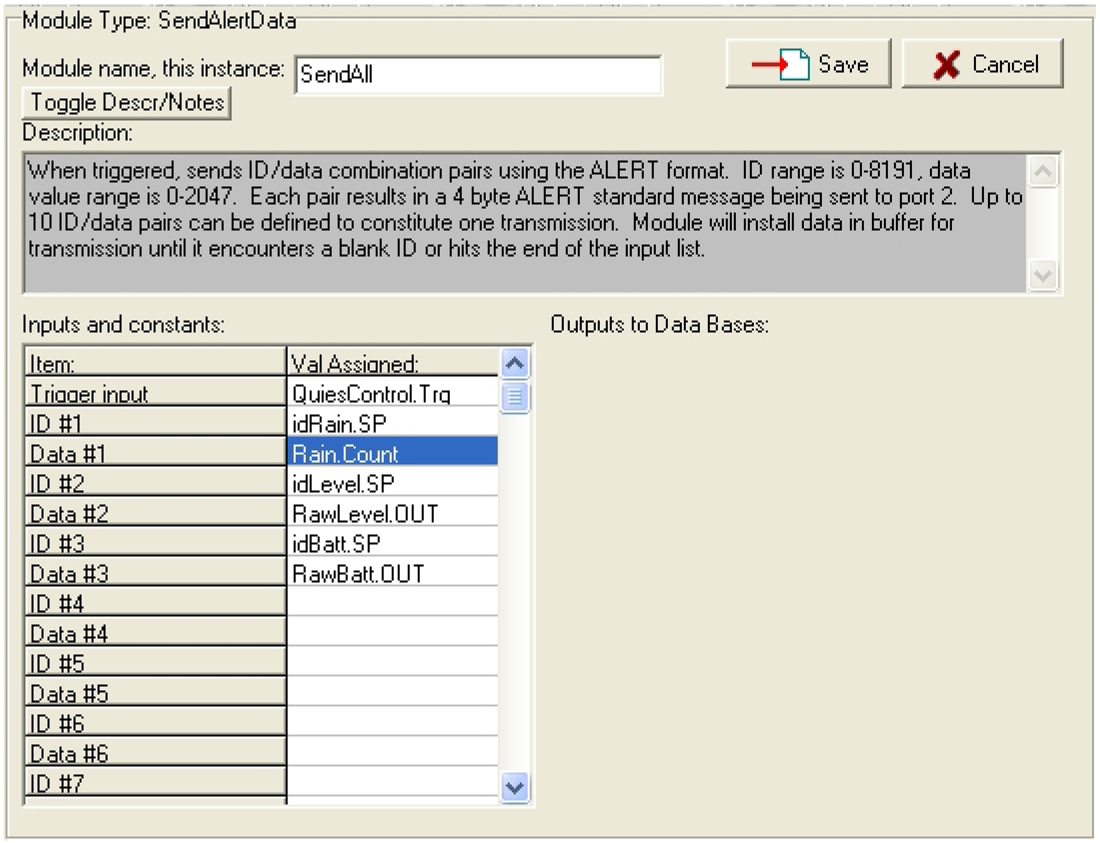


Figure 72 Alert Radio Remote Wiring Diagram

The key elements of this application rest in the method of getting the data transmitted in the ALERT format.

SendAlertData Module

Instead of the data residing in a transmit array as in the applications above, for the ALERT protocol we have the **SendAlertData** module whose job is to read the user's data and prepare the transmitted message when triggered. Typical use would be as shown below:



Module name, this instance: SendAll

Description:

When triggered, sends ID/data combination pairs using the ALERT format. ID range is 0-8191, data value range is 0-2047. Each pair results in a 4 byte ALERT standard message being sent to port 2. Up to 10 ID/data pairs can be defined to constitute one transmission. Module will install data in buffer for transmission until it encounters a blank ID or hits the end of the input list.

Inputs and constants:

Item:	Val Assigned:
Trigger input	QuiesControl.Trg
ID #1	idRain.SP
Data #1	Rain.Count
ID #2	idLevel.SP
Data #2	RawLevel.OUT
ID #3	idBatt.SP
Data #3	RawBatt.OUT
ID #4	
Data #4	
ID #5	
Data #5	
ID #6	
Data #6	
ID #7	

Figure 73 SendAlertData Module Setup

Note that the trigger is taken from a quiescent controller that handles transmission timing. Following that trigger in the figure above, each measurement is installed as an ID number followed by the actual measurement. The ID numbers are taken from setpoints so the installer can set them at installation time. Up to ten measurements can be made a part of a transmission; and any number of SendAlertData modules can be used to give different combinations of measurements.

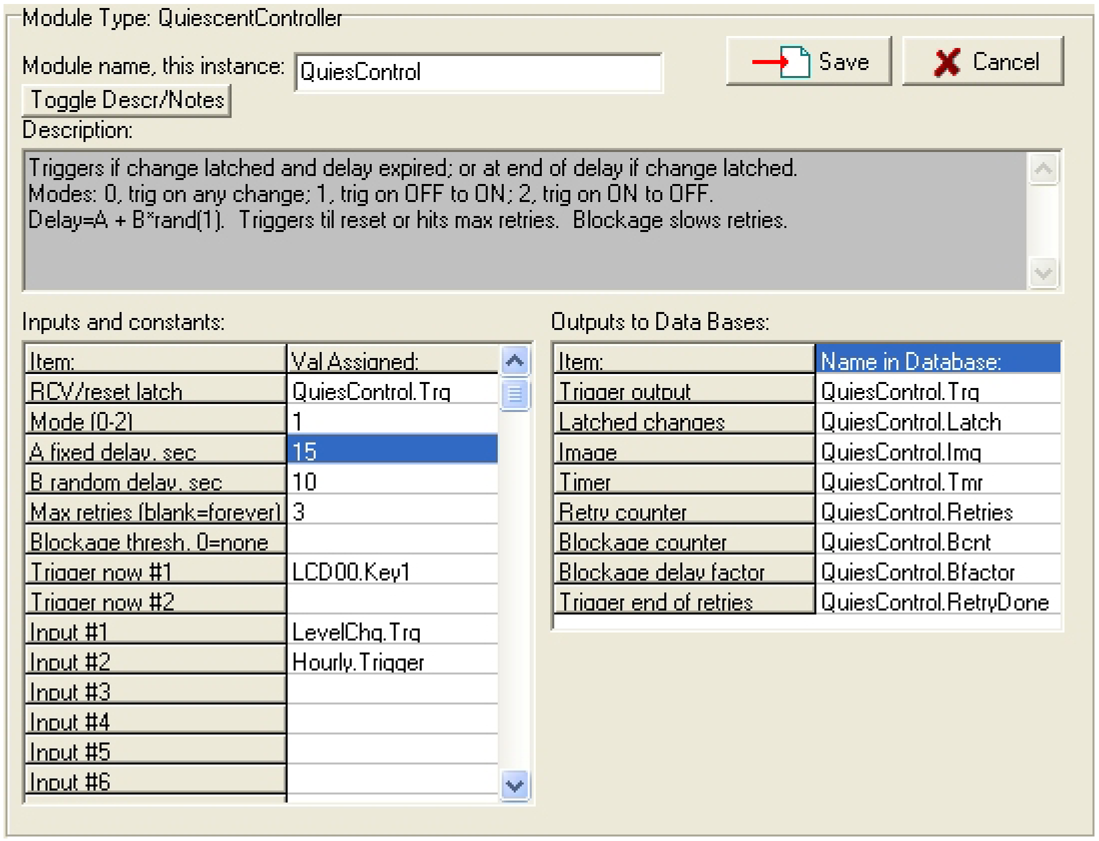
ComSetup Module

The **ComSetup** module is slightly different from that of the application above in that the ALERT protocol is chosen (Mode=8).

QuiescentController

Timing of transmissions is controlled by a **QuiescentController**, whose setup is illustrated below:

Sample Applications



Module name, this instance: QuiesControl

Description:

Triggers if change latched and delay expired; or at end of delay if change latched.
Modes: 0, trig on any change; 1, trig on OFF to ON; 2, trig on ON to OFF.
Delay=A + B*rand(1). Triggers til reset or hits max retries. Blockage slows retries.

Inputs and constants:

Item:	Val Assigned:
RCV/reset latch	QuiesControl.Trg
Mode (0-2)	1
A fixed delay, sec	15
B random delay, sec	10
Max retries (blank=forever)	3
Blockage thresh, 0=none	
Trigger now #1	LCD00.Key1
Trigger now #2	
Input #1	LevelChg.Trg
Input #2	Hourly.Trigger
Input #3	
Input #4	
Input #5	
Input #6	

Outputs to Data Bases:

Item:	Name in Database:
Trigger output	QuiesControl.Trg
Latched changes	QuiesControl.Latch
Image	QuiesControl.Img
Timer	QuiesControl.Tmr
Retry counter	QuiesControl.Retries
Blockage counter	QuiesControl.Bcnt
Blockage delay factor	QuiesControl.Bfactor
Trigger end of retries	QuiesControl.RetryDone

Figure 74 QuiescentController in ALERT Application

The QuiescentController monitors up to 16 status inputs to see if one or more has changed. If so, it initiates a transmission by asserting its trigger output. Once it does so, it starts a timer that will timeout after a delay that is the sum of a fixed delay and a random delay, taken from inputs A and B respectively. If a change is detected during that delay, the module will arm itself for issuing a new trigger at the end of that delay but will not assert the trigger until after the delay expires. Once the delay is done, if there was no change detected before it finished, then the module will issue a trigger immediately upon detecting the next change. In this application, the ALERT protocol does not provide for a reply from the destination station, so in this case, the trigger output (QuiesControl.Trg) is routed back to the RCV/reset latch input to fake the controller into thinking it has had a reply to its transmission so it will not cycle through its usual series of retries. You can see that the controller is monitoring for a change in tank/reservoir level, and the hourly trigger. The rainfall tipper is not monitored by this module because it is sent independently by itself to minimize transmission time.

AP Note #7: CAPTURING LOGGED DATA WITH A PDA

You can use a PDA (Personal Digital Assistant) such as the Dell Axim30, which we used in our testing, to act as a hand held terminal and to capture logged data dumped from the RUG3. Your PDA must have a serial port with a serial cable terminated in a DB9 connector; and serial communication software. For the Dell Axim30, we bought a serial cable, PN 18000S, from www.thesupplynet.com. A cable may be available from Dell also. We used communication software called vxHpc that we purchased from Cambridge Computer Corporation at www.cam.com. Before the cable will work, you must be sure to jumper pins 1,4 and 6 together, and separately, jumper pins 7 and 8 in the RUG3 serial to modular adapter as shown in the drawing below. We can supply the adapter with the proper jumpers in place.

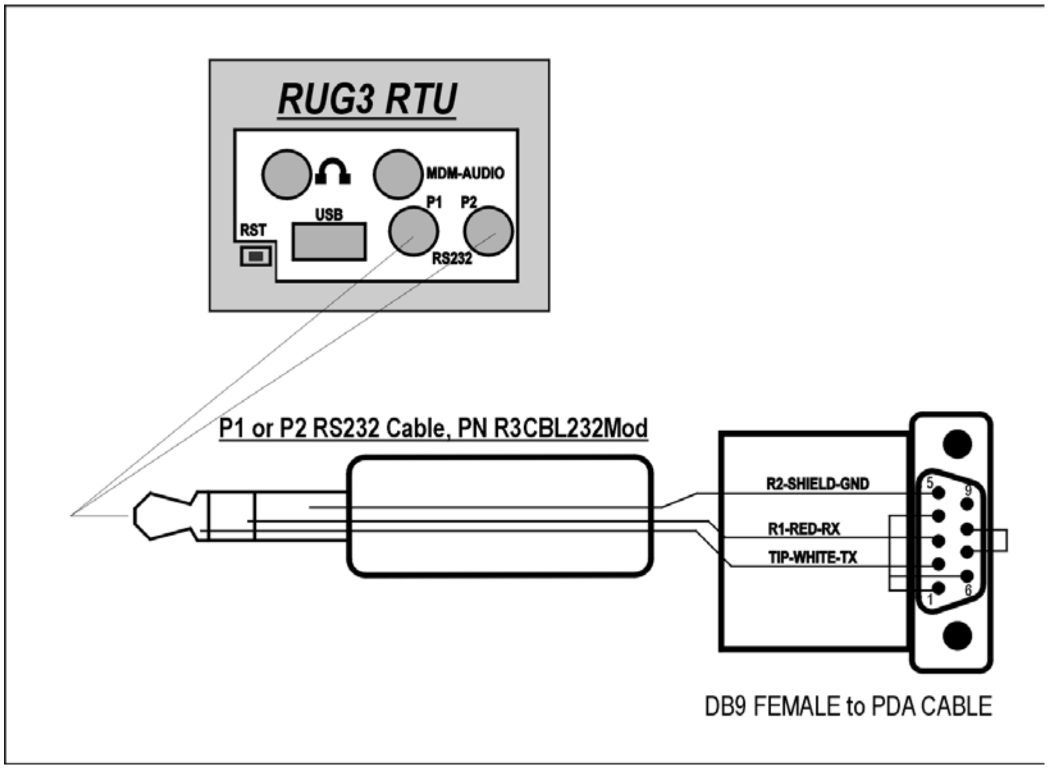


Figure 75 Serial Adapter to Dell Axim30

Install the communications software by following the instructions included with the software or included with your PDA. Then connect the adapter in the figure above to the PDA's serial cable DB9 male end and plug the other end into the PDA. Finally, launch the communications software and follow the procedure below to establish correct communication parameters. This procedure assumes the Dell Axim30 and the vxHpc communications software.

1) Tap 'Start', 'Programs', 'Communications', 'vxHpc' to launch vxHpc.
2) Tap 'Add New Session'
3) Name the session as you wish.
4) Under 'Select Comm. Interface', select 'Direct Connect-Async.
5) Tap the 'Async. Comm.' Tab
6) Under 'Select a Port, select 'Serial Cable on COM1'
7) Tap 'Configure' and select 9600 baud, 8 data bits, None parity, 1 stop bit, and None flow control, then tap 'OK'

8) Tap ‘OK’ to end Session Properties
9) Tap the name of your new session. Communications should commence immediately. To confirm communications, press the reset button on the RUG3. The normal “OS Test…OS Test passed” messages should appear on your PDA’s screen.

To capture a file from the RUG3 once the communications software is confirmed running and communicating with the RUG3, follow this procedure.

1) Tap ‘File’, then ‘Capture Text’.
2) Enter a file name in the ‘Filename’ field.
3) Select a folder and location. Then tap ‘OK’. We tested with a Secure Digital cartridge as the location so it could be later removed and inserted into a PC. It also has greater capacity than the onboard memory.
4) Trigger the RUG3 to commence dumping of its file.
5) When the file is finished, tap ‘File’, then ‘Capture Text’ and confirm ‘YES’ to terminate capturing text.
6) At this point, your captured file should be present on the PDA.

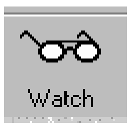
CHAPTER 9…TROUBLESHOOTING

Introduction

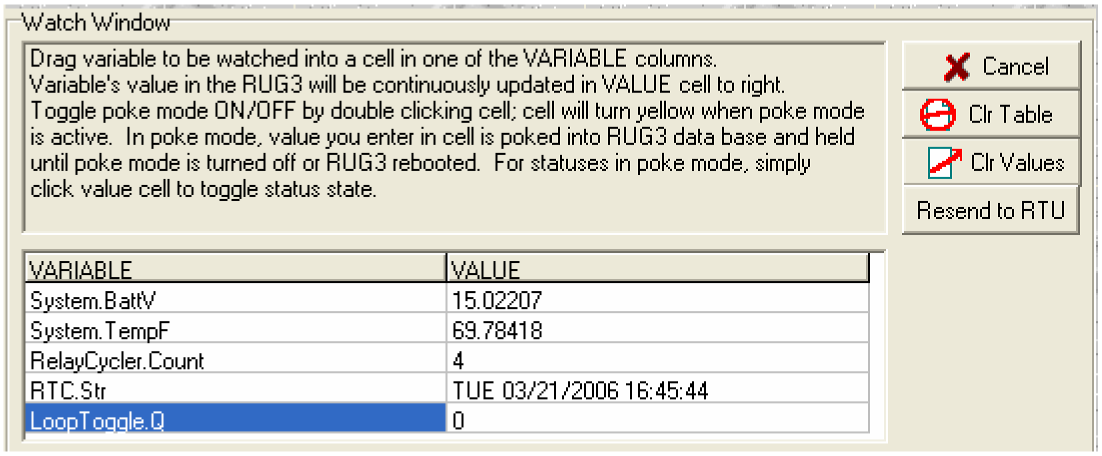
No matter how well you design your project, its software, and the interfaces to external equipment, these applications are often complex; and things can go wrong. This chapter attempts to provide assistance in the form of suggested troubleshooting techniques based on problems users have encountered in the first few years of RUG3 use. We will first address the use of the RUG3's primary troubleshooting aid, the watch window. Following that is a listing of symptoms and possible strategies for isolating and fixing problems.

Watch Window

After you have compiled and loaded your program into the RUG3, you can use the watch window to peer inside the databases to observe program internal operation. To use the watch window, be sure your program is running on the RUG3, then simply click on the watch window button to open the window. The button is on the upper tool bar and is shown below.



The watch window should become visible. Then, simply drag variables you wish to observe from the databases and drop them into the cells in the watch window's **Variable** column. Once you do that, R3SETUP will begin polling the RUG3 once per second for values of your variables, and will place them to the right of your variable names in the **Value** column. The figure below presents a typical watch window observing operation of a running program.



Drag variable to be watched into a cell in one of the VARIABLE columns.
Variable's value in the RUG3 will be continuously updated in VALUE cell to right.
Toggle poke mode ON/OFF by double clicking cell; cell will turn yellow when poke mode is active. In poke mode, value you enter in cell is poked into RUG3 data base and held until poke mode is turned off or RUG3 rebooted. For statuses in poke mode, simply click value cell to toggle status state.

VARIABLE	VALUE
System.BattV	15.02207
System.TempF	69.78418
RelayCycler.Count	4
RTC.Str	TUE 03/21/2006 16:45:44
LoopToggle.Q	0

Figure 76 Watch Window

Notice that the watch window enables you to observe statuses, integers, floating point values and strings. The watch window knows from your compiled program the type of variable you have requested in each cell, and formats the response it receives from the RUG3 accordingly. Since the watch window relies on data base addresses to capture data, it is imperative that the program you are running, and the one resident in R3SETUP be identical. If you have made even a minor change in your program, you should recompile and reload the program or the watch window could give you erroneous values. In addition, the watch window data requests are given high priority in the RUG3, which can interfere with other serial communications.

When you cancel the watch window, it simply stops polling and becomes invisible. If you later invoke the watch window, it will reappear with the same variables present as previously. Clicking the **Clr Values** button will erase all values without affecting the variable choices. The variable values should then rewrite within one second. If you click the **Clr Table** button, the watch window will be entirely erased. If you load a new program, the entire watch window will be cleared inside the RUG3 but can be re-established by hitting the **Resend to RTU** button.

Trouble Diagnosis

Basic Operation and Program Loading Problems

Unit Appears Dead

- Power up the unit. Check DC power at the PWR and GND connections. Voltage should be at least 12.0 VDC.
- If power is OK, remove power, connect computer or laptop running R3SETUP and click the terminal button. Then power up the unit. Unit should indicate that it is testing its operating system (OS). If it indicates that it is waiting for operating system load, it has encountered an operating system error, and you must reload the operating system.
- If unit does not give any message to the serial port, press the reset button to the left of the RUG3's program port. The unit should indicate that it is testing its OS. If it does not, check your cable and reboot your computer, then press the RUG3's reset button again. If you still get no indication of activity on the serial port, return the unit to RUGID for repair.

Unit Will Not Respond to Program Load

- While power is applied to the unit, press the recessed reset button next to the programming port. The unit should respond with its welcome message. If you get no message, remove power from unit, then, immediately after you reapply power, repetitively punch the recessed reset button for 5 seconds. If you still get no message, return the unit to RUGID for repair.

Program Appears to Load But Will Not Run

- Operating system in RUG3 may be incompatible with R3SETUP revision. Try reloading the operating system into the RUG3, then reload the program. If after loading, program will still not run, try loading and running one of the RUGID supplied programs such as 'R3Burnin'. If RUGID program runs, there may be an error in your project file. Try deleting sections of the program to isolate the problem. If RUGID program does not run, your RUG3 hardware may be defective.

Realtime Clock Will Not Keep Time

- The realtime clock/calendar should be accurate to two minutes per month. If it is outside that range, or is presenting random values, return your unit to RUGID for repair.

Unit Loses Time or Data During Power Outages

- The onboard lithium battery along with other components should retain the realtime clock/calendar and all RAM contents for outages of up to two years cumulative time. Replace the lithium battery with type CR2032. If the problem persists, return the unit to RUGID for repair.

Unit Loses Operating System

- Unit operating system and user configuration files are held in flash memory, which does not require battery power. If the unit experiences a power transient during loading of either the operating system, or configuration file, then the operating system could be corrupted. If the loss of operating system is unrelated to such loading events, then the flash memory could be defective.

Unit Stalls and Must Be Powered Down and Re-powered to Run

- Stalling is usually caused by local transients from switched inductive loads such as starters, relays, motors and solenoids. The RUG3's isolation and voltage clamping should block most transients that might try to come in through I/O. However, DC connections to the RUG3 provide a path for transients to get into the unit. Check that local inductive loads have snubbers installed, and that wiring to any switched AC loads does not run in parallel with any DC connections to the RUG3.
- The onboard watchdog timer should restart the program if the program hangs up for more than two seconds. If it is doing this repetitively, then there may be a problem with your application program or with the loaded OS. Try installing an earlier OS and then installing your program.

Unit Resets Occasionally

- Resetting is usually caused by local transients from switched inductive loads such as starters, relays, motors and solenoids. The RUG3's isolation and voltage clamping should block most transients that might try to come in through I/O. However, DC connections to the RUG3 provide a path for transients to get into the unit. Check that local inductive loads have snubbers installed, and that wiring to any switched AC loads does not run in parallel with any DC connections to the RUG3.
- Unit could be spending too much time on some software task. Try deleting recently added modules to solve problem.

LCD Display and Keyboard Problems

Contrast is Too Dark or Too Light

- LCD contrast is controlled by the RUG3's CPU and is affected by component values and voltages on the board. If you can read the LCD, you can set the contrast from the LCD unit's keyboard. Otherwise, you must use the terminal window in R3SETUP. To set LCD contrast, press the [-, minus] key and then key [7] to initiate contrast setting. You will be prompted to enter a new contrast setting in the range of 0 to 255. Numbers in the range of 120 to 140 usually work. If you cannot obtain acceptable contrast, return the unit to RUGID for repair.

Contrast Changes When Nearby Equipment Powers On/Off

- Nearby equipment could be causing surges or sags in main power to the RUG3, causing regulators to be unable to maintain clean power to the LCD controller. Check that main power to the RUG3 is 12 VDC.

Display Has a Line of Dark Blocks on the Top Line

- Remove power, connect computer or laptop running R3SETUP and click the terminal button. Then power up the unit. Unit should indicate that it is testing its operating system (OS). If it indicates that it is waiting for operating system load, it has encountered an operating system error, and you must reload the operating system.
- Check to be sure that your program has a display defined for the LCD port, port 0.

Keystrokes on Keyboard Get No Response

- Check to be sure that the keyboard ribbon cable is inserted all the way into the keyboard connector on the RUG3 board.
- Check to be sure that your program has a display defined for the LCD port, port 0.

I/O Problems

Analog Input is Inaccurate (4-20 ma)

- Check that the **AnalogInput** module you are using for the channel in question is set for the type of instrument you are using (4-20 ma. or 0-5V), and that its offset and span settings are proper for the measurement you are making. If they appear correct, disconnect the transducer from the channel in question, apply power, and with an ohmmeter, measure the resistance from the analog input channel in question to the GND terminal on the analog input screw header. If you are using 4-20 ma., it should read within 2 ohms of 221 ohms. If the channel is set for 0-5V operation, the input resistance should be about 20K ohms. If the resistance is out of this range, or, if none of these conditions exists, return the board to RUGID for repair.
- Check that another analog input on this same board is not out of range of 0.0 to 23.0 ma., or 0-5VDC An input voltage out of range can affect other channels.
- If the low pass filter time constant is long, the measurement will take a long time to settle to an accurate value.

Battery Voltage or Temperature Measurement is Inaccurate

- Battery voltage and temperature are calibrated at the factory during unit testing along with the rest of the analog inputs. Make sure that other analog inputs are not applying voltages outside the range of 0-5 V at the RUG3's analog input terminals. If they are in range, the unit should be returned to RUGID for retesting.
- Board temperature measurement measures the temperature inside the case. It will normally read higher than ambient temperature due to heating inside the case. If it appears out of range, return to RUGID for repair.

Relay Output Will Not Turn On

- Make sure that the power supply voltage applied to the RUG3 is at least 10.5 VDC. Relays may not pull in below that voltage.
- The relay output must be driven by a digital output module, and that module must be driven by a variable from the status database. If that signal has not been routed to the relay output module, or if it is not changing state as expected, then the relay output will appear to be inoperative. If the module is driven correctly, then return the unit to RUGID for repair.

DC Digital Input Does Not Correctly Sense Input

- Be sure that you are using a **Digital Input DC** type module to sense the digital input. The RUG3's digital inputs are designed to operate from dry contacts or logic in the range of 0-4 VDC to 0-12 VDC, with a threshold of 1.5 volts. When your signal is turned on, be sure that the voltage across the digital input is below 1.5 VDC.

Communications Problems

Modem Keys Continuously

- Make sure that you have a **ComSetup** module in your project referencing the modem port, port 2. Also, make sure that the **ComSetup** module has actually been triggered before the transmission.

Modem Will Not Key Radio

- Make sure that you have a **ComSetup** module in your project referencing the modem port, port 2. Also, make sure that the **ComSetup** module has actually been triggered before the transmission.
- Make sure that the **Poll** module is being triggered.
- Make sure that the program has a transmit array for the port and destination referenced in the poll module.
- Make sure that the transmit array referenced by the poll has at least the number of words defined as are requested in the poll.

Modem Receives But Will Not Transmit Reply

- Make sure that the transmit array referenced by the poll exists and has at least the number of words defined as are requested in the poll.

Modem Keys But Will Not Transmit

- Make sure that the transmit array referenced by the poll exists and has at least the number of words defined as are requested in the poll, and that it is defined for the correct port and address.
- Use stereo headphones to listen to the transmission. Connect them to the headphone jack to the left of the modem audio connection on the RUG3's serial connection panel. You should hear the transmission in one ear and any reception in the other ear. If no signal is present, or the signal is distorted, return the unit to RUGID for repair. If the signal is low in amplitude, adjust transmit amplitude in the ComSetup module to increase signal level.

Modem Transmits But Other Receiver Will Not Accept Message

- Use stereo headphones to listen to the transmission. Connect them to the headphone jack to the left of the modem audio connection on the RUG3's serial connection panel. You should hear the transmission in one ear and any reception in the other ear. If no signal is present, or the signal is distorted, return the unit to RUGID for repair. If the signal is low in amplitude, adjust transmit amplitude in the ComSetup module to increase signal level.
- Make sure that both the transmitting unit and the destination unit have matching **ComSetup** input properties and that their addresses are correct.
- Make sure that the number of transmitted words does not exceed the array size of the destination station.
- Make sure that the **ComSetup** transmit delay is long enough. For phone lines, use a delay of at least 1.5 seconds. For radios, use 1.5 seconds. Shorter times may work, but be conservative.

Modem Receiver Will Not Accept Message

- Make sure that the **ComSetup** module has been triggered to install channel parameters before the transmission comes in.

- Make sure that the sending unit's **ComSetup** transmit delay is long enough. For phone lines, use a delay of at least 1.5 seconds. For radios, use 1.5 seconds. Shorter times may work, but be conservative.
- Use stereo headphones to listen to the transmission. Connect them to the headphone jack to the left of the modem audio connection on the RUG3's serial connection panel. You should hear the transmission in one ear and any reception in the other ear. If no signal is present, or the signal is distorted, problem could be in the channel or in the transmitter at the other end.

RS232 Port Will Not Respond to Modbus Messages

- Verify that the **ComSetup** module has been installed for the port you are using, and that it is configured for Modbus use (Modbus slave mode, RS232, correct baud, parity, etc.). Make sure that the address installed in **ComSetup** matches the slave address your Modbus master is polling.
- Some PC's require that control signals be defaulted true for operation. You can assure that they are true by jumpering pins 1,4 and 6 together, and by jumpering pins 7 and 8 together on the DB9 connector.
- Make sure that the RUG3 transmit and receive arrays are configured for the board and port you are using. If you are using the modem's RS232 port, you should be using port 2
- Make sure that you are using source and destination addresses of 255 in the array setups. Modbus masters have no address.
- If you are sure that your setup is correct, use the MODBUS.EXE program to test the setup. Contact RUGID for assistance.

Other Unit Only Receives Part of Transmitted Data

- If this unit is polling, make sure the poll module specifies the correct beginning register and enough registers to be transferred to the destination.
- If this unit is responding to another poll, make sure that the polling unit is requesting the correct starting register and number of registers to be transferred.

This Unit Only Receives Part of Transmitted Data

- If this unit is polling, make sure the poll module specifies the correct beginning register and enough registers to be transferred from the destination.
- If this unit is responding to another poll, make sure that the polling unit's poll module is requesting the correct starting register and number of registers to be transferred.

Receive Array Has Wild Numbers

- If the unit has not yet received a message, the receive array will have undetermined values. This is normal. You must wait for a reception.
- If the unit has received a message, the received message may not have addressed the registers in question. If this unit is polling, make sure the poll module specifies the correct beginning register and enough registers to be transferred from the destination.
- If this unit is responding to another poll, make sure that the polling unit's poll module is requesting the correct starting register and number of registers to be transferred.

Unit Resets With Each Transmission

- If the unit is powered by the same source as a radio, the power source may not be able to source both the RUG3 and the radio when it transmits.

Data Logging Problems

Unit Will Not Log As Many Points As Logger Setup Specifies

- The log must hold both data samples and time tags. Each takes one word (4 bytes) of storage. The inclusion of time tags will use up some of the loggers data space. You must either log time tags less frequently or increase the size of the log.

Log Contains Lots of Wild Numbers

- If the log has been just installed, or has had its size changed, the compiler probably has moved the log so it is pointing to uninitialized memory. You must trigger the data logger's reset input to erase the log, or wait for samples to replace the erroneous data.

Module Problems

Cannot See Triggers on Watch Window

- Triggers are only on for one scan, so may be too short for watch window to capture. Try capturing the trigger with either a **FlipFlop** module or an **OrGateLatch** module, or a **Counter** module, then display that.

Flow Totalizer Records Flow When None Is Present

- Flow transducer or analog input may be indicating slight positive flow when none is present. If you are using one of the flow math modules, set the low flow dropout to a small positive value. Otherwise, run the flow value through the FlowConvert module and implement the low flow dropout there.

Pulse to Flow Module Does Not Work

- Make sure the pulse to flow module's trigger input is being triggered.
- Make sure that you have assigned a **DICount** module to the digital input that you are using to capture the flow pulses. Remember, the pulse to flow module works from pulse count per unit time, not pulse duration.

Setpoint Has Wild Numbers

- Setpoints are in uninitialized RAM, so they will assume random values until set by hand, or set by triggering their default inputs.

CHAPTER 10… WARRANTY, DIMENSIONS, SPECIFICATIONS

Warranty

RUGID COMPUTER, Inc. warrants that equipment manufactured and sold by us is free from defects in material and workmanship. Under this warranty, our obligation is limited to repairing or replacing, at our option, any equipment or parts returned, shipping prepaid and properly packed, to our plant and proving to be defective by our inspection within one year after sale to the original purchaser. This warranty shall not apply to equipment or parts thereof which are normally consumed in operation, or to any equipment which shall have been repaired or altered in any way outside our plant, so as to, in the judgment of RUGID COMPUTER, Inc., effect its stability, accuracy, or reliability, nor which has been operated in a manner or environment exceeding its specifications, nor which has been damaged, altered, defaced, or has had its serial number removed or altered. Under no circumstances shall RUGID COMPUTER, Inc. be liable for any loss or damage, direct, incidental or consequential, arising out of the use, misuse, or inability to use, this product. The liability of RUGID COMPUTER, Inc. shall not exceed the original purchase price of this product.

Return/Repair Policy

Specific warranty provisions are stated in the warranty section above. Under no circumstances are products returnable for credit. If a product is found to have failed, it must be returned to the factory for repair or replacement. The determination of whether to repair or replace the product is made by us after a period of testing. If it cannot be brought up to full operation with full certainty, it will be replaced. When repaired under warranty, we will return the repaired product using the same freight class as it was received no charge. We will make every effort to ship within 24 hours of receipt. The determination of whether a product's repair or replacement is covered under warranty will also be made by us. We give the benefit of the doubt to the customer in this determination. However, if there is any indication of physical damage, alteration, or misapplication of the product, then the repair will not be covered by the warranty.

It is in your best interest to accurately assess and report the circumstances of a failure. One of the most difficult determinations to make is whether a product has suffered over voltage stress in the field. If it has, the product can fail at sometime after being repaired and declared operational due to failure of a component that was stressed but did not reveal itself during testing. Also, any failure related information you can give us along with the product can reduce the time it takes to find the defective components. Therefore, it could save you money.

Warranty, Dimensions, Specifications

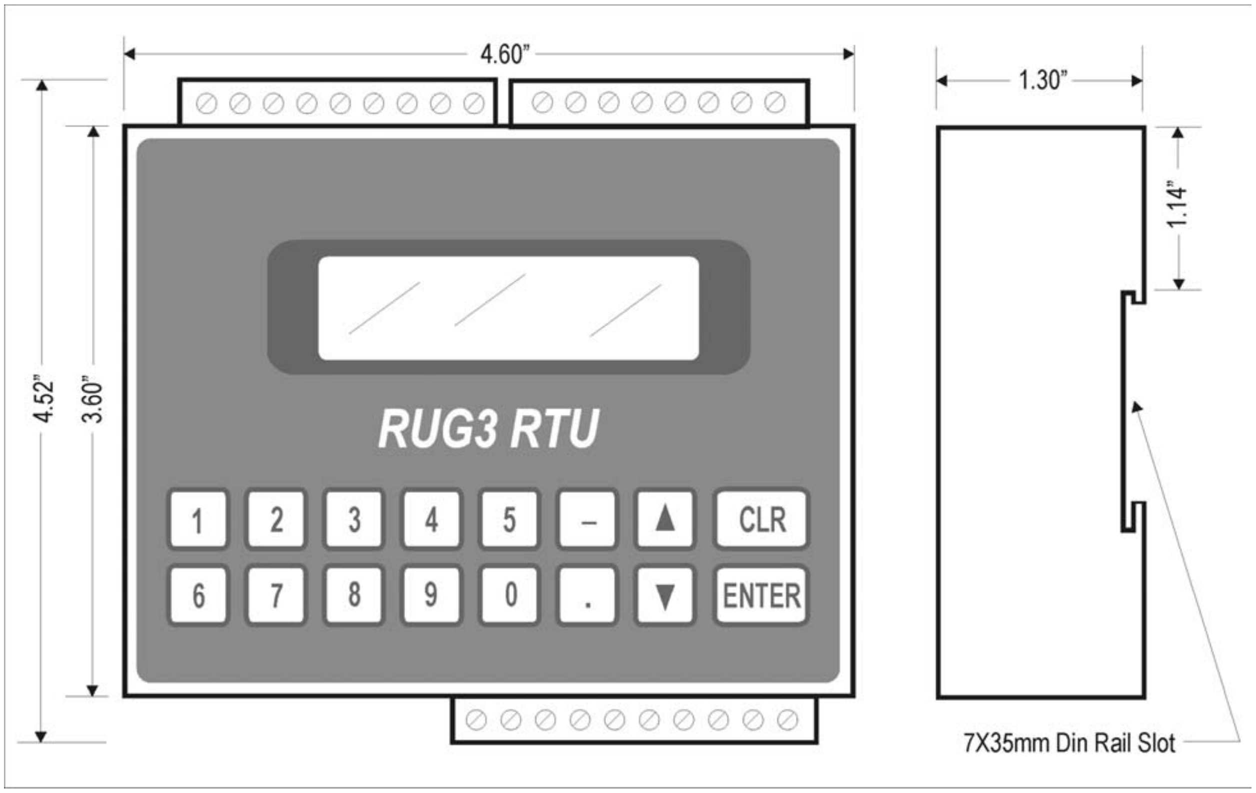


Figure 77 RUG3C/D Dimensions

Warranty, Dimensions, Specifications

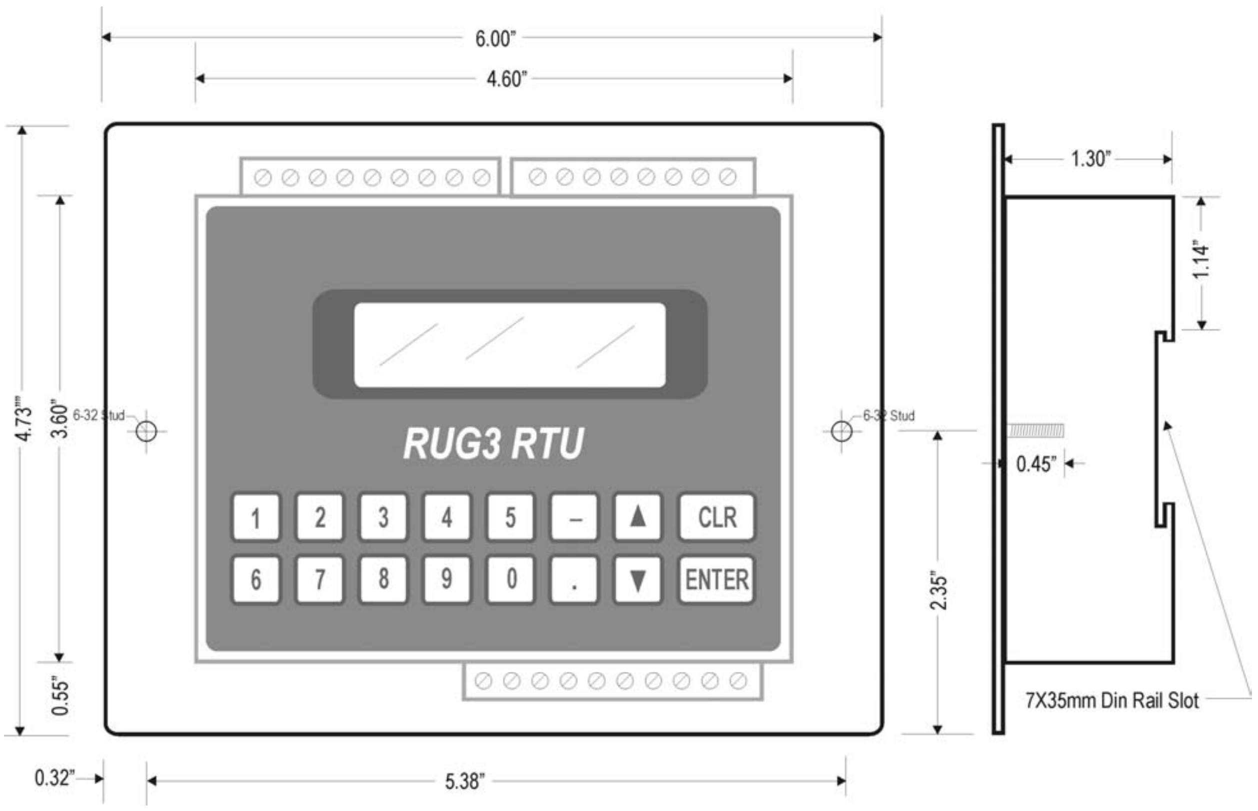


Figure 78 RUG3P Dimensions

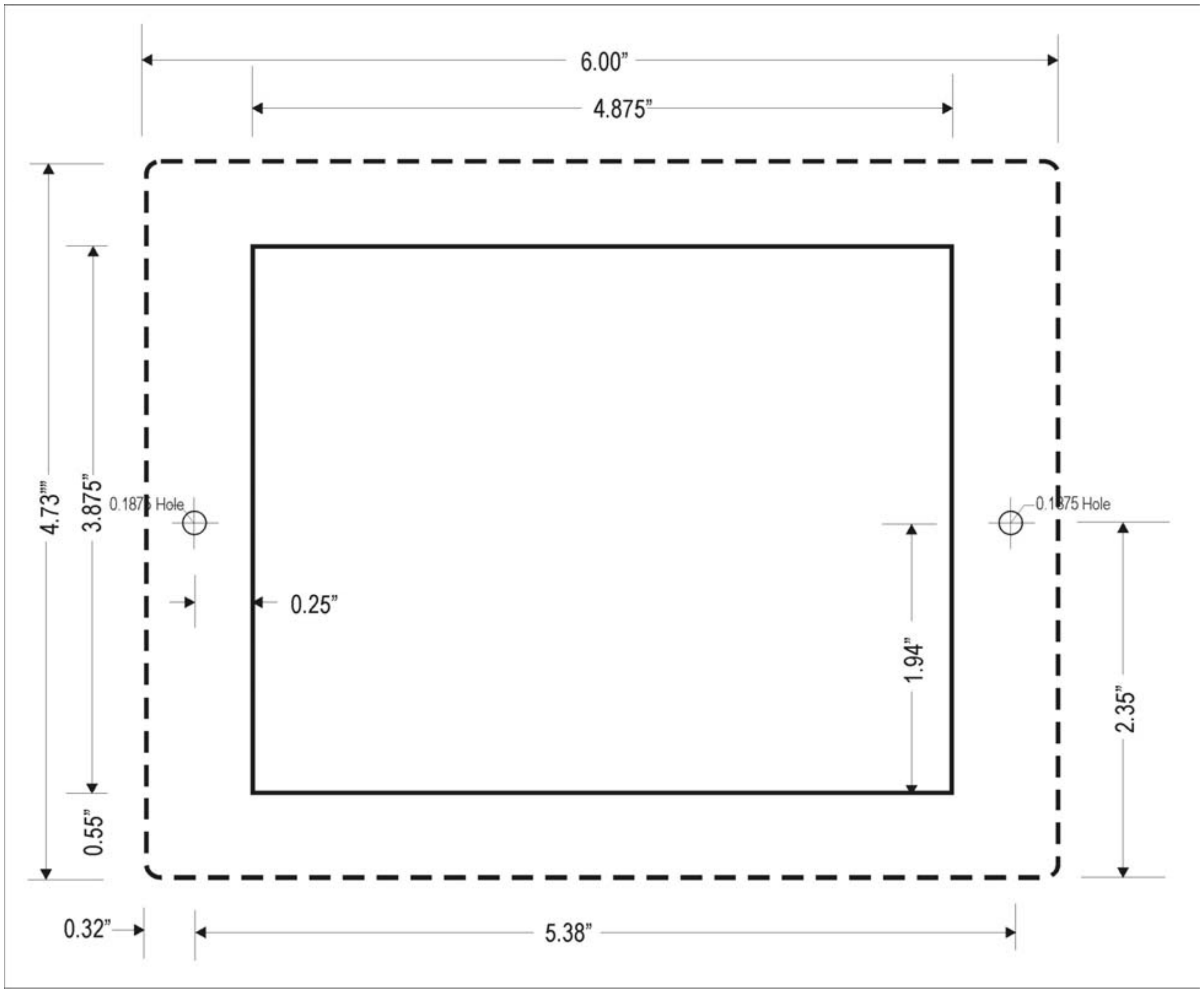


Figure 79 RUG3 Panel Mount Cutout Dimensions

RUG3 SPECIFICATION

LOGIC FAMILY
All low power CMOS
MICROCONTROLLER
16-bit MSP430, 8 Mhz, 16 bit data bus, 16 bit address bus
MEMORY
RAM-2 Kbytes battery backed low power static RAM
Program Flash-60 Kbytes
Logging Flash-2 Mbytes
Battery Backup-Onboard lithium coin cell backs up RAM & realtime clock/calendar min 2 years
DISPLAY
2 line X 16 char backlit LCD, sunlight readable, backlight switchable by software
KEYBOARD
16 key sealed tactile membrane with interrupt scanning
REALTIME CLOCK/CALENDAR
Battery backed clock/calendar 0.005% crystal accuracy
OPERATION SECURITY
Watchdog Timer-Hardware timer resets unit 0.5 sec. after interrupt fail. Cannot bedisabled.
Telemetry Watchdog-Resets rcv buffer if no character received within 1 sec.
Brownout Detector-Halts process if logic voltage falls below 2.7 V, restarts when voltage rises to 3 V
AUTOBOOTING
Auto startup on power application. Automatic OS test and user program test must pass before program will start. Retries every 60 sec.
I/O SURGE PROTECTION
All I/O is resistor isolated, meets IEEE surge protection rqmts.
ANALOG INPUTS
6 chan, 12 bit res., successive approx, Individually switchable as 4-20 ma or 0-5 v. Factory calibrated.
ANALOG OUTPUTS
2 chan optional, 12 bit resolution, optically isolated. Factory calibrated.
DIGITAL INPUTS
Status- 8 chan, dry contact compatible, self powered
Pulse Counting-all DI count 128 PPS
Pulse Duration Detecting-all can convert pulses to analog with 4ms resolution
Shaft Encoder-DI's in pairs used to decode shaft encoders

DIGITAL OUTPUTS
4 ch, 10 amp relays or 0.5 amp/48V SSR's. Pulse Duration Outputs-Relays can generate pulse width modulated or one shot signals with 4 ms res.
ANEMOMETER INPUT
AI6 connected to clipping amp, counted to derive wind speed
REFERENCE OUTPUT
5.0 VDC reference available to power potentiometers, shares pin with DI8.
INSTRUMENT POWER
Loop supply switchable to 24 VDC or battery voltage and can be switched on/off by software.
SERIAL PORTS-2 Standard RS232
One programming/gen purpose port plus one gen purpose RS232 port
MODEM-Optional
Bell 103 standard/ALERT standard
Radio Interface
4-wire audio, adj. gain, xformer isolated, optically isolated key line.
Low tones mode for splinter chan.
Phone Line Interface
4 wire audio adjustable gain, transformer isolated
Transmit power 0-4Vp-p, software adjustable in 32 steps
COMMUNICATIONS
ASCII std,
RUG9 Background CRC gen/decode, variable length messages, user defined message lengths. Can combine status, integer, float, in any message.
ALERT protocol-standard
Modbus RTU slave or master
ModbusTCP slave or master.
RUG6 protocol-standard
Eavesdrop Mode-R9 protocol, any RTU can accept data passing between any other stations
Peer to Peer- Full RTU to RTU or RTU to master or master to RTU messaging
Store and Forward- Initiating station sets path through up to 3 intermediary stations
Address Range-1 to 254, 1-65535.
POWER INTERFACE
12 VDC +/-20%, diode isolated. 2.8 ma normal operation (loop supply and relays off) to 440 ma. max.

LOOP SUPPLY
Builtin switchable regulated 24 VDC +/- 5%, 120 ma.
I/O CONNECTIONS
All I/O uses removable rising cage screw headers in banks of up to 10 each, 14 ga wire. RS232 and Modem signals use 3.5 mm jack
SOFTWARE
Storage-operating system, calibration, and all user configuration and programming stored in nonvolatile flash memory. Flash loader stored in flash protected boot block.
Security-memory integrity test on boot up, CRC gen/detect on serial ports. Program loading CRC protected.
Scanning-Built in software scans all I/O, ports, timers, realtime clock
PROGRAMMING
Modules-applications use precompiled modules resident in flash memory where programmer interconnects modules and sets properties using supplied Windows program. No procedural programming required for most applications.
LADDER LOGIC
Ladder logic is built in to the Windows configuration program to handle misc controls
DATA LOGGING
Logs floating point, integer and status samples with time tags to onboard flash eeprom. 2 Mbytes. Can dump logs to serial port as comma delimited ASCII or R9 protocol CRC-secured binary.
VARIABLES
Supports 16 bit integer, 32 bit floating point, boolean, strings.
ERROR MESSAGES
Configuration program handles all setup errors. Run time software is self protecting... no run time errors.
ENCLOSURE
16 ga. steel, blue powder coat DIN rail mountable.
Case: 4.5 X 3.5 X 1.3 in.
Panel mount flange 6.0 X 4.7 In.
TEMPERATURE RANGE
-40 to +85 deg. C logic
-20 to +60 C LCD display
DOCUMENTATION
240 page manual on CD
WARRANTY
1 year std limited warranty
REPAIR
Nominal 24 hr turnaround

Index

Index

Index

Index