RUG9/RUG5 TECHNICAL MANUAL

Release 6.32

By

RUGID COMPUTER
6305 Elizan Dr. NW
Olympia, WA 98502
(360) 866-4492
FAX (360) 866-8074
www.rugidcomputer.com

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Figures

Table of Figures

Table of Figures

List of Tables

CHAPTER 1…INTRODUCTION

Welcome to the latest in RUGID's series of small remote terminal units (RTU's). Because both the RUG9 and RUG5 units employ the same CPU and, therefore run the same software, this manual presents technical information on both units. Therefore, throughout this manual, references that apply to both units will refer to the RUG5/9. Where a topic only applies to one or the other, then the reference will state that it applies to the RUG5 or RUG9. The RUG5/9 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 RUG5/9 unit up and running within minutes of unpacking it. If you are new to the RUG5/9, 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 RUG5/9 you will need to run RUGID's R9SETUPD.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 R9SETUPD.EXE program is available for download free of charge from our web site address below. You will also need a straight through serial cable with a DB9 female connector on one end and a DB9 male connector on the other.

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 RUG5/9 running. It includes installing the support software, powering up the RUG5/9, loading configuration files to the RUG5/9, 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 RUG5/9, you may want to skip this chapter.

Installing RUG5/9 Support Software

In order to load configuration files and the RUG5/9 operating system, and to modify the RUG5/9 operating system, you will need the R9SETUPD.EXE program. This program is included at no extra charge with each RUG5/9 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 running on a PC compatible computer with a screen having at least 600 X 800 pixel resolution.
2) Insert the R9SETUP disk into your 3.5 inch floppy drive or CDROM drive, whichever applies. Double click on the SetupXXX.exe program, where XXX is the revision number such as 565. The program should self install into the directory C:\Programs\R9.
3) To start the program, go to START, PROGRAMS, R9 and double click the R9 icon.

The R9SETUP 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 RUG9

All RUG9 units can be powered from 120 VAC using a standard 12 VAC wall transformer with a 2.5 mm coaxial plug. Simply plug it into the round jack on the left side of the RUG9 card cage as shown in Figure 1. If your unit consists of more than one card cage, each card cage must be powered; the expansion cable does not carry power to any of the expansion card cages. A one amp transformer should be sufficient for most applications. However, if your application makes heavy use of loop power supplies or internal relays, you may need a heavier transformer. The maximum power draw of a card cage can be estimated from the following formula:

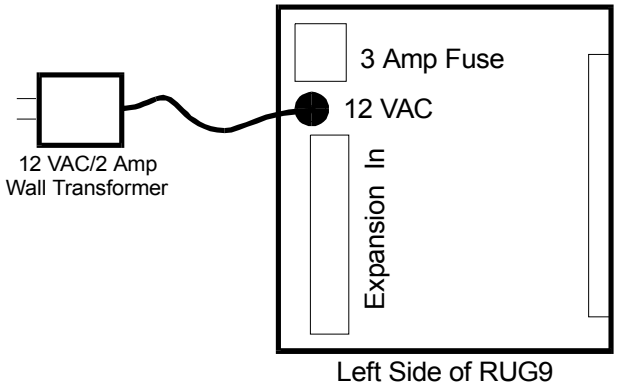


Figure 1 Powering the RUG9 from 120VAC

Current=0.14+0.28*(#relay boards)+0.28*(#loop supply boards)+0.12*(#displays)

For applications using DC power such as solar power applications, the RUG9 can either be powered by applying 12 to 15 VDC to the AC power jack as illustrated above; or you can apply power to the loop/charger board as illustrated in Figure 2. The current requirement is the same as estimated above. If you apply power to the AC jack above, be aware that the power supply ground will be approximately 1.5 volts above that present in the RUG9, since your power will be applied to the RUG9's full wave bridge, rather than directly to the RUG9's ground. This usually will be of little consequence since all external connections to the RUG9 are optically isolated from the RUG5/9's DC bus except for the battery connections on the loop/charger board.

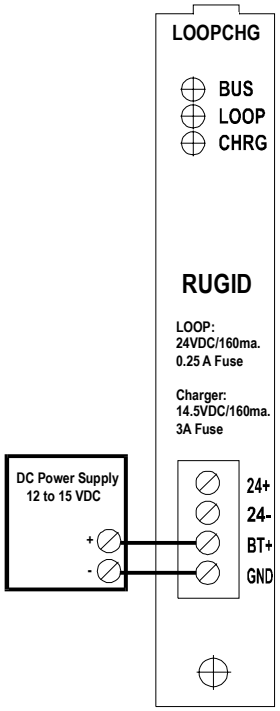


Figure 2 Powering the RUG9 from 12 VDC

Applying Power to the RUG5

Like the RUG9, all RUG5 units can be powered from 120 VAC using a standard 12 VAC wall transformer with a 2.5 mm coaxial plug. Simply plug it into the round jack on the rear of the RUG5 card cage as shown in Figure 3. The total power draw of the unit can be estimated from the following formula:

Current=0.14+0.28*(# of relay boards)+0.28*(#loop supplies)

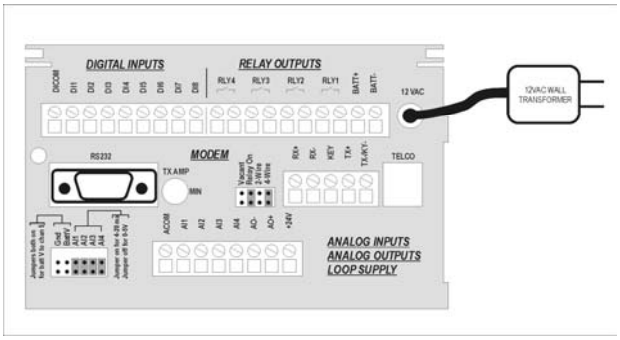


Figure 3 Powering the RUG5 from 120VAC

For applications using DC power such as solar power applications, the RUG5 can either be powered by applying 12 to 15 VDC to the AC power jack as illustrated above; or you can apply power to the battery terminals as illustrated in Figure 4. The current requirement is the same as estimated above. If you apply DC power to the AC jack above, be aware that the power supply ground will be approximately 1.5 volts above that present in the RUG5, since your power will be applied to the RUG5's full wave bridge, rather than directly to the RUG5's ground. This usually will be of little consequence since all external connections to the RUG5 are optically isolated from the RUG5's DC bus except for the battery connections on the loop/charger board.

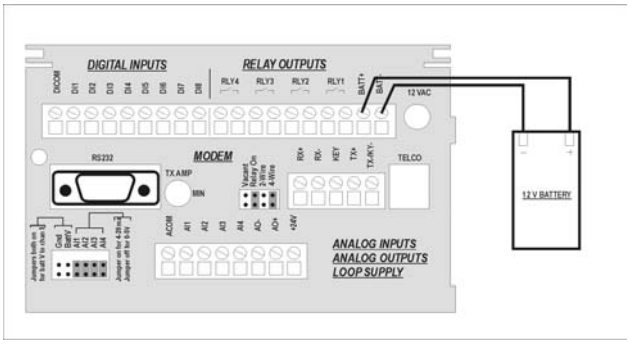


Figure 4 Powering the RUG5 from 12 VDC

Preparing to Communicate with the RUG5/9

The figure below illustrates how you connect a PC to the RUG5/9. With this connection, you can load configuration files, load the RUG5/9's operating system, and send screens of information from the RUG5/9 to the PC for observation by an operator.

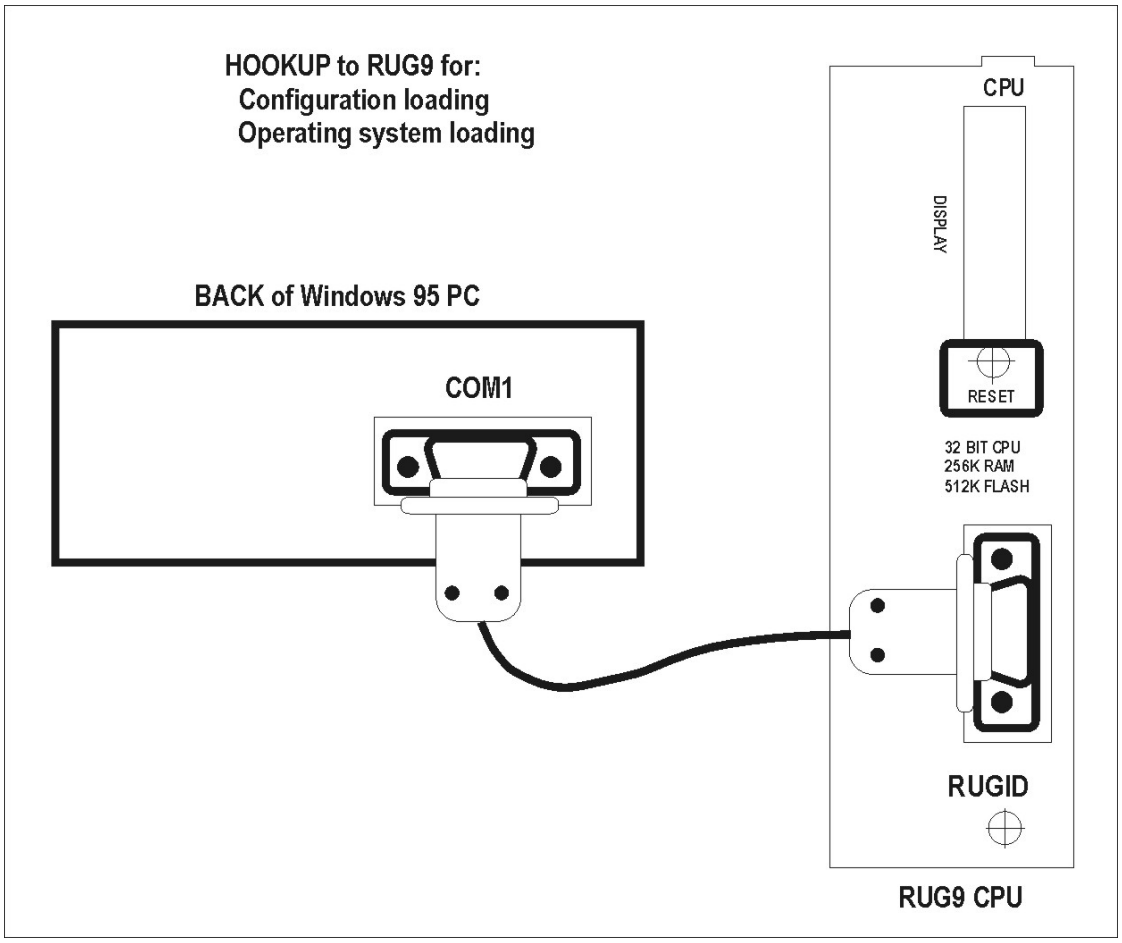


Figure 3 Serial Port Hookup to RUG5/9 CPU

RUG5/9 BASIC OPERATION

When you power up the RUG5/9, 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 port on the CPU board. This should rarely happen, but if it does, you must reload the operating system using the R9SETUPD program. If the RUG5/9 is not running its user program or has declared that the OS is faulty, it will retry the initial startup tests every 60 seconds. If it finds the OS is OK, it will start the OS. If the OS is OK it will test the user program. If it finds that the program is OK, it will then start the program. Therefore, the unit should not remain in a non-executing state for more than 60 seconds unless it has detected a failure in the OS or user's program, even if you have commanded it to stop the program.

If it finds an error in the user configuration file (user program), or if no file is present, or the unit was idle before it was last powered off, then the unit will present the following display on both the CPU's RS232 port and the unit's 320X240 LCD, if present:

```
*** WELCOME TO RUGID MONITOR ***
Revision Version 5.65  2/26/03
Date/Time Fri 02/24/03 16:17:21

Installed program:
R9Diagnostics  02/26/03 12:13:16 PM

1 Start program
2 Set realtime clock/calendar
3 Change LCD contrast
4 Toggle LCD normal/reverse
5 Read memory location
6 Clear all memory
Enter choice
```

This menu header indicates the revision date of the installed operating system and user configuration file. If no user configuration file is present, then the installed program name would not be shown, and the "1 Start program" prompt would be blank. If this is the case, then a new configuration file must be loaded from the R9SETUPD program. Here are the options from the above screen.

1) Run program…starts the user's program
2) 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.
3) Change LCD contrast. You will be shown the present LCD contrast value (0 to 511). Enter a new value to adjust LCD contrast. Low numbers give light contrast; high numbers give darker contrast. For the 320X240 LCD, common values are in the range of 80 to 180. A value of 200 usually gives good contrast. For the RUG5's 2X16 character LCD, a value of 20-30 usually gives good contrast
4) Toggle LCD normal/reverse. Each time you hit this key the 320X240 LCD will toggle from dark characters on light background to light characters on dark background and vice versa. This will have no effect on the RUG5's 2X16 LCD.
5) Read memory location…this is used for keyboard testing and is otherwise reserved for factory use.
6) Clear all memory…zeroes all RAM used by the user's program

THE RUG5/9 MENU SYSTEM

When the RUG5/9 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 RUG5/9. This menu also is presented whenever you press the minus [-] key on the keyboard.

System menu for serial ports and 320X240 LCD:

```
*** SYSTEM MENU ***

1 Displays menu
2 Setpoints menu
3 Log on for setpoint access
4 Log off setpoint access
5 Set clock/calendar
6 Tables menu
7 Adj LCD contrast, now=200(132)

Choose option…
```

For the RUG5 2 X 16 LCD:

```
1DSP,2SP,3LOGON
5CLK,6TBL,7LCD
```

If you have already logged on and the logon timer has not timed out, 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, clock values and tables will take effect immediately.

ACCESSING DISPLAYS

When you press key [1] from the system menu above, the RUG5/9 will present a display list of this form from which you may select a display to be presented. In the case of the 2 X 16 LCD, 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 Level trend
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 R9SETUPD. 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 RUG5/9. 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 RUG5/9 will immediately present the display you selected. Thereafter, each time you hit the [ENTER] key, the RUG5/9 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 RUG5/9 will return you to the system summary menu above. For the RUG5's 2X16 LCD, use the up and down arrow keys to scroll up and down the lines of any display.

Some displays such as trends and event logs involve multiple pages presenting successively older data. For those displays, hitting the [up arrow] and [down arrow] keys take you forward or backward in time, respectively.

ACCESSING SETPOINTS USING SERIAL PORTS or the 320X240 LCD

Hitting key [2] from the system menu will cause the RUG5/9 to present the first page of setpoints. The setpoints are arranged alphabetically by R9SETUPD before loading into the RUG5/9. 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 RUG5/9 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:

Hitting key [2] from the system menu will cause the RUG5 to present the first setpoint. Setpoints are arranged alphabetically by R9SETUP before loading into the RUG5. 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 enabled in the configuration file, hitting key [3] from the system menu will cause the RUG5/9 to prompt you for your access code. This code is set using the SysSetup1 module in R9SETUPD 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 RUG5/9 will allow you to alter setpoints. If you are not logged on, the RUG5/9 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 **SysSetup1** module.

LOGGING OFF SETPOINT ACCESS

Hitting key [4] from the system menu will cause the RUG5/9 to disable setpoint access until you log on again.

SETTING 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 first entering a value.

OBSERVING/CHANGING TABLE ENTRIES

Hitting key [6] from the system menu will cause the unit to present a list of tables used by the program, such as that shown below. If you are using the 2 X 16 LCD, only two lines at a time will be presented:

```
  *** TABLE LIST ***

0 ComStats
1 Cfail
2 RcvTime
3 PollTable

Choose table…
```

The example above indicates that the program uses four tables. To access the table named "ComStats", you would hit key [0]. The unit would then present the table's first column, which might look like this:

```
TABLE: PollTable  Col: 1

0 StaName               =Master
1 PollAddr              =123
2 PollMsg               =Master
3 RcvMessage            =Received from master
4 TxRegs                =5
5 RxRegs                =10
6 OnLineFlag            =0
7 FailCount             =0
8 FailFlag              =0
```

For the 20 X 40 LCD:
Hitting the [UP ARROW] key will take you to the next higher column of the table; hitting the [DOWN ARROW] key will take you to the next lower column. Hitting keys 0 through 9 will allow you to alter cells in the table that were designated by the programmer as RAM by leaving them blank in the table. For example, hitting key [6] in above example will enable you to change the on line flag to enable/disable polling of the named station. When you are done making alterations to the table, hit the [-] key to return to the main system menu.

For the 2 X 16 LCD:
Only one table cell at a time will be shown. For example, in the case of the above table, the second entry to be shown will be shown as:

```
PollAddr=
123
```

Hit the [DownArrow] key to move down to the next cell:

```
PollMsg=
Master
```

Hit the [UpArrow] key to move up to the previous cell. Hit the [ENTER] key to move up to the next table column. You cannot move to a previous column. To do that, you must hit the [-] key and reenter the table. To edit a cell's value, hit the [CLR] key to erase the cell's value and then key in the new cell value followed by the [ENTER] key. When you are done editing, hit the [-] key to return to the main menu.

ADJUSTING LCD CONTRAST

Hitting key [7] from the system menu enables you to adjust the LCD contrast. Note that in the system menu, the prompt is "7 Adj LCD contrast…now=200(132)", indicating that there are two components to the contrast setting. The first (200) is the user's last entry to the contrast register. The second (132) is the value actually used to set the contrast. You control the first entry; and the system calculates the second value based on your entry and the temperature reading from the loop supply/charger board, if present. When you hit key [7] from the system menu, you will be prompted for a new contrast setting. Full range is 0 to 511; but normal range is usually 150 to 250 depending on LCD temperature. A value of 180 usually gives acceptable contrast.

Configuring the RUG5/9

Once you have applied power to the RUG5/9 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 than its program. Unlike earlier RUGID units that were programmed in BASIC, the RUG5/9 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 RUG5/9's operating system. Our job as programmers now becomes one of connecting together the modules (i.e., designing the project) to do the tasks we wish the RUG5/9 to perform. When we tell the R9SETUPD program to compile our project, it produces a configuration file for loading into the RUG5/9.

INSTALLING R9SETUPD

R9SETUPD is the support software you will use to configure the RUG5/9. It is compatible with Windows 95/98/NT/2000/XP/Vista (hereinafter referred to as Windows) and is available at no charge with each RUG5/9 unit. It is also downloadable from our web site, www.rugidcomputer.com. When you receive it, it will be named SETUPXXX.EXE, where XXX is the software revision number such as 565 (representing revision 5.65). To install it, if you have downloaded it and it is on your hard drive, simply find it in your directory system and double click on it. If it is on floppy disk, insert it into your floppy drive and click on the **START** button and select **RUN…** Then type the name of the file on your floppy (e.g., SETUP429) and click **OK**. In either case, an installation wizard will guide you through the installation.

STARTING R9SETUPD

To start R9SETUPD, double click the 'R9' icon on your desktop, if it exists, or click on **START,** then **PROGRAMS,** then the **R9SETUPD** entry. When R9SETUPD starts, you should see the following screen:

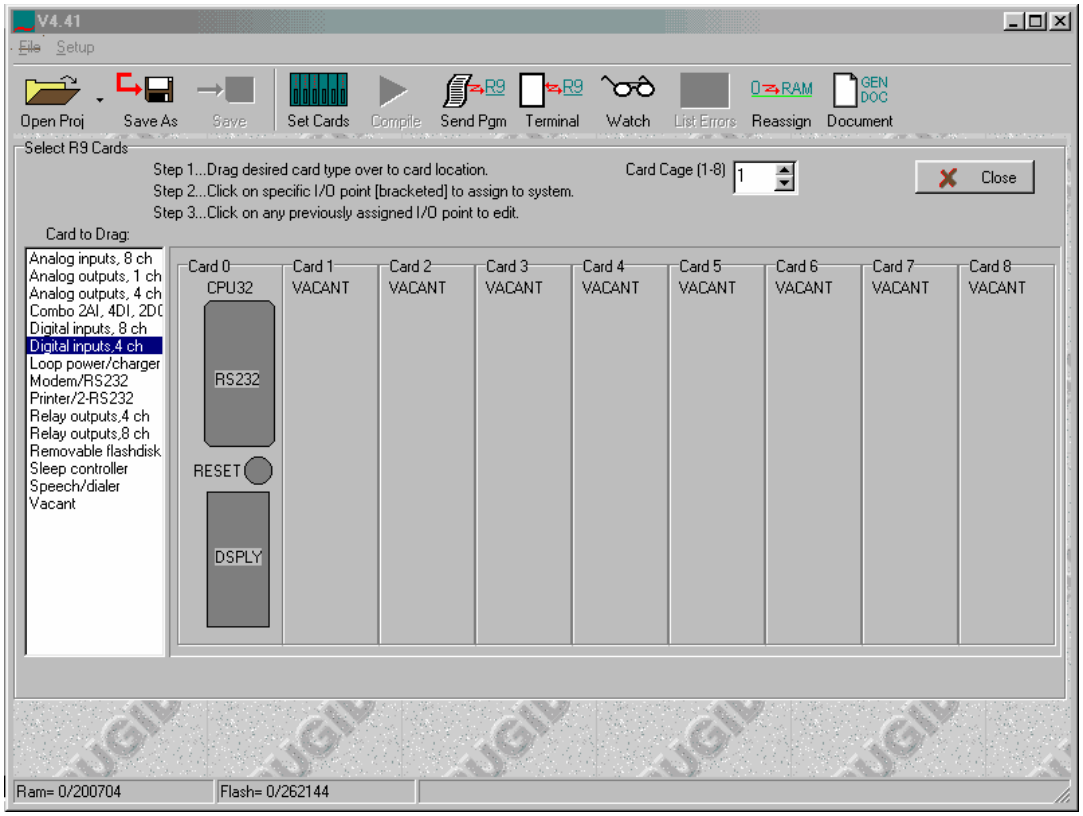


Figure 4 R9SETUPD Opening Screen

A Simple Application

To illustrate how you configure the RUG5/9 to do something useful, let's configure a RUG5/9 to read an analog value that we'll assume is a tank level and generate high and low alarms based on the value. We'll send the alarms to a pair of relays on the RUG5/9. We'll also make the RUG5/9 show the tank level and alarm states on its LCD display.

Specifying RUG5/9 Hardware

To begin, start the R9SETUPD program you installed on your Windows system above. To do this, click on **START…Programs…** and select "R9SetupD". You should see a screen appear as illustrated above. This is the screen where you tell the RUG5/9 what cards you have installed. Notice that card position 0 indicates that we have a CPU card installed. The remaining eight card slots indicate vacant. Position 0 of the first card cage must always have a CPU card. The other card cages cannot have CPU cards and will always show the card zero position as vacant. First, let's specify that we have an analog input card in card position #3. To do this, we must select the analog input card from the list at the left, drag it to the first card position, and then drop it. Go ahead and try it. Use the left mouse button to click on "Analog inputs, 8 ch" and hold it down as you drag the mouse pointer over to anywhere in card 3's space, then release the key. When you do, the vacant appearance of card 1 should change to that of **ANIN8** with a column of 8 channels and a COM connection. Similarly, drag the label **Relay outputs, 8 ch** from the list at the left over to card position number 6. A **RELAY8** card representation with 8 LED's and 8 channel connections should appear in card 6's location. Your screen should now look like this:

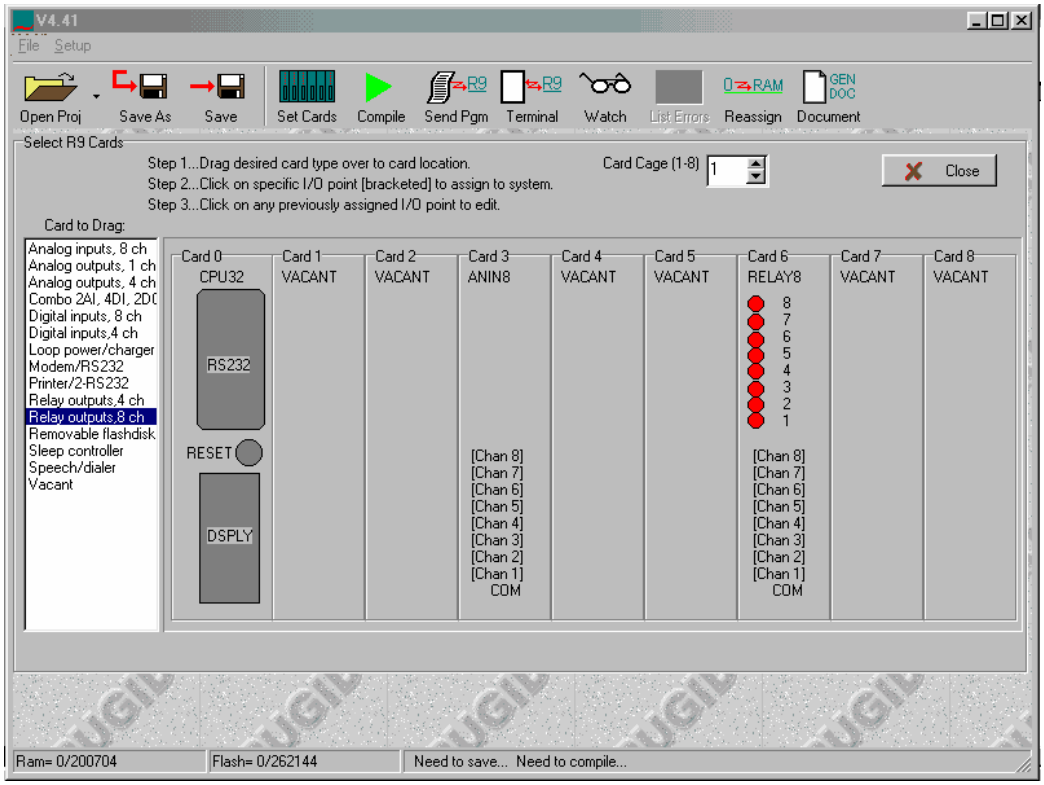
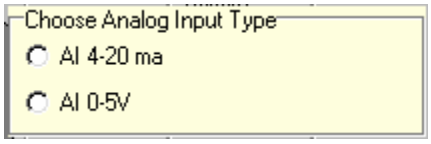


Figure 5 Card Cage with AI and Relay Cards Installed

We've now told the RUG5/9 what our hardware configuration is.

Specifying I/O Assignments

We must now assign names to the I/O points we just installed so that we can refer to them as we connect them together. First, let's set up the analog input. Click on the analog input card's **Chan 1** field. You should see the following small panel appear in the middle of the screen to enable you to select the type of channel to use. Select a 4-20 ma. type channel.



You should now see the following screen appear. It allows you to name the analog input and set parameters for it.

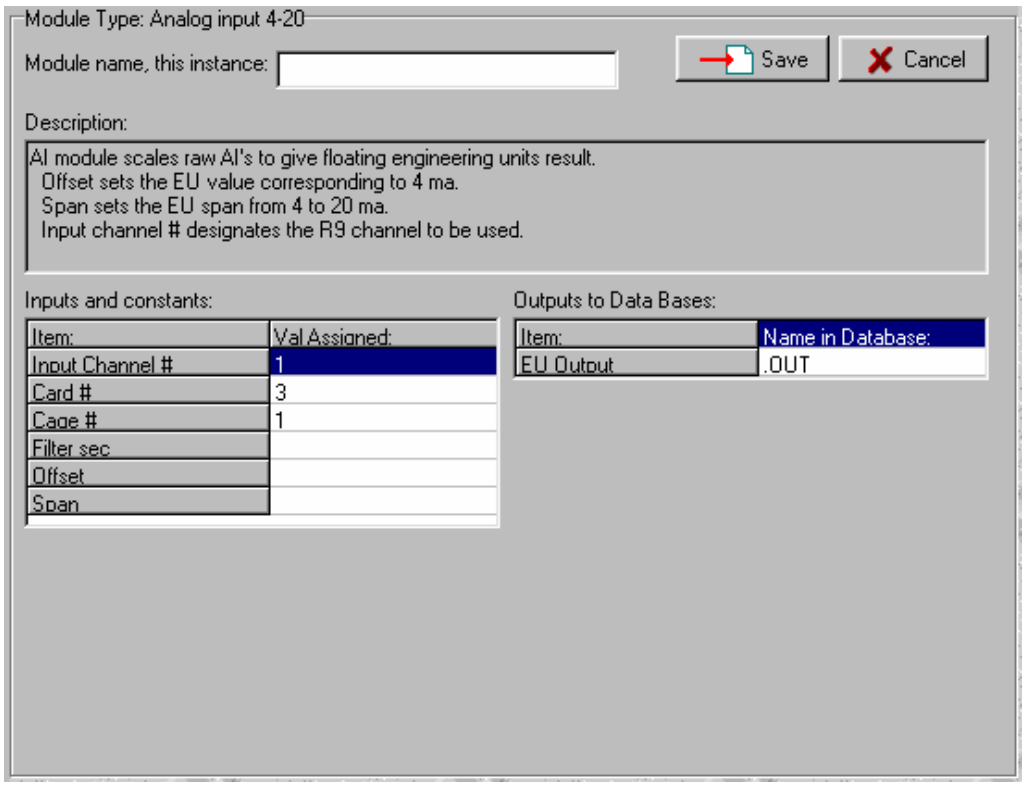
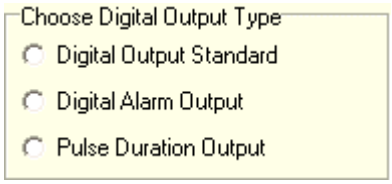


Figure 6 Analog Input Configuration Panel

This is the parameter entry screen for the 4-20 ma. analog input channel at card cage #1, card #3, channel #1. Your cursor should be sitting in the edit box where you enter the name of this channel. 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 RUG5/9's floating point data base. Down below, in the inputs and constants list box, click on the cell to the right of the title **Filter sec** and enter the value "5". This sets the analog input's filter time constant to 5 seconds to slow down its response to transients. In the other two cells below this one enter values of 0.0 for offset and 15.0 for span. 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. OK, we're done, click on the **Save**

button and we will return to the screen showing our circuit boards. 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 card 6's channel 1 entry and you should get a channel type selection panel such as shown below:



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

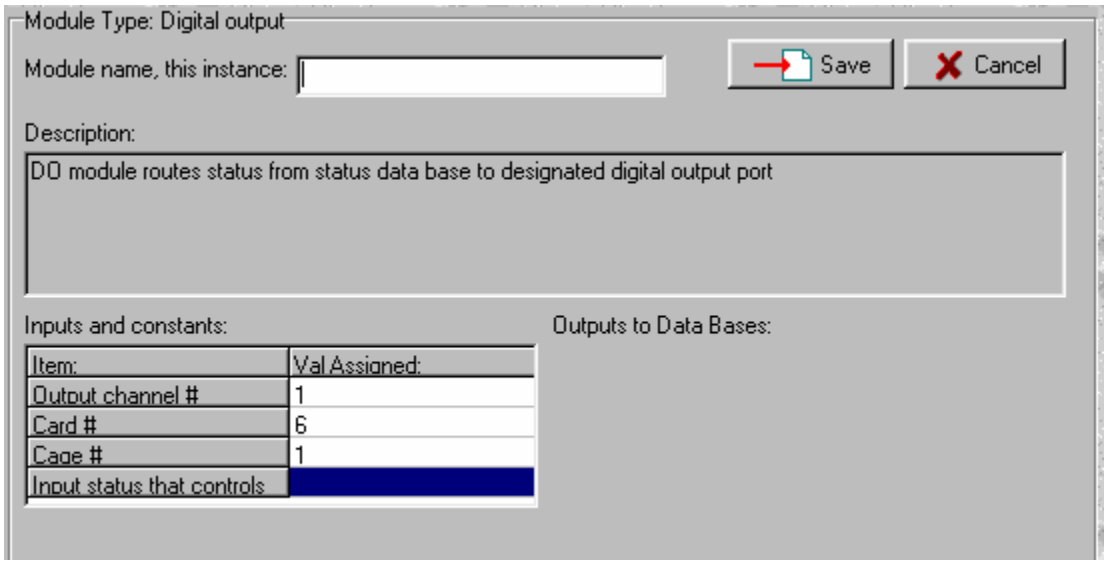


Figure 7 Digital Output Configuration Panel

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 assigning I/O for this project. Our card cage representation should look like this:

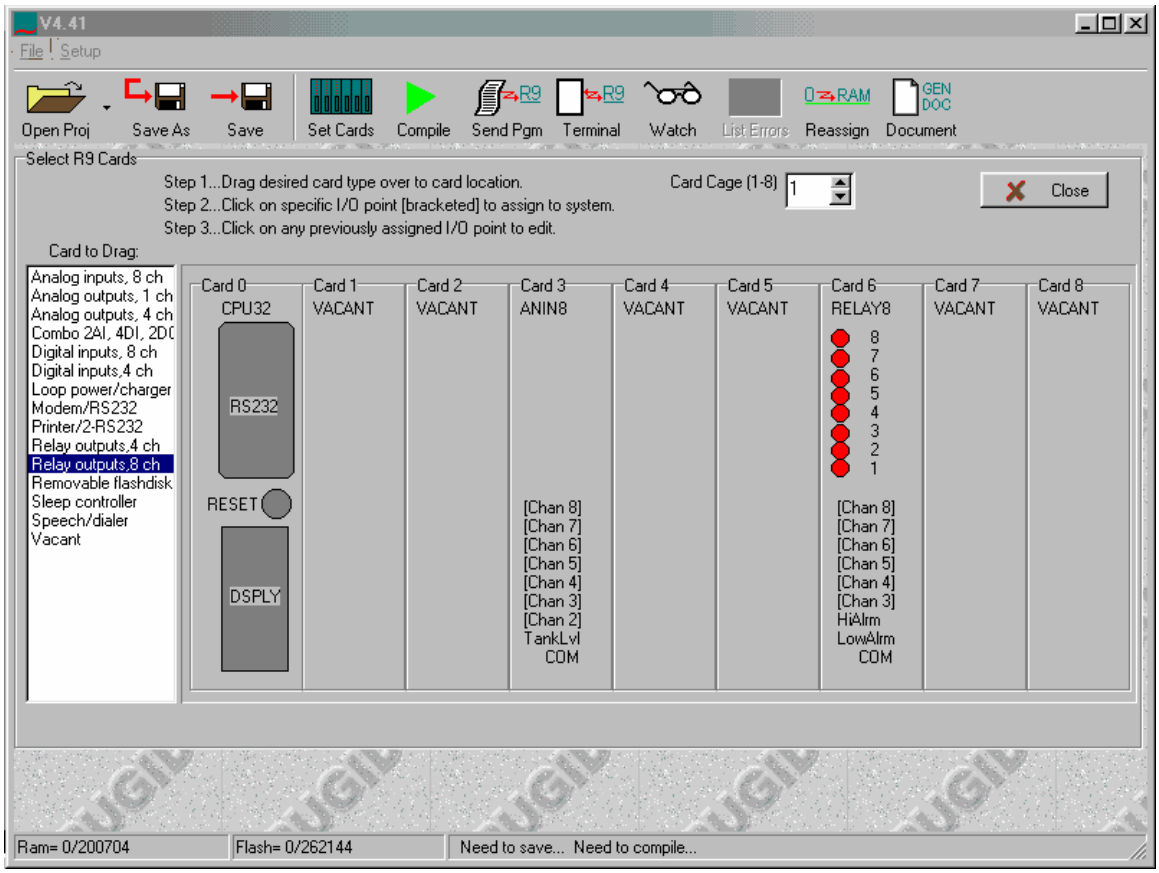


Figure 8 Card Cage With I/O Modules Installed

Notice that the first analog input channel has the name “TankLvl” and that the first two digital output channels are named “LowAlrm” and “HiAlrm” 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:

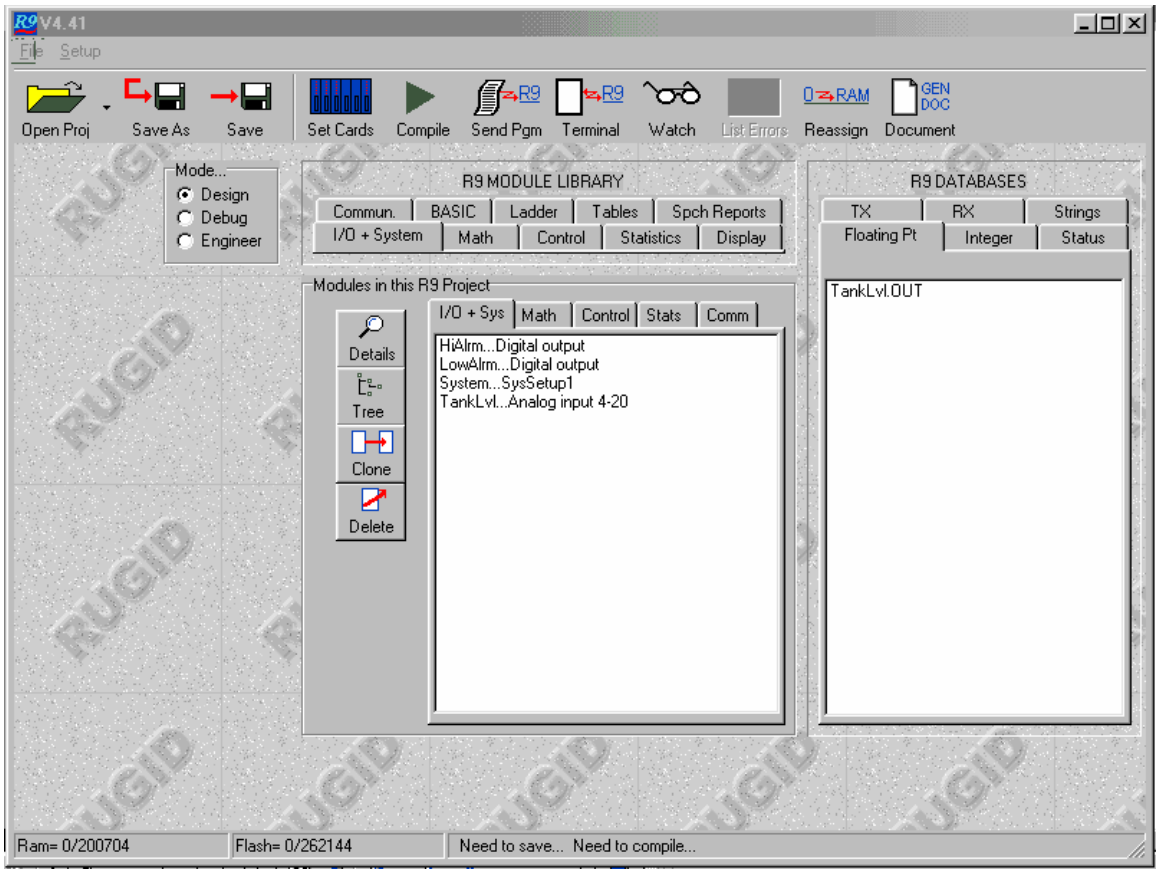


Figure 9 Main Project Panel

Notice a few things here. In the middle of the screen are two sets of tabs…**R9 MODULE LIBRARY**, and **MODULES in this R9 Project**. On the right side of the screen is a third set of tabs labeled **R9 DATABASES**. Here is what they mean:

R9 MODULE LIBRARY Preprogrammed modules we can use to design our project.
MODULES in this R9 Project…Modules we have configured and installed in our project.
R9 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 R9 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:

HiAlrm…Digital output
LoAlrm…Digital output
System…SysSetup1
Tanklvl…Analog input 4-20

We configured the **HiAlrm**, **LoAlrm** 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., "Analog input 4-20").

On the right of our screen we have the RUG5/9 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 a single entry named **TankLvl**, the name we gave our first analog input module. 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 **R9 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 **R9 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:

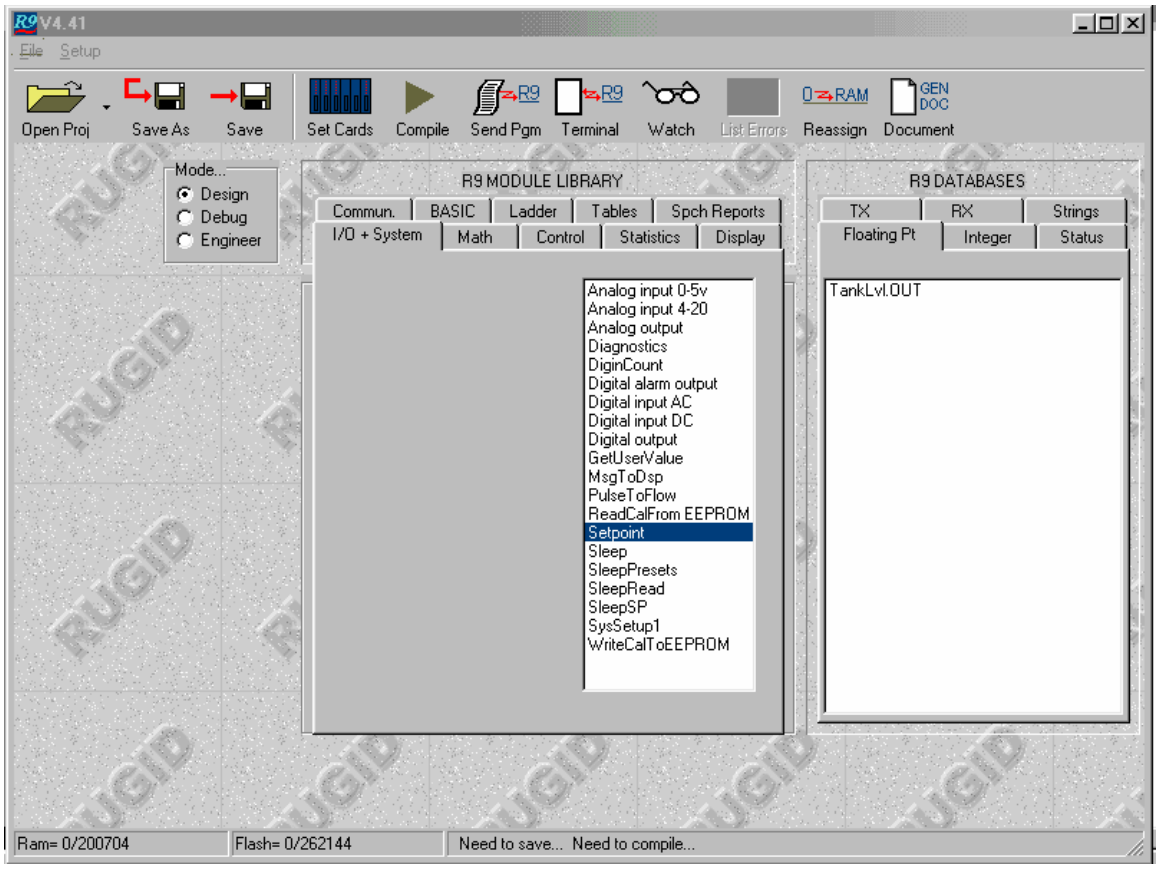


Figure 10 Selecting a Setpoint Module from Module Library

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

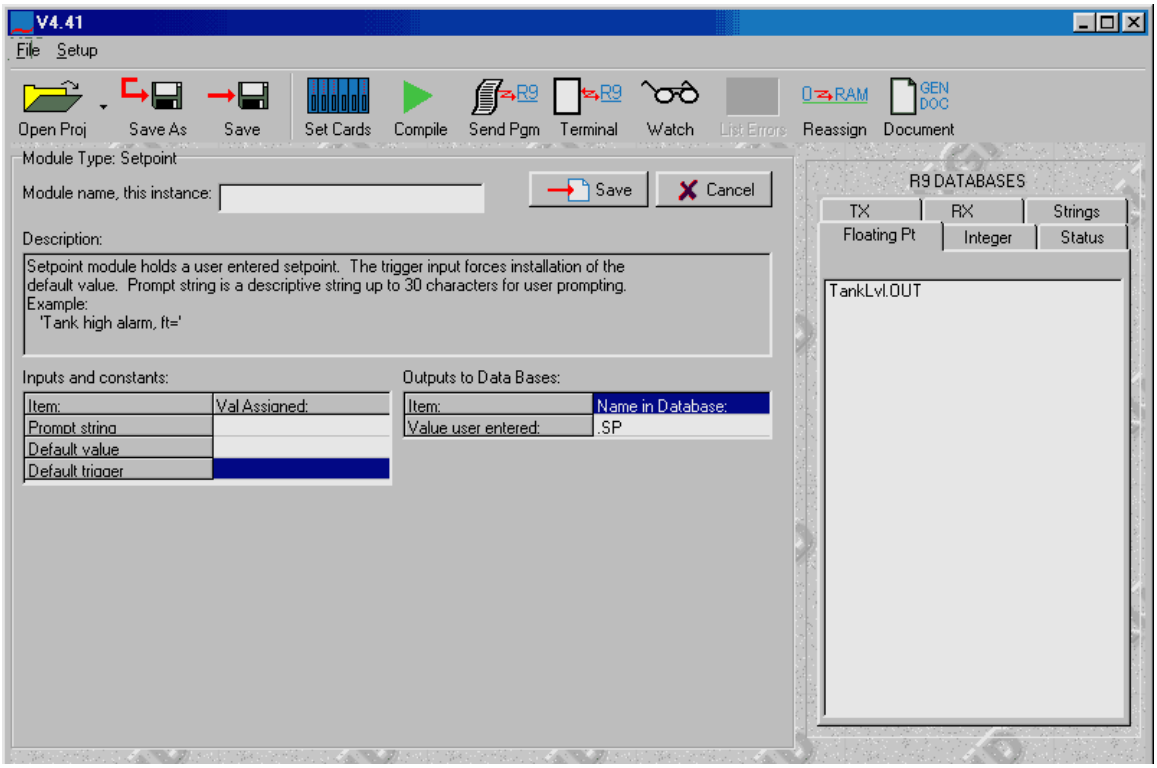


Figure 11 Setpoint Module Configuration Panel

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 RUG5/9's menu system. This must adequately describe what we want the operator to enter for a setpoint. Therefore, enter "Tank low alarm setpoint ft=". We'll not enter the Default value or default trigger 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.

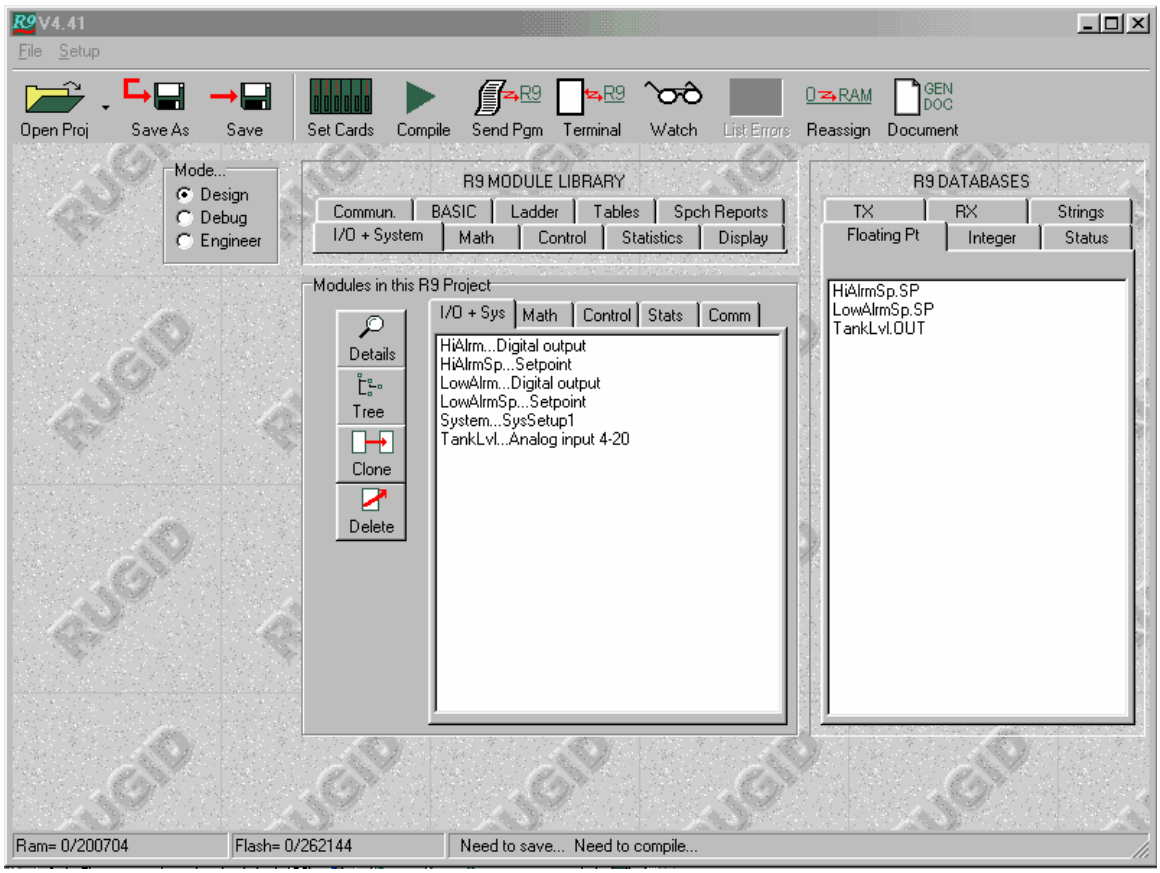


Figure 12 Example Project With Setpoints Configured

Connecting Modules Together

We now have six modules in our project and three entries in our floating point data base. It's time to begin hooking these modules together. We do that by taking the outputs of our 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 one we need is the **AlrmHi** module from the **Control** tab. Click on the **Control** tab in the **R9 MODULES LIBRARY**. Now, select **AlrmHi** from the control module list. The module configuration panel below should appear. Type "HiAlrmGen" in its module name edit box. It should then look like this:

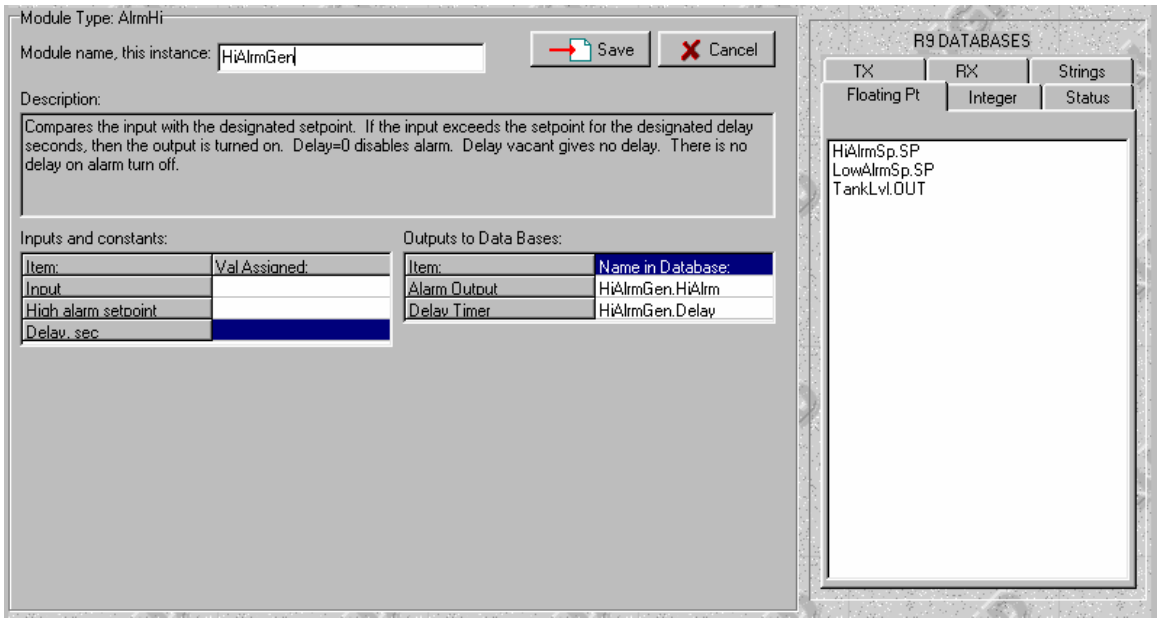


Figure 13 Hi Alarm Module Configuration Panel

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 output. 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. 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. Finally, enter the value "7" into the **Delay sec** cell. This sets the amount of time the tank level must exceed the setpoint before this module will turn on its alarm output. Our module should look like this:

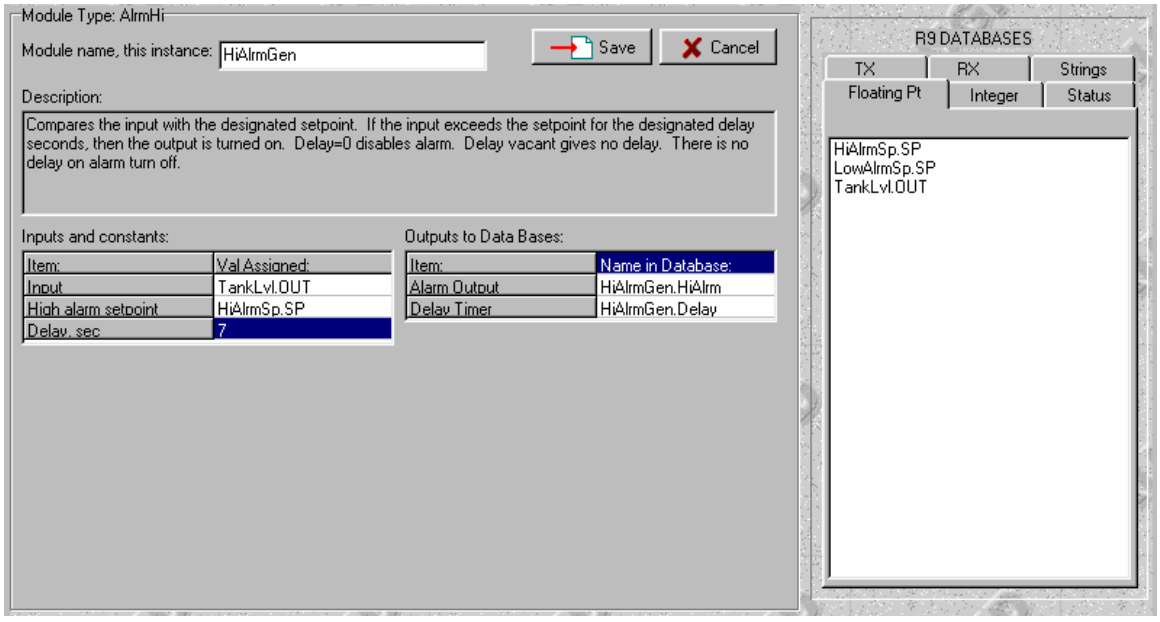


Figure 14 Configured High Alarm Generator Module

Notice that for each module input cell, you can either enter a constant value into the input cell, drag a signal from a data base into the cell, or leave it blank. If you leave it blank, it will be given a value that is transparent to the module's math (0.0 or 1.0). 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 constants or strings taken from the string database. Basically, the **AlrmHi** module we just configured compares its input (tank level) with its high alarm setpoint, and, if the alarm condition exists for at least 7 seconds, it will turn on its high alarm output. Now, click the save button to save this module to the project.

In a similar way, install a low alarm generator to the project using the **AlrmLo** module from the **Controls** tab. Name it LoAlrmGen and drag in the tank level and low alarm setpoint from the floating point database, enter a 7 second alarm delay, and then save it to the project. Your project screen should now look like this:

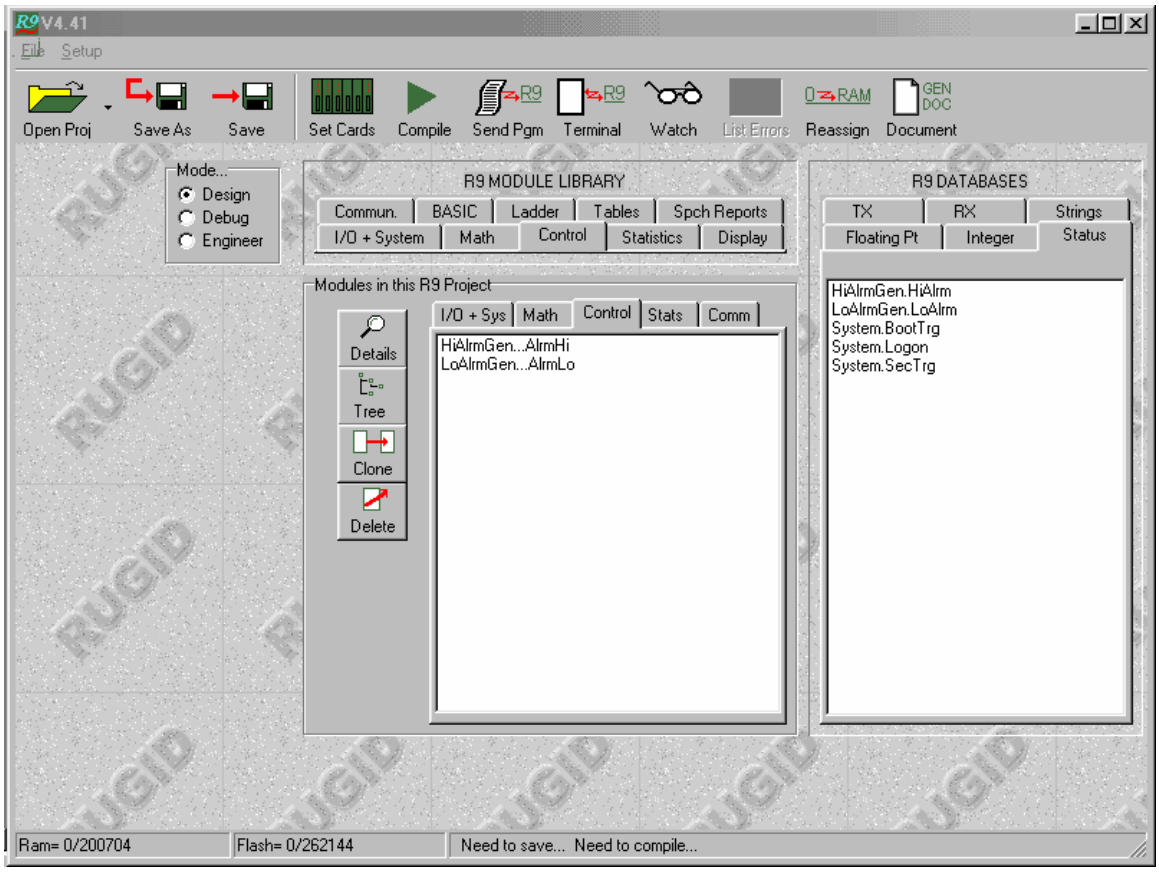


Figure 15 Project Screen After Alarm Generators Added

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.

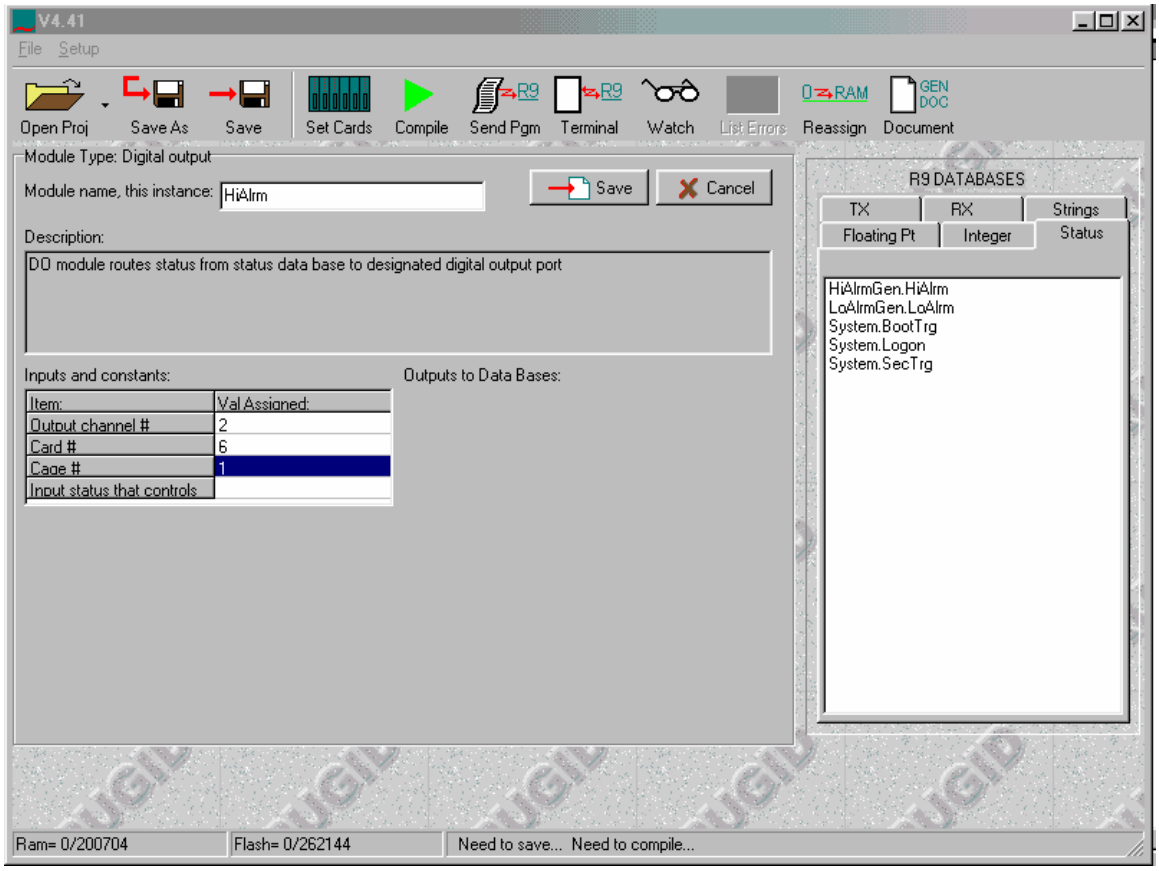


Figure 16 Digital Output Configuration Page With Status Data Base Showing

Now drag the **HiAlrmGen.HiAlrm** entry from the status database over to the relay output module's **Input status that controls** cell. This will cause the **HiAlrmGen.HiAlrm** status output to control the relay. Similarly, call up the **LowAlrm…Digital output** module from the project module list and drag the **LowAlrmGen.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 RUG5/9 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 **R9 MODULE LIBRARY** window. Your screen will change to this:

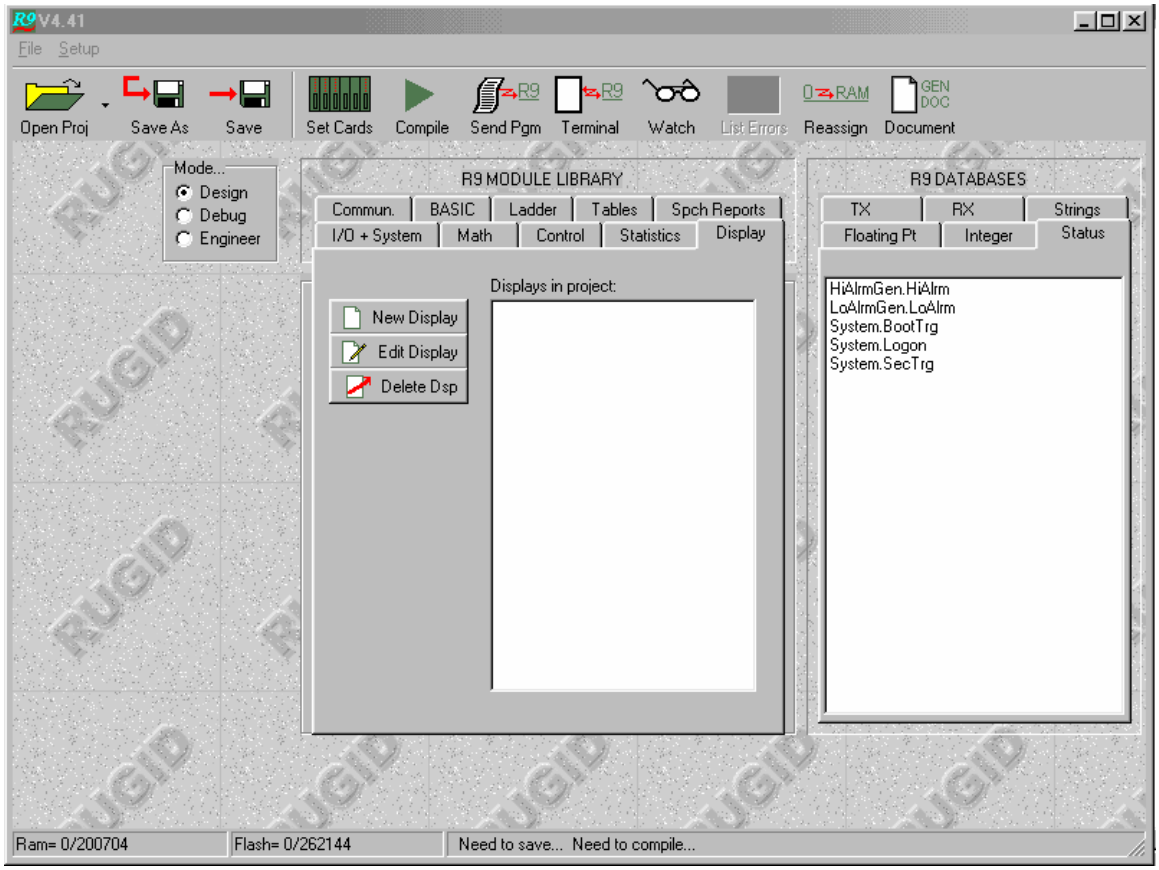


Figure 17 Display Selection Tab in Module Library

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

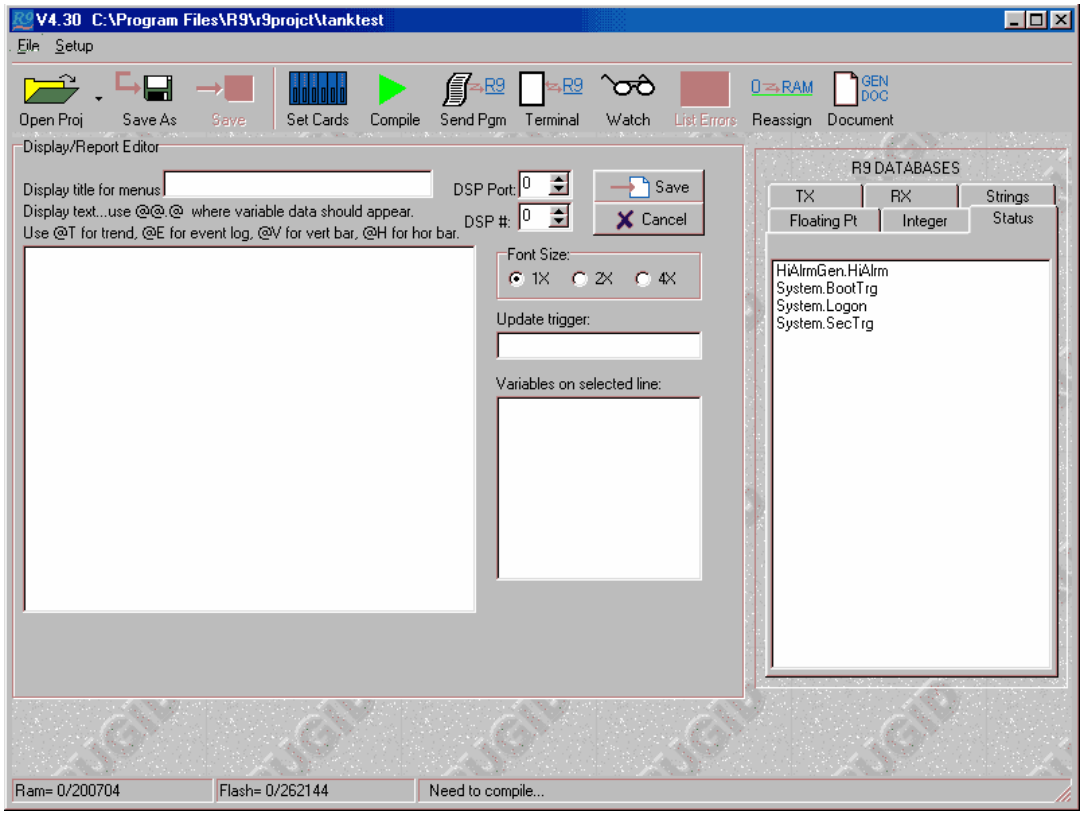


Figure 18 Display Editing Panel

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:

```
** Main Display **

Tank lvl  ft=@@.@@
Hi alarm setpt=@@.@
Lo alarm setpt=@@.@

Hi ON=@   Low ON=@
```

After you have done that, just for fun, click on the **2X** radio button so the RUG5/9 uses larger characters. Your screen should look like this:

Getting Started

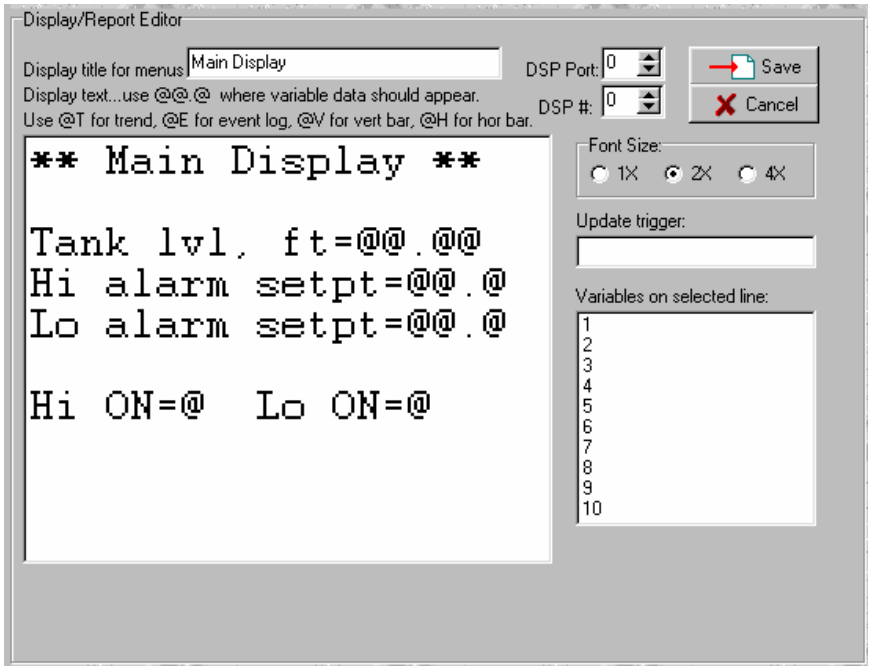


Figure 19 Display Defined

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 be placed on the LCD screen. But we haven't yet told the RUG5/9 what data to present in each of the **@@.@** type fields. We do that as follows. First, click on the "Tank lvl 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 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. Similarly, for each of the high alarm setpoint and low alarm setpoint lines in the large window, click on the line and then drag the corresponding alarm setpoint from the floating point data base and drop it on the #1 line of the list box. 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 ON=@ Lo ON=@". Here we must drag two entries from the status database and drop onto the top two locations in the variables list box because we have two **@** fields on that line. We still have one more thing to do before we're done…we must specify the display's update trigger. With the RUG5/9, 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, which 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 and generates a trigger once per second. Your screen should now look like this:

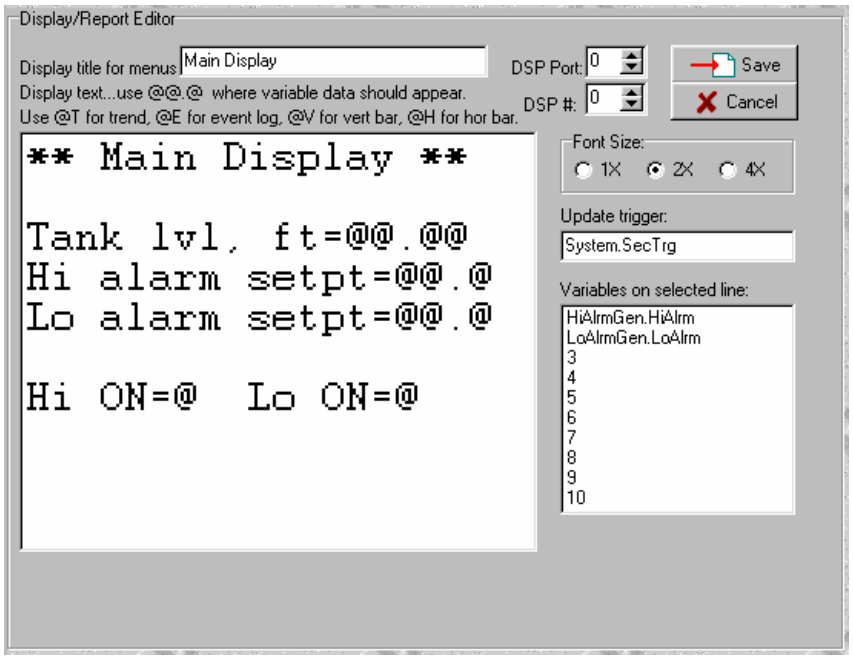


Figure 20 Display Panel with Alarm Bits Specified

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.

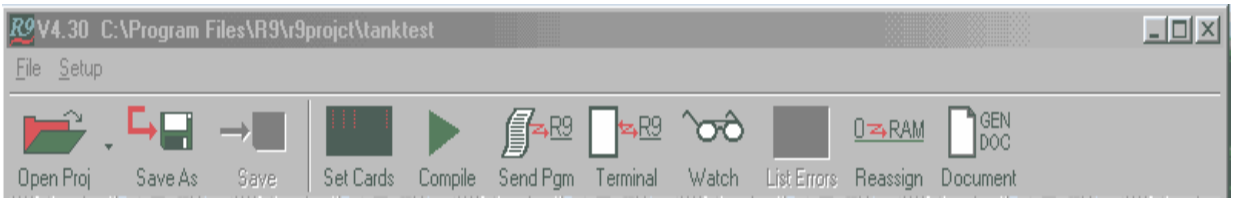


Figure 21 R9SetupD Toolbar

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 RUG5/9. If no errors are present, click on the **SendPgm** button to send the project to the RUG5/9. Note that you do not have to click the **Compile** button before sending the program to the RUG5/9, R9SETUPD 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 RUG5/9. It will look like the screen below:

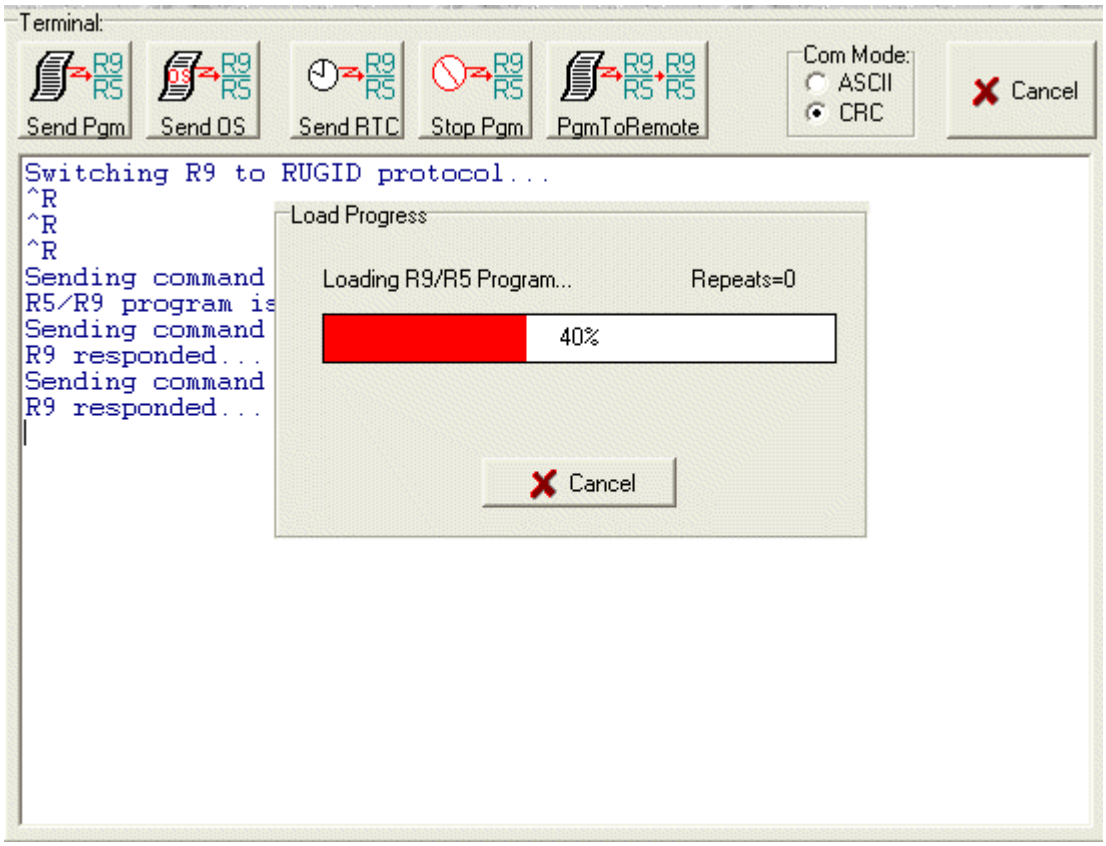


Figure 22 Example File Load Progress Screen

When done with loading, the RUG5/9 will begin running the program. Look at the RUG5/9'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.

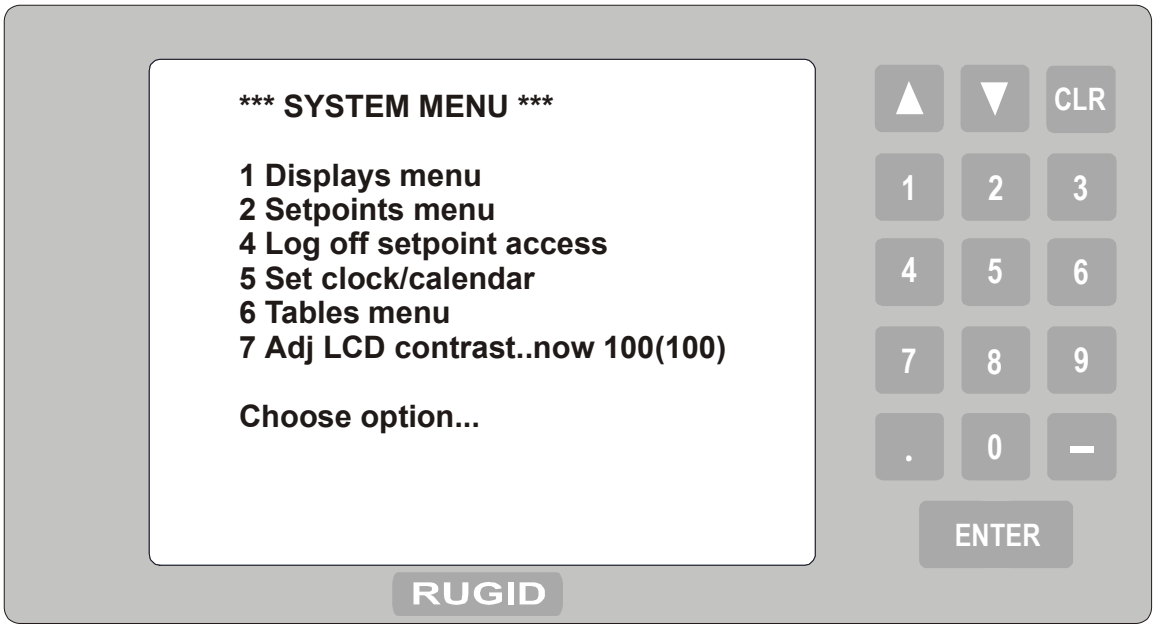


Figure 23 RUG5/9 Keyboard and System Menu

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 RUG5/9. 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, 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-RUG5

Each RUG5 consists of a minimum of a small card cage with a mother board in the center, a CPU board in the front, and a power board at the rear. The mother board mounts in the center of the card cage and has connectors into which the CPU and up to five I/O cards plug in. Each card slides into the card cage from the front or rear and is secured with a small plate and sheet metal screw. The CPU occupies the top position in the front of the card cage. The following table defines which cards are allowed in the front two expansion slots, as well as cards available for the rear slots.

Table 1 Cards Allowed in RUG5 Card Cages

FRONT of CARD CAGE		REAR of CARD CAGE		
Card in front top slot	Cards in front mid (slot 6) and bottom (slot 7), R9 boards	Top slot	Middle slot	Bottom Slot
CPU board (required)	Dialer Modem/RS232 Sleep board Flash disk Loop/Charger/Diag Analog inputs Analog outputs Digital inputs Relay outputs Printer/dual serial Combo board	Pwr/Charger Pwr/Digital I/O	Modem	Analog I/O, Loop

In addition to the above hardware, the RUG5 can also have either a 2 line by 16 character LCD and keyboard installed in the top of the card cage, or a RUG9 display/keyboard module can be attached to the LCD connector on the CPU card. The Rug9 display/keyboard module consists of a 320 by 240 pixel LED backlit LCD and 16 key tactile feedback keyboard enclosed in a steel housing. That module must be mounted remotely. A single 20 pin ribbon cable up to 5 feet in length interfaces the LCD/keyboard module to the CPU board. The following figure provides details of the RUG5 hardware.

Hardware

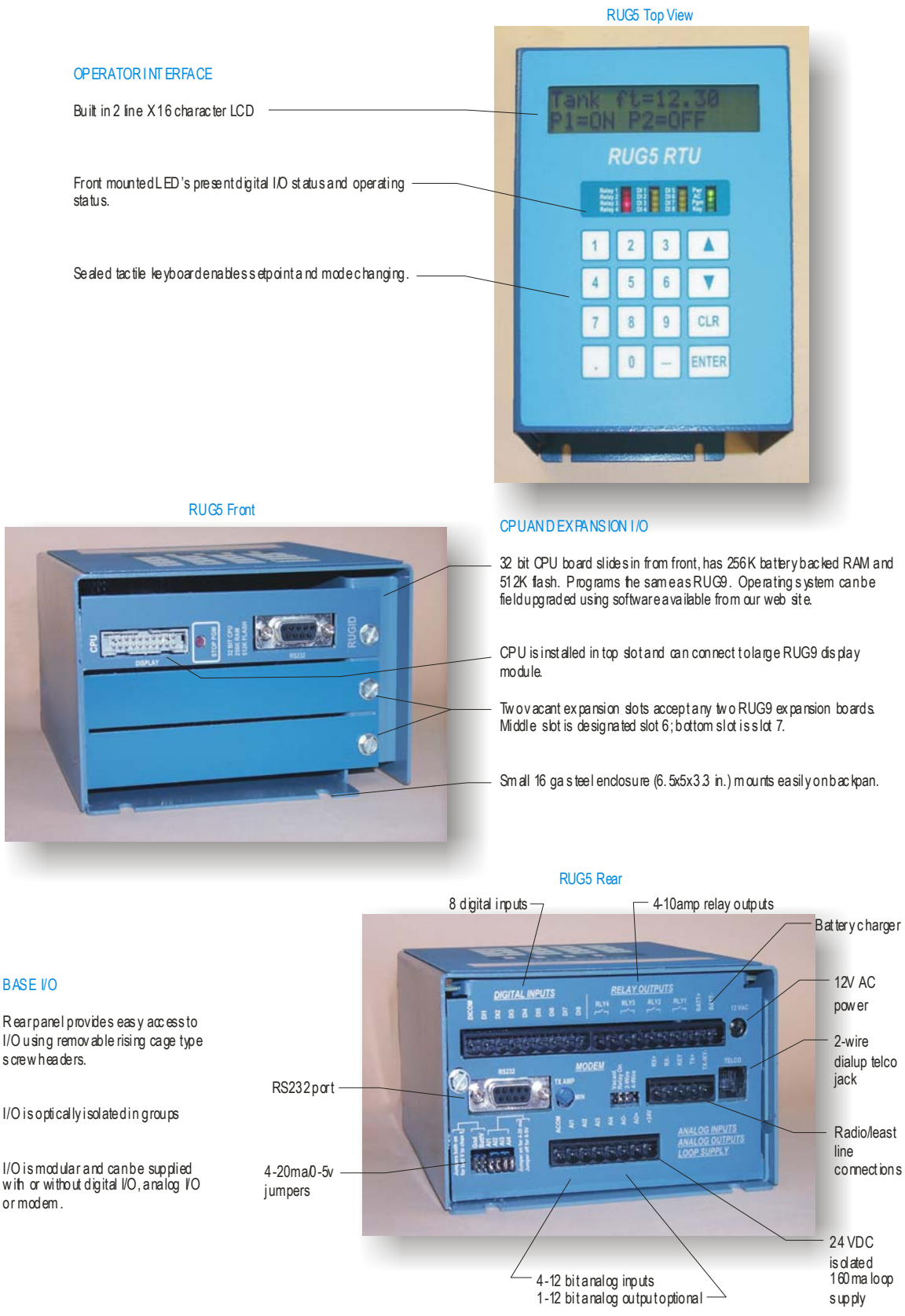


Figure 24RUG5 Features

Hardware

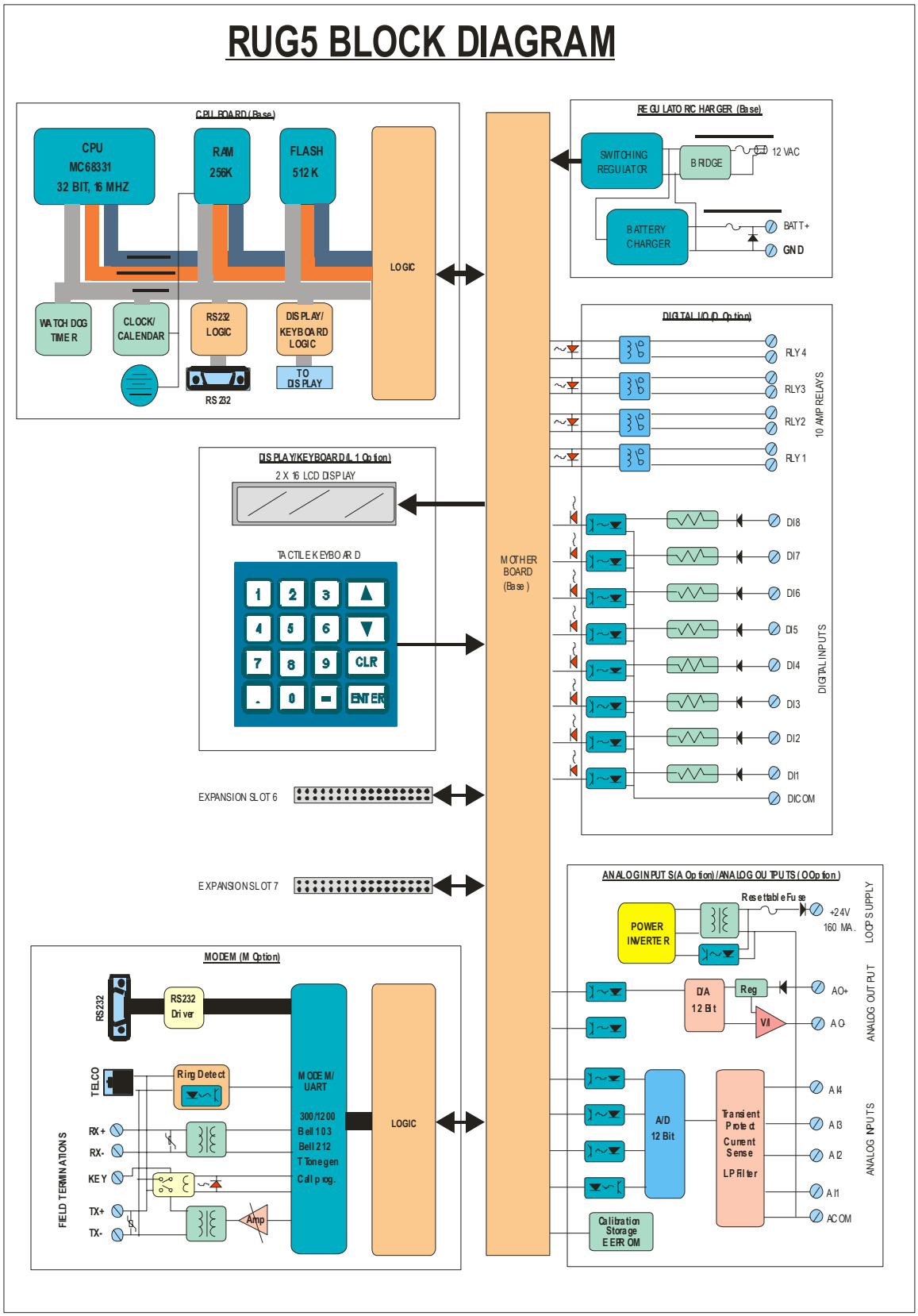


Figure 25 RUG5 Block Diagram

Applying Power to the RUG5

Like the RUG9, all RUG5 units can be powered from 120 VAC using a standard 12 VAC wall transformer with a 2.5 mm coaxial plug. Simply plug it into the round jack on the rear of the RUG5 card cage as shown in Figure 3. The total power draw of the unit can be estimated from the following formula:

Current=0.2+.035*(# of relays)+0.28*(#loop supplies)

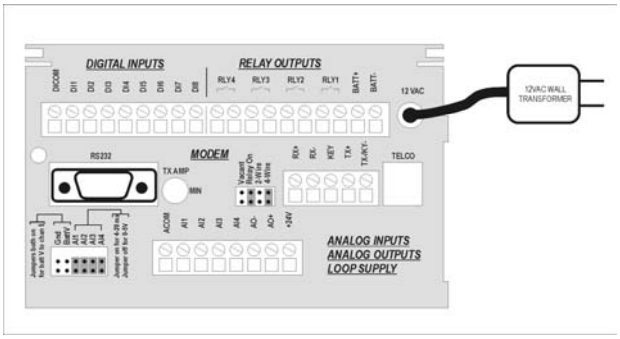


Figure 3 Powering the RUG5 from 120VAC

For applications using DC power such as solar power applications, the RUG5 can either be powered by applying 12 to 15 VDC to the AC power jack as illustrated above; or you can apply power to the battery terminals as illustrated in Figure 4. The current requirement is the same as estimated above. If you apply DC power to the AC jack above, be aware that the power supply ground will be approximately 1.5 volts above that present in the RUG5, since your power will be applied to the RUG5's full wave bridge, rather than directly to the RUG5's ground. This usually will be of little consequence since all external connections to the RUG5 are optically isolated from the RUG5's DC bus except for the battery connections on the loop/charger board.

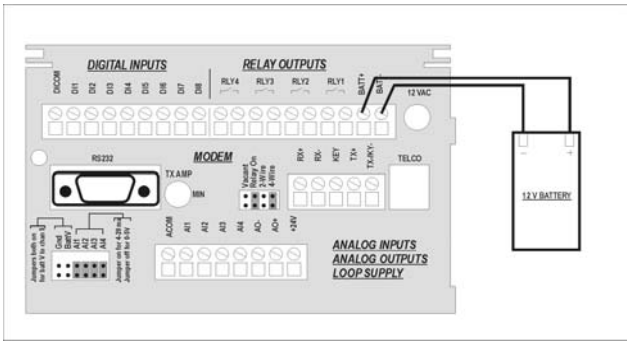


Figure 4 Powering the RUG5 from 12 VDC

Hardware

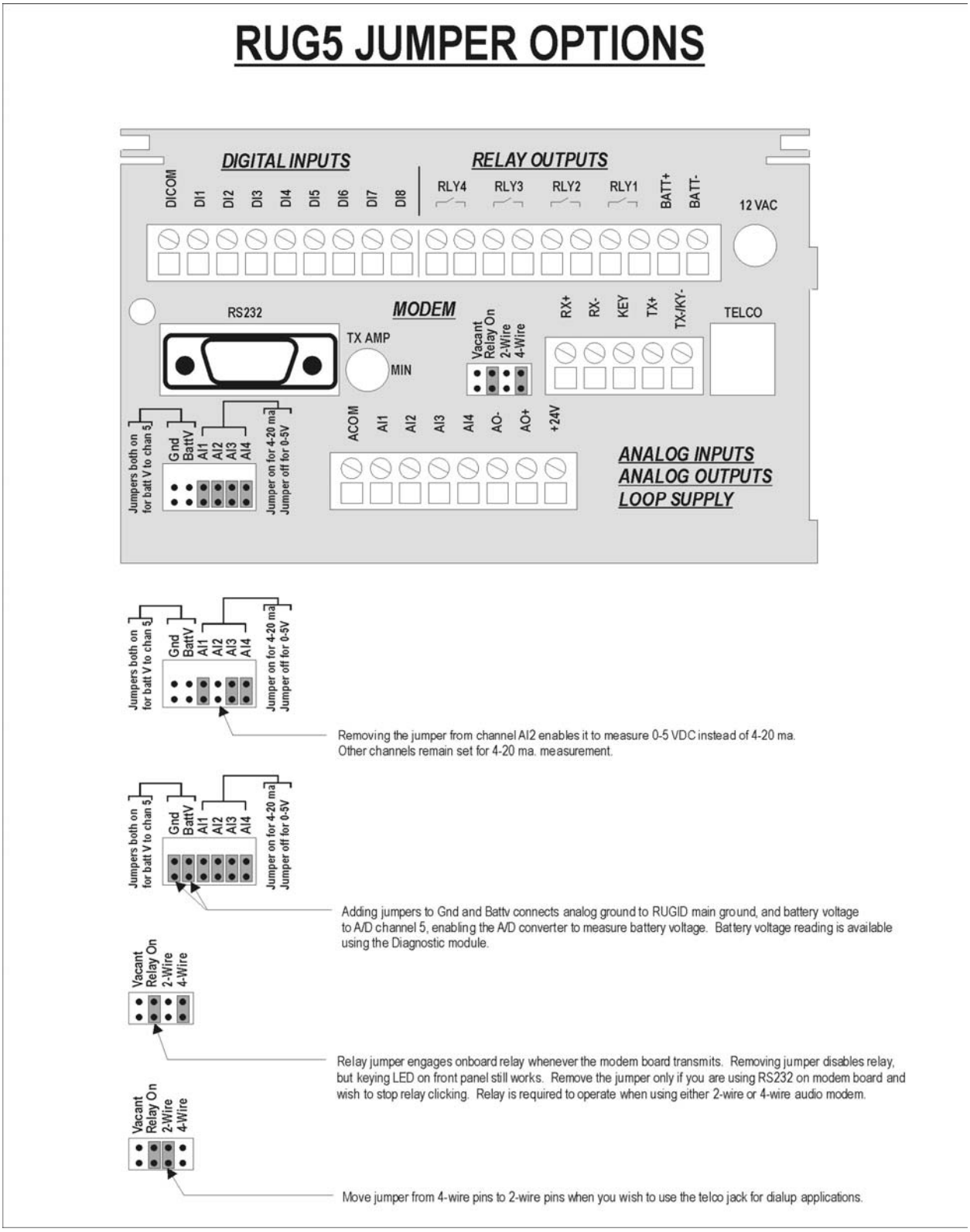


Figure 26 RUG5 Jumper Options

Overview-RUG9

Each RUG9 consists of a minimum of one card cage with a mother board and a CPU board. The mother board mounts against the back panel of the card cage and has nine connectors into which the CPU and up to eight I/O cards plug in. Each card slides into the card cage from the front and is secured with a small plate and sheet metal screw. The CPU occupies the leftmost position in the card cage. If more I/O points are required by an application than can be installed in the first card cage, then you can attach additional card cages to expand the I/O. Up to 8 card cages can be interconnected to constitute a single RTU. Only the first card cage needs a CPU card; the CPU card position in the remaining card cages must be left vacant. The following table defines which cards are allowed in the primary and expansion card cages. In general, the primary card cage can have any cards, but the expansion card cage can only have I/O channel cards.

Table 2 Cards Allowed in Expansion Card Cages

PRIMARY CARD CAGE #1		EXPANSION CARD CAGES #2 TO 8	
Card 0	Cards 1 to 8	Card 0	Cards 1 to 8
CPU board (required)	Dialer Modem Sleep board Flash disk Loop/Charger/Diagnostics Loop/Charger Analog inputs Analog outputs Digital inputs Relay outputs Printer/dual serial Combo board	Loop/Charger	Loop/Charger/Diagnostics Loop/Charger Analog inputs Analog outputs Digital inputs Relay outputs Combo board

In addition to the above hardware, an RTU can also have a display/keyboard module attached. The display/keyboard module consists of a 320 by 240 pixel LED backlit LCD and 16 key tactile feedback keyboard enclosed in a steel housing. The module can be attached with a hinge kit to the front of the primary card cage, or it may be mounted remotely. A single 20 pin ribbon cable up to 5 feet in length interfaces the LCD/keyboard module to the CPU board. The following sections provide details of the RUG9 hardware.

RUG9 Card Cage

The RUG9 card cage is a steel enclosure with integral card guides and a mother board mounted inside at the back of the cage. The card cage mounts to a panel, using teardrop shaped holes punched into its rear flange. The mother board is secured in the card cage with eight screws that tightly attach it to threaded studs pressed into the rear sheet metal of the cage. Up to 9 cards can be installed by sliding them in from the front of the cage so they mate with header strips soldered onto the mother board. The drawing below presents the card cage arrangement.

Hardware

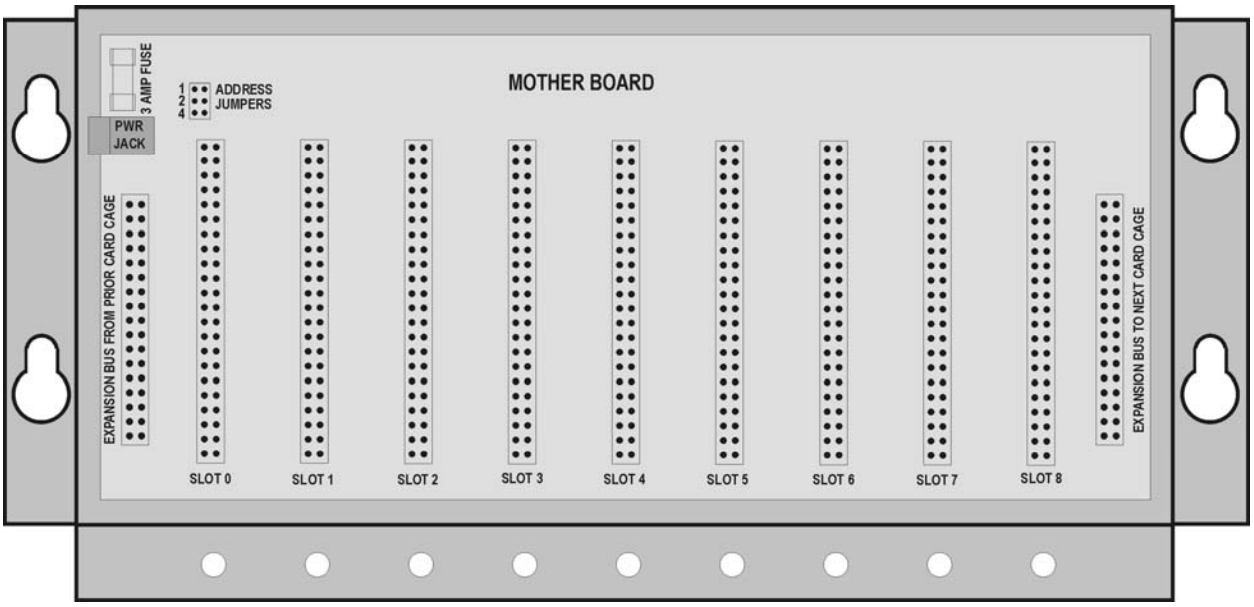


Figure 27 RUG9 Card Cage

Electronically, the card cage mother board contains power supply and address decoding functions as illustrated below. Note that late revisions of the mother board use a solid state breaker instead of the fuse for over current protection. If the card cage has had an over current condition, the solid state breaker will have to cool for a few minutes before the card cage will successfully power up. See Chapter 10 for card cage dimensions and mounting hole pattern.

Hardware

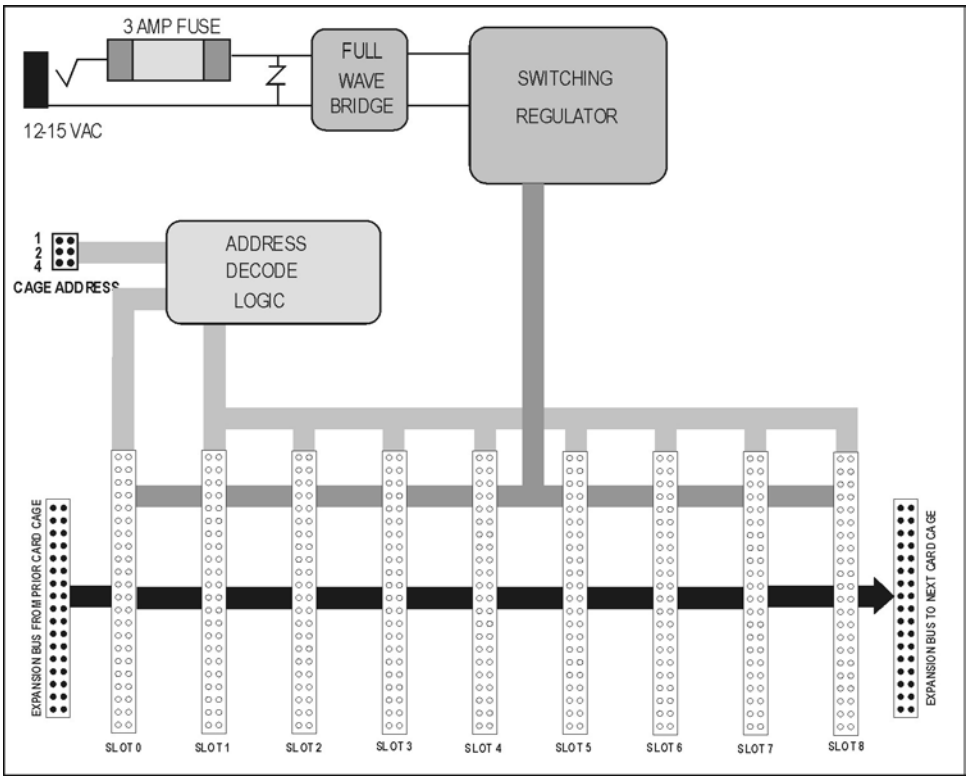


Figure 28 RUG9 Mother Board Block Diagram

Addressing

Each card cage in an RTU must be addressed uniquely. Specifically, the main card cage that contains the CPU board must be address 1; the first expansion card cage must be address 2, etc. Addressing is established by installing push on jumpers as illustrated below. All card cages delivered will come with all jumpers in place, which will address the card cages as though they are the main card cage. To address cages for use as expansion cages, simply remove one or more jumpers to obtain the addressing shown below.

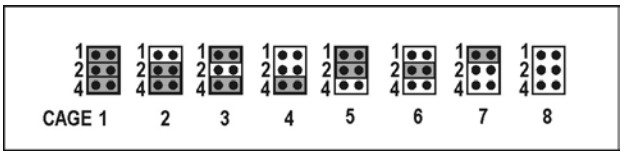


Figure 29 Card Cage Address Jumpers

Removing and Installing Cards

When your unit arrives, it should have the cards you ordered in place. You can easily rearrange the cards, remove them and add others. **Before removing or installing cards, be sure power is removed from the card cage and from all cards installed.** This is important because removing or installing cards while the card cage is powered can damage the card being moved or other cards in the card cage including the mother board. It is also important that I/O points be powered down so **you don't get shocked by adjacent cards**. To remove a card, remove its cover plate hex head hold down screw with a nut driver and remove the card cover plate by pulling it out from the bottom. Then, grip the card firmly and pull it out. It is sometimes convenient to insert a small screw driver

into the hole at the lower front corner of the card and pry against the lower lip of the card cage to remove the card. To install a card, make sure its protruding pins are straight. Then, insert it into the position it is to occupy and push it in gently until it is almost all the way in. The final quarter inch of its travel will require more force as its pins engage the mother board's connector. If it will not travel the last quarter inch, the card's pins may not be accurately aligned with the mother board's connector, or one or more of the board's pins may be bent. If the pins are straight but it resists the final travel, try jiggling the board to align the pins. If it still resists, remove the card to the right of the position you are working on and look to see if the pins are correctly aligned with the mother board. Once they are aligned properly, the board should slide into place with firm pressure. When fully seated, the card's front edge should be flush with the card cage's lower lip. With the card fully seated, install the card face plate by inserting the face plate's tab into the horizontal slot in the upper surface of the card cage, then install the hex head hold down screw with a nut driver or screw driver. If you have added a card, you may need to break away the leftmost position of the vacant slot cover plate that covers the unoccupied card cage positions before reinstalling it.

Applying Power

All RUG9 units can be powered using a standard 12 VAC wall transformer with a 2.5 mm coaxial plug. Simply plug it into the round jack on the left side of the RUG9 card cage as shown in the drawing to the right. Each card cage must be powered; the expansion cable does not carry power to any of the expansion card cages. A one amp transformer should be sufficient for most applications. However, if your application makes heavy use of loop power supplies or internal relays, you may need a heavier transformer. The maximum power draw of a card cage can be estimated from the following formula:

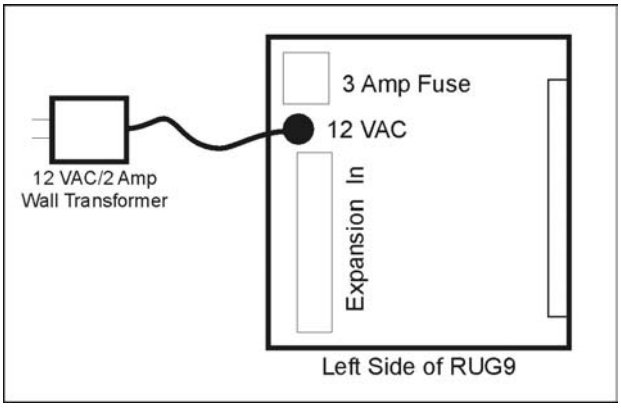


Figure 30 Applying AC Power to RUG9 Card Cage

Current=0.1+0.28*(# of relay boardss)+0.28*(#loop supply boards)+0.1*(#displays)

Hardware

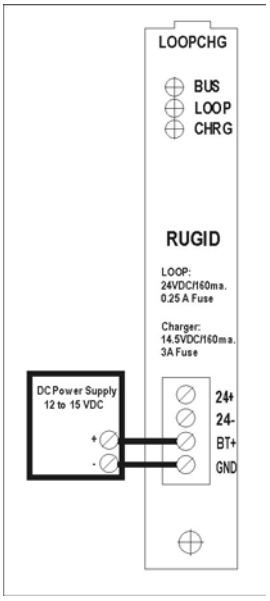


Figure 31 Applying DC Power to RUG9

For applications using DC power such as solar power applications, the RUG9 can either be powered by applying 12 to 15 VDC to the AC power jack as illustrated above; or you can apply power to the loop/charger board as illustrated above. The current requirement is the same as estimated above. If you apply DC power to the AC jack above, be aware that the power supply ground will be approximately 1.5 volts above that present in the RUG9, since your power will be applied to the RUG9's full wave bridge, rather than directly to the RUG9's ground. This usually will be of little consequence since all external connections to the RUG9 are optically isolated from the RUG9's DC bus except for the battery connections on the loop/charger board. If you have a sleep board installed, you must power the card cage using the power and ground pins on the sleep board.

Expanding A RUG9 RTU

A RUG5/9 RTU can have up to 7 card cages attached to it by daisy chaining them together using 34 pin ribbon cables as shown below. The ribbon cables should be kept as short as possible to avoid noise problems. Each card cage must be powered separately, only minor power is carried over the ribbon cable.

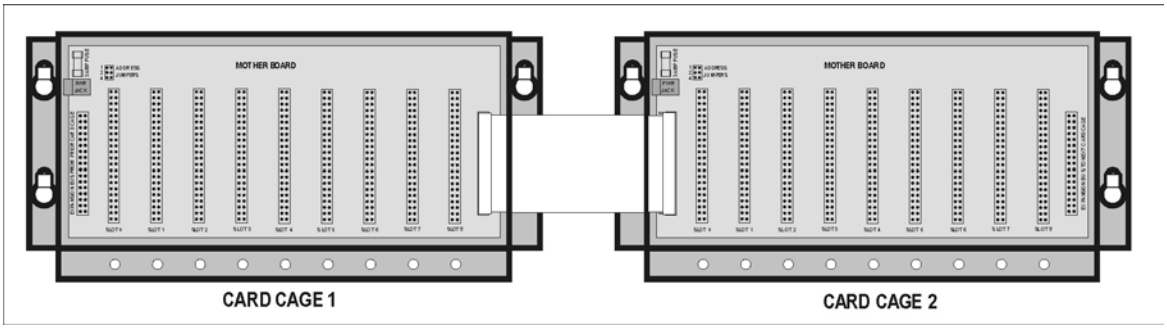


Figure 32 Adding A Card Cage

CPU

The RUG5/9 CPU contains the MC68331 microprocessor, flash memory for storing both the operating system and user's application file, battery backed up RAM, real time clock/calendar, display/keyboard interface, watchdog timer, an RS232 port, and some glue logic. The figure below presents the CPU card block diagram.

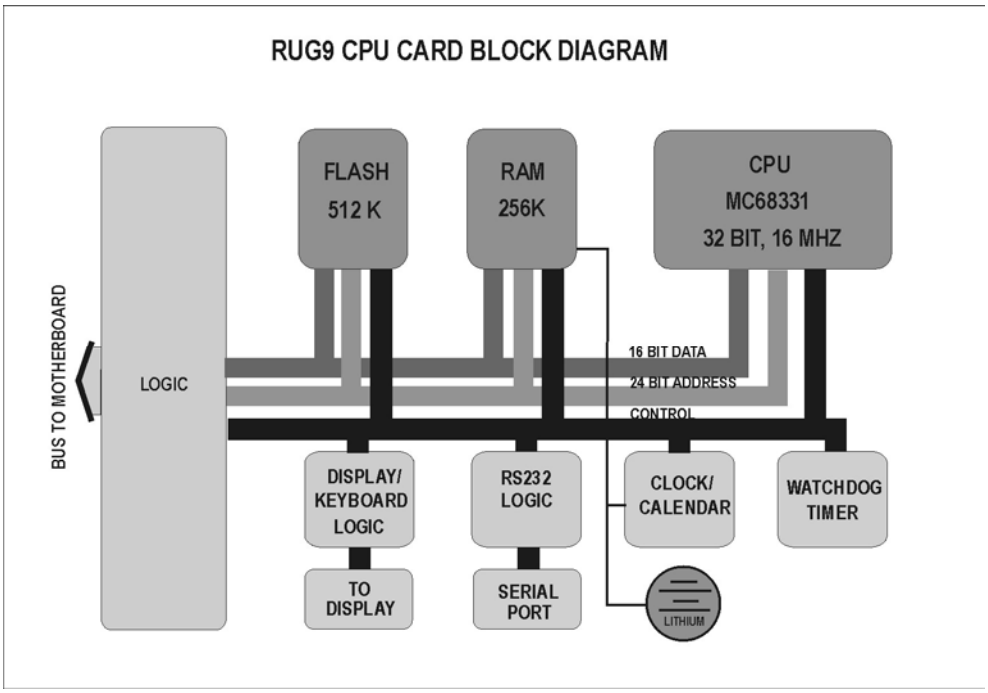


Figure 33 RUG5/9 CPU Block Diagram

There are no user installed components on this board except the lithium battery necessary to back up the RAM and real time clock/calendar to keep them powered in the event of a power outage. The battery required is a CR2032 type, sufficient to power the Ram and clock/calendar for up to 3 years of power outage. As long as the card cage has power applied, the lithium battery is not depleted.

Display Interface

The display/keyboard module connects to the CPU board using a 20 pin ribbon cable. The ribbon cable connects to the shrouded connector near the top of the CPU board. **VERY IMPORTANT: do not connect the ribbon cable to the card cage while the card cage is powered.** Remove card cage power before attaching the cable, then power up the card cage. Be careful to observe the proper polarization. For proper installation, the ribbon cable polarization key should face the right as you are looking at the front of the CPU board.

Reset Button

Near the center of the CPU board, a reset button enables you to stop the RUG5/9 program and return to the monitor mode from which you can reload the program, set the clock, etc. The button is recessed to guard against inadvertent resetting. To reset the unit press the button with a thin pointed object such as a ball point pen. Once you do so, you should see the following menu appear on both the LCD and on the CPU's serial port. If you take no further action, the program will restart automatically after 60 seconds

```
*** WELCOME TO RUGID MONITOR ***
Revision Version 4.31 10/08/99
Date/Time Fri 10/30/99 16:17:21

Installed program:
DemoAlone.R9C  10/15/99 12:13:16 PM

1 Start program
2 Set realtime clock/calendar
3 Change LCD contrast
4 Toggle LCD normal/reverse
5 Read memory location
6 Clear all memory
Enter choice…
```

RS232 Port

The DB9 connector near the bottom of the CPU board is the RS232 port included with all units and is used for local program and OS loading. It is always set for 9600 baud, no parity, 8 bit word, 1 stop bit.

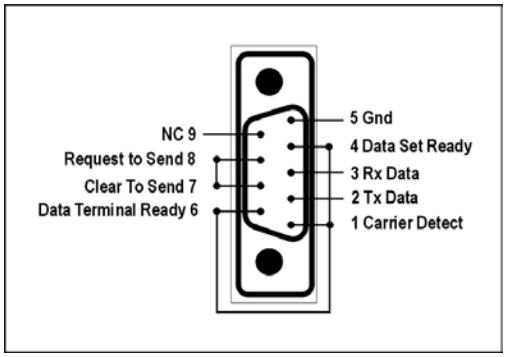


Figure 34 CPU RS232 Pinout

Flash Memory

The CPU's flash memory does not require power to retain its contents indefinitely. It holds the RUG5/9 operating system and user configuration file in partitions as defined in the following table:

Table 3 Flash Memory Partitions

Sector 0, 16K	Boot loader, programmable only at factory
Sector 1, 8K	Operating system, loadable in the field
Sector 2, 8K	
Sector 3, 96K	
Sector 4, 128K	
Sector 5, 128K	User configuration file, loadable in the field
Sector 6, 128K	

Changing the Battery

If the RUG5/9 is presently unpowered, power it up momentarily, and then remove power. This will charge the capacitor that will power the clock/calendar and RAM while the battery is being replaced. Then, remove the CPU card from the card cage and remove the battery from its holder. Install a fresh 3 volt CR2032 lithium battery in the holder with the positive terminal up. Re-install the CPU card in the card cage and 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.

Display/Keyboard Module

The LCD module is composed of a 320 by 240 pixel LED backlit LCD display, membrane keyboard with tactile feedback, controller/backlight driver board and steel enclosure. The display/keyboard module can be mounted on the RUG9 card cage using the display hinge and latch kit; or it may be mounted remotely up to five feet away from the RUG5 or RUG9 card cage on a 20 pin ribbon cable. The keyboard membrane overlay seals the front of the module against dust and moisture. For remote mounting, longer mounting studs are supplied on the back plate to accommodate thick panels and the supplied neoprene rubber gasket. Mounting the module on the supplied rubber gasket against a panel will seal the rear of the module. The display module should not be opened except by the factory as there are no user serviceable parts or maintenance items in the module. See Chapter 10 for display dimensions and mounting hole pattern.

Figure 35 LCD/Keyboard Module

Loop Supply/Charger/Diagnostics Board

Power for analog loops, isolated digital inputs, and to charge a battery can be obtained from the loop supply/charger/diagnostics board. It boosts the unregulated 12 volt bus to provide a regulated and isolated 24 VDC output of up to 160 ma. for analog loops and for digital inputs. The battery charger provides up to 320 ma. to charge an external lead acid battery. Three LED's on the board indicate the presence of bus voltage, loop power and battery charging. In normal operation, the bus LED and loop power LED should glow brightly. If the bus voltage LED is off then power to the card cage has failed or the card cage fuse has blown. If the loop power LED is off, then the loop supply has failed or its fuse has blown. The battery charger LED should glow brightly when substantial current is being sent to the battery. If the battery is disconnected or is fully charged, the LED should not glow. A dim glow indicates trickle charge current is present. This board also measures bus voltage, battery voltage and onboard temperature with a 12 bit A/D converter. Early versions of this board have two 2AG fuses, a 3 amp fuse for the charger and a 1/2 amp fuse for the loop supply. Later versions use resettable fuses that open up on over current and close again when they cool off. They look like large disk capacitors soldered onto the board and do not need to be replaced after an over current. Simply wait a few minutes for them to cool and they will reconnect. The drawing below shows the fuse locations. There are no other user changeable components on the board. The block diagram below illustrates the functions of this board.

To use the board's diagnostic capabilities, you will need to include a diagnostics software module in your project. The diagnostics software module is included in the I/O + System tab of the R9 Module Library. It provides calibrated bus voltage, battery voltage, temperature, low battery alarm and low bus alarm outputs.

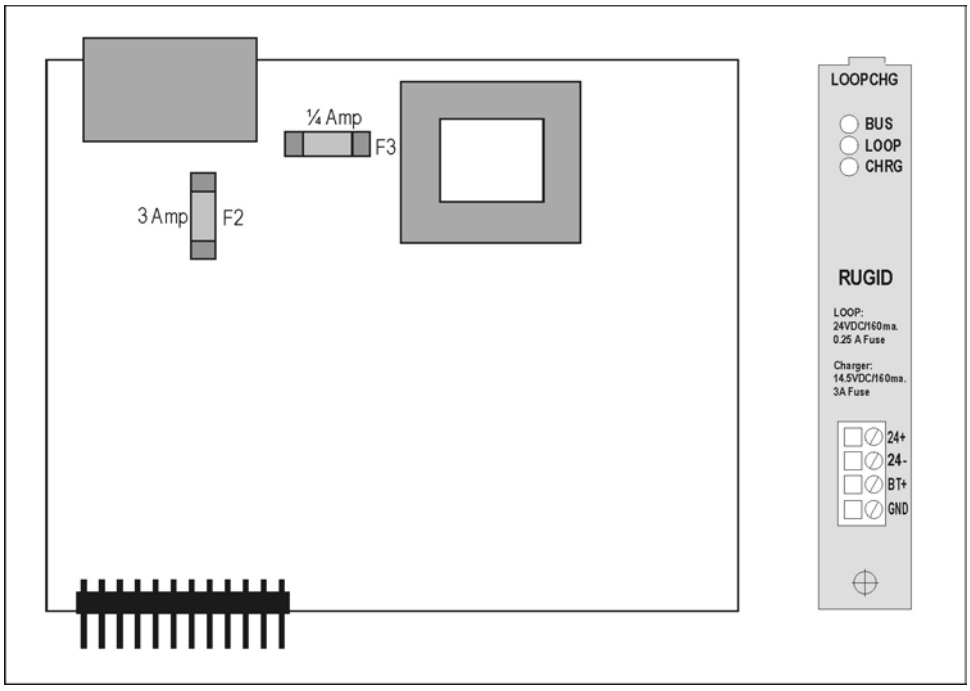


Figure 36 Loop Supply Board Fuse Locations

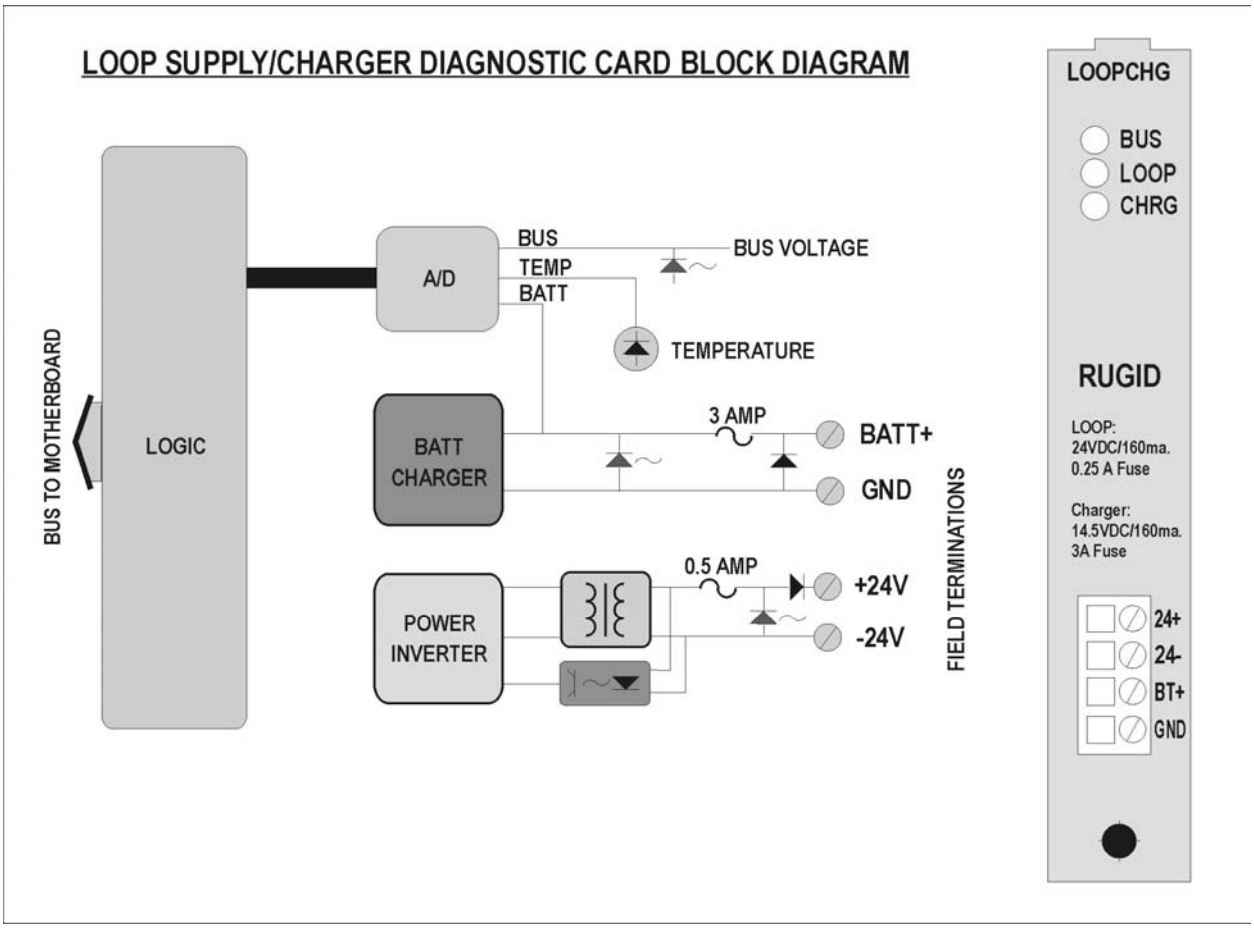


Figure 37 Loop Supply/Charger Block Diagram

Digital Input Board With Common Pin

The digital input board inputs are optically isolated from the rest of the RTU, but have a common return as illustrated in the block diagram below. Each digital input can be powered by 12VDC or 12 to 120VAC signals. For DC applications, the most positive voltage must be applied to the channel input, and the more negative voltage to the COM pin. Also, for noise control, safety, and isolation purposes, it is preferred practice not to have both AC and DC signal sensing on the same digital input board as that would require tying an AC source to a DC source return signal on the digital input board's COM pin. If your application requires channel to channel isolation or mixing DC and AC sensing on one board, see the fully isolated digital input board below.

Hardware

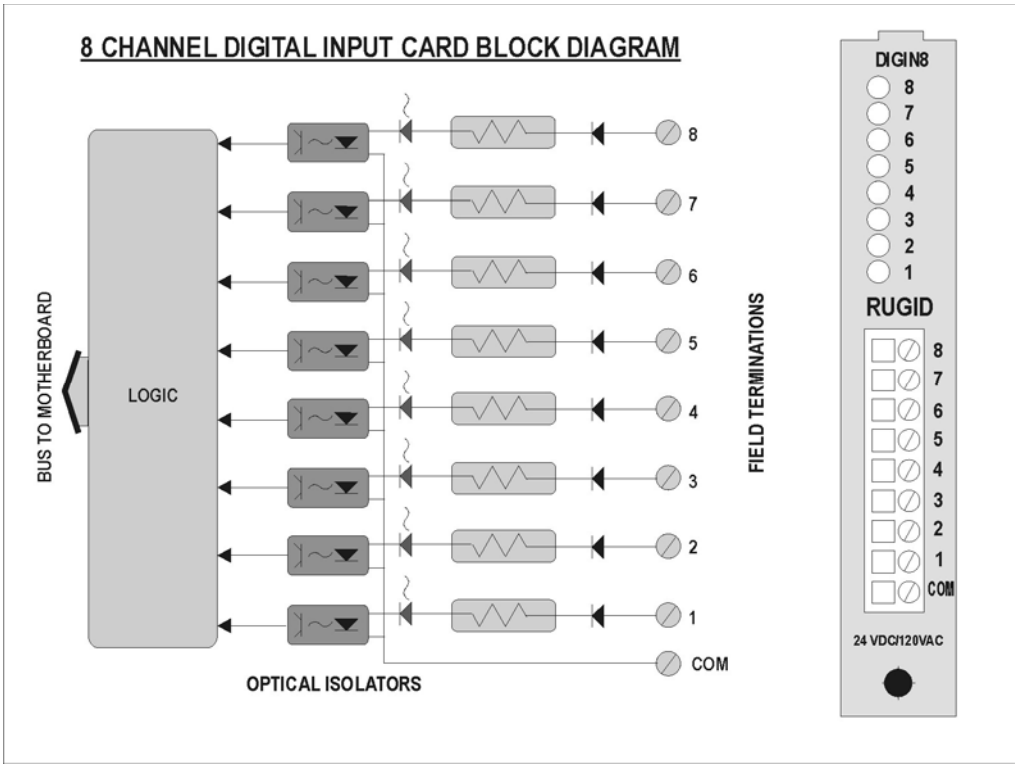


Figure 38 Digital Input Card Block Diagram

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:

Table 4 Digital Input Function Choices

APPLICATION TYPE	MODULE TO USE
Dry contact sensing	Digin DC
AC presence sensing	Digin AC
Counting up to 500 counts per second	DiginCount
Measuring pulse durations	PulseDurationIn

The figure below presents the proper method to sense presence of 120 VAC with the digital input board. In this case be sure to use the DiginAC software module rather than one of the others since the DiginAC module will properly detect presence of AC and give a true reading between individual AC cycles. Also, even though the board is optically isolated from the RUGID bus, the channels use a common terminal to minimize terminal space. Therefore, you should not attempt to read AC signals on the same board that you use for DC or other low voltage signals, since you would have to connect the AC and DC signals together on the common pin. The proper way to use DC signals with the digital input board is shown below. Note that both the 24 volt loop supply and the digital input channels are optically isolated from the RUG5/9 bus so the system remains properly isolated. Each channel draws approximately 3 ma. from the loop supply.

Hardware

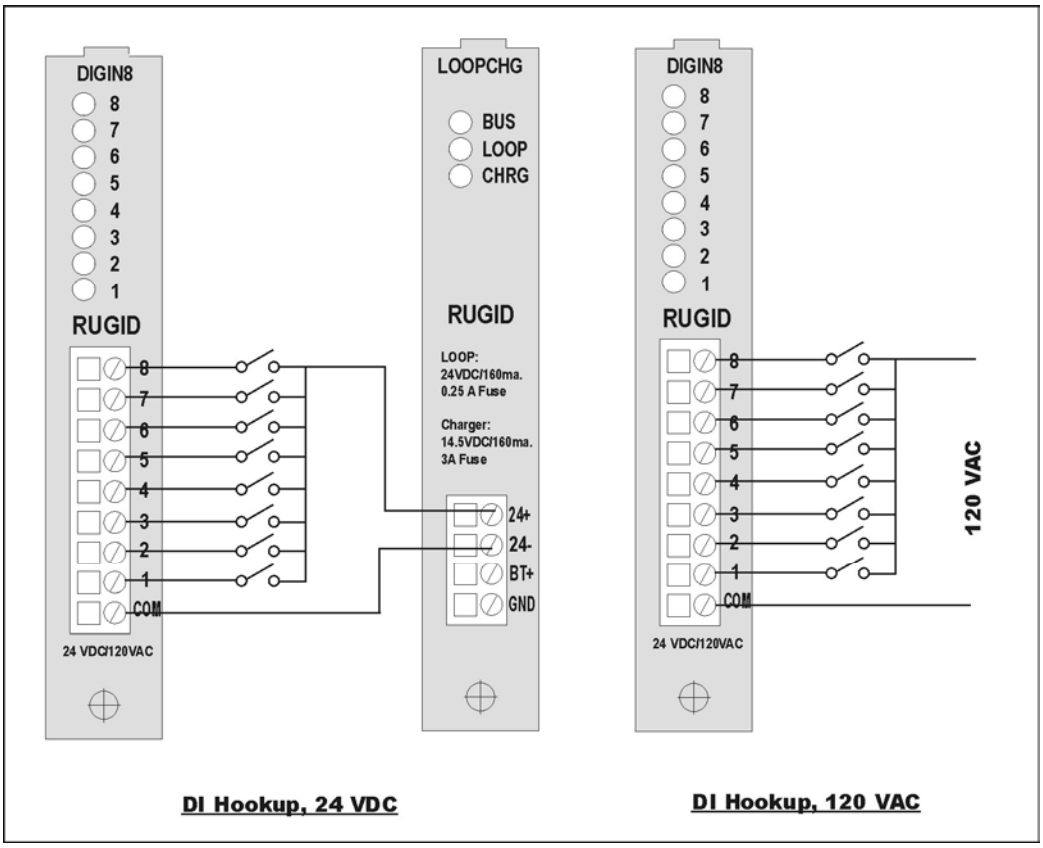


Figure 39 DI Hookup Examples

Fully Isolated Digital Input Board

If you need full channel to channel isolation or need to mix DC and AC type digital inputs, you should use the fully isolated digital input board illustrated in the figure below. Function choices for this board are listed in the table below:

Table 5 Full Isolation Digital Input Function Choices

APPLICATION TYPE	MODULE TO USE
Dry contact sensing	Digin DC
AC presence sensing	Digin AC
Counting up to 256 counts per second	DiginCount
Measuring pulse durations	PulseDurationIn

Hardware

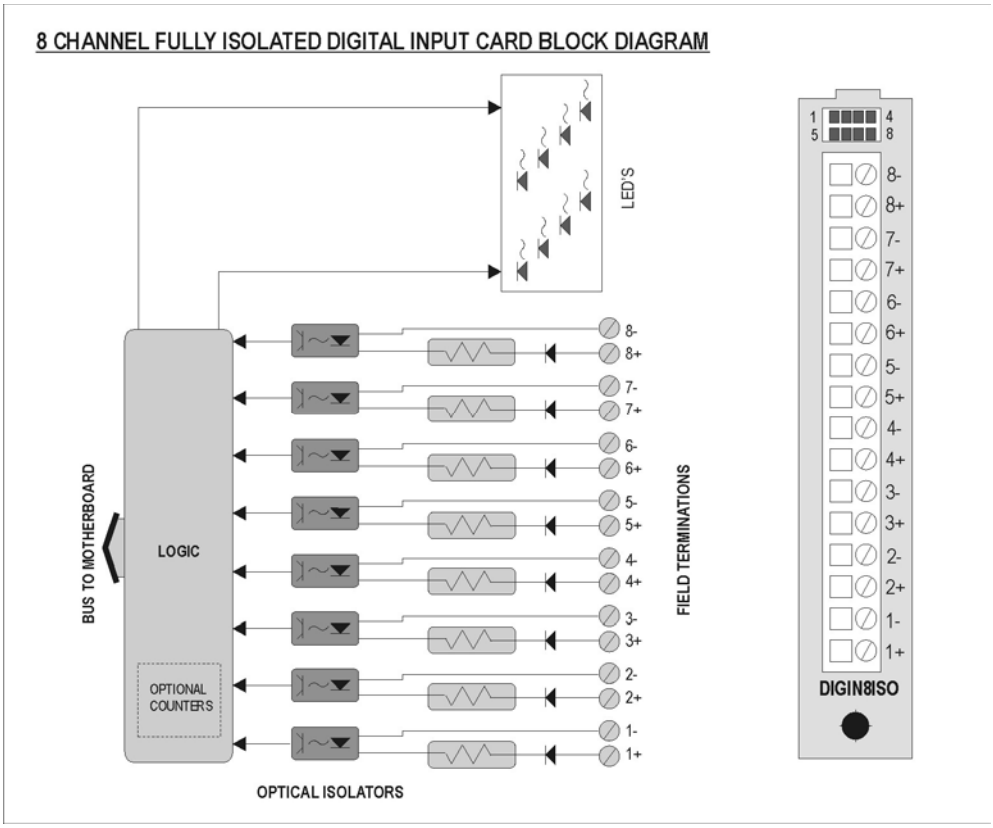


Figure 40 Fully Isolated Digital Input Board Block Diagram

Relay Output Board

The relay output board provides 8 channels of 3 amp relay outputs that are compatible with both 120VAC and 30 VDC. The relays provide 1500 volt isolation between the loads and the RUG5/9 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 6 Digital Output Function Choices

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

On the standard digital output board, the relay common pins are all returned to the board's COM pin. The following block diagram illustrates the board functional design.

Hardware

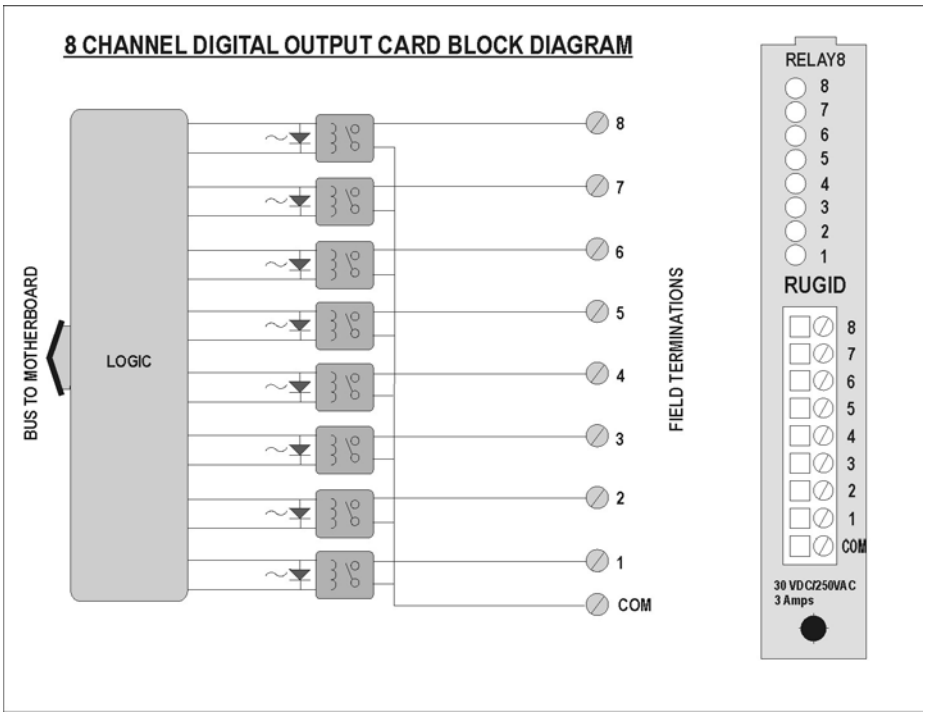


Figure 41 Relay Output Board Block Diagram

Fully Isolated 10 Amp Digital Output Board

If you need greater current capability than 3 amps, or you wish to drive both DC and AC loads with the same board, you should use the fully isolated 10 amp relay output board illustrated below. The available modules to control the digital outputs are the same as in the case of the 3 amp board and are listed below.

Table 7 Digital Output Function Choices

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

Hardware

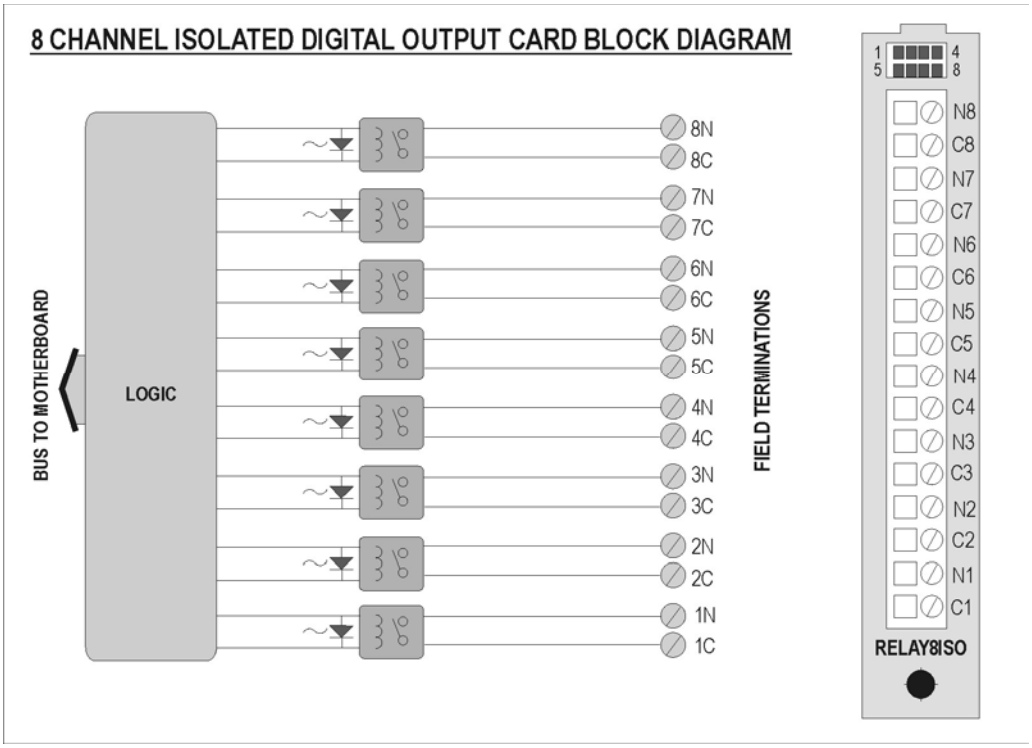


Figure 42 Fully Isolated Relay Output Board Block Diagram

Analog Input Board-8 Channel/12 Bit Resolution

The standard analog input board uses a single eight channel 12 bit A/D converter on the field side of the optical interface to provide accurate analog measurements in the range of 0 to 5 volts or 4-20 ma. The board has onboard precision current sensing resistors that can be engaged by installing push on jumpers, one per channel, to make the selected channel(s) compatible with the 4-20 ma. instrument standard. The jumpers are clearly labeled on the board as assigned to channels 1 through 8. Any channel with the jumper in place, i.e., shorting its two pins together, configures the channel for 4-20 ma operation; any channel with its shorting bar removed or offset so it does not short its two pins together is configured for 0-5 VDC operation. The board is shipped with the push on jumpers installed, so the channels are setup for 4-20 ma. when the board arrives. In addition, in order to maintain field isolation, the board has an onboard inverter to power the field side electronics, so loop current is not necessary for the functioning of the A/D electronics. This means that the channels can be used for 0 to 5 volt and 0 to 20 ma. measurements. Channel calibration for both 0-5V and 4-20 ma operation is held in an onboard EEPROM whose contents are set at the factory. The RUG5/9 reads the EEPROM on boot up and uses the calibration constants to linearize and scale the channel values thereafter in the preprogrammed modules. Depending upon your application, you will want to choose from the following modules to make use of the A/D channels.

Table 8 Analog Input Function Choices

DESIRED FUNCTION	MODULE TO USE
Read 0-5 volt input	Analog Input 0-5
Read 4-20 ma. input	Analog Input 4-20

Hardware

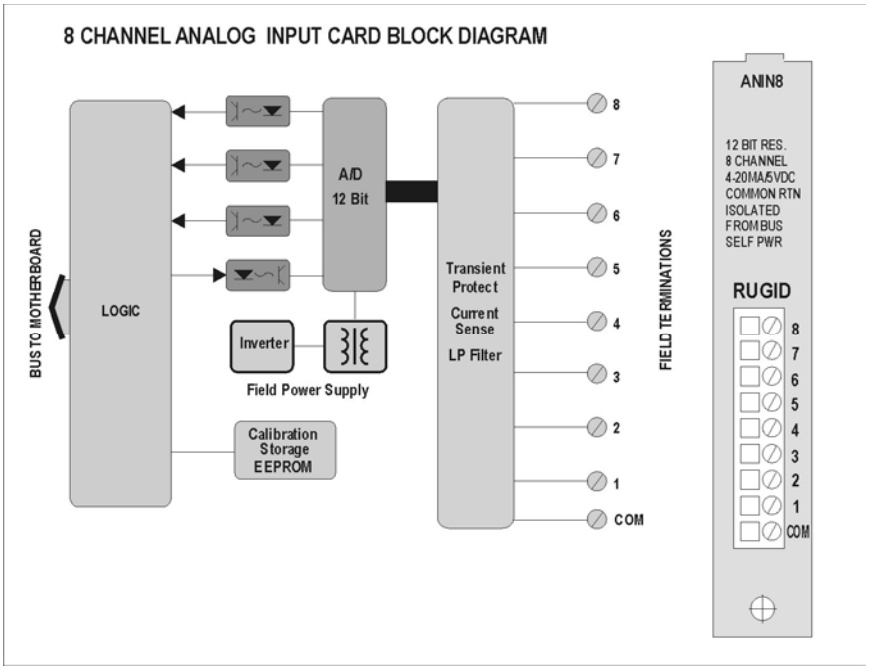


Figure 43 Analog Input Board (8 Chan/12Bit) Block Diagram

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 on the loop supply board and their negative terminals to individual analog input terminals on the analog input board. Be sure to observe the loop supply's capacity of 160 ma total current, or a maximum of eight analog 4-20 ma loops per loop supply board.

Hardware

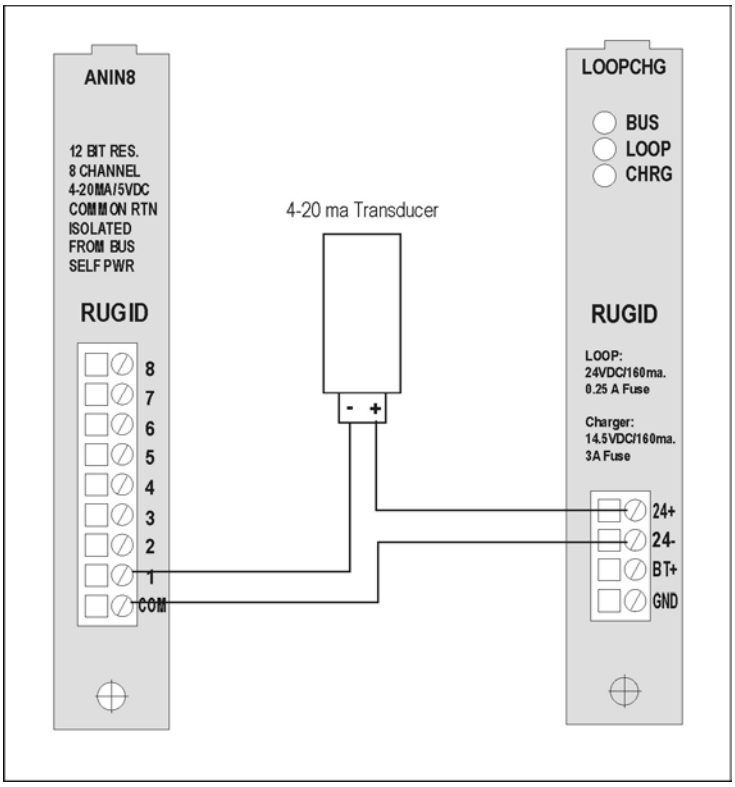


Figure 44 Transducer Hookup (4-20 ma) to 12 Bit Analog Input Board

Voltage type transducers and potentiometers can be connected as shown below. For each transducer you must remove the push on shorting bar for that channel so the RUG5/9 input will have high impedance and not draw excessive current from the transducer. You can mix 4-20 ma type transducers and voltage type transducers without limitation on a single RUG5/9 analog input board.

Hardware

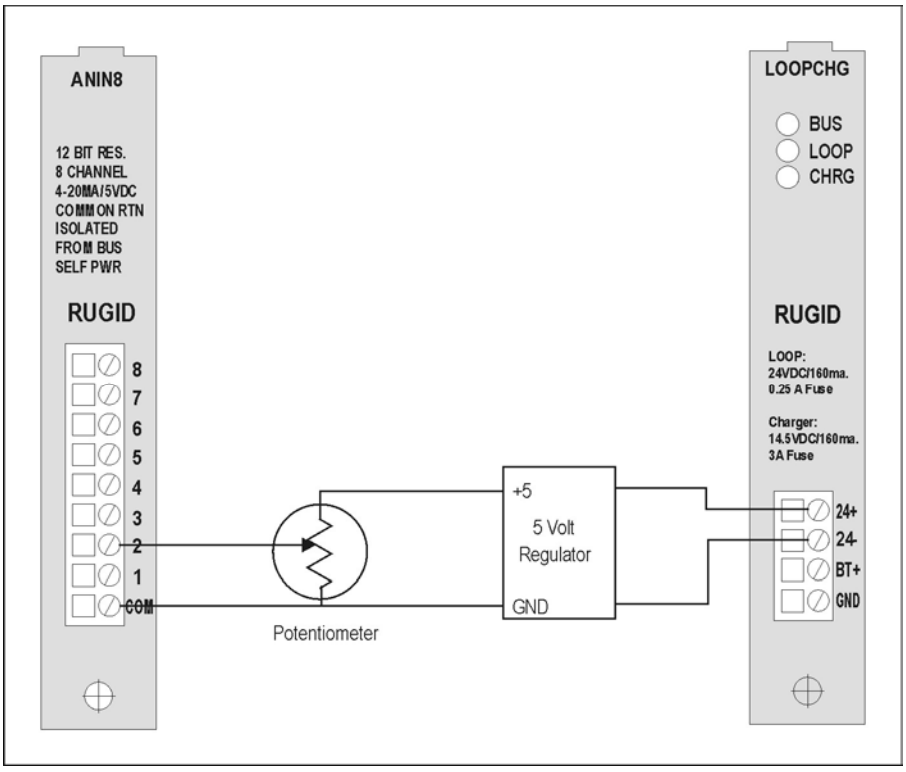


Figure 45 Potentiometer Hookup (0-5Vdc) to 12 Bit Analog Input Board

Analog Input Board-4 Channel/16 Bit Resolution

The high resolution analog input board uses four 16 bit resolution A/D converters on the field side of the optical interface to provide accurate analog measurements in the range of 4-20 ma. The board has onboard precision current sensing resistors to make the channels compatible with the 4-20 ma. instrument standard. Channel calibration for 4-20 ma operation is held in an onboard EEPROM whose contents are set at the factory. The RUG5/9 reads the EEPROM on boot up and uses the calibration constants to linearize and scale the channel values thereafter in the preprogrammed modules. You must use the "Analog Input 4-20" module for each of the A/D channels. The figure below presents the board's block diagram.

Hardware

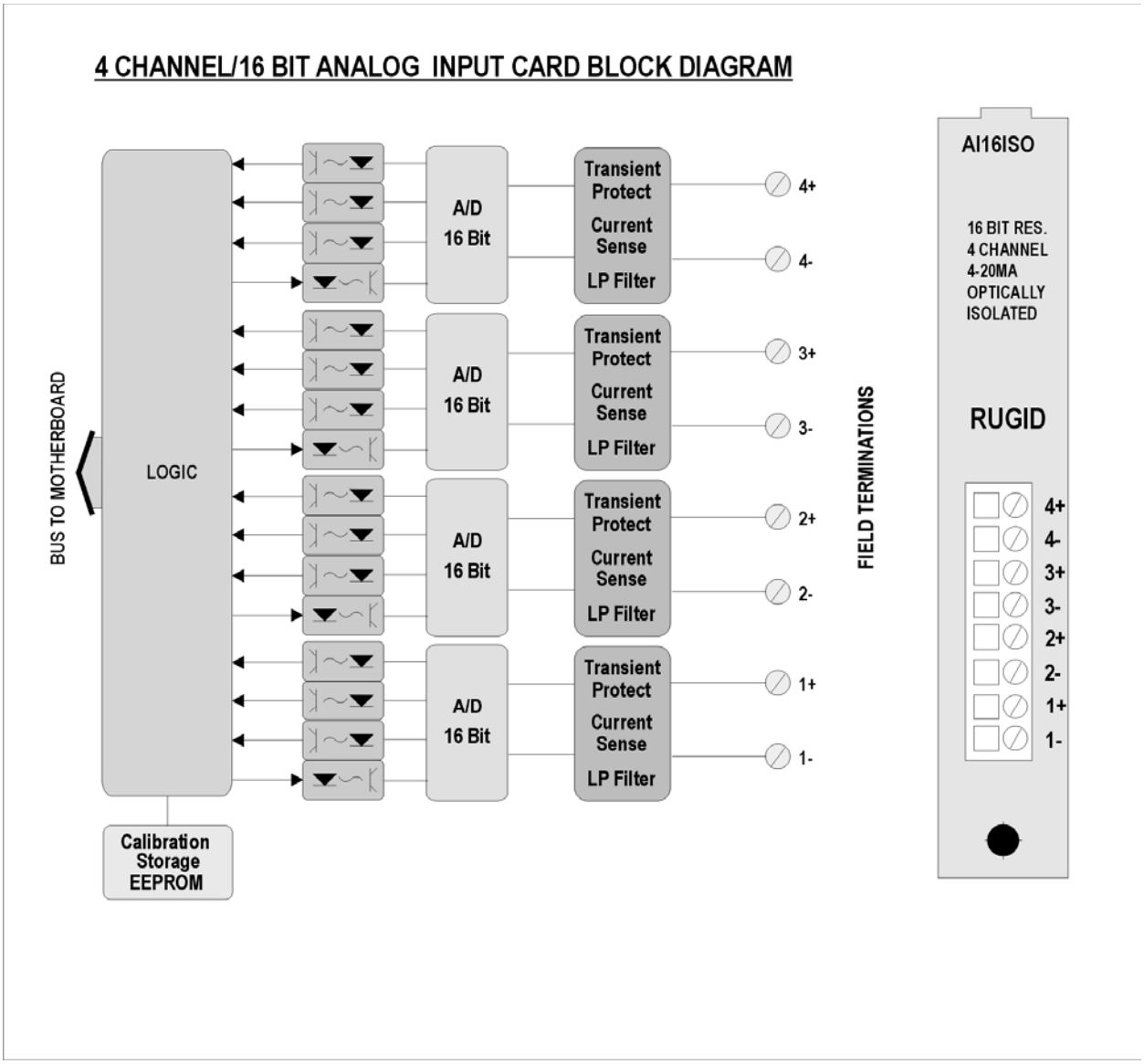


Figure 46 High Resolution Analog Input Board Block Diagram

This board is compatible with 4-20 ma only, since the field side of the opto-isolator is loop powered. A typical hookup using the RUG5/9 loop supply is presented below. Because of the A/D converter's high resolution, it is imperative that you observe low now noise wiring practices.

Hardware

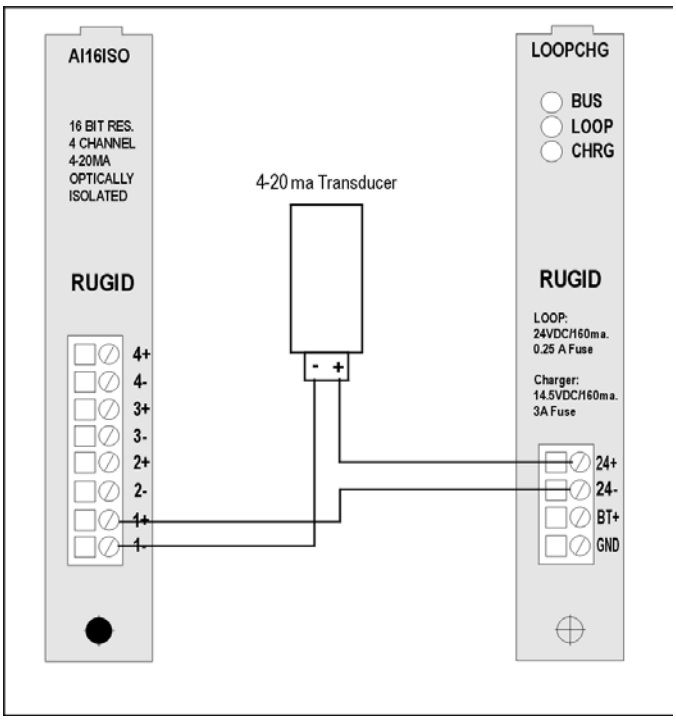


Figure 47 Transducer Hookup (4-20 ma) to High Resolution Analog Input Board

Analog Output Board

The analog output board provides four channels of 12 bit D/A conversion complying with the 4-20 ma. standard. There is a single D/A converter per channel, isolated from the RUG5/9 bus and from any other channel. The channels are loop powered, so appear to other equipment to be the equivalent of a 2-wire 4-20 ma. transducer. Channel calibration is held in an onboard EEPROM whose contents are set at the factory. Each channel will require 12 volts from its loop, and can tolerate up to 48 volts across its output. A blocking diode on each channel will protect the channel from reverse application of loop voltage. The RUG5/9 reads the EEPROM on boot up and uses the calibration constants to linearize and scale the channel values thereafter in the preprogrammed modules. You must use the "Analog Output" module to control each D/A channel.

Hardware

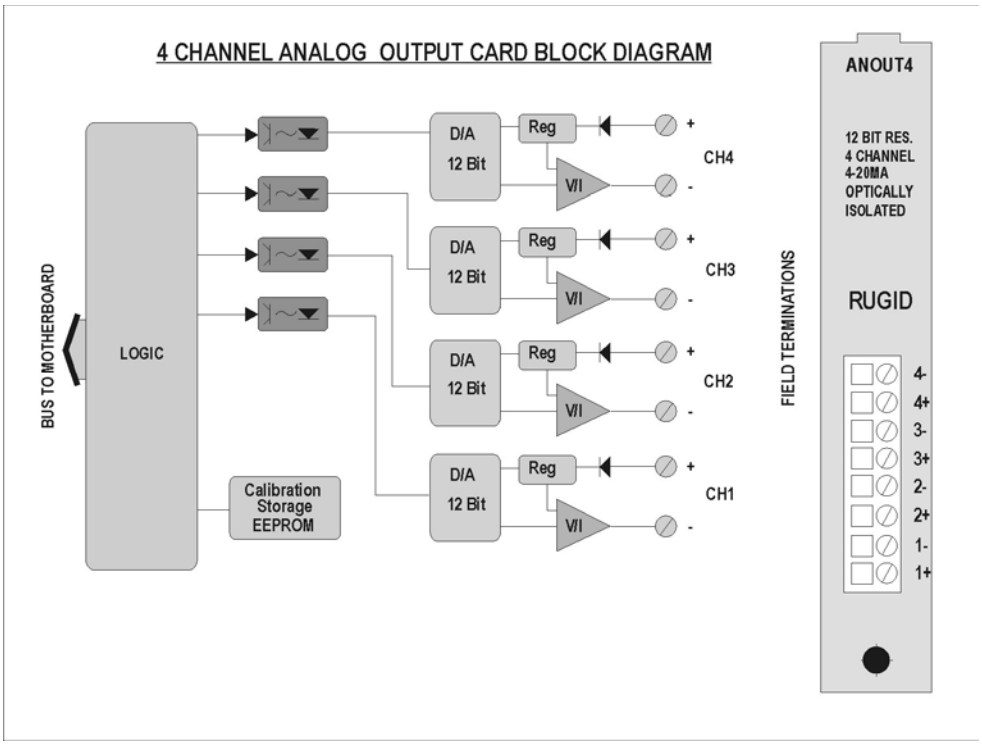


Figure 48 Analog Output Board Block Diagram

The proper way to connect a RUG5/9 analog output channel to another device is presented in the figure below. Each analog output channel is completely isolated from the RUG5/9 and any other channels in the unit; and is the equivalent of a two wire 4-20 ma transducer where external equipment is concerned. Therefore, it is important to note that the analog output does not supply power to the load. Instead, it must be inserted into a loop with a loop supply providing power.

Hardware

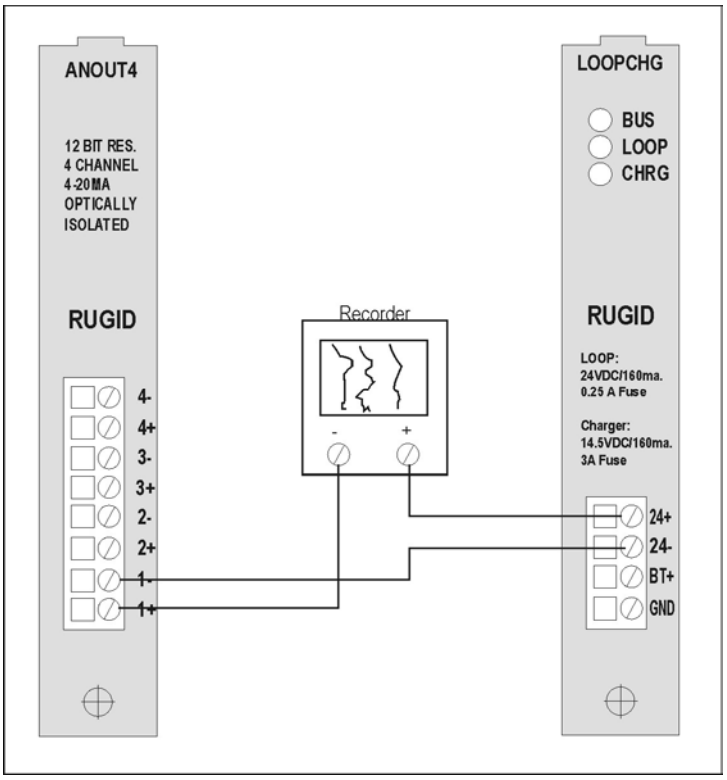


Figure 49 Typical Analog Output Hookup

Modem/RS232 Board

For audio communications, the modem board provides Bell 103, 300 baud or Bell 212, 1200 baud capability compatible with 2-wire or 4-wire wireline channels, or audio radios. An onboard adjustable transmit channel amplifier makes the board able to communicate with both leased lines or customer owned lines. The ring detector and touchtone generator enable the board to dial out or answer 2-wire calls. An onboard keying relay enables the board to work with standard audio radios. The relay and 600 ohm transformers completely isolate the board from field circuits. The choice of 2-wire versus 4-wire operation is made with a single push on jumper on the board. In 2-wire dial up operation, either the modular jack or TX+/- terminals may be used for both receive and transmit. In 2-wire or 4-wire leased line or radio operation, RX+/- and TX+/- must be used. The keying relay employs the KEY terminal and the TX- terminal. For high speed communications, the RS232 port can be connected to radio modems or external high speed phone line modems. The mode, baud rate, tone use, protocol, etc., are set using the **ComSetup** module. The figure below presents the board's block diagram.

Hardware

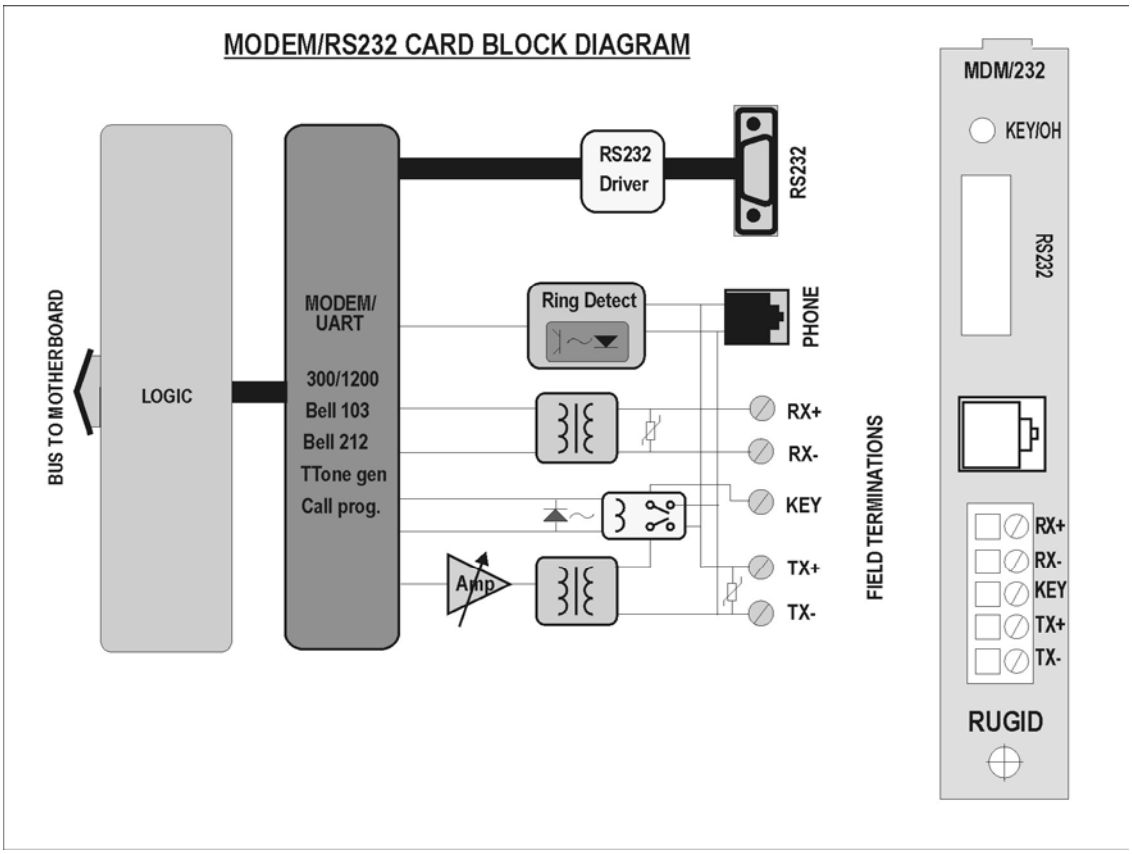


Figure 50 Modem Board Block Diagram

The diagram below presents the user alterable modem board functions. Jumper J6 engages the dialing relay when installed (as delivered) and disables the relay when removed. For RS232 operation, it is advisable to remove the jumper so the relay does not click with each transmission. Jumper J3 selects 2-wire or 4-wire operation. The center pin must be shorted to either the 2W pin or the 4W pin for 2-wire or 4-wire operation respectively. Potentiometer R2 adjusts transmitter amplitude over about a 3 to 1 range. Factory setting of midrange gives 2.0 v p-p, or –0.8 dbm. Full counterclockwise gives minimum amplitude (0.8 v p-p, or -8.7 dbm). Full clockwise gives maximum amplitude (2.4 v p-p, or +7.9 dbm). Transmit impedance is 600 ohms when transmitting; infinite when receiving. Receive impedance is 600 ohms.

Hardware

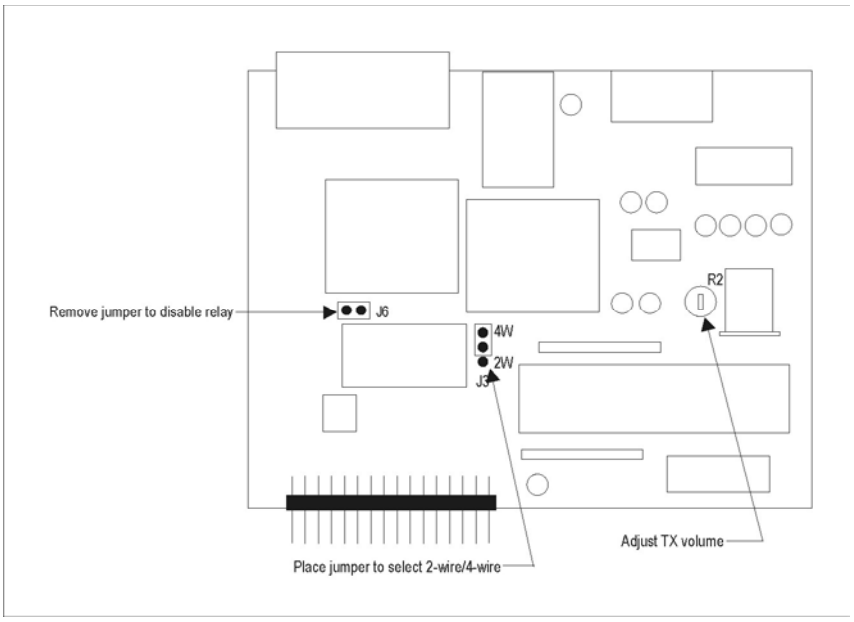


Figure 51 Modem Board User Alterable Features

The diagram below presents the RS232 connection to the modem board. The cable indicated is the standard cable available from RUGID, part number CBL232.

Hardware

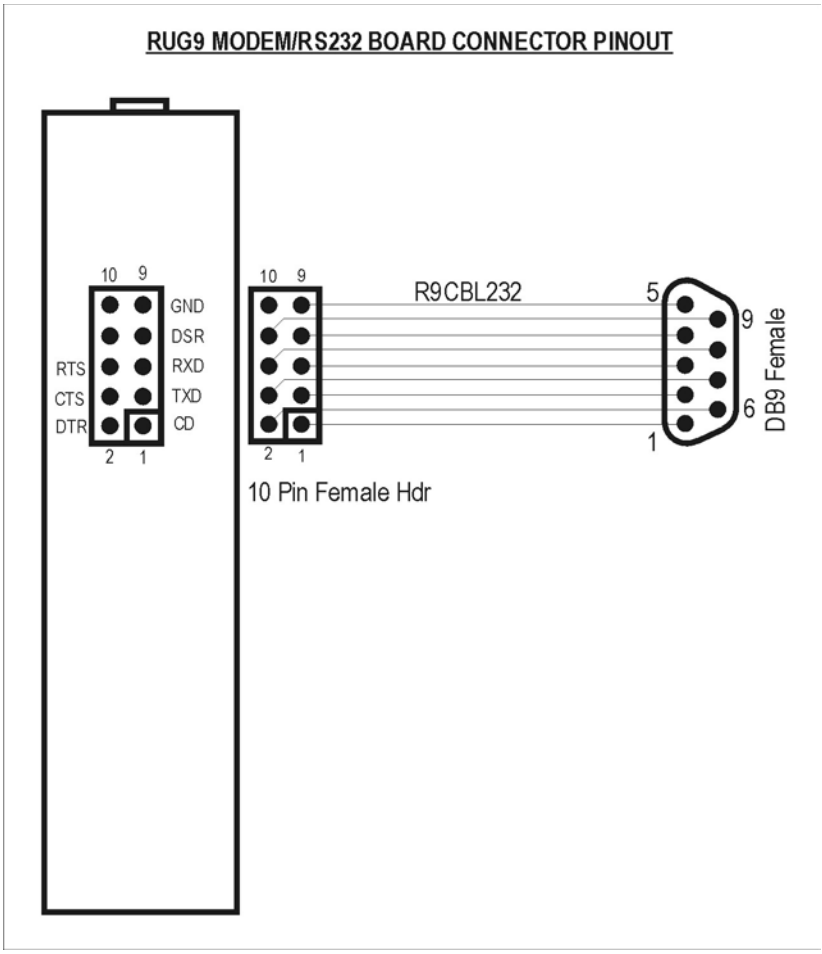


Figure 52 Modem Board RS232 Connections

Further examples of typical modem connections are presented in the following figure.

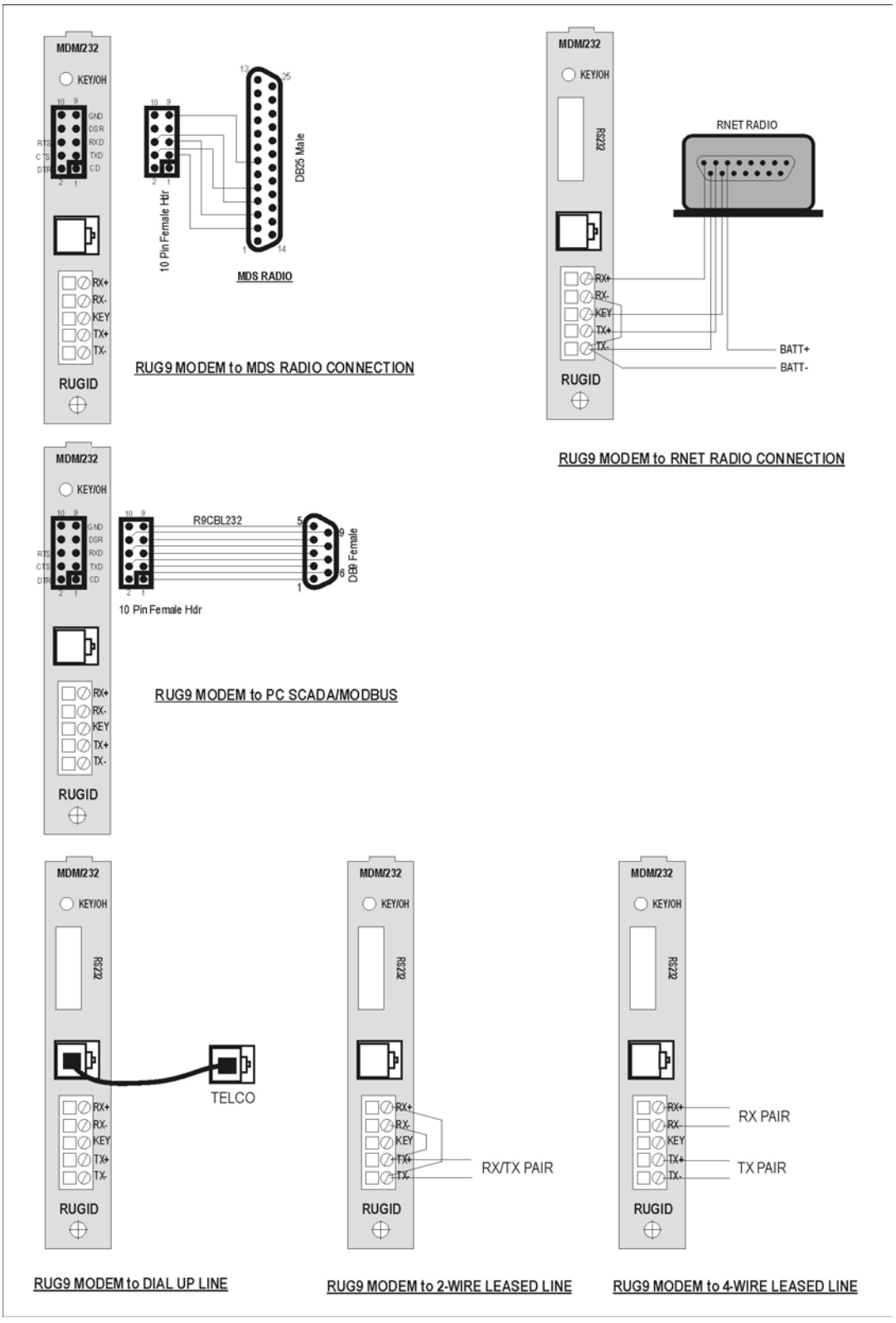


Figure 53 RUG5/9 Modem Typical Connection Examples

Speech Dialer Board

The speech dialer board interfaces the RUG5/9 to a standard dialup phone line to provide a speech interface to a user who has either called in, or who has been called by the RUG5/9. The dialer board includes speech recording and playback of up to 256 addressable messages. Each message can be of any length up to the remaining speech capacity of the board's onboard flash memory. The board also has phone line dialing capability, and touchtone transmission and decoding capability. The phone line must be connected to the onboard modular jack. When the RUG5/9 goes off hook, the KEY/OH LED will illuminate. Control information such as phone numbers to be dialed, delays, rings to answer, and which specific messages are to be spoken are set up using software modules in the communications page of R9SETUPD. An autodialer example configuration file, named "Autodialer" is included in the R9SETUPD project folder. Speech is recorded using the built in microphone in conjunction with the RUG5/9's menu system using either the RUG5/9 LCD display and keyboard or a terminal connected to the CPU's RS232 port. Up to 256 messages of varying length can be stored with a total speech storage time of 12 minutes. Storage uses FLASH memory so no batteries are required to hold speech indefinitely. Emitted speech can be heard by connecting an external 8 ohm speaker to the SPKR and COM terminals. Dialer board output volume is controlled by an onboard potentiometer. Be aware that the potentiometer controls both the speech volume as well as the touchtone transmission volume. If you turn the speech volume down, you may attenuate the touchtone amplitude too low to dial. To decrease the speaker volume without affecting the touchtone volume, install a resistor in series with the speaker. An 8 ohm resistor will cut the volume in half; a 16 ohm resistor will cut it to one third normal volume, etc. If you wish to transmit speech to a radio, the dialer board can be connected to an audio radio to emit speech to be received on other radios tuned to the same frequency. In that case, the KEY line is used to key the radio. The following diagram illustrates the location of the volume control potentiometer on the dialer board. Clockwise corresponds to full volume; counterclockwise gives zero volume.

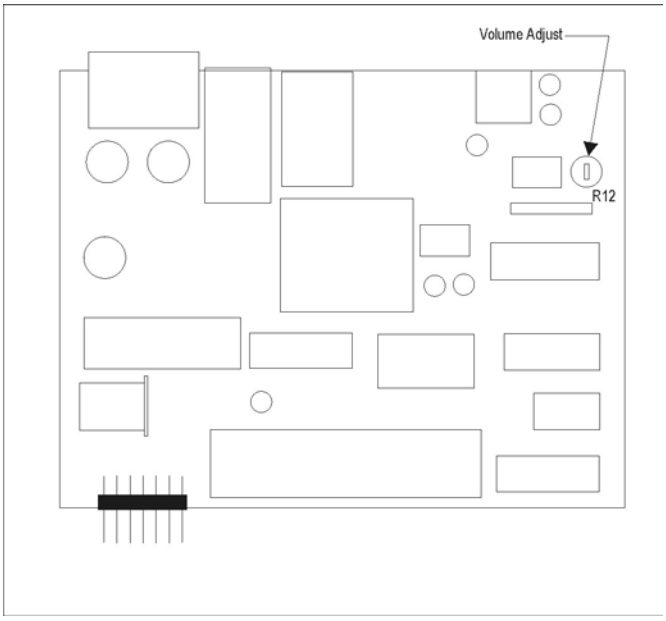


Figure 54 Dialer Board Volume Adjustment Potentiometer Location

Hardware

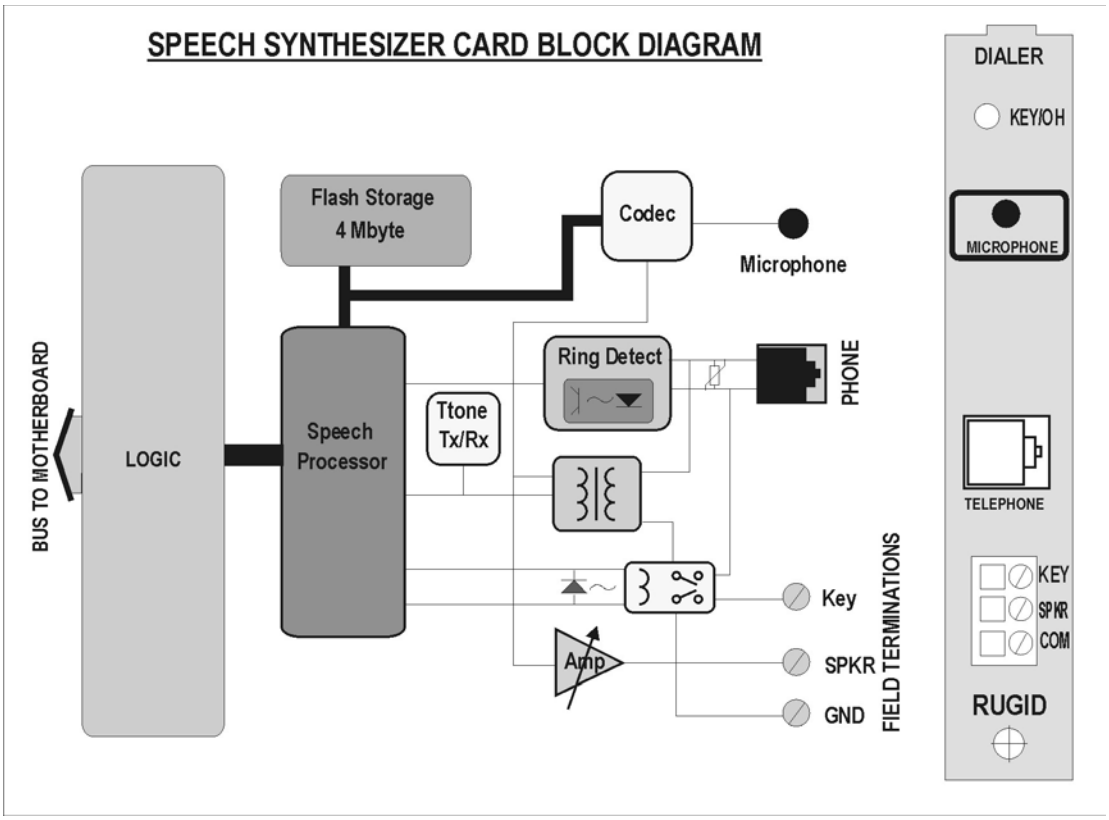


Figure 55 Dialer Board Block Diagram

Sleep Controller Board

The sleep controller applies or removes power to the rest of the RUG5/9 under control of software modules. When you use the sleep/controller board, power to the unit is applied through the sleep/controller board VIN and GND terminals, rather than to the card cage or loop supply/charger board. While asleep the RUG5/9 does no processing and draws no power; all processing is done by the sleep/controller board. This processing is limited to counting time for the next wake up, counting pulses on the sleep/controller board's digital inputs and anemometer input, comparing the sleep/controller board's analog inputs against setpoints, and watching for touch tones and ringing to wake up the RUG5/9 processor. These choices are made using the sleep software module in the controls category of R9SETUP. The table below defines the module functions. Whenever the sleep board powers up the RUG5/9, the 'WK' LED will illuminate. Similarly, whenever the board energizes the external 'AUX' output, the 'AX' LED will illuminate. The AUX output is intended to energize external equipment such as a radio receiver, in a duty cycled fashion while the RUG5/9 remains asleep. For example, a radio could be energized for one minute per hour to give another site the chance to send a touchtone address to wake the RUG5/9 and enable normal telemetry. When the RUG5/9 is asleep, the DI1 through DI4 LED's will remain extinguished. When the RUG5/9 is powered, those LED's will reflect the states of the corresponding inputs. Note that none of the I/O points on this board are optically isolated.

Some I/O pins on the sleep board have multiple uses. Digital inputs DI1/DI2 and DI3/DI4 can be used in pairs to read a shaft encoder. Analog input AI2 also has a clipping amplifier, enabling it to read an anemometer directly. When jumper J4 is installed, digital input DI4 becomes a 5 volt reference output for powering external potentiometers whenever the AUX output is energized. The TIP terminal is connected to the touchtone detector which can receive touchtones whenever the AUX output is energized.

Hardware

Table 9 Sleep Board Functions

Count pulses: anemometer and tipper bucket
Time next wakeup
Read shaft encoder level measurement
Read & test analog inputs (2 ea. 8 bit resolution)
Detect phone line ringing & wake up
Detect touch tones & wake up if address matches
Control external (AUX) radio and touch tone detector
Supply 5 volt reference for potentiometers

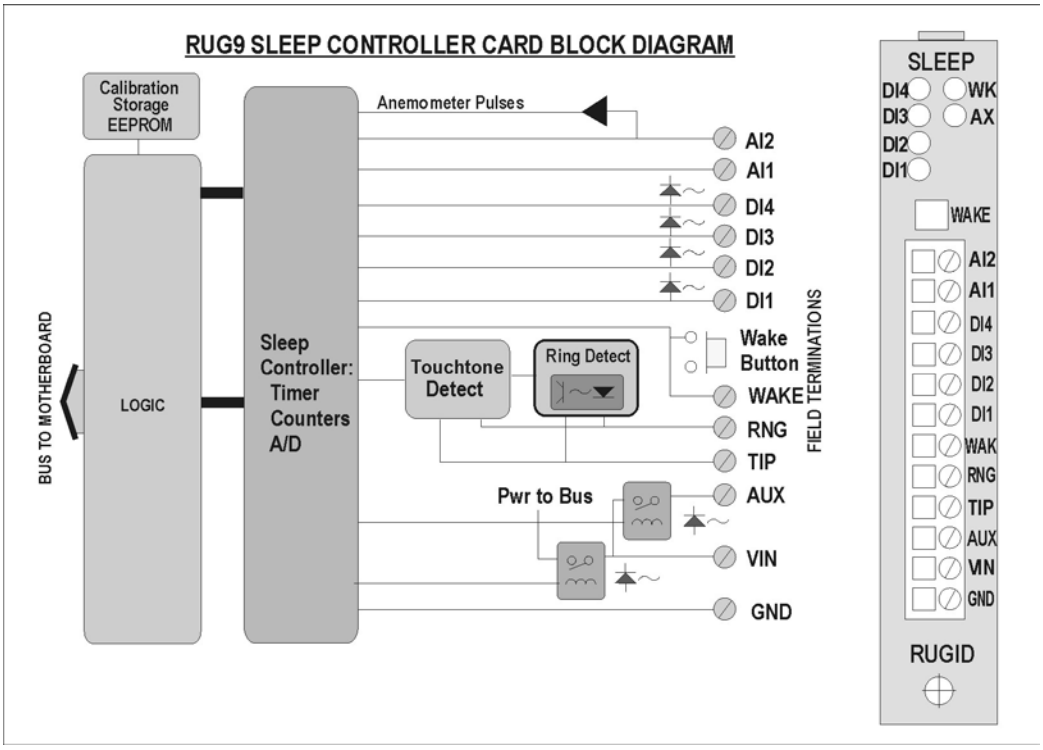


Figure 56 Sleep Board Block Diagram

The following diagram shows the locations of jumpers to engage selectable features.

Hardware

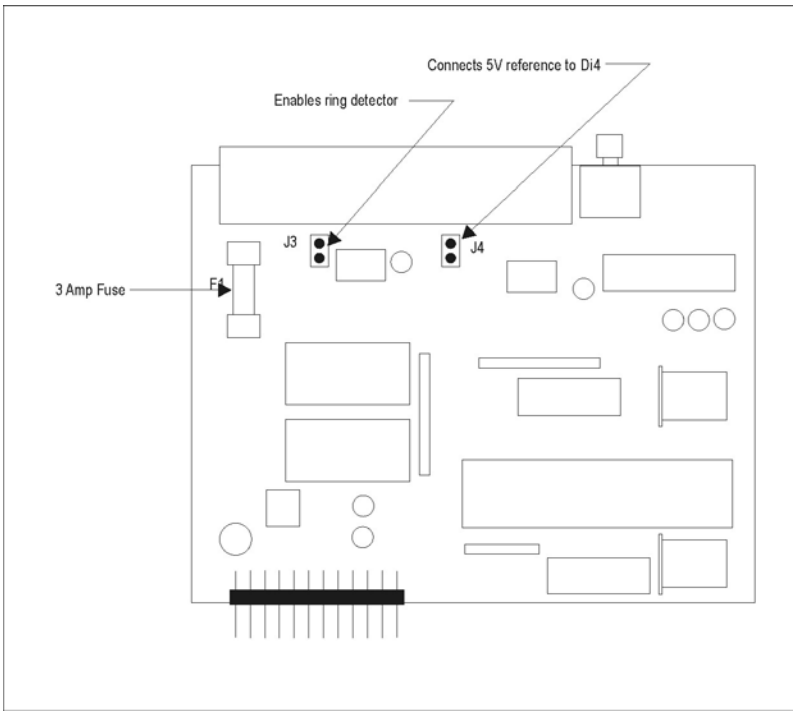


Figure 57 Sleep Board User Alterable Items

Flash Disk Board

The flash disk board accepts a removable flash memory cartridge of 4 Mbyte to 2 Gbyte capacity. The cartridge plugs into the RUG5/9 flash disk board front panel slot and requires no battery to retain its memory indefinitely. Flash cartridges of the Compact Flash standard are compatible with the flash disk board. They are normally used to log field data for long periods. The RUG5/9 stores data as comma delimited ASCII DOS files compatible with Microsoft EXCEL software. Once the data have been recorded, the flash disk cartridge can be removed and plugged into a PCMCIA carrier which, when plugged into a computer's PCMCIA slot, enables the data to be retrieved and saved on suitable archive media. When the flash disk board's 'READY' LED is illuminated, it indicates that the cartridge is seated in its connector properly. The 'ACCESS' LED illuminates whenever the RUG5/9 accesses the cartridge. The cartridge can be inserted and removed while power is applied to the RUG5/9 without harm and is keyed to prevent incorrect insertion. You should not remove the cartridge while the access LED is illuminated or file corruption may occur. RUG5/9 software can create and append to files but cannot delete or rename files.

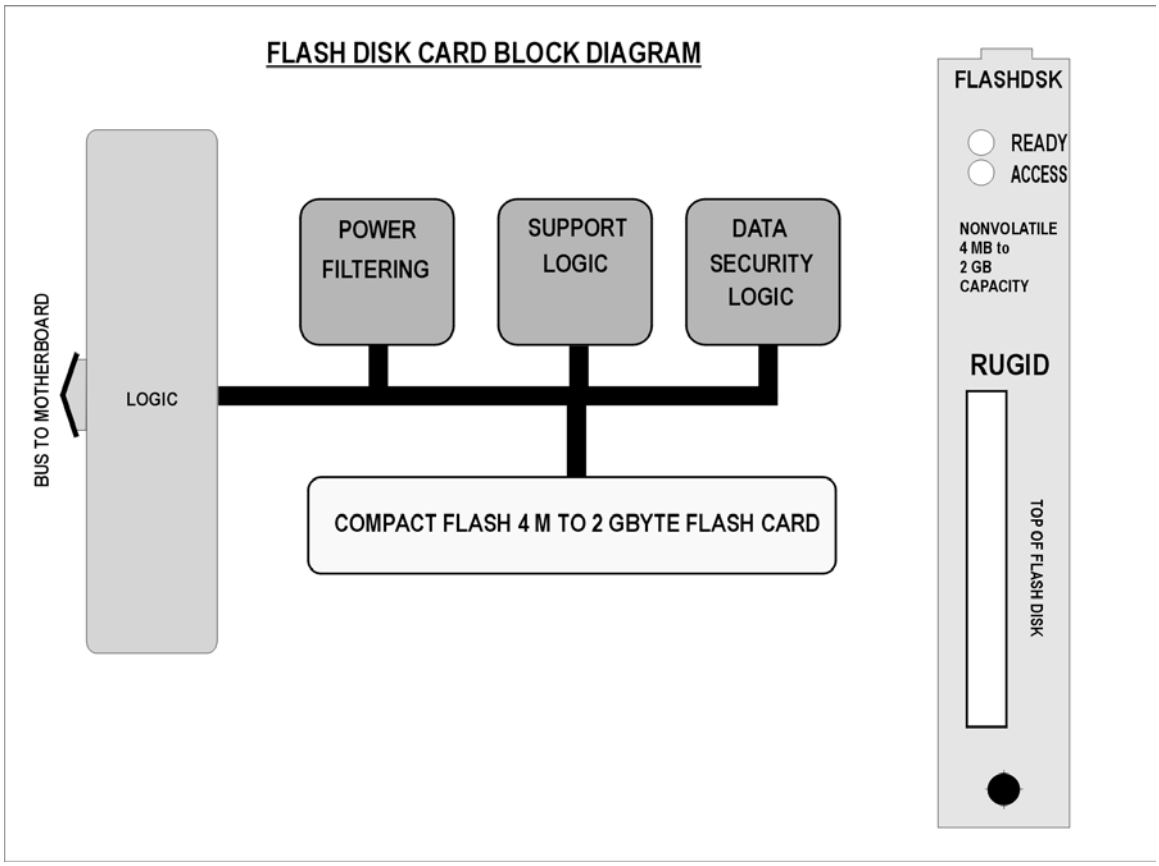


Figure 58 Flash Disk Board Block Diagram

Combo Board

The combo board combines portions of four other boards to provide I/O for many small applications with just a single I/O board. The I/O consists of two 12 bit resolution analog inputs, four digital inputs, two relay outputs, and a loop power supply.

Hardware

Combo Board Analog Inputs

The combo board analog inputs use a single two channel 12 bit A/D converter on the field side of the optical interface to provide accurate analog measurements in the range of 0 to 5 volts or 4-20 ma. The board has onboard precision current sensing resistors that can be engaged by installing push on jumpers, one per channel, to make the selected channel(s) compatible with the 4-20 ma. instrument standard. The jumpers are clearly labeled on the board as assigned to channels 1or 2. Any channel with the jumper in place, i.e., shorting its two pins together, configures the channel for 4-20 ma operation; any channel with its shorting bar removed or offset so it does not short its two pins together is configured for 0-5 VDC operation. The board is shipped with the push on jumpers installed, so the channels are setup for 4-20 ma. when the board arrives. In addition, in order to maintain field isolation, the analog inputs use the loop supply's onboard inverter to power the field side electronics, so loop current is not necessary for the functioning of the A/D electronics. This means that the channels can be used for 0 to 5 volt and 0 to 20 ma. measurements. Channel calibration for both 0-5V and 4-20 ma operation is held in an onboard EEPROM whose contents are set at the factory. The RUG5/9 reads the EEPROM on boot up and uses the calibration constants to linearize and scale the channel values thereafter in the preprogrammed modules.

Combo Board Digital Inputs

The combo board digital inputs are optically isolated from the rest of the RTU, but have a common return as illustrated in the block diagram below. Each digital input can be powered by 24VDC or 24 to 120VAC signals. For DC applications, the most positive voltage must be applied to the channel input, and the more negative voltage to the COM pin. Also, for noise control and isolation purposes, it is preferred practice not to have both AC and DC signal sensing on the same digital input board as that would require tying an AC source to a DC source return signal on the digital input board's COM pin. If your application requires channel to channel isolation or mixing DC and AC sensing on one board, use the fully isolated digital input board.

Combo Board Relay Outputs

The combo board relay outputs provide 2 channels of 3 amp relay outputs that are compatible with both 120VAC and 30 VDC. The relays provide 1500 volt isolation between the loads and the RUG5/9 bus and have large coil to conductor spacing to minimize transient to coil coupling. The relays share a common pin.

Combo Board Loop Supply

Power for analog loops and isolated digital inputs can be obtained from the combo board's loop supply. It boosts the unregulated 12 volt bus to provide a regulated and isolated 24 VDC output of up to 160 ma. for analog loops and for digital inputs. The loop supply's negative pin is shared with the analog input's (ACOM pin) so minimal wiring is required to connect 4-20 ma transducers. An LED on the board indicates the presence of loop power. In normal operation, the loop power LED should glow brightly. If the loop power LED is off, then the loop supply has failed or its fuse has blown. The required fuse is a 2AG, ½ amp fast blow fuse. The following diagrams present the combo board's block diagram and typical connections.

Hardware

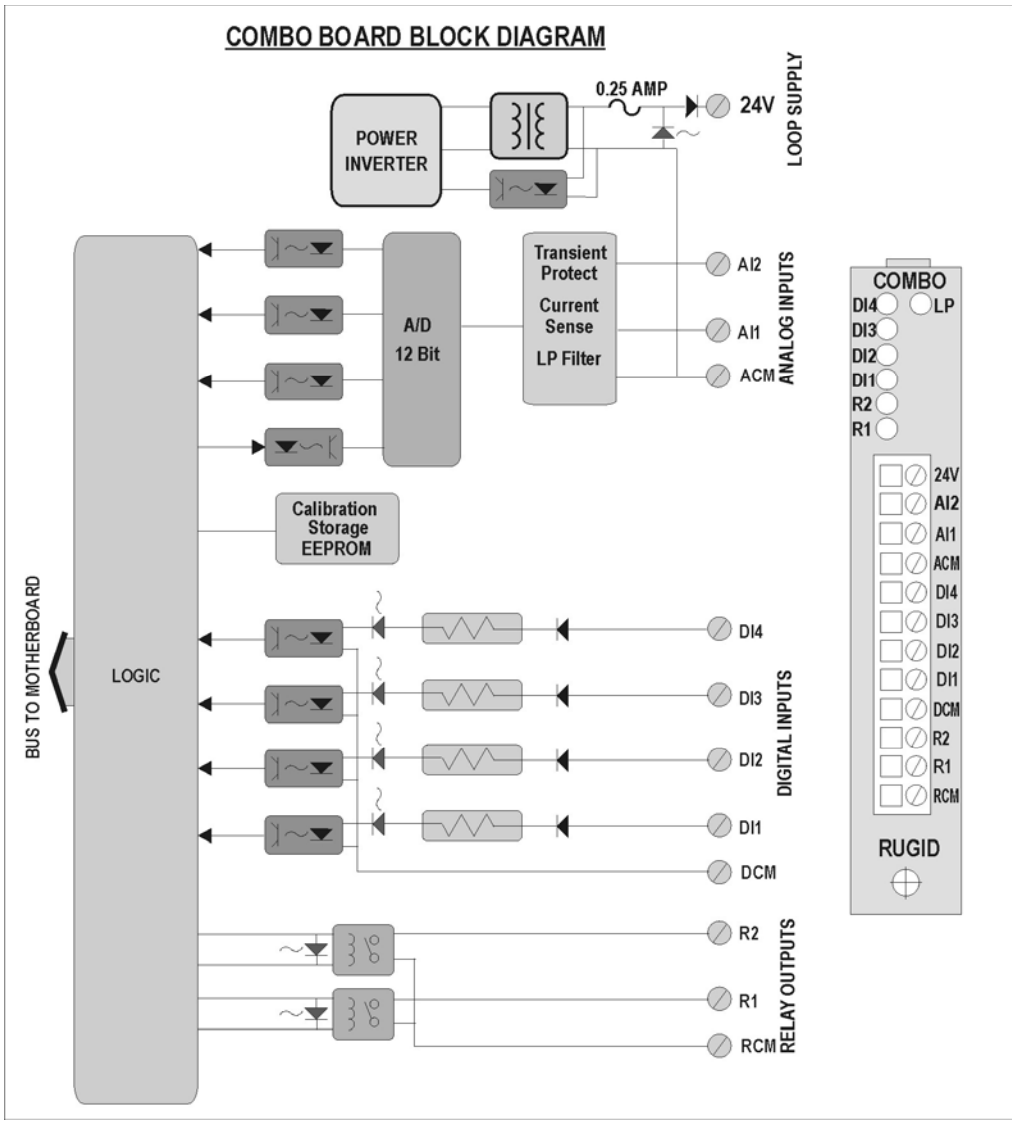


Figure 59 Combo Board Block Diagram

Hardware

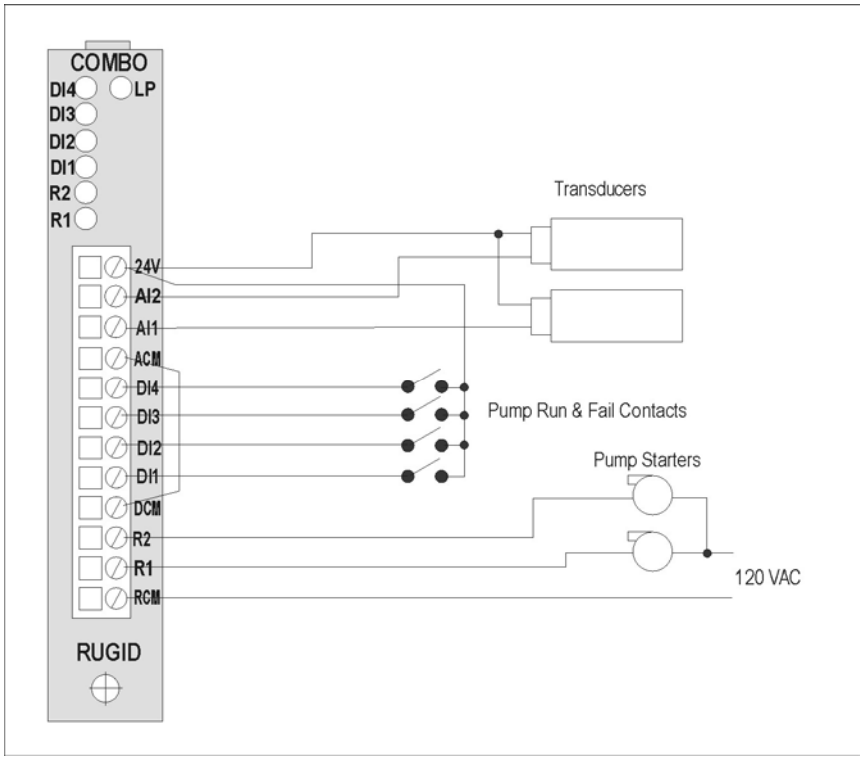


Figure 60 Combo Card Typical Hookup

Dual Serial Port/Printer Board

The dual serial port/printer board shown below provides two RS232 ports and a parallel printer port. Due to board space limitations, the board uses rectangular shrouded headers instead of DB9/DB25 connectors as is standard for these types of ports. Therefore, you will need the pigtail cables (R9CBL232 for serial ports, and R9CBLPRN for the printer port) to convert the shrouded header pinout to standard DB9/DB25 pinout for use with standard PC cables. Standard pigtail cable length is two feet.

Serial Ports

Serial port communication parameters, such as baud rate, parity, word length and stop bit count are set using the **ComSetup** module. You must have a **ComSetup** module for each serial port so that background software knows that the board has multiple interrupt sources. All serial protocols available in the **ComSetup** module (ASCII, RUG6, RUG5/9, MODBUS, etc.) are supported by these ports. All standard baud rates from 50 through 38,400 baud are supported. Other baud rates are supported without error if the value 192,000 divided by the specified baud rate is an integer. Port 1 supports both RS232 and RS485 standards. Maximum recommended line length is shown for each port standard. The alternate configuration is selected using jumpers on the board. The board block diagram presents the jumper configurations for the serial port alternate configurations. See the board layout for the jumper locations.

Hardware

Table 10 Dual RS232 Port/Printer Board Configuration Options

Port on board	Port Address on Board	Standard Configuration	Alternate Configuration
RS232-1	1	RS232, 50 ft.	RS485, 4000 ft.
RS232-2	2	RS232, 50 ft.	None
Printer	3	PC Parallel, 20 ft.	None

Printer Port

The printer port conforms to the Centronics standard PC parallel port for driving printers. The port cannot be used to connect other devices such as Zip drives, etc. You must use a "PrnSetupWatch" module to define for the compiler that a printer port exists and what size print spooler to allocate for it. That module will provide indications of printer busy, printer failed and out of paper. Note that the printer port will not work on boards installed in slot 8 of the card cage due to a port limitation of the RUG5/9 boot loader code. The serial ports can be used in slot 8. To have the RUG5/9 print a report to a printer, simply define a display for the printer port, as if it were a serial port, and trigger it when you want it to print.

Hardware

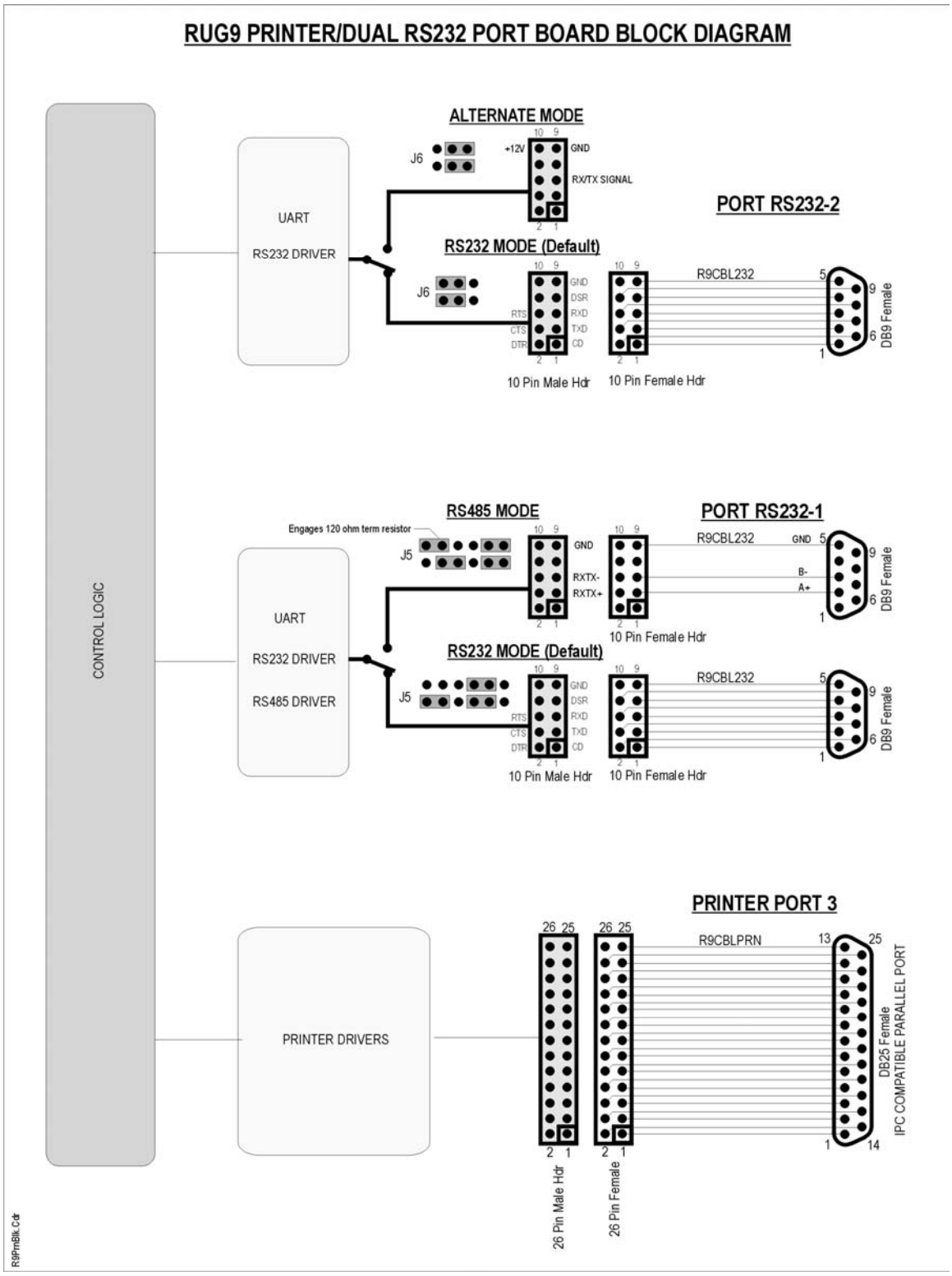


Figure 61 Dual Serial Port/Printer Port Board Block Diagram

Hardware

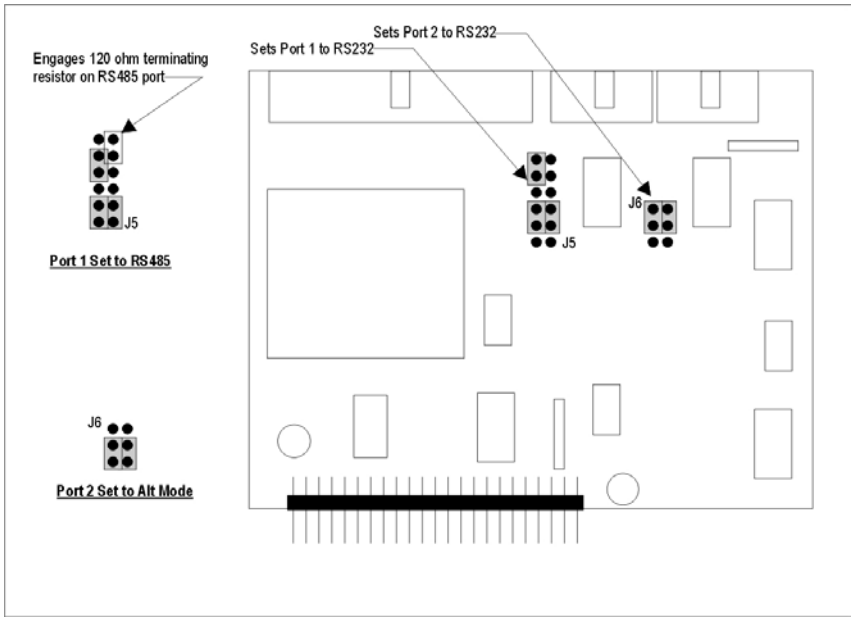


Figure 62 Dual Serial Port Jumper Location

CHAPTER 4…USING RUG5/9 SUPPORT SOFTWARE

Introduction

Unlike earlier RUGID products that you program using the procedural BASIC language, in the case of the RUG5/9, you configure preprogrammed modules using the R9SETUPD program that runs under Windows 95/98/NT/XP/Vista (herinafter referred to as Windows). After you complete your configuration, you save it and then press the 'Send To R9' button to send it to your RUG5/9 unit. The code for each module resides in the RUG5/9, so as soon as the configuration is successfully loaded into the RUG5/9, it will begin executing modules as established by your configuration file. Modules will execute in alphabetical order, but since the program cycles many 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 'Stop' key on the R9SETUPD program's terminal screen; or by pressing the recessed 'Reset' key on the front of the RUG5/9's CPU board.

Procedure for Setting Up A Project

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

1) Designate card locations in card cage(s)
2) Configure I/O modules
3) Install setpoints
4) Set up receive telemetry arrays
5) Configure math calculations
6) Design control strategies
7) Configure statistics (totalizations, data logging, running times, minima, maxima, etc.)
8) Set up event logging
9) Set up transmit telemetry arrays
10) 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 R9SETUPD Design Environment

To configure the RUG5/9, you must use the R9SETUPD program included with your unit 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, click on the **START** button at the lower left of your Windows screen; select **PROGRAMS**; then select **R9SETUPD** from the list of programs installed on your computer. You can also click on the 'R9' icon on your desktop. The icon looks like this:

The program should start and display the screen presented in the next figure.

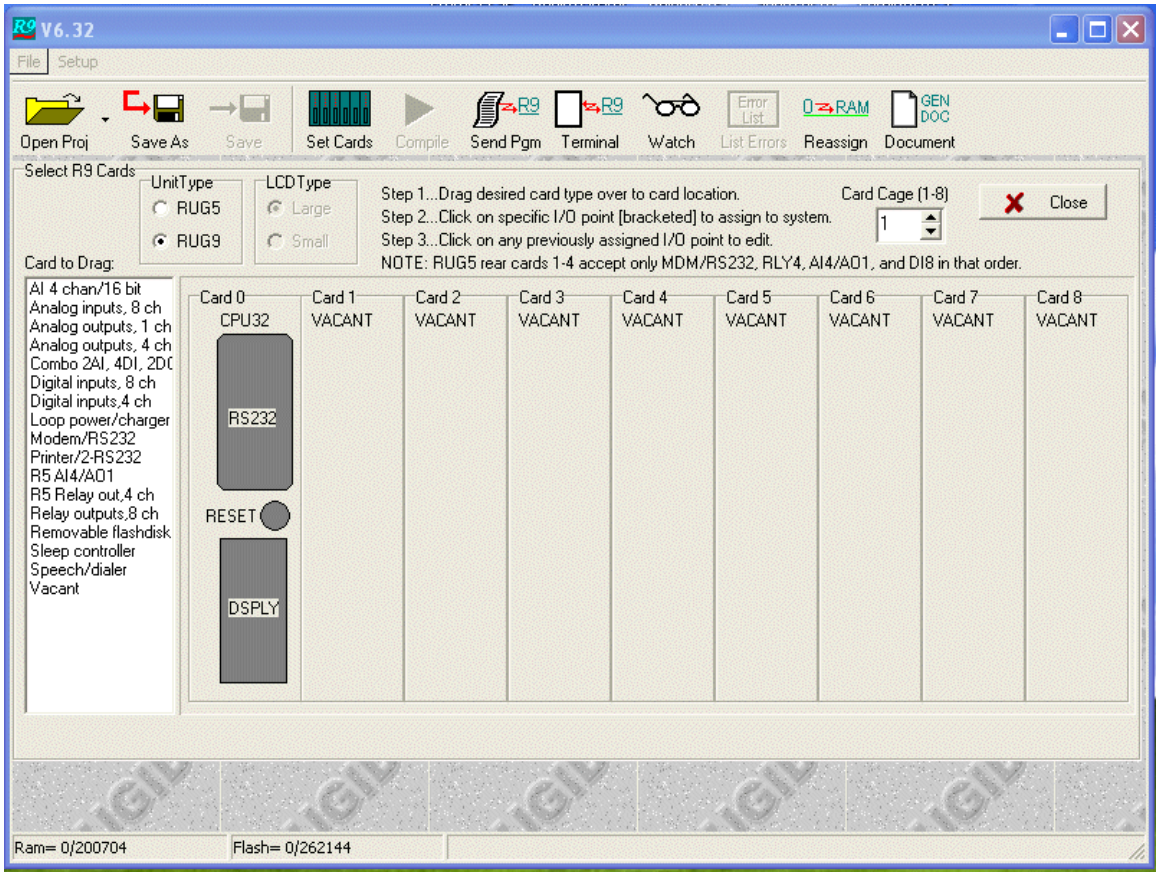


Figure 63 R9SETUPD Opening Panel...Card Installation

Revision

At the very top of the screen you will see a number like the '6.32' in the screen above. This is the OS revision of the support software. It is generally best to use the latest OS revision for a new project. You can always download the latest from our web site (www.rugidcomputer.com\downloads). It is also best to use the same OS revision in the support software as is present in the Rug5/9 unit. If the RUG5/9 has an older OS, you can send it a new one from the support software terminal screen as described later below.

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.

Set Cards: Presents the card cage representation so you can rearrange cards, add new ones, delete cards, and access card 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 RUG5/9.

Terminal: Opens the terminal panel from which you can communicate with the RUG5/9, send a file, reload the RUG5/9 operating system, set the RUG5/9's realtime clock, etc.

Watch: Opens the watch window which you can use to observe database items as the RUG5/9 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.

Designating Card Locations in Card Cage-RUG9

After starting R9SETUPD, you will see the preceding screen, which shows a card cage representation with no cards installed. You must tell the software the location of each card in your unit so that the compiler can set up buffers and interrupts for appropriately interfacing with the cards. You do this by first selecting a card from the list on the left labeled "Card to Drag", then dragging it over to the card location where you wish it to be installed, and finally dropping it into that location. If you wish to change the card installed in a particular location, simply drag the new card to that location and drop it. If your project requires more I/O than you can fit in one card cage, you must specify one or more additional card cages to be included in this unit. You do this by clicking the up arrow in the **Card Cage (1-8)** spin box. You can freely traverse among the card cages that are part of this unit by controlling the number in that spin box. Once you have changed to a new card cage, drag and drop additional cards into the new card cage and configure them as described below. If you wish to delete a card that you previously dropped onto the card cage, simply drag the 'Vacant' card type and drop it onto the desired slot.

Designating Card Locations in Card Cage-RUG5

Note that for the RUG5, you must select the RUG5 unit type in the radio box in the upper left of the card cage representation. When you do that, the card cage number spin box will fade because the RUG5 cannot accept any additional card cages. Also, the card titles will change from 'Card 1…Card 8' to specific titles indicating the allowed card types in each virtual card slot. If your unit has cards in the designated slots, you must still drag the appropriate card type and drop it into the slot. Any of the RUG9 card types can be dropped into slots 6 and 7, the front expansion slots.

Configuring the RUG5

The RUG5 configures the same as a RUG9 except for the smaller LCD display, and limitations on number and placement of I/O cards. The R9SETUP setup software is used to configure the RUG5 as well as the RUG9. On its opening screen, shown below, you must tell it that you are configuring a RUG5 by clicking the 'RUG5' radio button in the Unit Type box. In addition, if you are using the small LCD display and built in keyboard, you must click the 'Small' radio button in the LCD Type box. Both of those selections have been made in the illustration below.

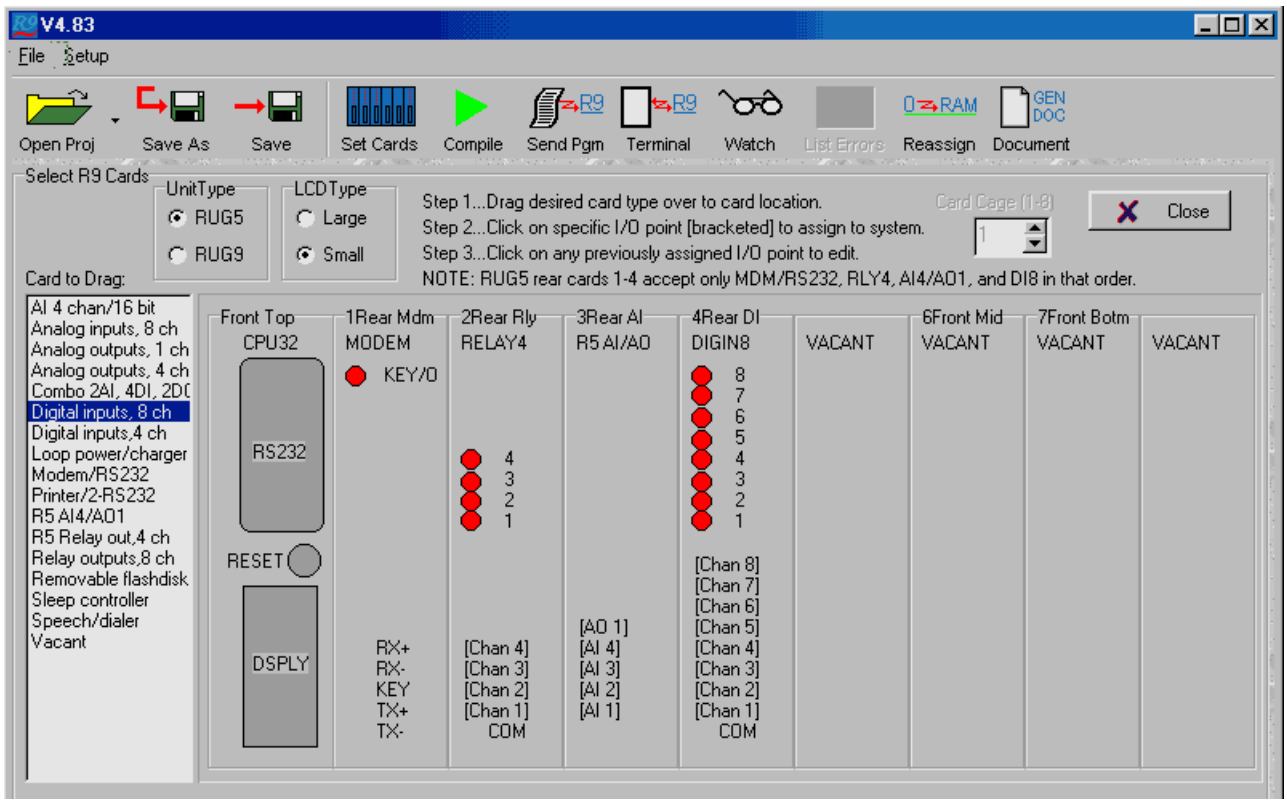


Figure 64 Setting Unit Type, LCD Type, and Installing RUG5 Cards

Also in the example page, notice the placement of the cards. Obviously, the rear of a RUG5 does not look like this, but, in terms of card addressing, this is the way the cards will be regarded by the software. Therefore, you must install cards you are using in the locations shown. Notice also that if you use the two expansion slots in the front of the RUG5, they are addressed as slots 6 and 7. If you attempt to drag cards to the wrong slots when the RUG5 has been designated, the cards won't drop.

Configuring RUG5 Small Displays

The RUG5's small display is limited to 2 lines by 16 characters. Also, it is designated as port 2 in the display addressing structure. (The large display is port 0, the CPU's serial port is port 1.) Therefore, when you configure a display for use on the small display, you must specify port 2, and keep your lines to 16 characters or fewer. You can place up to 50 lines on any display, but the RUG5 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 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 below, no line exceeds 16 characters, and the display port has been set to 2.

As with the RUG9, you can have as many displays configured as flash memory will hold; the operator simply hits the ENTER key to traverse to the next display, then uses the arrow keys to move up or down that display.

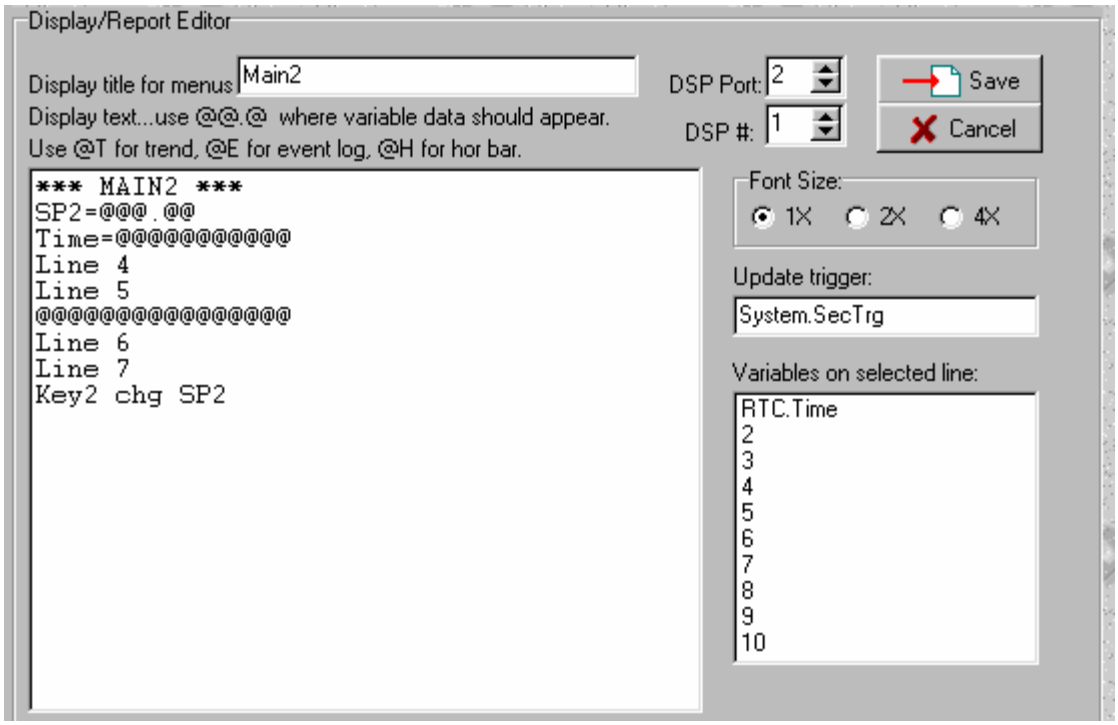


Figure 65 Configuring Display for Small LCD

Using the GetUserValue Module with Small LCD

The GetUserValue module is used to obtain an immediate value from an operator. In the case of the RUG9, you must specify which line the prompt will use on the LCD display. Because the RUG5 LCD has only 2 lines, when the GetUserValue module is executed, its prompt will appear on the top line; and the user's response will appear on the second line. Be sure to keep the prompt to 16 characters or fewer.

Designating Display Type in RUG5

You can choose either the standard RUG5 small 2 line X 16 character display or the larger RUG9 320 X 240 graphic LCD to be used by the RUG5. Do this by selecting 'Small' or 'Large' display in the 'LCD Type' radio box. You can have both attached to your unit at the same time, but only one or the other will be scanned by the software.

Configuring I/O Modules

After you have designated your cards in the card representation panel, you should identify each I/O point that you intend to use and name it. You do that by selecting an I/O point from one of the cards as illustrated in the following figure, where we have installed a RUG9D collection of cards and selected the first digital input. Since the RUG5/9 has several ways in which to use digital inputs, it presents you with those choices in a small panel near the middle of the screen.

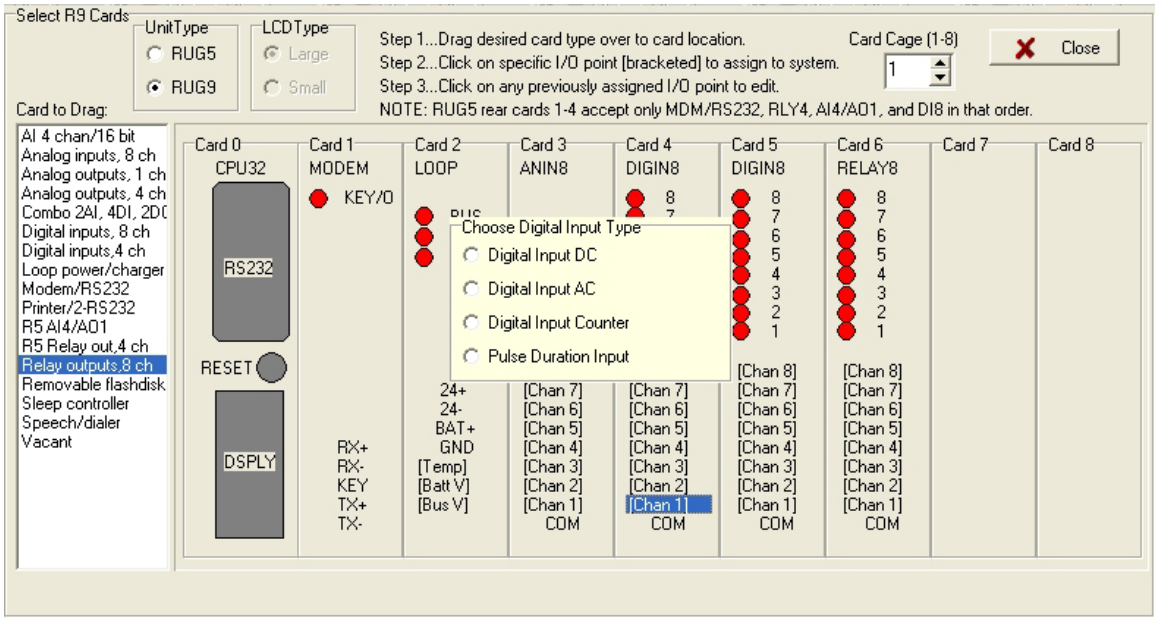


Figure 66 Selecting I/O Type

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.

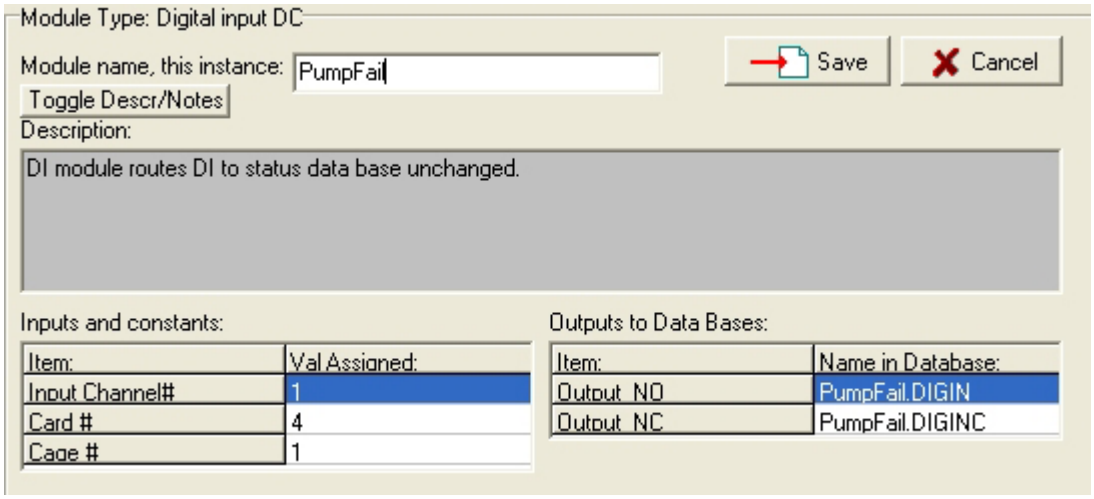


Figure 67 Digital Input Configuration Panel

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 *.DIGIN output basically follows the state of the external digital input. The *.DIGINC output is the inverse of the actual digital input. For the digital input module presented above, all the input properties (Input Channel #, Card #, and Cage #) are already 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. As you save each module, its point on the card cage representation will be displayed with the name you gave the module. In some cases, particularly in the cases of relay outputs and analog 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 card cage 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, table, etc., and it needs a signal from a database, simply drag the signal from the database and drop it in the input cell.

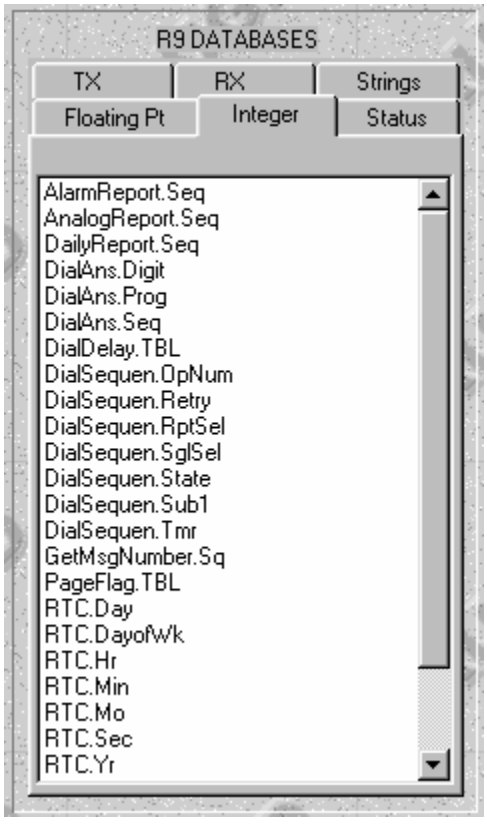


Figure 68 Databases Showing Integer Database

Adding a Module to a Project

You can add a module to a project by clicking on one of the tabs in the 'R9 Module Library', to pull down the module list for that category, and then clicking on one of the modules in the list. The figure below presents the I/O category module list, and indicates that the setpoint module has been selected.

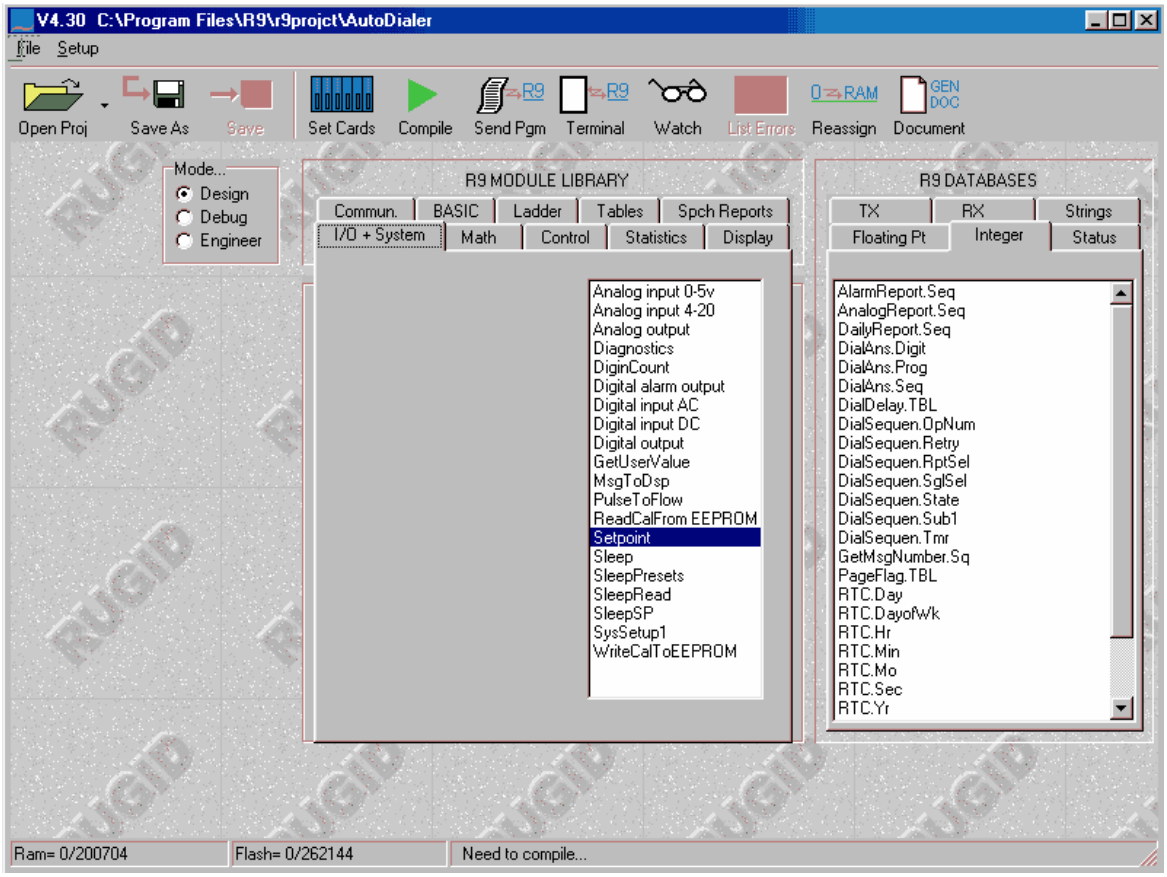


Figure 69 Selecting Module From Module Library

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

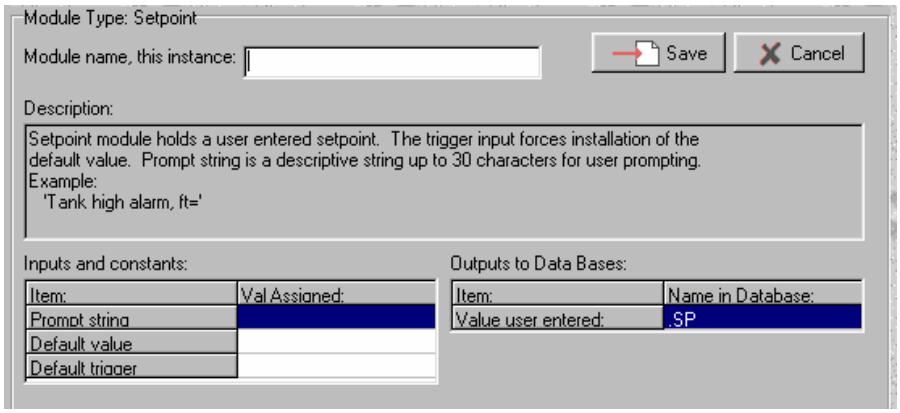


Figure 70 Module Configuration Panel

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 and must be unique, or when you attempt to save the module, the R9SETUPD 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. You can use special characters in the name, such as the underscore, minus, plus, etc., along with numerals. Do not use the decimal point or the vertical bar '|' characters.

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 'DialAns.Digit' is established as the input to the pictured module by dragging it from the integer database to the right:

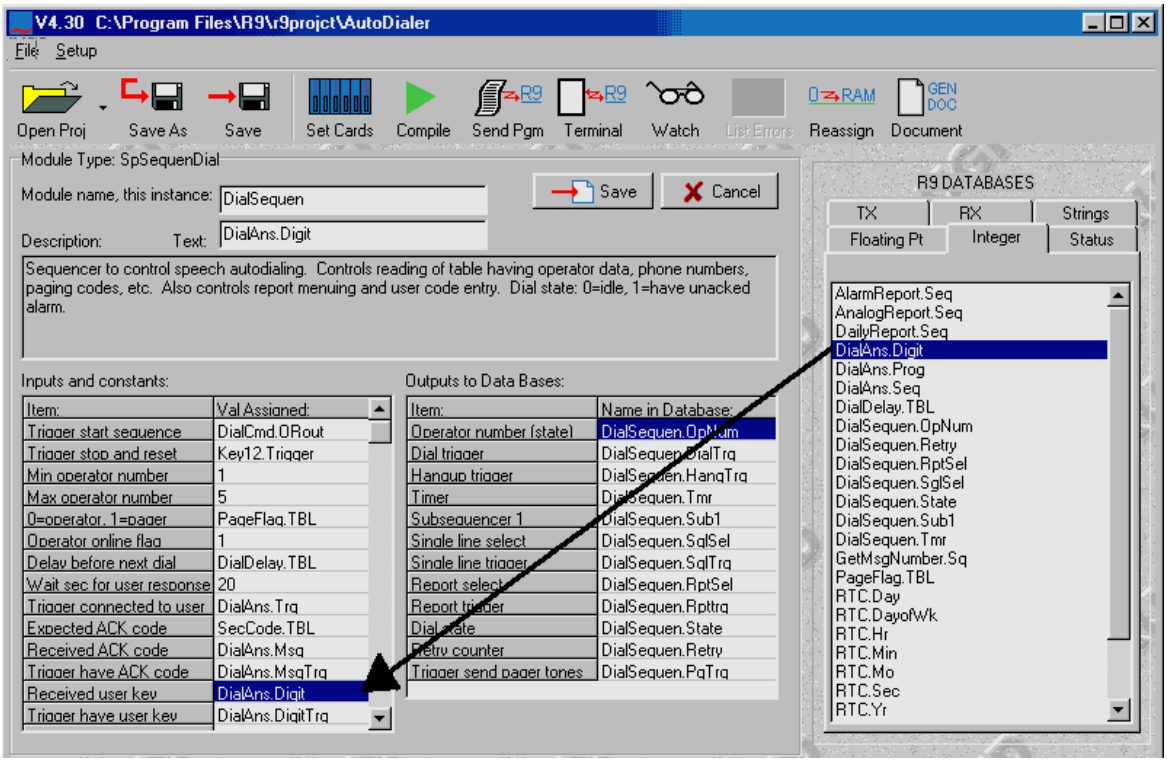


Figure 71 Dragging Database Item Into a Cell

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 board number, channel number or card cage number must be integer constants.
-Inputs that call for the number of bytes in an output string must be integer constants.
-Inputs that expect numbers of 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, or the inputs can be left blank. The exceptions are that card, cage and channel numbers for I/O type modules must be constants. These must be specified or the compiler will complain. If you fail to make an entry for a module input, the RUG5/9 will assume that it's zero unless it is a status input that should default to a one for logic reasons. 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 RUG5/9 will simply convert the input to the type needed and proceed with processing.

Triggers

Some modules generate triggers, which are statuses that are 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 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 R9 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 as illustrated below. Notice that each module is shown with the name you assigned the module followed by the module type.

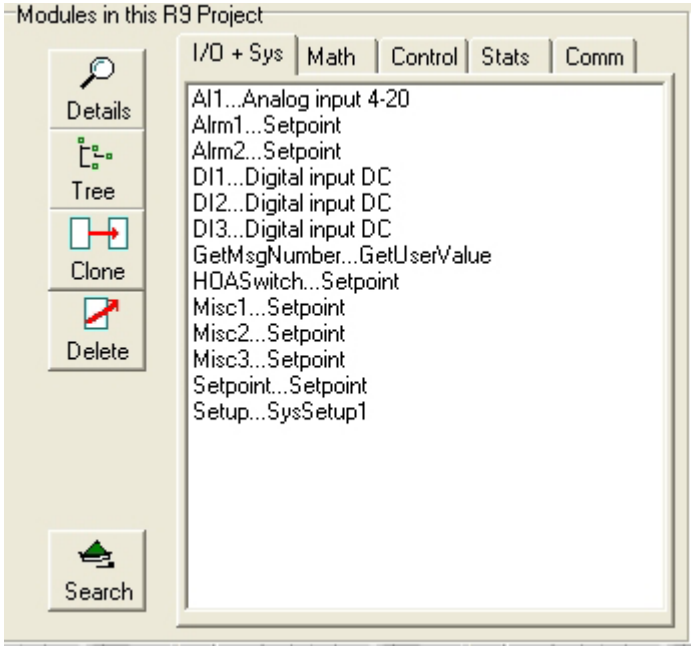


Figure 72 List of Modules Installed in Project

Observing Module Interconnects

When your project grows large, it is often difficult to remember which modules are connected to which others. If you wish to know where a module's output is being used, you can select the module from the project module list and then press the 'Tree' button. The system will present a page showing the module and where its outputs are being used. If you click on the 'Expand All' button, you will get the following module interconnect tree. Any change to a module shown will affect those to the right and below it on the tree.

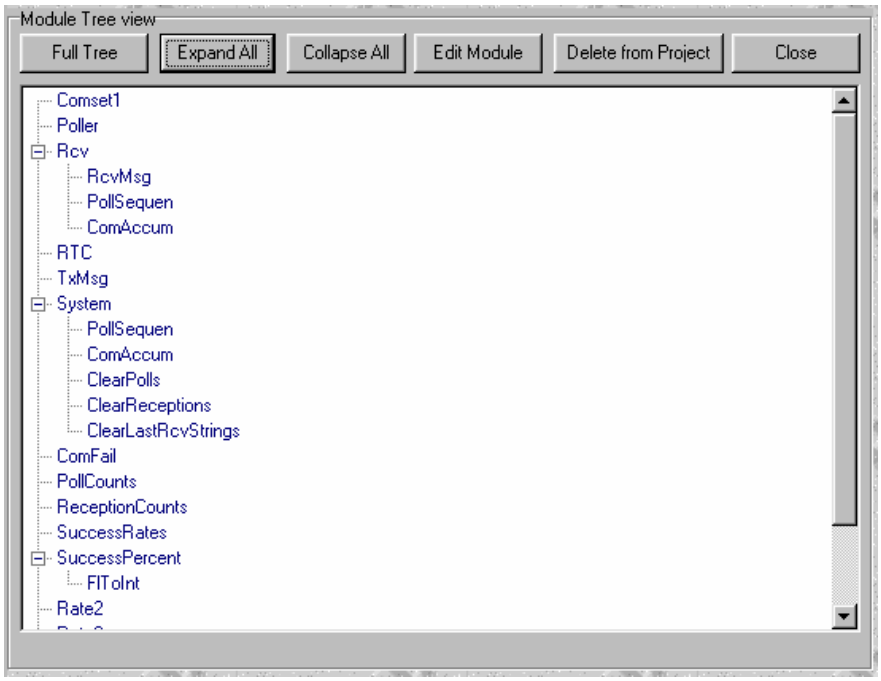


Figure 73 Module Tree View

Modifying an Existing Module

If you wish to change a module after you have installed it in the project, you must simply select it from one of the lists in the "Modules in This R9 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 similar 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 R9 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. 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 R9 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:

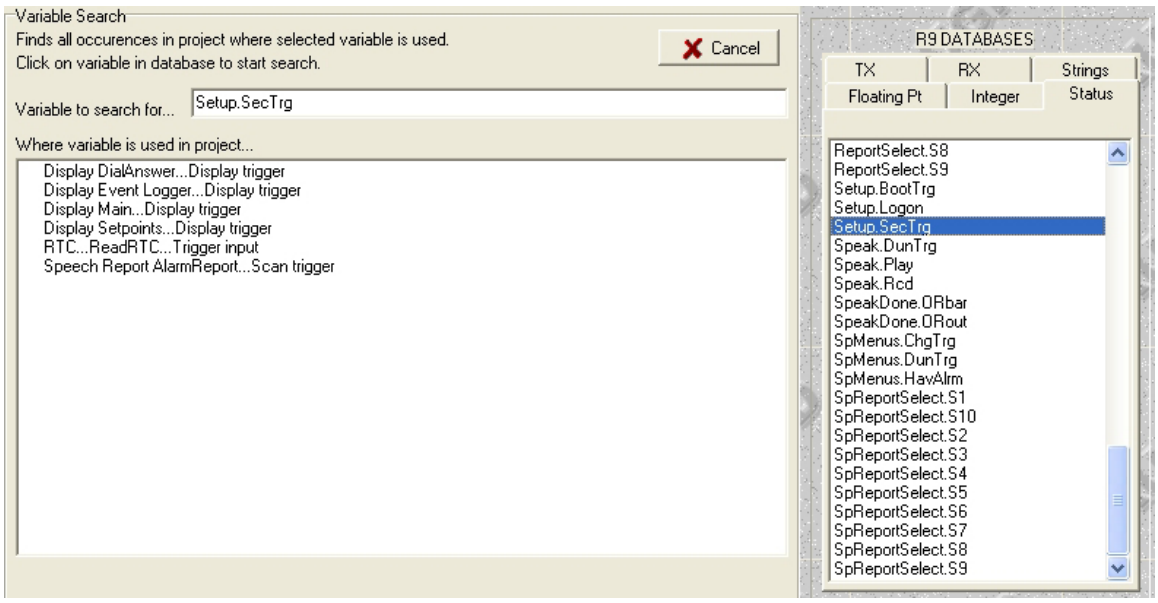


Figure 74 Searching for Output Usage

Returning to the Card Cage Panel

To return to the card cage representation of your project, click on the card cage button on the tool bar, labeled **Set Cards**.

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 RUG5/9, 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 RUG5/9 in a small panel in the lower left corner of the R9SETUPD form as illustrated below.

Figure 75 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. Flash is the area where the configuration file is stored and is not changed as the program is running.

Sending the Program to the RUG5/9

After you have designed your project and it compiles successfully, you are ready to send it to the RUG5/9 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 RUG5/9 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 RUG5/9, the compiler will command the RUG5/9 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 RUG5/9 and lets you observe ASCII responses from the RUG5/9. To open the terminal page, click the Terminal button on the tool bar. When you do so, R9SETUPD will open a communications port on your PC and begin communicating with the RUG5/9. The page looks like the figure below:

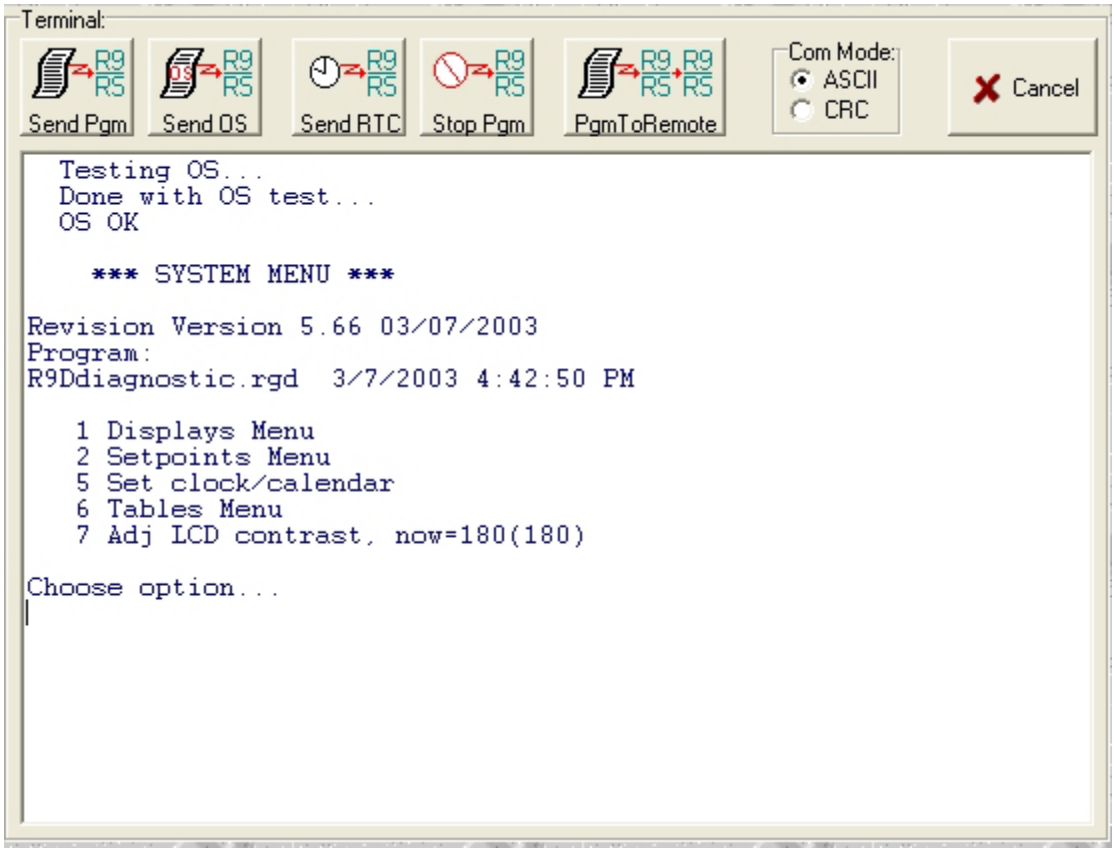


Figure 76 Terminal Page Showing Normal RUG5/9 Boot Up Messages

Sending the Operating System to the RUG5/9

Since the RUG5/9's operating system (OS) is held in flash memory, it can be reloaded serially. Each revision of R9SETUPD includes the latest RUG5/9 operating system. Clicking the **Send OS** button on the Terminal page will send the new OS to the RUG5/9. Loading the OS to the RUG5/9 will take several minutes. If it is interrupted, or if at any time the operating system integrity test on boot up fails, the RUG5/9 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 RUG5/9 matches the version of your R9SETUPD. The RUG5/9's operating system revision is listed on the terminal page any time the RUG5/9 issues the "Welcome to RUGID Monitor" message. On the terminal page illustrated above, the OS version is given as 5.66. R9SETUPD's revision is given in the upper left hand corner of the form as V5.66 for example.

Sending the PC's Realtime Clock to the RUG5/9

If the RUG5/9 has presented its "Welcome to RUGID Monitor", and you click the Send RTC button at the top of the terminal page, R9SETUPD will read your PC's clock/calendar and install it into the RUG5/9's realtime clock in CRC secured format. To return to ASCII mode, you will need to hit the space bar 4 times afterward.

Stopping the RUG5/9's Program

There are two ways to stop the RUG5/9's program: press the recessed reset button on the front of the RUG5/9's CPU board; or hit the **Stop** button on the top of the terminal page. Either way, the RUG5/9 should send the "Welcome to RUGID Monitor" message and wait for a command or program reload. If you take no further action, the program will restart automatically in 60 seconds.

Observing Program Execution With the Watch Window

You can observe the values of data base items while the RUG5/9's program is running by hitting the **Watch** button on the tool bar. When you do so, the following panel will appear:

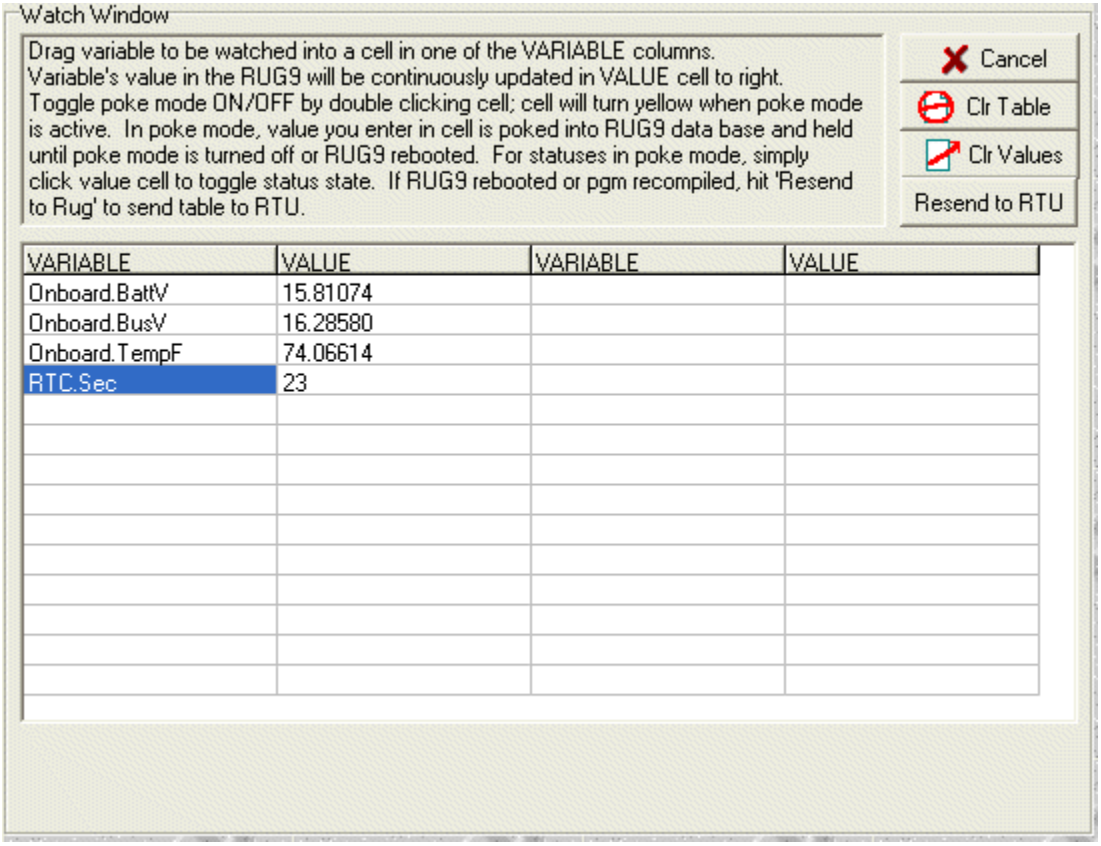


Figure 77 Watch Window

When you first see this window, all entries will be blank. To begin observing values from the RUG5/9, 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 RUG5/9 in the VALUE cells to the right of the variables. For testing, you can place any variable in the table into 'poke mode' by double clicking the value cell. The cell will change to a yellow background. Then you may type in a new value for the variable followed by the ENTER key. That value will replace the usual output value of the module until you turn off the poke mode or the RUG5/9 reboots. If your RUG5/9 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 click on the 'Resend to RTU' button.

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.

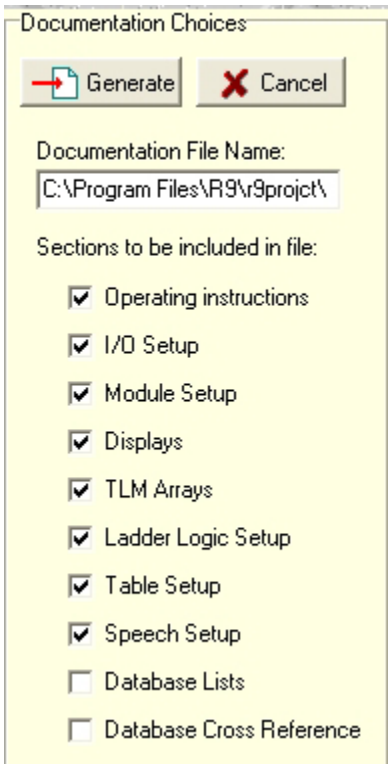


Figure 78 Documentation Generator Choices

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. The generated file is in simple ASCII text format without any word processor formatting.

Setting PC Communications Port Parameters

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

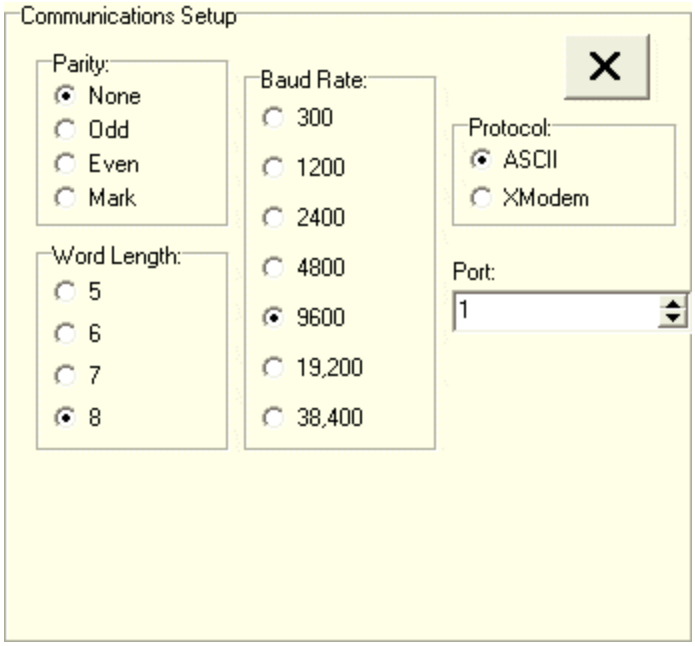


Figure 79 Com Port Setup Panel

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 R9SETUPD is started. The port range is 1 through 25. Most PC's use port 1 for a DB9 serial port. USB ports are usually port 3, 4 or 5.

Setting File Paths and Controlling Sorting

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

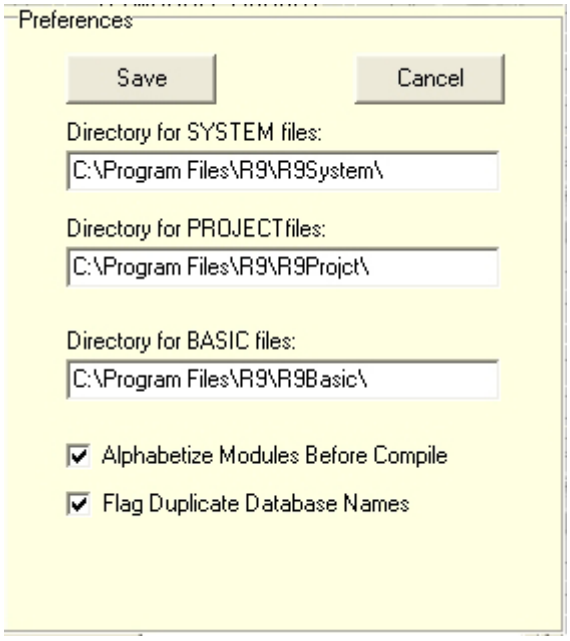


Figure 80 Preferences Panel

Here you can change the paths to folders where R9SETUPD 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. If you leave alphabetization checked (our default setting), the compiler will sort before compiling, meaning that program execution in the RUG5/9 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 RUG5/9 in the order that you installed them in your project; and setpoints will be presented in that order also. If you uncheck the 'Flag Duplicate Database Names' checkbox, the compiler will ignore duplicate database names. Generally, you should leave both checkboxes checked. Changes you make will be saved in an initialization file and reinstalled each time R9SETUPD is started.

SENDING PROGRAMS TO REMOTE RTU's

SUMMARY

Starting with operating system revision 5.04, programs can be reloaded into remote RTU's over their telemetry channel for all telemetry networks except dial-up networks. This includes leased phone line, audio radio, spread spectrum radio, RS232 and RS485 networks. Any path that works for normal telemetry communications can be used for program loading including store and forward paths. All that is required is that you connect a PC to the programming port on a RUG5 or RUG9 in the system that has telemetry access to the RUG5 or RUG9 into which you wish to send a program. You will also need to know the destination unit's address in the network, and which board and port number the local unit can use to communicate with the distant unit. Both units as well as any store and forward units must already be running OS revision 5.04 or later. Note that remote program loading only applies to loading application programs (configuration files); the operating system cannot be loaded in this manner…you have to access the unit directly to reload the OS.

Normally, you will load programs from the master site to remote sites; however, if the channel has infrequent traffic, as in quiescent systems, you can load programs from any site to any other site as long as they have communications access to each other. You could even reload the master site from a remote site.

For this to work, all sites involved must be using the RUG9 protocol. All messages from the PC to the RUG5/RUG9 units use this protocol which secures each transmission using CRC. If a transmission does not evoke a valid reply for any reason, then the message will be repeated. Each message contains the program segment's absolute memory address in the destination station, so a missed or corrupted message can be repeated and installed correctly.

PROCEDURE

To download a program to a remote site, connect your PC to a RUG5 or RUG9 in the system, start R9SETUPD software, call up the program you wish to send to the remote site, and click the **Terminal** button on the main toolbar. The terminal page should open with the following toolbar at the top of the panel.

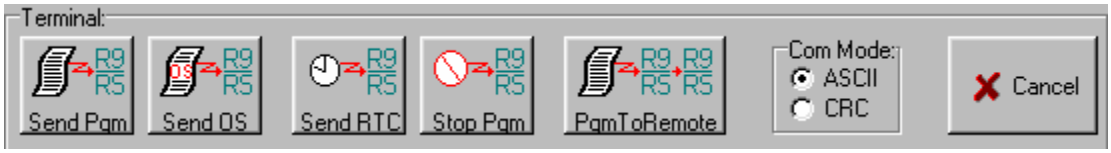


Figure 81 Terminal Toolbar

Here are the functions of the toolbar buttons:

- **Send Pgm**…sends the loaded program to the **LOCAL** RUG5/RUG9. This is the button you use to send programs to a RUG5/RUG9 to which you are directly connected on the RUG5/RUG9's CPU board DB9 connector.
- **Send OS**…reloads the operating system to the **LOCAL** RUG5/RUG9.
- **Send RTC**…sends the clock/calendar from the PC to the **LOCAL** RUG5/RUG9.
- **Stop Pgm**…sends ^K codes to the **LOCAL** RUG5/RUG9 to stop its program and invoke the monitor mode.
- **Pgm To Remote**…sends the loaded program to the **REMOTE** RUG5/RUG9. This button commands the local RUG5/RUG9 to pass the program load to a distant RUG5/RUG9.
- **Cancel**…stops the download sequence and closes the serial port and the terminal panel.

It is important to notice the distinction between the leftmost button for loading a program to a local RUG5/RUG9; and the fifth button, which is used to send the program to a distant site. If you mistakenly click the leftmost button when you intend to load the program to a distant site, you will instead load the program to the local RUG5/RUG9.

To start the process, click the **PgmToRemote** button. The following panel will appear:

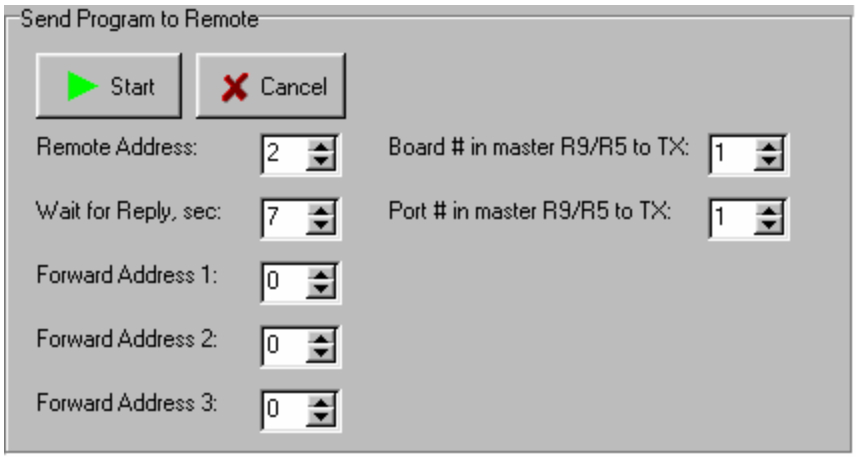


Figure 82 Send Program to Remote Path Definition Panel

This panel enables you to define the path to the remote site and some other parameters as described below:

- Remote Address…the address in the telemetry system of the distant remote site to which you wish to send the new program.
- Wait for Reply, sec…how long in seconds the PC should wait for a response from the remote site before retrying the transmission. This delay needs to be set to a period that is longer than it would normally take the remote site to reply. The default of 7 seconds is sufficient for a 300 baud rate to a site not involving store and forwarding. When downloading at 300 baud using store and forwarding, this time value should be increased proportional to the number of intermediary stations involved in each transmission.
- Board # in master R9/R5 to TX…the board number (1-8) in the local RUG5/RUG9 that is to be used to send messages to the distant site.
- Port # in master R9/R5 to TX…the port number (1-3) in the local RUG5/RUG9 that is to be used to send messages to the distant site.
- Forward Address1,2,3…when store and forwarding technique is to be used to relay messages to/from the distant site, these entries identify which remote station addresses are to be used to perform the relays. Unused forward addresses should be left at a value of zero. For example, to send a program to site 17 from the local site using address 9 as a store and forward relay site, the **Remote Address** entry should be set to 17, and **Forward Address 1** should be set to 9. In this case, **Forward Addresses 2, and 3** should be set to zero.

When all settings are correct, click the **Start** button to begin the process. An internal sequencer will take care of initiating the pass through mode in the local RUG5/RUG9, stopping and erasing the program in the distant unit, and installing the program there. It will also start the new program in the distant site. During the download, a progress panel will show a download progress bar and the number of repeats (failed messages). For radio or phone line paths, repeats are normal. Note that a low baud rate on the telemetry path will mean much longer than usual program download time. For example, a program that takes 30 seconds to download locally (9600 baud) will take at least 16 minutes to send over the telemetry channel. If the local unit was polling, during the download that polling will be disabled and restarted after the download is finished.

Note that the '**SendOS'** command is case sensitive; it must be entered exactly as shown.

Automatic Program/OS Loading Using Command Line Parameters

Launching R9SETUPD and Sending a Program Automatically

Normally, when you launch R9SETUPD from your desktop, it will begin running and load the last configuration file on which you were working. It will then wait forever for you to command it to take action by pressing a button with your mouse; and will only terminate when you tell it to manually. You can, however, have R9SetupD launch itself, load a program, send that program to a RUG5/9 and then terminate itself automatically using the Windows RUN feature. This could be useful if the PC and RUG5/9 unit are remotely located and no human is local to perform the manual keystrokes. To initiate the automated process, you must use the Windows RUN feature along with the complete path to R9SETUPD followed by the complete path to the application configuration file that you wish R9SETUPD to send to the RUG5/9. Note: the command line parser uses the blank character to delimit individual commands in the command line. Therefore, you cannot use file or folder names that contain blanks such as Windows' "Program Files". The following procedure illustrates the process:

1) Click START, then RUN on your desktop. Windows will launch the RUN dialog.
2) In the RUN dialog edit box, type the path to R9SETUPD followed by the path to your application file. For example, to launch R9DDiagnostic, you would type:
   `c:\R9SetupFiles\R9\R9SetupD.exe`
   `c:\R9SetupFiles\R9\R9Projct\R9DDiagnostic.rgd`.
3) At this point, windows should launch R9SETUPD. It will load R9DDiagnostic, compile it, and send it automatically to the RUG5/9 and then terminate.

Observing Result of Automatic Process Using the R9ResultLog File

After the sequence is finished you can use an editor to access the R9ResultLog file to see what happened in the last R9SETUPD session. The file will have the following format:

1: 2/12/2005 6:20:53 PM ...Launched R9Setup
4: 2/12/2005 6:21:10 PM ...Program load successful
2: 2/12/2005 6:21:10 PM ...Terminated R9Setup

Each line consists of a numeric code, a date and time and text describing the action taken. The numeric code is there to make it easy for a program to parse the file and ascertain the success of the procedure. The possible actions are:

1=Launched R9Setup
2=Terminated R9Setup
3=Successful OS load
4=Program load successful
5=Unsuccessful load
6=Unable to open referenced program
7=R9 not responding
8=Can not find R9.DWN file
9=Unable to open referenced table file
10=Project compile resulted in error(s)
11=Error in remote load parameters
12=Created and saved→ new program name

Controlling Repeat Attempts and Size of R9ResultLog File

In order that R9SETUPD does not try forever to connect with a RUG5/9 and does not create a huge R9ResultLog file, you can set limits in the RUG9SETUPD preferences menu item. To do so, within R9SETUPD click on the 'Setup/Preferences' menu item. The following panel should appear:

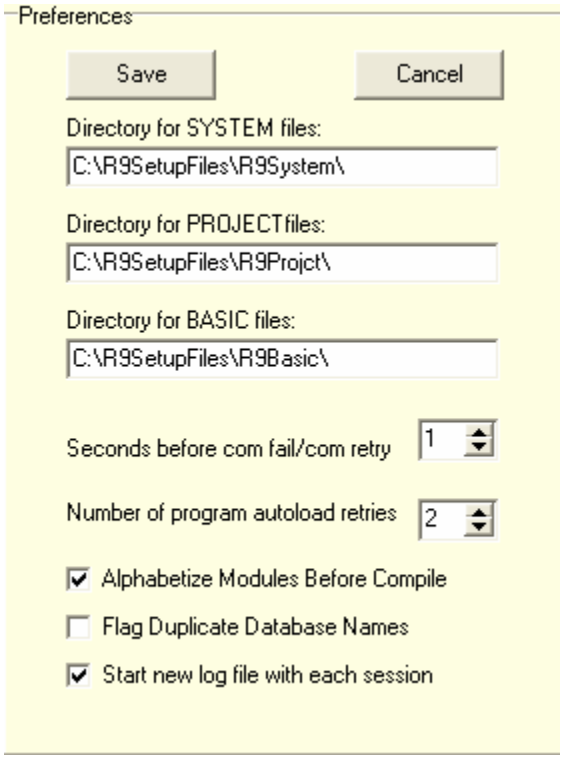


Figure 83 Auto Load Preferences

The '**Number of program autoload retries'** can be set to a range of 0 to 9. Values of 2 or 3 are practical. If the program cannot connect with the RUG5/9 after that number of tries, the program will terminate. Clicking the last item on the panel, '**Start new log file with each session'**, will cause the R9ResultLog file to be erased at the beginning of each session so that only the most recent session's events are recorded in the file. If left un-clicked, the log file will accumulate events from multiple sessions until it reaches 100 events. After that time, with each new event, the oldest event at the top of the list will be deleted and the

most recent event will be added at the bottom. Note that the log file records manual sessions as well as automatic sessions.

Automatically Loading Tables Before Sending to RUG5/9

You can have R9SETUPD load a file of tables into your program before compiling and sending to the RUG5/9. This could be useful if your application requires the same program but different tables for each field unit; or if your application needs different programs to use the same table data. To do this, first design and install the table(s) into a RUG5/9 program and save the program. With a programmer's editor, cut the tables from your '.rgd' file and save them into a new file. Note that the '.rgd' file is just a text file that contains your program. To illustrate, the listing below is a segment from our Autodialer.rgd example:

```
.
.
.
Misc 2=@@@.@@|Misc2.SP|2|3|4|5|6|7|8|9|10
Misc 3=@@@.@@|Misc3.SP|2|3|4|5|6|7|8|9|10
!TX telemetry definition
!RX telemetry definition
!Ladder definition
!Tables
~OperatorList|DialSequen.OpNum|||6|5|20|0|$40F2C4|$40F463|
String|OperName||Albert|Bill|Cheryl|Diana|Eileen|OperName.TBL|
String|PhNumber||||||PhNumber.TBL|
Integer|SecCode||100|200|300|400|500|SecCode.TBL|
Integer|PageFlag||||||PageFlag.TBL|
String|PageCode||||||PageCode.TBL|
Integer|DialDelay||||||DialDelay.TBL|
!Speech
Zero
One
Two
Three
.
.
.
```

The tables consist of everything between '!Tables' and '!Speech'. It may contain multiple tables or a single table as in the example above. Each table begins with the '~' character. After you cut the table that you wish to reuse and paste it into a new text file, remove the address references from end of the first line of each table. For example, the first line of the above table is:

```
~OperatorList|DialSequen.OpNum|||6|5|20|0|$40F2C4|$40F463|
```

You need to change it to:

```
~OperatorList|DialSequen.OpNum|||6|5|20|0|||
```

Now, to specify that the saved table(s) are to be loaded into the program before it is compiled and sent to the RUG5/9, your command line to the Windows RUN command would be the same as before followed by a space and then the path to the file containing the table:

```
c:\R9SetupFiles\R9\R9SetupD.exe
c:\R9SetupFiles\R9\R9Projct\R9DDiagnostic.rgd
c:\R9SetupFiles\R9\R9Projct\TableFile
```

Automatically Sending a New OS to a RUG5/9

You can have R9SETUPD automatically send the operating system (OS) to the local RUG5/9 in a manner almost identical to the automatic program loading process describe above. You simply substitute the text '**SendOS**' in place of the program name to be loaded. The following procedure illustrates the process:

Click START, then RUN on your desktop. Windows will launch the RUN dialog.

3) In the RUN dialog edit box, type the path to R9SETUPD followed by the 'SendOS' command. For example, you would type**: `c:\R9SetupFiles\R9\R9SetupD.exe SendOS.`**
4) At this point, windows should launch R9SETUPD. It will send the OS automatically to the RUG5/9 and then terminate.

Note that the '**SendOS'** command is case sensitive; it must be entered exactly as shown.

Automatically Sending a New Program to a Remote RUG5/9

You can have the system command a local RUG5/9 to forward loading commands and the program to a distant RUG5/9 by appending a command of the form:

```
RemoteLoad,A,W,F1,F2,F3,B,P
```

to the command line. The field "RemoteLoad" is case sensitive and must be entered exactly as shown. Arguments of the remote load command must be delimited with a single comma each with no spaces. They have the following meanings:

A=remote station address
W=wait time in seconds for a response from the remote site
F1=first forwarding address to reach remote unit (use 0 for no forwarding)
F2=second forwarding address to reach remote unit (use 0 for no forwarding)
F3=third forwarding address to reach remote unit (use 0 for no forwarding)
B=board number in local RUG5/9 having serial access to remote units
P=port number in local Rug5/9 having serial access to remote units

As an example, in order to send a program to remote station 35 with a wait time of 7 seconds for a reply, and assuming that board 1 and port 1 in the local Rugid unit communicates with the remote units (most common configuration) with no forwarding, the command would be the following:

```
RemoteLoad,35,7,0,0,0,1,1
```

The full command line containing the above remote loading command would be:

```
c:\R9SetupFiles\R9\R9SetupD.exe
c:\R9SetupFiles\R9\R9Projct\R9DDiagnostic.rgd
c:\R9SetupFiles\R9\R9Projct\TableFile RemoteLoad,35,7,0,0,0,1,1
```

Another example with forwarding through stations 17 and 25, and assuming board 6, port 1 in the local RUG5/9:

```
RemoteLoad,35,7,17,25,0,6,1
```

The full command line with this command would be:

```
c:\R9SetupFiles\R9\R9SetupD.exe
c:\R9SetupFiles\R9\R9Projct\R9DDiagnostic.rgd
c:\R9SetupFiles\R9\R9Projct\TableFile RemoteLoad,35,7,17,25,0,6,1
```

Automatically Compiling and Saving a File With Merged Table Without Loading to RUG5/9

You can have R9SETUPD automatically load a program, merge a table file into it, compile the merged program, save the new file with a new name, and finally exit without sending to a RUG5/9. To do that, in your command line, you would simply follow the table file name with a command of the form: "CompileSaveAndExit,Newfilename". The new file name is a parameter to the "CompileSaveAndExit" command following the comma delimiter. A full command line would look like this:

```
c:\R9SetupFiles\R9\R9SetupD.exe
c:\R9SetupFiles\R9\R9Projct\R9DDiagnostic.rgd
c:\R9SetupFiles\R9\R9Projct\TableFile CompileSaveAndExit,R9DMerged.rgd
```

CHAPTER 5…SOFTWARE MODULES

Introduction

This chapter provides detailed descriptions of the preprogrammed software modules resident in the RUG9. With a 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 tables, speech reports, and 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.

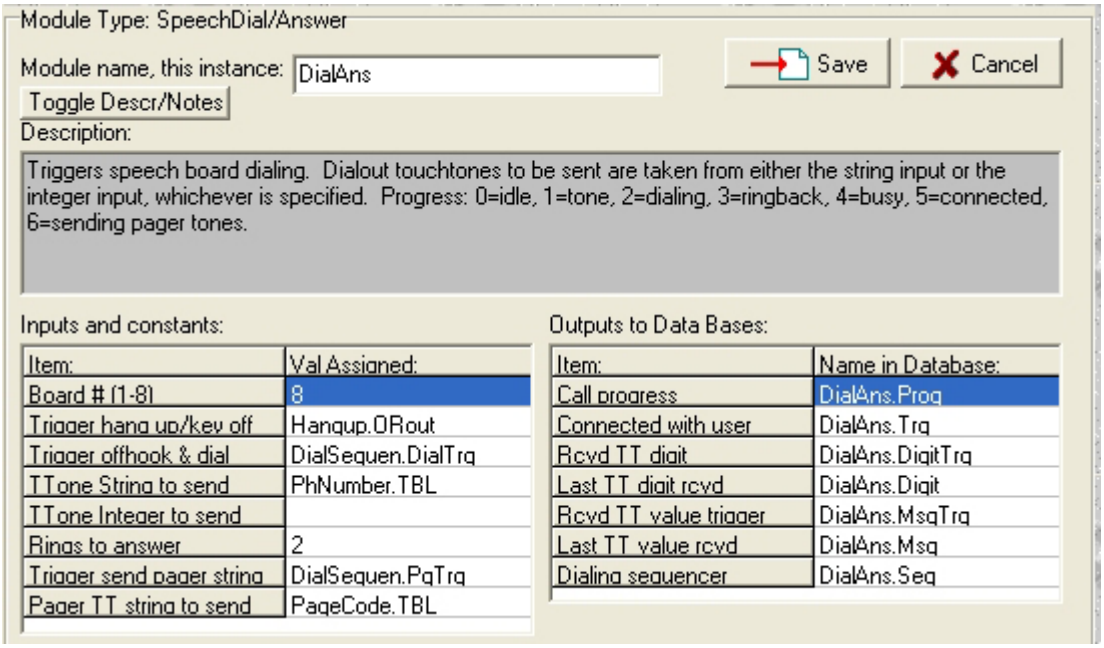


Figure 84 Module Editing Panel

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 and sixth entries have had constants typed into them by the programmer, the fifth input has been left blank, 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 (Board #) must be entered as a constant to tell the compiler where the board resides that this module controls. Generally, constants are only required to tell the compiler which board, card cage or channel number a module uses, or how much space to reserve for data logs and string outputs.
- **Blanks:** you may leave blank any input that is not needed in a calculation or other use. In the above example the module requires a phone number either in the form of a string or an integer. Since the phone number is being supplied as a string (the input above the blank one), then the integer version can be left blank. Generally, blank entries are regarded as 0.0 if they are terms, 1.0 if they are factors.
- **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 cannot drag in an entry from the string database. In the above example panel, inputs labeled "Ttone string to send" and "Pager TT string to send" must be dragged from the string database. (They could also be typed in, in which case they would be saved as string constants.)
- **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

Analog Input 0-5V

The **Analog Input 0-5V** module is used to add a voltage type analog input to a project. One of these modules is required for each analog input in your RTU that must sense a 0 to 5 volt input. The module performs raw count to engineering units conversion and low pass filtering. The offset entry sets the engineering units value that corresponds to a zero volt input from the field. The span entry sets the engineering units span that corresponds to the zero to 5 volt span from the field. The filter time constant sets the time in seconds 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 one second essentially provide no noise immunity. Typical useful filter constants range from 5 to 30 seconds. The engineering units output from this module will be available in the floating point data base. Note that for a RUG9 analog input to function as a 0 to 5Vanalog input, the black shorting bar that engages the current sensing resistor on the analog input board should be removed to avoid unnecessarily loading the analog input source.

Inputs That Must be Constants:

- Input channel #…which analog input channel on the board this module uses
- Card #…which card in card cage (1-8)
- Cage #…which card cage (1-8)

Other Inputs:

- Filter sec…low pass filter time constant in seconds. 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.
- Offset…the engineering units value that corresponds to 0.0 volts applied to the input
- Span…the engineering units span that corresponds to the signal span of 0.0 to 5.0 volts.

Primary Outputs:

- EU output…Name.Out…floating point engineering units representation of this analog input

Expected Applications:

Use this module for analog inputs on either the 12 bit/8 channel analog input board, or the analog inputs on the combo board. Do not use for 16 bit/4 channel analog input board. Use setpoints for span and offset if the operator needs to adjust calibration.

Analog Input 4-20 ma

The **Analog Input 4-20ma**. module is used to add a current type analog input to a project. One of these modules is required for each analog input in your RTU that must sense a 4-20 ma. input. The module performs raw count to engineering units conversion and low pass filtering. The offset entry sets the engineering units value that corresponds to a 4 ma. input from the field. The span entry sets the engineering units span that corresponds to the 4-20 ma. span from the field. The filter time constant sets the time in seconds 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 one second essentially provide no noise immunity. Typical useful filter constants range from 5 to 30 seconds. The engineering units output from this module will be available in the floating point data base. Note that for a RUG9 analog input to function as a 4-20 ma. analog input, the black shorting bar that engages the current sensing resistor on the analog input board should be installed or an external current sensing resistor must be used.

Inputs That Must be Constants:

- Input channel #…which analog input channel on the board this module uses
- Card #…which card in card cage (1-8)
- Cage #…which card cage (1-8)

Other Inputs:

- Filter sec…low pass filter time constant in seconds. 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.
- Offset…the engineering units value that corresponds to 4.0 ma. applied to the input
- Span…the engineering units span that corresponds to the signal span of 4.0 to 20.0 ma.

Primary Outputs:

- EU output…Name.Out…floating point engineering units representation of this analog input

Expected Applications:

Use this module for analog inputs on either the 12 bit/8 channel analog input board, the 16 bit/4 channel analog input board, or the analog inputs on the combo board. Use setpoints for span and offset if the operator needs to adjust calibration.

Analog Output

This module takes a variable from the floating point database and scales it properly to give the corresponding 4-20 ma. signal on the designated analog output channel. You must supply constants defining the card, card cage and channel the module is to control. You must also define the floating point variable that is to control this output, and what engineering units values of the chosen floating point variable are to correspond to 4.0 ma. and 20.0 ma. For example, if variable "VFDspeed.Out" is to control this analog output and a speed of 25% is to give 4.0 ma on the output, and 75% speed is to give 20.0 ma. on the output, then 25 and 75 must be entered in the "EU and 4 ma" and "EU at 20 ma" cells, respectively. Also, the variable "VFDspeed.Out" must be dragged from the floating point data base into the "Source" input cell. You must use constants for channel, card and cage numbers. You can use constants or variables for any other cell entries.

Inputs That Must be Constants:

- Channel #…which analog output channel (1-4) on the board this module uses
- Card #…which card in card cage (1-8)
- Cage #…which card cage (1-8)

Other Inputs:

- EU Source…engineering units input signal that is to control this output
- EU at 4 ma…what engineering units value should give 4.0 ma. at output
- EU at 20 ma…what engineering units value should give 20.0 ma. at output
- Select 0=EU, 1=Raw…set to zero or leave blank to control output from "EU source" with module doing scaling based on EU at 4 ma and EU at 20 ma values. Set to 1 if "Raw count to send" is to be routed to output directly without any rescaling.
- Raw count to send…value in range of 0 to 4095 to be sent directly to analog output if above flag is set to 1

Primary Outputs: None

Outputs for Internal Use: None

Limitations:

Minimum analog output current is approximately 3 ma., so low raw counts cannot make the output go below 3 ma. Loop voltage across analog output must be at least 12 VDC. Output to D/A converter is sent whenever there is at least a one bit change in raw count, or each 5 seconds, whichever is more frequent.

Expected Applications:

Use one of these modules to control each analog output channel on the 12 bit/4 channel analog output board. Basically, a signal (PID output, math output, or other floating point output of a control strategy) that needs to control an analog output must be used as the EU Source of one of these modules to properly control the analog output channel.

Bargraph

The **Bargraph** module is used to establish the engineering units range of a value to be displayed as a horizontal bargraph on the RUG9's LCD, and to clamp the value to be displayed to that range. You need a bargraph module for each variable you wish to display as a bargraph on the LCD. If you are showing the same variable as a bargraph on several different displays, you only need one bargraph module to set the range for that variable. The **Bargraph** module's output is placed in the floating point database. That is the value you drag into the display's 'Variables on this line' box so the display function can find the **Bargraph** module at display time.

Inputs That Must be Constants: None

Other Inputs:
- Input…floating point input value to be displayed on LCD as a bargraph
- Min value…floating point value corresponding to zero length bar on the LCD. This does not have to be 0.0, but may be any value below the max value below.
- Max value…floating point value corresponding to the maximum length bar on the LCD. This must be greater than the min value above.

Primary Outputs:
- Output…Name.Bar…floating point value clamped to the min/max range defined by the min and max values above. This is the item from the floating point data base that must be referenced by the LCD setup.

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Use one of these modules for each variable you wish to display as a bargraph on the LCD.

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 clear RAM…status trigger input that when true will clear all user memory.

Primary Outputs: None

Outputs for Internal Use:
- Copy of trigger input…Name.Copy…status copy of trigger input to enable rising edge detection.
-

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.

Diagnostics

If you use the Loop/Charger/Diagnostics board in your RUG9, this software module will provide bus voltage, battery voltage and temperature monitoring for you without any required calibration. Those outputs will appear in the floating point database. The module also supplies low bus voltage and low battery voltage alarm status outputs based upon comparison of the bus and battery voltages with a pair of setpoint inputs. The low bus voltage alarm can be used to signal loss of AC power if the card cage is normally powered from AC, since the bus voltage will drop when AC power fails. The low bus and battery alarm statuses will appear in the status database. Raw counts from the A/D converter and calibration counts will appear in the integer database, and are generally only used by the factory for calibration.

Inputs That Must be Constants:

- Card #…which card in card cage (1-8)
- Cage #…which card cage (1-8)

Other Inputs:

- Batt V low alarm SP…battery voltage low alarm setpoint, sets the voltage below which the battery low alarm will be declared.
- Bus V pwr fail SP…bus voltage low alarm setpoint, sets the voltage below which the power fail alarm will be declared.
- Filter sec…low pass filter to be applied to all diagnostics board measurements. The larger this number, the slower the board will be to respond to changes and the more immune it will be to transients.

Primary Outputs:

- Bus voltage…Name.BusV…floating point unit bus voltage in volts, normally about 14 VDC
- Batt voltage…Name.BattV…floating point voltage present on battery charger terminals. Will be close to bus voltage if battery is not connected to battery and ground terminals.
- Temperature…Name.TempF…floating point unit internal temperature, degrees F. Usually higher than ambient due to heat from internal power regulator.
- Low battery status…Name.LowBat…status low battery voltage alarm. Declared if battery voltage is below battery voltage low alarm setpoint.
- Power fail status…Name.PwrFail…status low bus voltage alarm. Declared if bus voltage is below bus voltage low alarm setpoint.

Outputs for Internal Use:

- Raw Bus V…Name.Rawbus…raw count from A/D converter for bus voltage. Used by factory for installing calibration on board.
- Raw Batt V…Name.Rawbatt…raw count from A/D converter for battery voltage. Used by factory for installing calibration on board.
- Raw Temp…Name.RawTemp… raw count from A/D converter for temperature. Used by factory for installing calibration on board.
- EP9-EP19…Name.EP9-19…readback from onboard EPROM for calibration confirmation at factory.

Limitations:

Use no more than one of these modules for each loop supply/charger/diagnostics board in your unit.

Expected Applications:

Use the power fail alarm to detect AC power failure in battery backed up system. Use the battery voltage measurement and the low battery alarm to detect a failing battery, or nearing full discharge when AC power is off. Onboard temperature can be used to detect excessive temperatures in the panel.

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 –2,147,483,648 through 2,147,483,647. 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
- Card #…which card in card cage (1-8)
- Cage #…which card cage (1-8)

Other Inputs:

- Preset status input…trigger input that causes the preset count input to become the present count
- Preset count…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 status…Name.Digin…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. 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 input boards as well as the digital inputs on the combo board.

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. There are no outputs from this module since its output is a relay.

Inputs That Must be Constants:

- Channel #…which input channel (1-8) on the board this module uses
- Card #…which card in card cage (1-8)
- Cage #…which card cage (1-8)

Other Inputs:

- Status input…the status that constitutes the alarm this module is presenting.
- ACK trigger input…trigger input used to acknowledge the alarm, i.e., make the output stop flashing on and off

- Lamp test 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: None

Outputs for Internal Use: None

Limitations: None

Expected Applications:
This module implements standard alarm annunciation functionality. It works with all digital output modules as well as the combo board relay outputs.

DigitalInputAC

This module is used to bring AC type digital inputs into the system. One of these should be used for each AC type digital input to be sensed. When AC voltage is present on the digital input, this module's true output is 1; when AC voltage is absent, 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
- Card #…which card in card cage (1-8)
- Cage #…which card cage (1-8)

Other Inputs: none

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

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Use this module to sense contact closures energized by AC voltage. Can also be used to sense the presence of AC at a load.

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
- Card #…which card in card cage (1-8)
- Cage #…which card cage (1-8)

Other Inputs: none

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

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Use this module to sense contact closures energized by DC voltage such as from the Rug9 loop supply.

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 #…which output channel (1-8) on the board this module uses
- Card #…which card in card cage (1-8)
- Cage #…which card cage (1-8)

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.

GetStringGP

This module is used to capture characters from the SpeechDialAnswer module. It accepts a character each time its 'Trigger get character' input is true. The character is appended to the output string. If the watch character is specified, the module will watch for the occurrence of the watch character and issue a trigger when that particular character is detected on the character input.

Inputs That Must be Constants:
- Max string chars…integer to tell the compiler how many bytes to reserve for the module's string output

Other Inputs:
- Enable…status that when true or blank enables full module operation; false prohibits operation.
- Character input…integer character to be captured when the get character trigger input is true
- Trigger get character…status that when true causes the module to append the character present on the character input to the output string.
- Trigger clear string…status that when true clears the output string.

- Character to watch for…integer that when matched with the character input will cause the 'Have watch char trigger' output to go true.
- Flags…integer of packed flags: value=1…omit the watch character from the output string. Value=2...translate the input to ASCII before appending to output string. Value=3…do both

Primary Outputs:
- String output…Name.Str…output string accumulated so far by module.
- Have watch trigger…Name.WatchTrg…trigger issued when watch character received.

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Use to capture user strings entered by a user by touchtones from the SpeechDialAnswer module

GetUserValue

When triggered, this module issues a prompt on the designated port and accepts the user's floating point response. The response is made available in the floating point database; and a trigger is issued after the user's response is accepted. If the user only hits [ENTER] key, no action results. High and low limits clamp the user's response to a range specified by the programmer. If the **log in** flag is nonzero, then the module will not issue the prompt unless an operator is logged on. If a poke destination is specified, the response is poked to that destination as well as being sent to the floating point database.

Inputs That Must be Constants: None

Other Inputs:
- Prompt string to send…string to prompt operator, such as "New high alarm setpoint="
- Display port #…port number where prompt to be sent and user response to be captured: 0=LCD, 1=CPU serial port, (board# * 3 + port # on board-1) for modem and dual serial/printer boards. E.g., if modem board is in slot 4, then port number is 4*3=12. If dual serial/printer board is in slot 7 and board's second port is to be used, port is (7 * 3 +2-1)=22.
- Display # (-1 if all)…which display number for this port that this module is to use; or all displays for this port if (-1).
- Line # on LCD if port 0…if port 0 (LCD) specified above, then this specifies which line on LCD the prompt message is to use. Top line on LCD is line 0.
- Char loc on LCD if port 0…if port 0 (LCD) specified above, then this specifies number of characters from left side of display where prompt to start. Leftmost character is character 0.
- Trigger input…trigger to cause prompt to be issued and sequence to start to capture user value.
- Poke variable destination…variable dragged from any non-string data base where user's value to be placed upon entry.
- High limit…maximum value user allowed to enter.
- Low limit…minimum value user allowed to enter.
- Login flag…0=allow any user to make entry; 1=user must be logged on before issuing prompt.

Primary Outputs:
- User value…Name.Val…last value user entered.
- Have new input…Name.Trg…trigger issued after new value entered.

Outputs for Internal Use:
- Internal sequencer…Name.Sq…module's sequencer

Limitations:
Line number and character location specifiers only apply to LCD use.

Expected Applications:
Use to gather control inputs and setpoints from operator on screen without the operator having to leave display and access the setpoint list.

GlobalSetpoint

The GlobalSetpoint module provides the same local setpoint entry function as the standard Setpoint module but adds functions necessary to enable the system to pass setpoint values around the communication system and install new values into identically configured GlobalSetpoint modules in other RTU's. It uses an index entry to identify the same GlobalSetpoint in all RTU's that are to maintain the latest value; and it uses an age code to assure that only more recent entries will be accepted by a module from the telemetry system.

Inputs That Must be Constants: None

Other Inputs:
- Prompt string…string the user will see in the RUG9's setpoint list
- Default value…floating point value to be installed in the setpoint when the trigger to install default is true
- Default trigger…trigger status to cause default value to become the setpoint value.
- Max allowed value…floating point value that limits the highest value the module will accept
- Min allowed value…floating point value that limits the lowest value the module will accept
- Index in system (1-1023)…integer value that identifies this particular global setpoint in the system. Anywhere in a network where this value is to be used, the GlobalSetpoint module in the RTU must have this exact same index.
- Multiplier when TX…floating point value by which this module's floating point output will be multiplied before its value is transmitted. Upon reception, the received value will be divided by this number before installation in the receiving GlobalSetpoint's floating point output. This must be one of the following power of 10 numbers: 0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0

Primary Outputs:
- Latest value entered/received…Name.SPG…present floating point value held by setpoint register.
- Trigger had local change…Name.TrgLocChg…trigger status output asserted the first scan after a user has entered a new value locally. Use to trigger or flag need to transmit value.
- Trigger had remote change…Name.TrgRemChg…trigger status output asserted the first scan after a new value has been installed as a result of reception by telemetry.

Outputs for Internal Use:
- Copy of last SP value…Name.Copy…floating point value of setpoint last entered or received to be used to compare against the '.SPG' entry to detect changes in value.
- Age code…Name.Age…integer value in range of 0-7 indicating age if present value for comparison with incoming values to assure that old values are not accepted.

Limitations:
Value range is +/-10E+/-38. Module must be used in conjunction with GlobalSetpointSetup module installed in the same unit, see below.

Expected Applications:
Use the GlobalSetpoint module to give operator a way to input values to be used in control strategies in this and remote units, or to provide the operator a way to enter a setpoint in one RTU that is to be used in one or more other RTU's. Values are saved as floating point, but operator entries can be either floating point or integer. You must include at least one GlobalSP field entry in your telemetry transmit and receive arrays for global setpoint values to be transferred.

GlobalSetpointSetup

The GlobalSetpointSetup module establishes and maintains transmit and receive lists for handling the interface between GlobalSetpoint modules and the various RX and TX communications arrays. When a user changes a global setpoint's value, or when the program pokes a new value into the setpoint, the setpoint's value, index, age code and multiplier are placed into the transmit list for inclusion in the next transmission. If the unit receives a global setpoint from another unit with the same index as one existing in the transmit list, that entry in the transmit list will be deleted by this module. Global setpoints that are received from other units are placed by the receiving function into this module's receive list and held there for one scan so that they can be tested and used by the GlobalSetpoint modules to see if they are newer than the values they already have installed.

Inputs that must be constants:
- Setpt buffer size…Integer sets size of both transmit and receive lists of global setpoints waiting to be sent, and received and waiting to be installed by GlobalSetpoint modules. Should be set to greater than the largest number of setpoint changes that could occur between transmissions of receptions, whichever is greater.

Other inputs:
- Trigger rotate buffer index…Status input that will cause the module to increment the pointer to the queued transmit list. This trigger should be issued once per reception from the master site to assure that all queued global setpoints awaiting transmission will get sent.

Primary Outputs: None

Outputs for Internal Use:
- TX buffer…Name.Buf…Integer array pointer to top of list of setpoints to be sent.
- RX buffer…Name.Rbuf…Integer array pointer to top of list of setpoints just received.
- TX buffer index…Name.Tidx…Integer pointing to next setpoint to send from transmit buffer list.

Limitations:
Buffer size must be greater than 1

Expected applications:
One and only one GlobalSetpointSetup module must be installed in any unit that is to use GlobalSetpoint modules to transfer setpoint values to other units.

MsgToDSP

This module issues a string to the designated port when triggered. String can be up to 79 characters, but should not exceed display width.

Inputs That Must be Constants: None

Other Inputs:
- String to send…string, up to 79 characters to be sent to port or LCD.
- Display port #…port number where prompt to be sent and user response to be captured: 0=LCD, 1=CPU serial port, (board# * 3 + port # on board-1) for modem and dual serial/printer boards. E.g., if modem board is in slot 4, than port number is 4*3=12. If dual serial/printer board is in slot 7 and board's second port is to be used, port is (7 * 3 +2-1)=22.
- Display # (-1 if all)…which display number for this port that this module is to use; or all displays for this port if (-1).
- Line # on LCD if port 0…if port 0 (LCD) specified above then this specifies which line on LCD the prompt message is to use. Top line on LCD is line 0.
- Char loc on LCD if port 0…if port 0 (LCD) specified above then this specifies number of characters from left side of display where prompt to start. Leftmost character is character 0.
- Trigger input…trigger to cause module to issue string to designated port.

Primary Outputs: None

Outputs for Internal Use: None

Limitations:
Line number and character location specifiers only apply to LCD use.

Expected Applications:
Use this module to send individual strings to ports in response to events such as telemetry receptions, alarms, etc.

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 flow meters and convert to engineering units.

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

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.

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

Other Inputs:
- Enable…when true or blank will enable pulses to be generated
- Trigger pulse…trigger to start each pulse if cycle time blank
- Pulse duration sec…floating point duration in seconds for each pulse when triggered or for each pulse to be issued repetitively if cycle time specified.
- Cycle time sec…floating point input setting the expected pulse repetition time

Primary Outputs:
- Output state…Name.Out…status copy of present output state.

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 digital input to flow rate and sends to floating point database. Calculation is performed at each trigger. For each calculation, the module reads number of pulses, and number of milliseconds since prior trigger, then applies user's flow per pulse factor to calculate average flow rate per second between triggers. Output flow rate units is 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, output in this case must be multiplied by 60 (or the flow per pulse could be multiplied by 60). Module should not be triggered too frequently or output will be very granular. 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) on the board this module uses
- Card #…which card in card cage (1-8)
- Cage #…which card cage (1-8)

Other Inputs:

- Trigger input…trigger input that triggers module to read count and calculate flow rate
- Volume per pulse…floating point factor defining how many gallons are represented by each input pulse
- 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
- 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

Primary Outputs:

- Flow rate per second…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
- Old time count…Name.Oldtime…count in internal timebase units to compare against new time reading
- Temp volume…Name.Temp…floating point temporary total accumulator
- 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 262,144 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.

ReadCalFromEEPROM

This module is used by the factory to force reading of EEPROM calibration constants from all boards with onboard calibration. This is used during factory test; not for user application.

Setpoint

The setpoint module enables you to specify a setpoint to be added to the RUG9's setpoint list that an operator can use to enter values into the RTU's system to control operations. Access to setpoints in the RUG9 is secured with an access code that must be entered correctly before access is granted. 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 RUG9'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

Notice that the setpoint module has default value and default installation trigger inputs. Since the RUG9 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. 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 TriggerOnKey module).

Inputs That Must be Constants: None

Other Inputs:

- Prompt string…string the user will see in the RUG9's setpoint list
- Default value…floating point value to be installed in the setpoint when the trigger to install default is true
- Default trigger…trigger status to cause default value to become the setpoint value.

Primary Outputs:

- Value user entered…Name.SP…present floating point value held by setpoint register.

Outputs for Internal Use: None

Limitations:

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

Expected Applications:

Use the setpoint module to give 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.

Sleep

The sleep module is used to install the sleep interval and numerous flags that specify under what conditions the unit is to awaken, and then to put the unit to sleep, i.e., to power down the main RUG9 bus with only the sleep board remaining powered. The sleep board will then count time for the next wakeup, and monitor the specified measurements and statuses to detect whether to awaken the full unit on the basis of a change of state or alarm condition. For this to work, DC power for the unit must be applied to the sleep board and not to any other boards. In addition, before triggering the sleep module, trigger the **SleepSP** module to install setpoints. Note that it is imperative that a sleep board be installed in the slot referenced by this module because, if missing, the RUG9 watchdog timer may timeout waiting for the board to respond.

Inputs That Must be Constants:

- Card #…which card in card cage (1-8)

Other Inputs:

- Trigger to install setup…status trigger that installs the listed flags and wakeup time, and then causes the sleep board to power down the main RUG9 bus
- Next wakeup, sec…integer seconds from present trigger until the unit is to be re-powered unless another condition demands that the unit be awakened earlier
- Random offset seconds…integer random number of seconds to be added to wakeup interval so that multiple units in the field will not repetitively wake up simultaneously
- Enable AUX cycling…status to enable AUX relay cycling to power up external equipment and onboard touch tone receiver while the RUG9 remains asleep

- Enable touch tone wakeup…status to enable wakeup on reception of a 3 digit touch tone code
- Enable ringing wakeup…status to enable wakeup on reception of a telephone ring signal
- Wake if A/D 1>SP…status to enable wakeup if analog input #1 on sleep board exceeds setpoint
- Wake if A/D 2>SP…status to enable wakeup if analog input #2 on sleep board exceeds setpoint
- Wake if A/D 1<SP…status to enable wakeup if analog input #1 on sleep board is less than setpoint
- Wake if A/D 2<SP…status to enable wakeup if analog input #2 on sleep board is less than setpoint
- Wake anemom rate>SP…status to enable wakeup if wind speed exceeds setpoint
- Wake on DI1 cnt…status to enable wakeup if specified change of state detected on sleep board DI1
- Wake on DI2 cnt…status to enable wakeup if specified change of state detected on sleep board DI2
- Wake on DI3 cnt…status to enable wakeup if specified change of state detected on sleep board DI3
- Wake on DI4 cnt…status to enable wakeup if specified change of state detected on sleep board DI4
- Wake on encoder 1>SP…status to enable wakeup if value on encoder 1 exceeds setpoint
- Wake on encoder 2>SP…status to enable wakeup if value on encoder 2 exceeds setpoint
- Wake on encoder 1<SP…status to enable wakeup if value on encoder 1 is less than setpoint
- Wake on encoder 2<SP…status to enable wakeup if value on encoder 2 is less than setpoint
- Type: 0=quadrature, 1=acro…integer to select type of shaft encoder algorithm
- DI count: 0=rising, 1=falling…status to select wakeup on DI rising edge or falling edge

Primary Outputs: None

Outputs for Internal Use: None

Limitations:
Wakeup interval must be less than 32,768 seconds. Once this module is triggered, other sleep module communications with sleep board are suspended so that this module has top priority to communicate with the sleep board. Any or all above flags can be enabled simultaneously; whichever flagged condition occurs first will awaken the unit. If none of the flagged conditions exists before the wake interval times out, then the unit will awaken at that time. The wake button is always enabled to manually awaken the unit.

Expected Applications:
Use the sleep board to put the unit to sleep and wake occasionally in order to save power in power limited applications such as solar or battery only applications. This module is necessary to set the wakeup conditions and put the unit to sleep.

SleepPresets

This module is used to preset registers internal to the sleep board. Those registers are the four digital input counters, which are normally used to count tipper bucket rain gauge pulses; and the two shaft encoder up/down counters so they accurately represent measured analog values. Note that it is imperative that a sleep board be installed in the slot referenced by this module because, if missing, the RUG9 watchdog timer may timeout waiting for the board to respond.

Inputs That Must be Constants:
- Card #…which card in card cage (1-8)

Other Inputs:
- Trigger to preset DI's…status trigger to install counts in digital input counters
- DI 1 count preset…integer value to install into counter for DI 1
- DI 2 count preset…integer value to install into counter for DI 2
- DI 3 count preset…integer value to install into counter for DI 3
- DI 4 count preset…integer value to install into counter for DI 4
- Trigger to preset Enc 1…status trigger to install value into encoder 1 register

- Encoder 1 preset value…floating point EU value to install into encoder 1 register
- Trigger to preset Enc 2…status trigger to install value into encoder 2 register
- Encoder 2 preset value…floating point EU value to install into encoder 2 register

Primary Outputs: None

Outputs for Internal Use: None

Limitations:
Digital input counter range is 0 to 32,767. Encoder presets are in engineering units. When encoder preset trigger occurs, background software sets the encoder counters to half scale and translates to engineering units for readout by **SleepRead** module.

Expected Applications:
This module enables you to set the DI counters and encoder registers to known engineering units values.

SleepRead

Module reads sleep board internal registers and presents values in engineering units form. It also sets calibration for sleep board analog inputs, encoders and anemometer. Note that it is imperative that a sleep board be installed in the slot referenced by this module because, if missing, the RUG9 watchdog timer may timeout waiting for the board to respond to polls by this board.

Inputs That Must be Constants:
- Card #…which card in card cage (1-8)

Other Inputs:
- A/D 1 EU at 0 volts…floating point setting for A/D number 1 low end calibration
- A/D 1 EU at 5 volts…floating point setting for A/D number 1 high end calibration
- A/D 2 EU at 0 volts…floating point setting for A/D number 2 low end calibration
- A/D 2 EU at 5 volts…floating point setting for A/D number 2 high end calibration
- Encoder 1 EU per count…floating point setting for value of each pulse of encoder 1
- Encoder 2 EU per count…floating point setting for value of each pulse of encoder 2
- Anemometer MPH per Hz…floating point anemometer scale factor for MPH/Hertz

Primary Outputs:
- A/D 1 value EU…Name.AI1…floating point engineering units value of A/D channel 1
- A/D 2 value EU…Name.AI2…floating point engineering units value of A/D channel 2
- DI 1 count…Name.Cnt1…integer count present for sleep board DI channel 1
- DI 2 count…Name.Cnt2…integer count present for sleep board DI channel 2
- DI 3 count…Name.Cnt3…integer count present for sleep board DI channel 3
- DI 4 count…Name.Cnt4…integer count present for sleep board DI channel 4
- Encoder 1 EU…Name.Enc1…floating point engineering units value of encoder 1 register
- Encoder 2 EU…Name.Enc2…floating point engineering units value of encoder 2 register
- Windspeed now EU…Name.Wind…floating point wind speed in engineering units
- Wakeup reason…Name.Wak…integer code for reason unit woke up:
  - 0=power application
  - 1=time
  - 2=touch tone code
  - 3=phone line ringing
  - 4=anemometer rate > setpoint
  - 5=DI 1 rising or falling edge

- 6=DI 2 rising or falling edge
- 7=DI 3 rising or falling edge
- 8=DI 4 rising or falling edge
- 9=A/D #1 outside specified level
- 10=A/D #2 outside specified level
- 11=Encoder 1 outside level
- 12=Encoder 2 outside level
- 13=wake button

- Last sleep time, sec…Name.Time…how long in seconds unit was asleep
- A/D #1 raw count…Name.Raw1…A/D channel 1 raw count
- A/D #2 raw count…Name.Raw2…A/D channel 2 raw count
- Digital input #1…Name.DI1…present status of DI #1
- Digital input #2…Name.DI2…present status of DI #2
- Digital input #3…Name.DI3…present status of DI #3
- Digital input #4…Name.DI4…present status of DI #4

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Use this module to read values measured by the sleep board and to determine why it woke up. Module reads the sleep board approximately once per second.

SleepSP

This module is used to install setpoints necessary to compare against various analog register contents in order to wake the unit up if one of the values falls outside its designated range. These values should be installed at least one second before putting the unit to sleep. Channels that are not enabled for wakeup in **Sleep** module can be left blank.

Inputs That Must be Constants:
- Card #…which card in card cage (1-8)

Other Inputs:
- Trigger to install setup…status trigger input to force setpoints to be installed onto sleep board.
- A/D 1 setpoint EU…floating point value that analog input 1 must fall below or exceed to cause wakeup
- A/D 2 setpoint EU…floating point value that analog input 2 must fall below or exceed to cause wakeup
- Encoder level 1 SP…floating point value that encoder 1 must fall below or exceed to cause wakeup
- Encoder level 2 SP…floating point value that encoder 2 must fall below or exceed to cause wakeup
- Ttone code (3 char)…integer 3 character code that must be received by sleep board to wake up unit
- Rings to wake…integer count of rings that must occur before unit wakes up
- AUX ON interval sec…integer seconds the AUX relay is to remain on during each AUX cycle
- AUX ON duration sec…integer seconds the AUX relay is to remain off during each AUX cycle
- Anemom rate SP EU…wind speed that must be exceeded to cause wake up

Primary Outputs: None

Outputs for Internal Use: None

Limitations:
Trigger this module at least one second before putting unit to sleep. Touch tone code must be exactly 3 characters in range of 0 through 9. AUX ON interval and duration must be in the range of 0 through 32,767 seconds. Blank disables AUX cycling.

Expected Applications:
This module is used to install setpoints that are used to determine if analog values are to cause the unit to wake up.

SysSetup1

The system setup 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 RUG9 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.

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
- Security code on SP…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.
- LCD bklight timeout sec…integer duration in seconds that backlight will stay on after keystroke. Leaving cell blank or setting to zero will keep LCD backlight on continuously.
- Trigger LCD ON…status trigger turns on LCD backlight.
- Trigger LCD OFF…status trigger turns off LCD backlight.
- LCD contrast (1=temp ctrl)…integer contrast setting control: 0 or blank=contrast controlled by user menu entry; 1=background software adjusts LCD contrast based on loop supply board's temperature measurement (your unit must have a loop supply/charger board installed, and you must have a **Diagnostics** module in your project for this to work); >1=value is used to set contrast.
- Customizer…integer packed bits to enable custom titling on some menus.

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

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. Temperature control of LCD will give bad contrast if loop board is not present in base card cage.

Expected Applications:
Used in all projects to provide commonly used services.

WriteCalToEEPROM

This module is used by the factory to force writing of EEPROM calibration constants to all boards with onboard calibration. This is used during factory test; not for user application.

Math Modules

Arccos

The **Arccos** module provides a floating point angle output in radians, Y, calculated from the cosine value input X, Y=arccos(X). Acceptable input range is –1.0 to +1.0. Output range is – pi/2 to +pi/2. Calculations that would give undefined or infinite results return the largest or smallest floating point value possible.

Inputs That Must be Constants: None

Other Inputs:
- X…floating point cosine whose angle is to be calculated.

Primary Outputs:
- Y…angle in radians whose cosine is the module input X

Outputs for Internal Use: None

Limitations:
Input range is –1.0 to +1.0

Expected Applications:
Sometimes used in flow and volume calculations.

Arcsin

The **Arcsin** module provides a floating point angle output in radians, Y, calculated from the sine value input X, Y=arcsin(X). Acceptable input range is –1.0 to +1.0. Output range is – pi/2 to +pi/2. Calculations that would give undefined or infinite results return the largest or smallest floating point value possible.

Inputs That Must be Constants: None

Other Inputs:
- X…floating point sine whose angle is to be calculated.

Primary Outputs:
- Y…angle in radians whose sine is the module input X

Outputs for Internal Use: None

Limitations:
Input range is –1.0 to +1.0

Expected Applications:
Sometimes used in flow and volume calculations.

Arctan

The **Arctan** module provides a floating point angle output in radians, Y, calculated from the tangent value input X, X=arctan(Y). Output range is – pi/2 to +pi/2. Calculations that would give undefined or infinite results return the largest or smallest floating point value possible.

Inputs That Must be Constants: None

Other Inputs:
- X…floating point tangent whose angle is to be calculated.

Primary Outputs:
- Y…angle in radians whose tangent is the module input X

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Sometimes used in flow and volume calculations.

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 0 to 65,535.

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.

Characterization Table

Implements short 16 row table of value pairs that you can use to implement a lookup table to convert an input value to an output value based on table entries. Module finds nearest input pair in input value list, then interpolates to output correct conversion of input based on output values in list.

Inputs That Must be Constants: None

Other Inputs:
- Input value to lookup…floating point value to compare against input list and convert based on corresponding output values in list.
- Table input value 1 to 16…floating point input values against which input above is compared and interpolated.
- Table output value 1 to 16…floating point output values corresponding to input values above.

Primary Outputs:
- Table output value…Name.OUT…floating point output calculated from table lookup based on input value.

Outputs for Internal Use: None

Limitations:
Input values must ascend or descend in value. Outputs can represent any function. Input outside range of input values table results in extrapolation based on first pair of input values or last pair of input values.

Expected Applications:
Module is used to linearize transducer values that cannot be represented by other math modules. Module is also used to assist volume calculation of irregular shaped containers.

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.

Cosine

The **Cosine** module provides a floating point output in the range of –1.0 to +1.0, Y, calculated from the angle input X, Y=cos(X), where X is in radians. Acceptable angle input range should be 0.0 to 2pi, however, module will correctly calculate cosine of any input value. Calculations that would give undefined or infinite results return the largest or smallest floating point value possible.

Inputs That Must be Constants: None

Other Inputs:
- X…floating point angle whose cosine is to be calculated.

Primary Outputs:
- Y…floating point cosine of X

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Sometimes used in flow and volume calculations.

Cotangent

The **Cotangent** module provides a floating point output in the range of –1.0 to +1.0, Y, calculated from the angle input X, Y=cotan(X), where X is in radians. Acceptable angle input range should be 0.0 to 2pi, however, module will correctly calculate cotan of any input value. Calculations that would give undefined or infinite results return the largest or smallest floating point value possible.

Inputs That Must be Constants: None

Other Inputs:
- X…floating point angle whose cotan is to be calculated.

Primary Outputs:
- Y…floating point cotan of X

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Sometimes used in flow and volume calculations.

FloatToInteger

The **FloatToInteger** module converts an input floating point or 32 bit integer 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. The module also splits the floating point input into two 2-byte words for use in telemetry arrays. The FloatSwap output is the input value reproduced with the first 16 bit word swapped with the last 16 bit 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
- MS word of float…Name.FloatMS…integer, 16 bits, of most significant two bytes of input floating point binary word.
- LS word of float…Name.FloatLS…integer, 16 bits, of least significant two bytes of input floating point binary word.
- Float with wds swapped…Name.FloatSwap…floating point (32 bit) representation of original input value with the first 16 bits swapped with the last 16 bits.

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:
- 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, 0.1 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:

- 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:
- 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.

FlowManning

This module calculates gravity flow through an open channel using the Manning flow calculation algorithm. Any of three types of channel can be selected: round pipe, U-shaped pipe, or rectangular pipe. 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:
- 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.
- Channel diameter or width, ft…floating point pipe diameter or channel width in feet.
- Slope…floating point channel/pipe ratio of fall divided by run
- Roughness factor…floating point Manning channel roughness factor (usually 0.01 to 0.04)
- Type…integer type of channel: 1=round pipe, 2=U-shaped pipe, 3=rectangular pipe

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 channel.

FlowOvershot

This module calculates flow through an overshot gate type weir. 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:
- 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 hinge of gate.
- Gate width, ft…floating point width of movable gate, ft.
- Gate angle from hor. deg…floating point angle of submerged gate from horizontal in degrees
- Gate height at 90 deg. Ft…floating point gate height from hinge to lip, feet
- Approach chan width, ft…floating point width of approach channel, ft.

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 submerged gates hinged on bottom of channel.

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:
- 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.

FlowParshall

This module calculates the flow through a Parshall flume of 1 to 36 inches width. 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:
- 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:
- 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:
- 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.

FlowVNotchWeir

This module calculates the flow through a V-notch weir of user specified notch angle. 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:
- 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.
- Weir angle degrees…floating point weir angle

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.

GasFlow

This module calculates gas flow using the algorithm specified by API MPMS 14.3.3. This calculation will be performed when triggered, or will be performed each scan if the trigger is left blank. Each calculation results in gas flow result of standard cuft/hr and base cuft/hr.

Inputs That Must be Constants: None

Other Inputs:
- Trigger calculation…status if true or blank, calculation will be performed
- Upstream pressure PSI…floating point gage pressure upstream of orifice plate
- Upstream temp F…floating point temperature near upstream tap in deg F, converted to deg R by program.
- Orifice diff press in H2O…floating point differential pressure across orifice plate in inches of water at 60F
- Orifice plate diam in…floating point orifice plate diameter in inches at 68 degrees F
- Tube diam in…floating point tube diameter in inches at 68 degrees F
- Gas base pressure psia…floating point base pressure upon which user wishes to express flow rate, psia
- Gas base temp F…floating point base temperature upon which user wishes to express flow rate, deg F
- Gas CH4 moles/mole…floating point methane constituent of gas in moles/mole
- Gas C2H6 moles/mole…floating point ethane constituent of gas in moles/mole

- Gas C3H8+ moles/mole…floating point propane and heavier gases constituent of gas in moles/mole
- Gas CO2 moles/mole…floating point carbon dioxide constituent of gas in moles/mole
- Gas N2 moles/mole…floating point nitrogen constituent of gas in moles/mole
- Orifice metal type (1-3)…integer orifice metal type 1=stainless steel, 2=monel, 3=carbon steel
- Tube metal type (1-3)…integer tube metal type 1=stainless steel, 2=monel, 3=carbon steel

Primary Outputs:
- Qv…Name.Qv…floating point flow result in std cuft/hr
- Qb…Name.Qb…floating point flow result in base cuft/hr

Outputs for Internal Use: None

Limitations:
Any intermediate results that would result in a math error such as divide by zero or square root of a negative will result in flow being set to zero.

Expected Applications:
Used to accurately calculate flow in gas pipelines.

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:
- 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.

LimitInput

The **LimitInput** module range tests and limits a value to the range set by the upper and lower setpoints specified, then pokes the range limited result back to the register from which it was read. It will not work on received TLM words.

Inputs That Must be Constants: None

Other Inputs:
- 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: None

Outputs for Internal Use: None

Limitations:
Does not work on receive telemetry array entries.

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

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:
- 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 11 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:
- 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 RUG9 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, 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 both to the left and to the right, 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:
- 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 value specifying leading characters if characters are needed to the left of the output numeric result: 0=no fill, 1=use blanks, 2=use the zero [0] character.
- Trailing fill flag…integer value specifying trailing characters if characters are needed to the right of the output numeric result: 0=no fill, 1=use blanks, 2=use the zero [0] character.

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.

N-th Order Polynomial

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:
- 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, for linearizing transducers among other applications.

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:
- 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.

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:
- 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.

Tangent

The **Tangent** module calculates Y=tan(X) where X is in radians. Acceptable angle input range should be 0.0 to 2pi, however, module will correctly calculate tan of any input value. Calculations that would give undefined or infinite results return the largest or smallest floating point value possible.

Inputs That Must be Constants: None

Other Inputs:
- X…floating point angle whose tangent is to be calculated.

Primary Outputs:
- Y…floating point tan of X

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Sometimes used in flow and volume 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.

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 (0-31)…integer specifying which bit of the input integer constitutes the least significant bit of the value to be unpacked.
- MS bit to include (1-31)…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 (0-31)…integer specifying which bit of the input integer constitutes the least significant bit of the value to be unpacked.

- MS bit to include (1-31)…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

AlarmHi

The **AlrmHi** module compares a floating point value with a setpoint and, if the value exceeds the setpoint for the period of a specified delay, turns on its output. If its delay is unspecified, then no delay will be applied. If the delay is specified and is set to zero, then the alarm 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 below the setpoint, the status output will turn off without delay.

Inputs That Must be Constants: None

Other Inputs:
- Input…floating point value to be compared against a setpoint.
- High alarm setpoint…floating point setpoint against which the input is compared.
- Delay, sec…integer period in seconds that the setpoint must be exceeded to declare an alarm. If the delay is zero, then the alarm is disabled.

Primary Outputs:
- Alarm output…Name.HiAlrm…status that becomes true if alarm declared by comparison

Outputs for Internal Use:
- Delay timer…Name.Timer…integer countdown delay timer counting 10^{th}'s of a second

Limitations: None

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

AlarmLo

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

Inputs That Must be Constants: None

Other Inputs:

- Input…floating point value to be compared against a setpoint.
- Low alarm setpoint…floating point setpoint against which the input is compared.
- Delay, sec…integer period in seconds that the input value must be below the setpoint to declare an alarm. If the delay is zero, then the alarm is disabled.

Primary Outputs:

- Alarm output…Name.LoAlrm…status that becomes true if alarm declared by comparison

Outputs for Internal Use:

- Delay timer…Name.Timer…integer countdown delay timer counting 10^{th}'s of a second

Limitations: None

Expected Applications:

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

Alarm Mismatch

The alarm mismatch 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.

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

Primary Outputs:

- Alarm output…Name.Alrm…status indicating: 0=input statuses match, 1=statuses have mismatched for the dalay period.

Outputs for Internal Use:
- Delay timer…Name.Timer…integer countdown delay timer counting 10th's of a second

Limitations: None

Expected Applications:
Used to detect pump failure or other control element failure.

Alternator

The Alternator module alternates which pump is called whenever a new pump call is received. Whenever a pump output is switched ON or OFF, the delay timer will delay any further switching until the delay has timed out. If either pump lockout is true, that call output will be held off until the lockout is cleared; and only the pump that is not locked out will be called. The module will not allow both pumps to be called at the same time. If the Force alternate now status input becomes true, the pump call outputs will alternate independently of the pump call input.

Inputs That Must be Constants: None

Other Inputs:
- Pump call…status indicating whether an output pump is to be called.
- Delay, sec…integer period in seconds following a pump on/off switch event during which no pump switching is allowed.
- Pump 1,2 lockout…status inputs that when true prohibit the corresponding pump output from being called.
- Force alternate now…status trigger that will cause the output pump call to switch to the alternate pump.

Primary Outputs:
- Pump 1,2 call…Pump calls

Outputs for Internal Use:
- Delay timer…Name.Dly…integer countdown delay timer counting 10th's of a second
- Lead status image…Limage…status of last lead designator
- Force alternate imate…Fimage…status image of last force alternate status input

Limitations: None

Expected Applications:
Used to control two pumps with call alternating each time the pump call input turns on.

AND2X8

The And2X8 module provides eight 2-input AND gates. For each of the 8 gates, if both specified status inputs are ON (1A is on AND 1B is on), then that gate's true status output (ANDout1) will be on. If either of the status inputs is OFF, then the true status output will be off. The ANDbar1 output is the complement of the true output, i.e., it will be off when ANDout1 is on, and it will be on when ANDout1 is off. Each of the eight gates is completely independent of the others.

Inputs That Must be Constants: None

Other Inputs:
- Input 1A…status input to be ANDED with other
- Input 1B…status input to be ANDED with other
- …

Primary Outputs:
- AND output #1 …Name.ANDout1…status output that will be true only if both specified inputs for that gate are ON
- Inverted output #1…Name.ANDbar1…status output that will be true if any specified input is OFF

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Use the AND2X8 when you need multiple 2 input AND gates, for example, to enable/disable up to 8 alarms independently.

ANDgate

Performs the logical AND of the 16 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:
- Input #1…status input to be ANDED with others
- Input #2…status input to be ANDED with others
- …
- Input #16…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).

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 32 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:
- 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 –2,147,483,648 to +2,147,483,647.

Expected Applications:
Use for general event counting and sequencing.

CounterStack

Stack of five counters, each with a trigger input to count, and a preset value that will be installed whenever the trigger to install all preset values is true. Each counter counts up whenever its trigger input is true. A continuously true status input to a counter would make the counter count up on each program scan. Each counter can be cleared, by triggering the "trigger clear counters" status input. The counter stack enables you to install multiple counters using one module rather than as many as five to accomplish the same function. The counter stack is typically also used in conjunction with a table and the polling controller to aid in keeping communications statistics.

Inputs That Must be Constants: None

Other Inputs:
- Trigger clear counters…trigger status input to set all counts to zero.
- Trigger preset counters…trigger status input to preset all counters to their corresponding preset input values
- Counter 1…5 preset values…integer values to be installed as the counts when the "Trigger preset counter" input is true.
- Counter 1…5 count triggers…trigger status input that will cause the counter to count when true.

Primary Outputs:
- Counter 1…5 value…Name.V1…Name.V5…Integer count values for the 5 counters.

Outputs for Internal Use: None

Limitations:
Count range is –2,147,483,648 to +2,147,483,647.

Expected Applications:
Use for general counting, or along with table and poll sequencer to keep communications statistics.

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 32 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 preset trigger is true.

Primary Outputs:
- State…Name.State…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 –2,147,483,648 to +2,147,483,647.

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:

- 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 0.1 second. Timer range is 2,147,483,65 seconds.

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 10^{th}'s of a 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.

EventLogger

The **EventLogger** module accepts up to 10 status inputs and logs a message for each status that changes state. 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 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
- 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 establishes a circular buffer to hold event messages, time tags, on/off states and analog values associated with the occurrences of statuses changing state. You may have as many **EventLogSetup** modules as you need. Each establishes a separate event log that can be independently displayed. In addition to establishing the logger buffer, the module also provides for automatic logging of setpoint changes, and for logging alarm acknowledgements using the dialer. Aside from these, this module does no other actual logging. You must use one or more **EventLogger** modules described above to log other status events. You can make the event log as large as you wish within the RAM capacity of the RUG9. Each event occupies 16 bytes: 4 bytes for time tag, 4 bytes to point to the event's message, 4 bytes for the on/off state, and 4 bytes for capturing an optional floating point value. Therefore, if you specify a 100 element log, it will require 1600 bytes of RAM. When the log fills up, the next event will overwrite the oldest event in the log. If you set the flag to log setpoints, the **EventLogSetup** module will store the time of occurrence, the setpoint prompt string and the value the operator entered. If you set the flag to log dialer ACK events, the module will log "Alarms acknowledged by operator X" each time an operator acknowledges alarms using the dialer.

Dumping Event Log to a Port

This module also provides a way to send the log or individual messages to a serial or printer port. (Note that the LCD display provides separate means to display the event log using the @E command. See display setup.) If you specify board and port number inputs, then a trigger on the "Trigger send log to port" input will cause the RUG9 to dump the entire log to the port.

Dumping Events as They Occur

If you specify board and port number inputs and set the mode to one, then the module will send each logged event to the port as it happens. This works for both serial and printer ports.

Inputs That Must be Constants:
- Number events to log…integer that establishes how many events this log is to hold.

Other Inputs:
- Trigger erase log…trigger status input that, when true, will erase the log.
- 1=log setpoint changes…integer input that when set to one causes the module to log setpoint changes. Entry blank or zero disables setpoint logging.
- 1=log dialer ACK events…integer input that when set to one causes the module to log dialer alarm acknowledgements. Entry blank or zero disables operator acknowledgment logging.
- Trigger send log to port…trigger status input that will cause the module to dump the entire log to board and port specified below.
- Port # to send to…integer input that specifies to which port on a board the module is to send log when triggered to dump log or individual events.
- Board # to send to…integer input that specifies to which board the module is to send log when triggered to dump log or individual events.
- Mode…integer input that, when set to one, causes module to dump individual events as they occur to board and port specified above.

Software Modules

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.
- Have change in log…Name.Chg…Status trigger indicating that new event has been logged.

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.
- Temp index…Name.Tdx…Integer count of number of events that have been dumped.
- Old index…Name.Old…Integer index of last event logged
- Last index sent…Name.Last…Integer index of last event transmitted

Limitations: None

Expected Applications:
Used to establish and supervise operation of each event log.

FlipFlop

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 latched by the flip flop, and will become its new output. 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:
Used to latch statuses and to provide a toggle (alternating) function.

FlipFlopRS

The **FlipFlopRS** module provides a two state latch that can be set, cleared, or cleared. The set and clear inputs can be used to unconditionally set the flip flop state to 1 or 0 respectively. The dominance state 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 (sets Q=1)…status input that when 1 causes the Q output to go to one.
- Clear input (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. If left blank, the dominance state defaults to 0.

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:

Used 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 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.

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 is 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…floating point delay before intrusion declared if not acknowledged.
- Auto reset delay setpoint sec…floating point delay to clear alarm after assertion
- Hold off delay sec…floating point delay to disable alarm for maintenance
- Reset alarm status input…status input to silence alarm

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 for 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.

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.

Latch32Floats

When triggered, the Latch32Floats module captures the input value and saves it in one of 32 output latches. The output latch addressed is set by the sum of the Index input plus the Offset input. If that sum is not in the range of 0 to 31, then no action is taken. The latch done trigger output will be asserted whenever a new value is latched. It can be used to clear the input source. The input to preset all latches when true, will set all latches to the Preset value. This module is useful for capturing values based upon the index, such as capturing the daily rainfall total for each day of a month. In that case, the index would be the day of the month and the input value to latch would be the total rainfall for that day.

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 value to latch…Floating point input to be captured when the trigger input is true.
- Index…integer that when added to the offset specifies which output is to latch the input value. If Index+Offset is not in the range of 0-31, no action is taken.
- Offset… integer that when added to the index specifies which output is to latch the input value. If Index+Offset is not in the range of 0-31, no action is taken.
- Preset all latches trigger…status input that when true causes the preset value to be transferred to all outputs.
- Preset value…floating point value that will be transferred to all outputs when the Preset all latches trigger is true.

Primary Outputs:
- Latch done trigger…Name.Trg…status trigger output indicating that a value has been latched. Used to clear an input after its value is captured.
- Latched 0-31…Name.L0-L31…floating point outputs that hold the last input value present when the trigger occurred and index+offset specifed the particular output

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Use to capture analog values such as totalizations for any of several index+offset combinations.

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 type 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:
Used 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.

Lead Lag Sequencer

The lead lag sequencer performs lead pump rotation of up to 8 pumps. It accepts up to 8 input pump calls, usually from pump up or pump down control modules. 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. The block diagram below illustrates a typical use of the LeadLag sequencer in a pump up application with four pumps. 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.

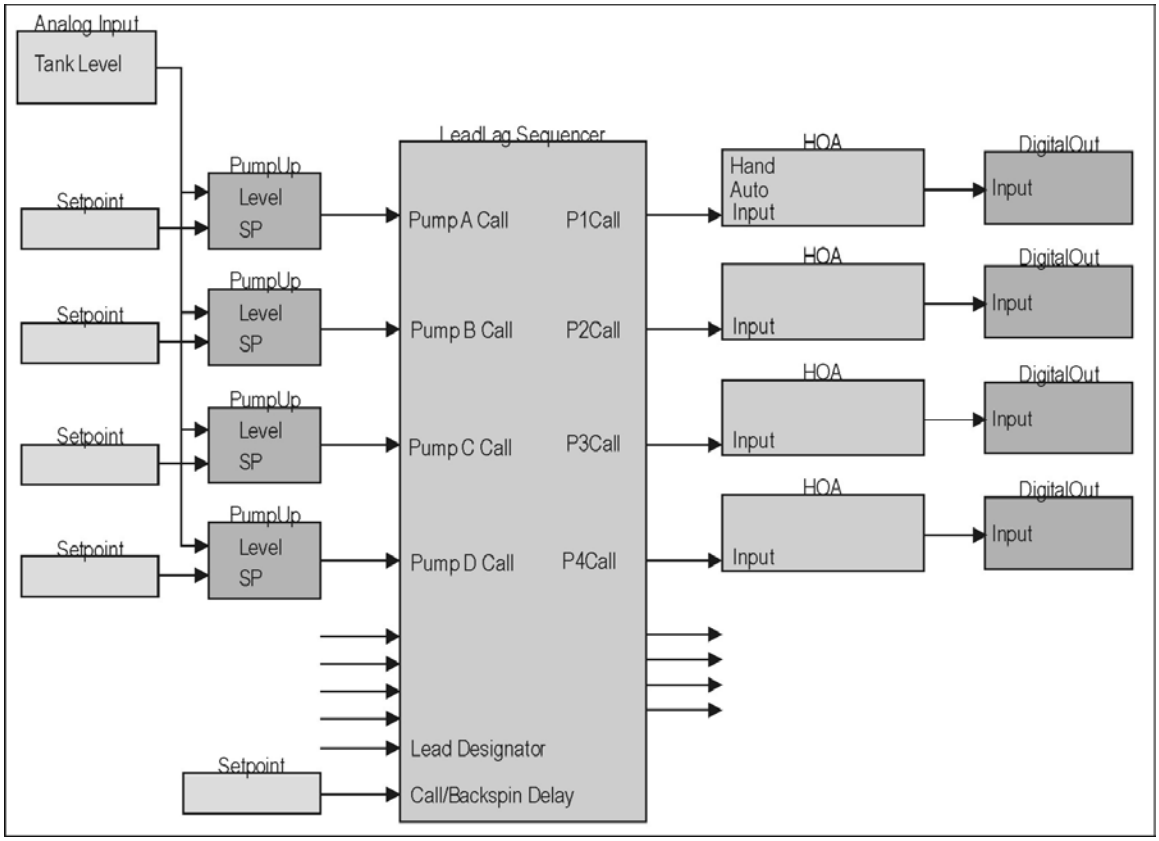


Figure 85 Lead Lag Sequencer Typical Use

Inputs That Must be Constants: None

Other Inputs:

- Pump A-H call…status input from pump up or pump down controller indicating the need to call one or more pumps.

- 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 1-8 call…Name.P1…Name.P8…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 the modules in the above figure to implement multi-pump controls with lead lag sequencing, lead pump rotation, and backspin delay.

Lead Lag Sequencer 4

The LeadLagSeq4 module performs lead pump rotation of up to 4 pumps with pump lockout capability. It accepts up to 4 input pump calls, usually from pump up or pump down control modules. 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 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 input from pump up or pump down controller indicating the need to call one or more pumps.
- Pump1-4 lockout…status that when non-zero, will cause the sequencer to skip that pump output in its calling sequence. Blank of zero entries will not cause lockout.
- 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. If this input is derived from a setpoint, the operator can specify the lead pump; or he can set it to zero enabling the module to rotate the lead pump.
- 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 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 16 inputs/constants to be sent to its output. If the inputs are taken from a database, then the module constitutes a 16 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 1 through 16, then the module outputs a value of 0.0.

Inputs That Must be Constants: None

Other Inputs:
- Control (1-16)…integer input to select one of the 16 analog inputs/table entries.
- Entry 1-16…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 16 entry inputs by the control index.

Outputs for Internal Use: None

Limitations:
Control index out of range of 1-16 returns value of 0.0.

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

MakeString

When triggered, the MakeString module produces a formatted string from the format definition and up to 10 inputs. The format definition is the same as on displays; i.e., it uses a fixed string with embedded @@@.@ symbols to specify the format of each input to be included in the output string. Example of a format string: "Tank level=@@.@feet, flow=@@@@GPM". In that case, input 1 would be presented as floating point with one decimal place, and input 2 would be presented as a 4 digit integer. Use <xx> format for special ASCII characters. I.e., <12> specifies FF character. To append two string inputs together such as Input 1='ABCDEF' and Input 2='GHIJKL', place the null character <0> between the two @ fields. I.e., use a format such as: Output=@@@@@<0>@@@@@. The result would be: Output=ABCDEFGHIJLK. Max string is 79 characters.

Inputs That Must be Constants:
- Output string max chars…integer that sets how much RAM the compiler must set aside for the string outputs of the module.

Other Inputs:
- Trigger input…status trigger input that forces the module to prepare a new string output.

- 1=keep leading zeroes…status input that when true will prepend leading zeroes to numeric fields.
- String format definition…string that defines the format of the output string as described above.
- Input 1..10…Any type of input to be formatted and included in the output string. Blank inputs or inputs for which there is no @@@ field in the format definition are ignored.
-

Primary Outputs:

- Output string…Name.Str…string of formatted output
- Trigger image…Name.Img…copy of last trigger

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Use for general purpose string preparation and to prepare string file names for logging to compact flash disk.

Mismatch Latch

The MismatchLatch module compares two statuses and turns on its output if the two statuses fail to match each other after a designated time delay. In addition, the module will latch the alarm and hold it even if the alarm condition is removed. 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 latch module. The latched alarm output can only be reset by triggering the reset latch status input.

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 when true will reset the latched alarm output

Primary Outputs:

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

Outputs for Internal Use:

- Delay timer…Name.Timer…integer countdown delay timer counting 10^{th}'s of a second

Limitations: None

Expected Applications:
Used to detect pump failure or other control element failure.

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. Time output is seconds since input last went 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…floating point 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
- Time…Name.Time…floating point output number of seconds output has been false

Outputs for Internal Use: None

- Timer…Name.Tmr…integer timer for delay

Limitations: None

Expected Applications:

Use to eliminate intermittent OFF states from status. Also to provide incrementing time that status has been off.

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. Time output is seconds since input last went true. 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…floating point 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
- Time…Name.Time…floating point output number of seconds output has been false

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

Limitations: None

Expected Applications:
Use to eliminate intermittent ON states from status. Also to provide incrementing time that status has been off.

OR Gate

The OR gate will accept up to 16 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 #1-16…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.

OR Gate Latch

The OR gate latch module will accept up to 16 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 #1-16…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.

Software Modules

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 RUG9 PID module implements a standard independent gain proportional/integral/differential (PID) linear control algorithm. It implements the following:

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 in 1/seconds
Kd=derivative gain in seconds
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 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

The derivative term is used to speed the response to load changes. Its use is beyond the scope of this manual.

Inputs That Must be Constants: None

Other Inputs:

- 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 fall. If the action is set to reverse (0), then if the process variable is greater than the setpoint, the output will rise.
- 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.

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.

- Output above HI limit…Name.HI…status output indicating that the calculation attempted to exceed the high limit. Use to control pulse type equipment.
- Output below LO limit…Name.LO…status output indicating that the calculation attempted to extend below the low limit. Use to control pulse type 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 the 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. 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.

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 the lead lag sequencer which will perform backspin timing and lead lag sequencing.

Inputs That Must be Constants: None

Other Inputs:
- 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:
- 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:
- 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…floating point time in seconds that pump call condition must remain true before the pump will be called.
- Off steady delay sec…floating point 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.

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.

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
- Day of week string…Name.Days…string of day of week: SUN…SAT
- 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.

ReadTableRowFloat

The read table row float module is used to read up to 10 cells of a table's floating point row. Tables provide easy access to a single column, but no built in access to a given row. This module enables you to identify the table, which row to read and the starting column. Then, when triggered, it reads up to 10 cells, starting with the one you designate. For example, the table below contains a floating point row named "Setpoint". The **ReadTableRowFloat** module would enable you to read all entries in that row by dragging the name "Setpoint.TBL" into the module's row name entry and setting the beginning column entry to 1. Then, when triggered, the outputs of the module would capture the contents of the setpoint row of the table. You may have as many of these modules as you need, accessing the same table, and even overlapping in cell addresses.

Address	Integer	1	2	3	4	5	6
City	String	Olympia	Tumwater	Lacey	Shelton	Aberdeen	Elma
Setpoint	Float	1.23	4.56	7.89	10.12	11.34	12.56

If the **ReadTableRowfloat** module were named GetTable, then, when triggered, the result would be:

- GetTable.C1=1.23
- GetTable.C2=4.56
- GetTable.C3=7.89
- GetTable.C4=10.12
- GetTable.C5=11.34
- GetTable.C6=12.56
- GetTable.C7=undefined
- GetTable.C8=undefined
- GetTable.C9=undefined
- GetTable.C10=undefined

Inputs That Must be Constants: None

Inputs That Must be Pointers:

- Row name…name of row to read, dragged from the floating point data base. Entry will have the form "Name.TBL", designating it as the output of a table.

Other Inputs:

- Starting column…integer specifying which column is the first to be read. Its value will appear in the output with the ".C1" extension.
- Trigger read…status trigger to cause table to be read. Leaving this entry blank will result in reading of the table with each scan.

Primary Outputs:

- Value at starting column…Name.C1…floating point value captured from first column read.
- Value at start col +1…9…Name.C2 to Name.C10…floating point values captured from columns to right of starting column.

Outputs for Internal Use: None

Limitations: None

Expected Applications:

Use this module to capture cells of a table row in parallel.

ReadTableRowInt

The read table row integer module is used to read up to 10 cells of a table's integer row. Tables provide easy access to a single column, but no built in access to a given row. This module enables you to identify the table, which row to read and the starting column. Then, when triggered, it reads up to 10 cells, starting with the one you designate. For example, the table below contains an integer row named "ComCount". The **ReadTableRowInt** module would enable you to read all entries in that row by dragging the name "ComCount.TBL" into the module's row name entry and setting the beginning column entry to 1. Then, when triggered, the outputs of the module would capture the contents of the ComCount row of the table. You may have as many of these modules as you need, accessing the same table, and even overlapping in cell addresses.

Address	Integer	1	2	3	4	5	6
City	String	Olympia	Tumwater	Lacey	Shelton	Aberdeen	Elma
ComCount	Integer	456	789	34	129	998	12345

If the ReadTableRowInt module were named GetTable, then the result would be:

- GetTable.C1=456
- GetTable.C2=789
- GetTable.C3=34
- GetTable.C4=129
- GetTable.C5=998
- GetTable.C6=12345
- GetTable.C7=undefined
- GetTable.C8=undefined
- GetTable.C9=undefined
- GetTable.C10=undefined

Inputs That Must be Constants: None

Inputs That Must be Pointers:

- Row name…name of row to read, dragged from the integer database. Entry will have the form "Name.TBL", designating it as the output of a table.

Other Inputs:

- Starting column…integer specifying which column is the first to be read. Its value will appear in the output with the ".C1" extension.
- Trigger read…status trigger to cause table to be read. Leaving this entry blank will result in reading of the table with each scan.

Primary Outputs:

- Value at starting column…Name.C1…integer value captured from first column read.
- Value at start col +1…9…Name.C2 to Name.C10…integer values captured from columns to right of starting column.

Outputs for Internal Use: None

Limitations: None

Expected Applications:

Use this module to capture cells of a table row in parallel.

ReadTableRowString

The ReadTableRowString module is used to read up to 10 cells of a table's string row. Tables provide easy access to a single column, but no built in access to a given row. This module enables you to identify the table, which row to read and the starting column. Then, when triggered, it reads up to 10 cells, starting with the one you designate. For example, the table below contains a string row named "City". The **ReadTableRowString** module would enable you to read all entries in that row by dragging the name "City.TBL" into the module's row name entry and setting the beginning column entry to 1. Then, when triggered, the outputs of the module would capture the contents of the City row of the table. You may have as many of these modules as you need, accessing the same table, and even overlapping in cell addresses.

Address	Integer	1	2	3	4	5	6
City	String	Olympia	Tumwater	Lacey	Shelton	Aberdeen	Elma
ComCount	Integer	456	789	34	129	998	12345

If the ReadTableRowInt module were named GetTable, then the result would be:

- GetTable.C1=Olympia
- GetTable.C2=Tumwater
- GetTable.C3=Lacey
- GetTable.C4=Shelton
- GetTable.C5=Aberdeen
- GetTable.C6=Elma
- GetTable.C7=undefined
- GetTable.C8=undefined
- GetTable.C9=undefined
- GetTable.C10=undefined

Inputs That Must be Constants:

- Output number of chars…Integer specifying output string length

Inputs That Must be Pointers:

- Row name…name of row to read, dragged from the integer database. Entry will have the form "Name.TBL", designating it as the output of a table.

Other Inputs:

- Starting column…integer specifying which column is the first to be read. Its value will appear in the output with the ".C1" extension.
- Trigger read…status trigger to cause table to be read. Leaving this entry blank will result in reading of the table with each scan.

Primary Outputs:

- Value at starting column…Name.C1…string captured from first column read.
- Value at start col +1…9…Name.C2 to Name.C10…strings captured from columns to right of starting column.

Outputs for Internal Use: None

Limitations: None

Expected Applications:

Use this module to capture cells of a table row in parallel.

SelectByValue

When enabled, the SelectByValue module selects output based on highest/lowest value (run time). Main enable=0 disables all outputs. Mode chooses whether control is based on lowest or highest value. Deadband prohibits switching output until value demands change by more than deadband. You can use this module to choose pumps based on run time, number of starts or other related characteristic. This module can also be used to generally find the minimum or maximum value from a collection of up to 8 values.

Inputs That Must be Constants: None

Other Inputs:

- Main enable…status that will enable scanning when TRUE and disable scanning and turn off all outputs when FALSE

- Call/Off delay setpoint sec…floating point setpoint that will delay changing the output statuses until a time delay from the last change.
- Mode 0=lowest, 1=highest…status input: 0=choose output whose input value is lowest, 1=choose output shoes input value is highest.
- Value deadband to switch…floating point input by which a value must change before it will be allowed to switch the output selection.
- P1..8 enable…status inputs that enable the individual status outputs
- P1..8 value…floating point values to be tested by this module to determine which output to engage

Primary Outputs:
- Last item selected 1-8…Name.Sel…Integer output with the number in range of 1-8 designating the last selection from the set of 8 inputs
- Last value selected…Name.Val…Value associated with the last output selected
- P1Call...P8Call…Name.P1…P8…Status outputs, one of which will be on to corresponding to the current selection

Outputs for Internal Use:
- Delay timer…Name.Tmr…Timer to time output switching

Limitations: None

Expected Applications:
Use this module to select lead pump from among up to 8 pumps in pump control strategy.

SequenBatch

The SequenBatch module is a sequencer for managing batch and test sequences and is intended to work with a table that defines strings, test phases and timing. Each sequence would consist of up to 8 phases with multiple states per phase. The state advances whenever either a trigger to advance the state is received, or the state times out. At the end of each phase (Next state to jump to=0), the result input is read and latched into one of 8 outputs, and a result string is latched into one of 8 result strings. The 'Trigger full sequence' input will trigger the full sequence starting with phase 1and continuing until a phase does not have a trigger input. As each phase is completed, its result will be latched. If the reset trigger is asserted, all results will be set to zero and result strings will be set to the reset string. If a single phase trigger is asserted, that phase will be run until the end of phase marker, then its result will be posted, and the sequence will stop. Normally the following inputs are taken from the table: Out string at start of phase, Out string if result=0, Out string if result=1, Max time in state sec, Next state to jump to, and Result of phase. Note that the state output should be used to set the column selector of the table, and when a trigger is received to start a phase, the state will be set to a value of 1-8 depending on which phase is to commence. Therefore, the 'Next state to jump to' entry in each of the first 8 columns must designate the beginning column higher in the table that is the actual starting point of each phase.

Inputs That Must be Constants:
- Output string max chars…Integer specifying output string length

Other Inputs:
- Trigger full sequence…status that will start the full sequence beginning with phase 1.
- Reset trigger…status that will terminate the sequence.
- Out string at reset…string that will be presented on string result outputs after reset asserted.
- Out string during phase…result string that will be presented during execution of phase in progress, e.g. ‘TESTING’
- Out string if result=0…result string if result of phase is 0, e.g., ‘FAILED’

- Out string if result=1…result string if result of phase is 1, e.g., 'OK'
- Max time in state sec…floating point time to spend in a single state of a phase
- Trigger advance state…trigger to override time delay and advance the state
- Next state to jump to…integer state to advance or jump to at end of present state
- Result of phase…status input indicating result, if any, of current phase
- Trigger run phase 1…8…status trigger input to immediately jump to a single phase of sequence then return to idle

Primary Outputs:
- State now…Name.State…Integer present state of sequencer (column selector for table)
- Phase now…Name.Phase…Integer (1-8) indicating present sequencer phase
- Result 1-8…Name.Result 1…Result 8…statuses latched at the end of each phase indicating result of that phase of sequence
- Result string 1-8…Name.Str 1…Str 8…String latches at end of each phase indicating result of that phase of sequence

Outputs for Internal Use:
- Temp sequencer…Name.Temp…internal sequencer
- Delay timer…Name.Tmr…Timer to time sequences

Limitations: None

Expected Applications:
This module works best with a table to define test and other batch sequences.

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 0.1 second. Timer range is 214,748,365 seconds. The sequencer state count range is -2,147,483,648 to +2,147,483,647. 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 a 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…floating point 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.

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 +2,147,483,647 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 -2,147,483,648 to +2,147,483,647. 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 a 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.

StringSwitch

The string switch uses an index to select one of up to 17 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 16 that selects a string from the input string list.
- String 0 to 16…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:

- 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

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, the module emits the 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. In this manner, the module will present the string corresponding to the highest priority alarm input bit (mode=0), or it will rotate through all input bits, presenting the output string corresponding to any that are on one after the other (mode=1).

Inputs That Must be Constants:
- Max string bytes…integer identifying space to be set up for this module's output string. This value should be greater than the number of characters in the longest input string.

Other Inputs:
- Trigger new test…status trigger that will cause the module to examine the input status bits and issue a new output string.
- Mode (0,1)…integer that specifies whether: Mode=0: the string corresponding to the first non-zero bit will be sent out, or Mode=1: when trigger new test is 1, the module will cycle to the next input bit that is on and present its string as the output.
- All off string…string that will be presented as the output if none of the input bits is on. E.g., 'Alarms off'.
- String 1 to 10…string inputs from which one that will be presented as the module output if its corresponding bit input is on.

Primary Outputs:
- Output string…Name.String…string output selected from the string input list by the latest test.
- Have true bit status…Name.HaveBit…status that indicates that at least one input bit is on

Outputs for Internal Use:
- Cycle counter…Name.Cnt…integer counter that keeps track of next input bit to test

Limitations: None

Expected Applications:
Use to present multiple alarms in one display line either based upon input priority or by equal priority cycling through all inputs.

SyncManyValues

When triggered, SyncManyValues scans 10 input values and compares them with associated output values. If any value has changed, the module sets the associated 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:

- Trigger scan…status trigger that will cause the module to compare the inputs with the outputs. Any input found to not match the corresponding output will cause the module to install the new input into both the corresponding output and into the other input (A or B).
- Input 1..10A, 1..10B…Floating point inputs that are compared with corresponding Value1..10 outputs.

Primary Outputs:

- Value 1..10…Name.1..10…floating point outputs that hold the most recent value present on the corresponding inputs.
- Change A trigger…Name.Atrig…status trigger indicating that a new value has been installed from one of the input1..10A inputs
- Change B trigger…Name.Btrig…status trigger indicating that a new value has been installed from one of the input1..10B inputs

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, input A entries could come from local setpoints; input B entries could come from the telemetry receive array. Whichever was entered last would be captured in the output values.

SyncToRTC

This module issues uniformly spaced triggers synchronized to the realtime clock. When enabled, it issues triggers on the interval specified, synchronized to realtime clock (RTC). To use it, 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 the zero time 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. If you specify an interval that is not evenly divisible into the period, the triggers will restart at the next zero event. For example, if you specify a trigger every 19 seconds, triggers will be issued at 0, 19, 38, 57 seconds of the minute, and then again at the zero second of the next minute.

Inputs That Must be Constants: None

Other Inputs:

- Enable…status input, =1 or blank enables generation of triggers; =0, disables triggers.
- Seconds interval (1-30)…integer number of seconds between triggers.
- Minutes interval (1-30)…integer number of minutes between triggers.
- Hours interval (1-12)…integer number of hours between triggers.

Primary Outputs:
- Trigger output…Name.Trg…status trigger at each specified time tick.

Outputs for Internal Use:
- Temp time…Name.Temp…Temporary copy of RTC integer.

Limitations: None

Expected Applications:
Use to trigger sampling or other events on definite RTC time ticks.

Toggle

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 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 0.1 second resolution and range of 214,748,365 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.

Trigger Every X Minute

This module issues a trigger event at specified intervals synchronized with the real time clock minutes register. The module will count minutes until the designated number of minutes has passed. Then it will issue a trigger and reset the minutes 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 minutes counter. If no preset trigger is received, the module will issue triggers indefinitely on the interval specified. This module is typically used to trigger events every X minutes; for example, to force a poll every 5 minutes.

Inputs That Must be Constants: None

Other Inputs:
- X interval minutes…integer number of minutes between successive trigger outputs.
- Preset status input…status input to preset the minutes counter to X.
- Enable status input…status input that if missing or true will enable counting; if zero, will inhibit counting.

Primary Outputs:
- RTC minutes image…Name.Min…integer copy of RTC minutes register.
- Trigger output…Name.Trg…status trigger output that will be issued every X minutes.

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

Limitations: None

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

Trigger Every X Second

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 trigger events 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.
- Preset status input…status input to preset the seconds counter to X.
- Enable status input…status input that if missing or true will enable counting; if zero, will inhibit counting.

Primary Outputs:
- RTC seconds image…Name.Sec…integer copy of RTC seconds register.
- 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 edge 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, rather than repetitively as long as the status is on.

Inputs That Must be Constants: None

Other Inputs:
- Status input…status whose rising 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.

Trigger on Boot

The Trigger on Boot module issues a trigger event each time the RUG9 boots up; i.e., each time the program starts. This trigger event is used to preset sequencers and latches on boot up so they don't start with unknown states. This module is a duplicate of the System.Boot output of the system setup module already included in each project.

Inputs That Must be Constants: None

Other Inputs: None

Primary Outputs:
- Trigger event output…Name.Trigger…status trigger issued for the first scan following start up of the program.

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Used to initialize latches, counters, sequencers…

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:
- 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.

TriggerOnKey

The trigger on key 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 keystroke will be accepted on any display on that port. The port number must be entered as an integer in the range of 0 through 26. Port 0 is the LCD display, port 1 is the CPU's serial port, ports 3,4, and 5 correspond to ports 1,2 and 3 on board 1; ports 6,7 and 8 correspond to ports 1,2 and 3 on board 2, etc. There are 3 ports per board. Port 0, the LCD display is most commonly used by this module.

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 Trigger On Key module to issue a trigger when display 3 is presented on port 0, and key "2" is designated; and you use the output of that Trigger On Key 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.

Inputs That Must be Constants: None

Other Inputs:

- 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.
- Keystroke, 0-9…integer numeral indicating which numeric key is to cause the trigger to be issued.

Primary Outputs:

- Trigger event output…Name.Trigger…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.

TriggerOnKeyLog

The trigger on key log 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, and while an operator is logged on. This module is identical to the **TriggerOnKey** module above, except that an operator must be logged on for the keystroke to cause any action. If no operator is logged on, the keystroke will be ignored. If the keystroke is recognized, the programmer designated action will be taken and the logon timer will be restarted.

Inputs That Must be Constants: None

Other Inputs:

- 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.

- Keystroke, 0-9…integer numeral indicating which numeric key is to cause the trigger to be issued.

Primary Outputs:
- Trigger event output…Name.Trigger…status trigger issued immediately after the user pressed the designated key on the designated port.

Outputs for Internal Use: None

Limitations:
Operator must be logged on for keystroke to be recognized.

Expected Applications:
Module enables authorized operator to cause changes in control actions, acknowledge alarms, etc.

TriggerOnRTC

This 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.

TrigOnBitThenClr

This 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:
Used 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 10 inputs, provides an individual change trigger on the first 9 inputs, and provides a composite trigger that indicates if a change has occurred on any of the 10 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 1-10…floating point values that are being monitored for a change.
- Deadband 1-10…floating point amounts by which the input values must change before a change trigger is issued.

Primary Outputs:
- Value 1-9 changed…Name.Trg1-9…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 10 inputs.

Outputs for Internal Use:
- Value 1-10 copy…Name.Val1-10…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.

TrigOnKeyMany

The trigger on key many 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 26. Port 0 is the LCD display, port 1 is the CPU's serial port, ports 3,4, and 5 correspond to ports 1,2 and 3 on board 1; ports 6,7 and 8 correspond to ports 1,2 and 3 on board 2, etc. There are 3 ports per board. Port 0, the LCD display is most commonly used by this module.

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 key "2" is designated; and you use the output of that 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.

Inputs That Must be Constants: None

Other Inputs:
- 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.

TrigOnKeyManyLog

The TrigOnKeyManyLog is identical to the TrigOnKeyMany module above except that output triggers will only be issued if the operator is logged on. The 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 26. Port 0 is the LCD display, port 1 is the CPU's serial port, ports 3,4, and 5 correspond to ports 1,2 and 3 on board 1; ports 6,7 and 8 correspond to ports 1,2 and 3 on board 2, etc. There are 3 ports per board. Port 0, the LCD display is most commonly used by this module.

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 key "2" is designated; and you use the output of that 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.

Inputs That Must be Constants: None

Other Inputs:

- 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.

ValueEqual

The **ValueEqual** module compares a value with a setpoint, and, if equal for a user designated time, turns on a status output. If the value becomes different from the setpoint, the status output will turn off without delay. Notice that this is a floating point comparison, so the two values must be **exactly** equal for a true result. Time delay zero or blank disables the delay. See the **ValueTest** module below.

Inputs That Must be Constants: None

Other Inputs:

- Input…floating point input value to be compared against the setpoint.
- Setpoint to compare…floating point setpoint that the input must equal to cause the status output to turn on.
- Delay sec…integer time the input must equal the setpoint before the status will be turned on.

Primary Outputs:
- Status output…Name.Equal…status that will become true after the input has been equal to the setpoint for the designated time delay. Output will stay true until the comparison result is false.

Outputs for Internal Use:
- Delay timer…Name.Timer…integer timer to time the comparison

Limitations: None

Expected Applications:
Used to detect when a controlled variable exactly matches a setpoint.

ValueTest

The value test 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.

ValueTestValueOut

The ValueTestValueOut module compares two test inputs A and B, then outputs one of six input choices based on result. Inputs can be integers, floating point values or data base variables. Updates output whenever enable is either true or unspecified.

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.
- Output value if A>B…Value to be sent to output if test satisfied.
- Output value if A>=B… Value to be sent to output if test satisfied.
- Output value if A=B… Value to be sent to output if test satisfied.
- Output value if A<>B… Value to be sent to output if test satisfied.
- Output value if A<=B… Value to be sent to output if test satisfied..
- Output value if A<B…. Value to be sent to output if test satisfied.

Primary Outputs:
- Output value... Name.Val…Floating point value selected based on test.

Outputs for Internal Use: None

Limitations: None

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

WriteTableRow

This module enables you to preset any number of cells on a table's row to a user specified value. You may use it to initialize any integer, floating point, or status table row to a value. If used to initialize a string row, it will clear the cells. In order to establish the proper connection to the table row, you must drag the name of the row from the appropriate database into the row name input cell. This name will always have the ".TBL" extension, indicating that it is the output of a table. You must also specify the starting column number, number of cells to be written, and value to be written. The cells will be written when the module is triggered. If you specify more cells than exist on the row, the module will preset all cells up to and including the last one.

Inputs That Must be Constants: None

Inputs That Must be Pointers:
- Row name…name of row containing cells to be written. Drag this entry from the appropriate database. Name will have the ".TBL" extension.

Other Inputs:
- Trigger to write cells…status trigger input that will cause cells to be written when true.
- First cell to write…integer column number of first cell to write on row.
- Number of cells to write…integer number of cells to write.
- Value to write…floating point, integer or status value to write in all cells. If a string value is referenced, the cells will simply be cleared.

Primary Outputs: None

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Usually used to clear RAM cells in a table on first use.

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:
- 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

Expected Applications:
Used to keep and latch average flow, pressure, etc.

DataLogger

The **DataLogger** module is a complex and powerful data logging tool. Its job is to sample an input at a user defined time interval, and save the sampled values and periodic time tags to memory. Once saved to memory, the data values can be used to show trends on the LCD, can be dumped to a serial port or flash cartridge, and can be used by a few other statistical modules. You specify the number of samples to be saved, the signal that is to be sampled, the sample interval, and various other required parameters. Samples and time tags are saved in a circular memory area. When the allocated memory fills up, the new samples will overwrite the oldest samples. Each sample is saved in floating point form, so it can be used for adjustable-scale trends and other purposes without loss of resolution. Each sample or time tag occupies 4 bytes of RAM. The RAM is battery backed up, so loss of power will not cause loss of logged data.

Specifying Sample Interval

You have two choices for specifying the logger's sample interval: 1) you can use the sampling software internal to the module, or 2) you can use the module's external trigger to accept a sample and/or time tag triggers that you generate using other modules.

To use the first method, you must set the "Sample tick choice" and "Number of ticks between samples". The sample tick choice basically enables you to choose between a number of convenient time bases such as seconds, minutes, hours, etc. Once you have chosen one of the tick time bases, then you simply specify how many of those ticks are to occur before a sample is taken. For example, if you wish to take a sample every 7 seconds, you would choose a sample tick choice of zero (seconds) and then set the number of ticks between samples at seven. Note that this method has precedence over the second method below.

To use the second method, i.e., externally setting the sample interval, leave the sample tick choice and number of ticks between samples entries blank. At the bottom of the input list, drag in from the status database a trigger generated on the time basis you require. For example, you could use a **TriggerEveryXSeconds** module, with its timeout set to 7 seconds to generate a sample every 7 seconds as in the example above.

Specifying Time Tags

The logger will imbed time tags in the data if you wish. If you are using sample tick choice method number 1 above, then you should set the "Trend time tick choice" to at least one higher number than the sample tick choice you made for data sampling. For example, if you have chosen a sample tick choice of zero (seconds) as in the example above, then you should choose a trend time tick of 1 (minutes) so you will have a time tick once per minute.

If you are externally controlling sampling as in method two above, then you must supply any time tag triggering externally as well. You can choose any time tag frequency you wish, including having a time tag per sample by installing the same external trigger for the time tag as for the data sample.

Setting Up Trending Inputs

One of the main uses for this module is to log data to be trended on the LCD. Some of the data logger's input parameters are used to define the trend features such as dot pattern, scales, etc., for when the logged data will be presented on the LCD. The trend high and low scales establish the scale factors on the LCD so the display software knows where each data base dot is to be placed in relation to the top and bottom of the LCD. You should specify high and low scales that are outside the maximum and minimum excursions of the data. That is not a limitation, however. You are free to specify any range you wish as long as the high scale is above the low scale. If data falls outside the specified scale, the trend will simply flat line at the scale. In the case of multiple trends simultaneously being displayed, you will want to use a different dot pattern for each. The dot pattern is entered as part of the logger setup in the form of a single decimal number that, when converted to binary, represents the bits of a single byte. Here are some useful examples:

PATTERN	DECIMAL	BINARY
Solid line	255	11111111
Fine dots	85	01010101
Long dashes	127	01111111
Short dashes	119	01110111
Uneven dashes	123	01111011

The trend horizontal tick EU entry sets the location of the first horizontal grid line up from the bottom of the trend. The display software then fills in the rest of the horizontal grid lines evenly spaced up from that line based on its distance from the bottom of the trend. For example, if your trend vertical scale is from zero to 100% and you wish to have horizontal grid lines at 20%, 40%, 60% and 80% in addition to the bottom and top of the chart, then you only need to set the first horizontal tick EU input to 20. Display software will calculate the rest. A fine dot pattern is used to present the horizontal grid lines. Vertical time

ticks are also displayed using a fine dot pattern and are plotted whenever a time tag is encountered in the data. The method of insertion of time tags in the data is discussed above. Finally, the trend time compression entry is used to specify degree of decimation of data samples to implement horizontal time compression. Normally, if you leave this entry blank or set it to 1, no time compression will occur. If you set it to 4, for example, then only every fourth sample will be trended, resulting in a 4 to 1 time compression of the trend.

The LCD will show at most 320 samples at one time. If you specify more than 320 words in the logger's database, then when you display the trend on the LCD, you can scroll forward and back in time using the up and down arrow keys.

Inputs That Must be Constants:

- Number of words in database…integer sets number of floating point RAM locations to be set aside for use by module. This includes time tick entries so include extra words for your anticipated data plus time ticks.

Other Inputs:

- Input to be logged…floating point data base point whose value is to be logged at the designated time interval.
- Enable sampling…status input that enables sample storage when true, disables when false.
- Sample tick choice…integer sets choice of time base type for sample storage…0=seconds, 1=minutes, 2=hours, 3=days, 4=months
- Number of time ticks between samples…integer to set sample interval. Example: to sample every 5 minutes, set sample tick choice above to 1=minutes, and set this entry to 5.
- Trend high scale…floating point value that sets engineering units of trend plot top of scale.
- Trend low scale…floating point value that sets engineering units of trend bottom horizontal axis.
- Trend dot pattern…integer in range of 0 to 255. Some examples: 255=solid line, 127=dashed line, 63=dashed line with dashes half as long as 127, 55=dots, zero blanks the trend.
- Trend 1st horizontal tick EU…floating point value that sets the engineering units level of the first horizontal dotted line. Others up to 10 will be at an even interval from the first. For example, if scale is 0 to 100, setting this entry to 25 will give a horizontal dotted line at 25, 50, 75, and 100.
- Trend time tick choice…integer that sets choice of time tick for vertical dotted lines so trend can be related to the real time clock. For example, setting this entry to 2 will cause the trend to have a vertical dotted line every hour of the trend.
- Trend time compression…integer that sets trend decimation. Example: 1=show every point, 2=show every other point, 10=show every 10th point.
- Reset input…status trigger input: true erases the data base and resets the indices to the beginning of storage.
- Trigger sample…status trigger input used in lieu of the sample tick choices above to externally trigger data sampling.
- Trigger time tag…status trigger input used in lieu of trend time tick choice above to externally trigger storage of a time tick in the data.
- ID number…integer identifier that must be unique to system for transfer of logged data over telemetry channel.

Primary Outputs:

- Logger database…Name.Log…integer pointer to beginning of log. Other modules that use the logged data use this name to find the data.

Outputs for Internal Use:

- Sample tick counter…Name.Cnt…integer counter for internal sampling time base.
- Last sample time tag…Name.Last…integer packed time tag
- First index…Name.First…integer of index to first sample in the log
- Next index…Name.Next…integer of index to next sample location to be stored. If same as first index, then log is empty. If one less than first index, then log is full.

- Last sample tick flag…Name.Sflag…integer copy of time of last sample
- Last trend tick flag…Name.Tflag…integer copy of time of last time tick

Limitations:
- Trend high scale value must be higher than trend low scale value.

Expected Applications:
Use for general data logging. Data can be trended, dumped to port, dumped to log and used by other statistics modules.

GetFromLogger

The GetFromLogger module reads referenced data logger module and extracts 30 samples beginning with designated start sample. Embedded time tags are skipped. Logger reference must be .Log output of DataLogger module. Sample index=1 is newest.

Inputs That Must be Constants: None

Inputs That Must be Pointers:
- Logger where data saved…integer pointer to data logger's data. It is the logger's Name.Log output.

Other Inputs:
- Start sample index…integer specifying with which sample in the database the calculation is to begin. An index of one specifies the latest sample.
- Trigger extraction…status input to cause next calculation to be performed.

Primary Outputs:
- Trigger have new outputs…Name.Avg…floating point average value of samples specified.
- Done…Name.DunTrg…status trigger output indicating that the calculation is complete and the new sliding average has been transferred to the module's output.

Outputs for Internal Use:
- Temp average…Name.Trg…Trigger that new values are output.
- Sample 1-30…Name.L01-L30…Floating point values extracted from logger.

Limitations: None

Expected Applications:
Use where you need to extract samples from the logger back in time.

LogMany

The LogMany module is used to log multiple items at the same time. With each sample it stores a time tag and up to 20 values. The time tag and each floating point or integer sample consume 4 bytes of storage each. Status inputs to be logged are packed all into a single 4 byte field. When the log is full, new samples will begin overwriting the oldest samples.

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

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 will erase log
- Input sample #1-20…floating point inputs to be logged. Logging of a record stops if blank input encountered.
- ID number…integer identifier that must be unique to system for transfer of logged data over telemetry channel.

Primary Outputs:
- Logger database…Name.Log…log where data and time tags stored, integer database
- 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:
- 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:
Used 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:
- 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:
Used to capture minimum reservoir level, flow rate, temperature, etc.

SlidingAvg

This module is used to calculate the sliding average of data taken by a data logger module. To use it, you must specify the beginning and ending sample numbers (indices) in the log, and you must reference the particular data log to use. The beginning and ending sample numbers are relative to the most recent sample that has been logged. When triggered, the module will scan the logged values and perform the calculation. If you set the trigger status input to 1, the module will perform its calculations on each program scan. When the module has scanned all the specified indices, it will transfer the result to its output and issue a trigger. An input is provided to control the maximum number of indices to be sampled on any one scan in order that calculations involving large numbers of samples do not take so long that the watchdog timer times out.

An example of an instance where you would use this module is the following. Suppose you have setup the Rug9 to keep a database of stream flow samples every minute for the last 24 hours (1440 samples), and you wish to know the flow rate for the last hour, last 3 hours, and last 6 hours to predict flooding potential. You would use three of the sliding average modules to calculate the three averages. In each case, the beginning index would be zero, representing the most recent data sample taken. For the one hour sliding average, the ending index would be 60 (60 samples in one hour). For the 3 hour sliding average, the ending index would be 180. For the 6 hour sliding average, the ending index would be 360. Since the samples are taken once per minute, you could trigger this calculation once per minute and specify a maximum of 20 indices to be calculated each scan. Notice that the **AvgValue** module above will not work for this type of application.

Inputs That Must be Constants: None

Inputs That Must be Pointers:
- Logger where data saved…integer pointer to data logger's data. It is the logger's Name.Log output.

Other Inputs:
- Trigger calculation…status input to cause next calculation to be performed. Usually set to the same time interval as the logger's data is being sampled.
- Start sample in log…integer specifying with which sample in the database the calculation is to begin. An index of zero specifies the latest sample.
- End sample in log…integer specifying the last database sample to be included in the calculation.
- Max samples per scan…integer specifying how many samples are to be included in the calculation on any one scan of the program. Usually set in the range of 10 to 50.

Primary Outputs:
- Final average…Name.Avg…floating point average value of samples specified.
- Done…Name.DunTrg…status trigger output indicating that the calculation is complete and the new sliding average has been transferred to the module's output.

Outputs for Internal Use:
- Temp average…Name.TAvg…Floating point intermediate average result.
- Temp count…Name.Tcnt…integer count of values included in calculation so far.
- Temp index…Name.Tidx…integer index of next index to be included in calculation.

Limitations:
- Specifying too many samples per scan could cause the watchdog timer to time out.

Expected Applications:
Use where an average over a specific time span is required.

SlidingRate

The sliding rate module is used to calculate a sliding rate, and produce high and low alarms of data logged in a data logger module. Basically, this module takes the difference between a data logger's most recent sample and one earlier in time as specified by you. It outputs that result and compares it against rising rate and falling rate alarm setpoints and issues the resulting alarms.

An example of where this module applies is in the case of a reservoir whose excessive rising rate would indicate a potential flood incident, and whose excessive falling rate would indicate a possible burst dam. An example setup would be the following: If your application requires you to detect a rising or falling reservoir whose rising rate exceeds 1 foot in 20 minutes or whose falling rate exceeds 2 feet in 20 minutes, you will need your data logger to sample once per minute. This sliding rate module would then need its number of samples in past input set to 20 (20 minutes), its rising rate alarm setpoint set to 1.0, and its falling rate setpoint set to –2.0.

Inputs That Must be Constants: None

Inputs That Must be Pointers:
- Logger where data saved…integer pointer to data logger's data. It is the logger's Name.Log output.

Other Inputs:
- # samples in past…integer designating how many samples down in data log the where the old sample exists that is to be compared with the latest sample.
- Enable calculation…status input: zero disables calculation; blank or 1 enables calculation.
- Rising rate alarm SP…floating point rising rate alarm setpoint. Calculation result exceeding this value will turn on alarm output without delay.
- Falling rate alarm SP…floating point falling rate alarm setpoint. This should be a negative number. Calculation result falling below this value will turn on falling rate alarm output without delay.

Primary Outputs:
- Change…Name.Change…floating point rate of change output.
- Old sample value…Name.Oldsamp…floating point value of old sample from data base.
- High rising rate alarm…Name.RiseAlrm…status true if calculation indicates rate of rise exceeds setpoint
- High falling rate alarm…Name.FallAlrm…status true if calculation indicates rate of rise is below setpoint.

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Use to generate rate of change values and associated alarms over specific time span.

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 each 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…not used.
- Sample counter…Name.cnt…integer count of samples taken this minute.
- 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 RTC 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

AlertRcvAnalog

The AlertRcvAnalog module is used to decode ALERT format messages and provide outputs in engineering units. It decodes binary format ALERT messages with specified ID #. Types: 1=analog, 2=precip, 3=up/down, 4=wind. For wind: direction=raw value*10; speed=multiplier divided by time in hrs between counts.

Inputs That Must be Constants:
- Card # (1-8)…integer designating which card slot has the port this module references.

- Port # (1-3)…integer designating which port on the board this module references.

Other Inputs:
- ID# (1-8191)…integer designating ID# for which this module is watching.
- Type (1-4)…integer designating type of message of message this module to convert: 1=analog, 2=precipitation, 3=up/down counter, 4=wind speed.
- Multiplier…floating point value used to convert received raw value to engineering units result
- Offset…floating point value used to offset received raw value to produce engineering units result
- Max EU value…floating point value that is maximum module is to allow output
- Min EU value…floating point value that is minimum module is to allow output
- Count change deadband…value by which new count is to change before being used to calculate new result
- Preset value trigger…status trigger input to preset output value
- Preset value…floating point value to use as preset value
- Trigger to recalculate…status trigger input to cause recalculation

Primary Outputs:
- Trigger have this ID…Name.Trg…status output that, when true, indicates a new value has been processed.
- Last ID received…Name.LastID…integer value of last ID accepted
- Value or wind direction…Name.Val…floating point result of EU calculation or wind direction for wind receptions.
- Wind speed…Name.WSpeed…floating point value of wind speed
- Last raw value received…Name.Raw…last value received before processing

Outputs for Internal Use:
- Reception timer…Name.Tmr…integer timer used to calculate wind speed
- Base accumulator…Name.Base…floating point base for counting values

Limitations: None

Expected Applications:
Used in ALERT base station applications to decode ALERT messages and convert to EU values.

AlertRcvStatus

The AlertRcvStatus module is used to decode ALERT format messages and provide status outputs. It decodes binary format ALERT messages with specified ID # containing up to 8 bits of status information. Validity count specifies over how many receptions in succession a bit must remain unchanged before the bit will be transferred to the output. Auto clear time sets the time over which no receptions have been made at which the status outputs will be set to zero.-

Inputs That Must be Constants:
- Card # (1-8)…integer designating which card slot has the port this module references.
- Port # (1-3)…integer designating which port on the board this module references.

Other Inputs:
- ID# (1-8191)…integer designating ID# for which this module is watching.
- Validity count…integer designating how many successive receptions must have the same statuses before status value change will be accepted.

- Auto clear time sec…floating point value used to timeout failure to receive and then clear output statuses.

Primary Outputs:
- Trigger have this ID…Name.Trg…status output that, when true, indicates a new value has been processed.
- Last ID received…Name.LastID…integer value of last ID accepted
- Status 1-8…Name.S1-S8…status outputs received
- Last raw value received…Name.Raw…last value received before processing

Outputs for Internal Use:
- Reception timer…Name.Tmr…integer timer used to calculate wind speed
- Image…Name.Img…image of oldest statuses received
- Image2…Name.Img2…image of newest statuses received

Limitations: None

Expected Applications:
Used in ALERT base station applications to decode ALERT status messages.

ComFailProcessor

The ComFailProcessor module accepts poll triggers and receive triggers which uses to keep track of running communications performance of the last 30 polls/receptions for each of up to 10 stations. Each second, it increments the 'Addr X last com seconds' output for each channel. Upon a poll, the history list will be shifted left and a zero inserted in the LSB. Upon a reception, the LSB will be set to a 1 and the corresponding 'Addr X last com sec' set to zero, restarting the time counter for that channel. At each poll trigger and rcv trigger, the success rate for the corresponding channel will be recalculated. The success rate ranges from 0 to 100%. Fail mode=0: fail bit is set when number of polls without response exceeds fail threshold input. Fail mode=1: fail bit is set when number of seconds since last com exceeds fail threshold input. Fail mode=2: fail bit is set when success rate (0-100) falls below fail threshold input. Com fails are packed into Name.Comfails. Route that output to NumericToBits module to obtain an individual com fail bit for each channel. Address offset is subtracted from address inputs before processing. Address offset=0 gives address range of 1-10; address offset=10 gives address range of 11-20, etc. After subtracting address offset, if poll address or rcv addrss falls outside range of 1-10, the corresponding trigger is ignored.

Inputs That Must be Constants: None

Other Inputs:
- Poll triggers…status input that indicates that a poll has occurred.
- Poll address…integer address that was most recently polled, usually from the SequenPoll module or counter.
- Rcv triggers…status input that indicates that a reception has occurred.
- Rcv address…integer address of the latest reception, usually obtained from the TrigOnRcv module.
- Fail mode…integer that specifies failure mode choice: Fail mode=0: fail bit is set when number of polls without response exceeds fail threshold input. Fail mode=1: fail bit is set when number of seconds since last com exceeds fail threshold input. Fail mode=2: fail bit is set when success rate (0-100) falls below fail threshold input.
- Fail threshold…integer specifying threshold for declaration of com failure.
- Address offset…integer that is subtracted from poll address and rcv address to determine which output of this module would be effected by latest poll trigger of rcv trigger.
- Clear all trigger…trigger status that, when true, will set all outputs to zero.

Primary Outputs:

- Addr N last com sec…Name.Addr1-10Sec…Seconds since last successful communications for this address (1-10).
- Addr N success rate…Name.Addr1-10Rate…communications success rate in range of 0-100% for this address (1-10).
- Packed com fails…Name.PackedFails…integer with one bit for each of the 10 addressed being monitored. Route this output to NumericToBits module to obtain a single com fail status for each of the 10 addresses.

Outputs for Internal Use:

- Addr N history…Name.Addr1..10hist…accumulation of communications success/fail bits of last 30 polls.

Limitations: None

Expected Applications:

Use this module in master units to track communications success rates for banks of 10 remote sites and to generate com fails for the sites.

ComSetup

The communications setup module is used to establish the protocol, baud rate, tone use, etc. of each serial communication port in the RUG9. 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 the RS232 port on the CPU board which defaults to 9600,N,8,1, ASCII on boot up. Board number and port number inputs must be constants. Note that there is no cage number input since communication boards can only reside in the base card cage. 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 SysSetup1 module's Name.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. Also note that the MDM/RS232 board has only one port; and that port can be designated to use either RS232 or its onboard modem. The dual serial port/printer board has three ports total: two serial ports and one printer port. This module can be used with the two serial ports on that board, but not for the printer port. For the printer port, use the **PrnSetupWatch** module.

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 RUG9 to RUG9 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 RUG9. This protocol enables a RUG9 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 RUG9 maintains the transmit and receive arrays separate; i.e.,

commands to read from the RUG9 will obtain data from the Rug9's transmit array, and commands to send data to the RUG9 will send data into the RUG9'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 RUG9 holding registers will obtain their values from the RUG9's receive array rather than the RUG9'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 RUG9.
- 7…ASC user. In this case, the RUG9's operating system will not act upon characters received, but will instead leave them in the input buffer for parsing by other modules. Menuing functions for this port are disabled. This mode should be used for communications with instruments that use ASCII to send and receive data.
- 8…ALERT This protocol make the RUG5/9 compatible with transmitters and receivers using the ALERT protocol, a transmit-only protocol used for reporting weather and flood warning data.
- 9…TW This is the True Wireless protocol used by certain industrial applications
- 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:

- Card # (1-8)…integer designating which card slot has the port this module references.
- Port # (1-3)…integer designating which port on the board this module references.
- Port buffer size bytes…integer that defines buffer size allocated to this port

Other Inputs:

- 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 operation for all baud rates 300 or below; and for PSK operation for baud rates above 300 baud. Recommended: 300 or 1200 baud for modem use, 9600 baud for RS232 use.
- Wd length (5-8)…integer specifying word length. Recommended: 8 bits.
- Parity…integer specifying parity choice: 0=none, 1=odd, 2=even, 3=mark, 4=space. Recommended: 0=none.
- Stop bits (1,2)…integer specifying number of stop bits per word. Recommended: 1
- Tone use…integer designating which tone set the modem is to use: 1=originate, 2=answer, 3=low tones. Use answer when the unit is to answer a commercial modem. Use originate when the unit is calling a commercial modem. Use low tones in systems of RUGID units that must communicate with each other (in addition to communicating with a master RUGID unit), especially in audio radio systems.
- Address (1-255)…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.
- Mode (1-7)…integer to select communication protocol. See above.
- TX delay tenths of sec…integer to set delay between the time the RUG9 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 2; for radio applications, set it to 7.
- 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.
- UART (0,2,4)…integer to establish connection between the UART and other hardware on the modem board. Use this to tell the system whether the UART is connected to the RS232 port (0), the 2 wire

channel (2), or the 4 wire channel (4). Use mode 0 for direct RS232 connections to PC's or to radios with built in modems such as spread spectrum radios. Use mode 2 to connect to the dial up telephone system, or to a customer owned or leased system wherein Rug9's must communicate among themselves. Use mode 4 to connect to audio radios or to 4-wire leased line phone systems.

- Rings to answer…integer sets the number of rings that must be counted on the 2-wire channel before the unit answers the line (goes off hook). The value depends on your application.
- Com flags…integer to set hardware handshaking in form of packed bits: bit 0 (LSB)=use RTS/CTS handshake, bit 1=suppress trailing nulls, bit 2=suppress reply if #ReplyRegs=0. Usually leave blank. Possible states:
  - 0=no action,
  - 1=use RTS/CTS,
  - 2=suppress trailing nulls,
  - 3= use RTS/CTS and suppress trailing nulls,
  - 4=suppress reply if #ReplyRegs=0,
  - 5=use RTS/CTS handshake and suppress reply if #ReplyRegs=0,
  - 6=suppress trailing nulls and suppress reply if #ReplyRegs=0,
  - 7= use RTS/CTS handshake, suppress trailing nulls, and suppress reply if #ReplyRegs=0

Primary Outputs:

- Audio carrier present…Name.Audio…status output will be true if audio carrier detected by modem.

Outputs for Internal Use: None

Limitations: None

Expected Applications:

ComSetup must be used for each serial port in the system except for the RS232 port on the CPU board.

ComWatch

The ComWatch module is for debugging communications problems. It presents internal communications information. It issues an output trigger if all specified addresses match the received message. Unspecified addresses are wildcards. When enabled, the module will read receive and transmit buffers for the specified port and make the buffer contents available as output strings. Buffer strings are hex/ASCII versions of buffer contents. The strings can be sent to the unit's local display or out a serial port for examination to determine if messages are formatted properly.

Inputs That Must be Constants:

- Output string max chars…integer setting the size of each output string.

Other Inputs:

- Enable…input status that enables module operation and conversion of buffer strings.
- Board number…integer that specifies board number in card cage (1-8)
- Port number…integer that specifies port on board (1-3)
- Source address…integer message source address (blank=wildcard)
- Destination address…integer message destination address (blank=wildcard)
- Forward #1 address…integer message forwarding address (blank=wildcard)
- Forward #2 address…integer message forwarding address (blank=wildcard)
- Forward #3 address…integer message forwarding address (blank=wildcard)
- Trigger clear rcv buffer…status trigger to fill receive buffer with zeroes
- Trigger clear tx buffer…status trigger to fill transmit buffer with zeroes
- Trig xfer rcv to shadow…status trigger to copy the receive buffer to a temporary buffer called the shadow buffer

Primary Outputs:

- Trigger event output…Name.Trg…status trigger output indicating that a message was received matching the address specifications.
- Rcved source address…Name.Src…integer address of source of received message
- Rcved destination addr…Name.Src…integer address of destination of received message
- Fwd#1 address…Name.Fwd1…integer address of first forward address in received message
- Fwd#2 address…Name.Fwd2…integer address of second forward address in received message
- Fwd#3 address…Name.Fwd3…integer address of third forward address in received message
- Rcv buffer string 1-5…Name.RxStr1-5…strings with hex ASCII contents of receive buffer
- Trigger CRC pass…Name.CRCPass…trigger that a received message passed the CRC test
- Trigger CRC fail…Name.CRCFail…trigger that a received message failed the CRC test
- Msg length byte received…Name.LenByte…integer copy of the message's length byte
- Bytes received…Name.BytesRcvd…integer indicating the number of unprocessed bytes in the receive buffer
- Trigger buffer reset…Name.Reset…trigger indicating that the buffer indices were set to zero
- Tx buffer string 1-5…Name.TxStr1-5…strings with hex ASCII contents of transmit buffer
- Rx buffer string 1-5…Name.RxStr1-5…strings with hex ASCII contents of receive buffer
- Shadow buffer string 1-5…Name.ShadowStr1-5…strings with hex ASCII contents of shadow buffer

Outputs for Internal Use: None

Limitations: None

Expected Applications:
This module is used to reveal if receptions are occurring and to show format of transmitted and received messages. The shadow buffer can be used to capture the receive buffer for later comparison. Whenever a new string is generated (the module's enable is set to 1), the shadow buffer string will indicate any differences between it and the receive buffer by placing a minus ('-') symbol ahead of any bytes that differ from the corresponding byte in the receive buffer. To use the CRC pass/fail output triggers, route them to a pair of counters and observe the counts. If a message is received and neither trigger is generated it means that the message either was not received at all or was too short. The Trigger buffer reset trigger is issued whenever the buffer indices are set to zero. This happens any time CRC is calculated, any time ComSetup is installed, or 0.3 seconds after the last character is received.

DecodeBinaryMessage

When an incoming message meets criteria for completion (enough bytes and type 6 bytes match), the DecodeBinaryMessage module decodes the message and presents message data values on floating point outputs. Type entries specify length and type of each field of incoming message: Type 1=1 byte integer; 2=2 byte integer; 3=3 byte integer; 4=4 byte integer; 5=4 byte IEEE float; 6=byte that must match corresponding input value (address, sync, start, or stop byte). Each field whose type is less than 6 is a variable type which will be read from the incoming message and saved as the next variable field value. Type entries and associated inputs can be fixed values or entries obtained from the databases. First type entry that is blank or zero terminates the message definition. Flags set certain characteristics: bit 0=blank or zero sets MS...LS byte order; bit 0=1 sets LS...MS byte order for integers.

Inputs That Must be Constants:

- Board #…integer indicating location in card cage of board to be used board (0-8).
- Port #…integer indicating port to use (1-3).

Other Inputs:

- Trigger clear rcv buffer…status trigger input that sets the receive buffer indices back to the beginning of the buffer.
- Flags (byte order)…integer or status that sets the byte order: LSB=0 sets MS…LS byte order; LSB=1 sets LS…MS byte order.
- Field N type…integer that sets the conversion length and type for each input: 1=1 byte; 2=2 byte integer; 3=3 byte integer; 4=4 byte integer; 5=4 byte IEEE float, 6=byte has value that must be matched in the incoming message.
- Input N…status, integer or floating point inputs that provide values to be matched if corresponding field type is specified as type 6.

Primary Outputs:

- Trigger rcvd new msg…Name.Rcvd…trigger issued after received message is decoded and all type 6 fields match.
- 1st…9th variable field value…V1-V9…floating point variable value extracted from the decoded message

Outputs for Internal Use: None
Limitations: None

Expected Applications:

Use to decode a binary message and extract its variable data.

DialMdm

The **DialMdm** module is used to control modem board communications on its 2-wire channel when used with the telephone dial up system. With it, you can have the Rug9 dial a distant modem and communicate, or answer the phone and communicate. Once off hook and connected, the unit can employ any of its built in protocols.

To call a distant modem, you must trigger a **ComSetup** module to install modem communications parameters into the port. Then apply the listed inputs below and trigger the **DialMdm** module. Note that the phone number is applied as either a string or an integer, using two separate inputs in the list below. Only one is necessary. If both are defined, the integer form will have precedence. The **DialMdm**'s sequencer output will transition through its states, identified below, ending with either state 20 (connected) or state 0 (idle) depending upon whether it was able to connect or not, respectively.

To cause the RUG9 to answer an incoming call, again trigger the **ComSetup** module to install parameters, then wait for the RUG9 to answer after the number of rings have been received as defined in the **ComSetup** module. If the **DialMdm** module indicates off hook and the sequencer transitions to state 20, then the modem is connected. The **DialMdm** module does the ring counting and timing of the answer tone, so the **DialMdm** module is required, even though the RUG9 doesn't actually dial in this case.

To terminate a call, simply issue a trigger to the "Trigger hang up/key off" input.

Inputs That Must be Constants:

- Board…integer board number (1-8) indicating the location of the modem board in the RUG9
- Port…integer port number (1-3), although the modem board has only port 1 implemented.

Other Inputs:

- Trigger off hook and dial…status trigger input to initiate dialing.
- Ttone string to send…string of phone number digits to dial. The minus character causes a 2 second delay in dialing.
- Ttone integer to send…integer form of phone number. Only usable up to seven digits.

- Dial delay, tenths of second…integer specifying how many tenths of a second the unit is to delay after going off hook before it commences dialing. Recommended 5 to 10.
- Tone ON tenths of sec…integer duration of each tone in dialing cadence. Recommended 3 for standard dial up system, 6 for cell phones.
- Tone OFF tenths of sec…integer duration of each silent period between tones during dialing cadence. Recommended 2 for standard dial up system, 4 for cell phones.
- Trigger hang up/key off…status trigger input that will terminate a call by hanging up the phone and returning the sequencer to the idle state.
- Answer tone tenths of sec…integer duration of the tone the RUG9 will issue immediately after answering the phone. This tone is necessary for some modems to acquire the RUG9's modem tones and prepare to synchronize. Recommended 40 (4 seconds).
- Trigger force answer…status to force off hook as if unit received rings

Primary Outputs:
- Done dialing trigger…Name.Done…status trigger output signaling the end of issuance of touch tones.
- Dial/ans sequencer…Name.Seq…integer indication of the status of the incoming or outgoing call. The states are: 0=idle, 1=waiting to dial, 2=dialing, 10=issuing answer tone, 20=connected.
- Off hook…Name.Offhook…status whose true state indicates the dialing relay is energized and the phone is off hook.
- Ring count…Name.Rngcnt…integer indication of the number of rings accumulated in present ringing sequence. If no rings received in 10 seconds, the ring counter is automatically cleared.

Outputs for Internal Use:
- Old trigger dialing status…Name.Oldd…status echo of trigger off hook and dial input to detect rising edge.
- Old trigger hang up status…Name.Oldh…status echo of trigger hangup/key off input to detect rising edge.
- Sequence timer…Name.SeqTmr…integer timer of internal sequencer functions.

Limitations: None

Expected Applications:
Use to control dialing to other 2-wire dial up modems, and to provide certain services, such as ring counting.

DumpLogToFlashDisk

This module controls writing logged data to the flash disk cartridge. It basically interfaces a **DataLogger** module to the flash disk board to enable dumping logged data samples and time tags to the cartridge in ASCII form using the MSDOS 5.0 file structure. For this to work, you must reference a data logger's Name.Log output in the floating point database, so this module can find the data. You must also supply an eight character or shorter file name to use on the cartridge, with no extension. When triggered, the module will find the data, and find the file on the cartridge, if it exists, then format the data and append it to the file on the cartridge. When done storing data samples, the module can optionally erase the log created by the **DataLogger** module, to avoid duplicate logging of data to the cartridge. If the file cannot be found on the cartridge, this module will create it. If no cartridge is inserted into the flash disk board's receptacle, this module will not attempt to store any data, nor will it erase the log if that option has been selected. This module is not capable of erasing files on the cartridge. The cartridge must adhere to the Compact Flash standard, and must be preformatted with the MSDOS 5.0 file structure.

Data stored on the cartridge by this module will be stored in the following format:

Optional Header

11/15/1999 13:14:36,1.15,1.56,2.78,4.76,4.77,4.79,5.1,5,5.01,5.12
11/15/1999 13:14:45,5.15,5.25,5.26,5.78,4.95,4.34,4.02,3.78,3.41

The optional header string can either be a string constant, or can be assembled from other strings including the realtime clock/calendar. Each line will consist of a time tag, taken from the logged data file, and a series of data samples converted to numeric strings and formatted as specified in this module. The delimiter 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. The file can grow without limit up to the capacity of the cartridge.

In practical applications, you should consider keeping data samples in short logs and letting this module erase the log after each storage operation. For example, if you are required to log every 5 minutes, consider sizing the log to hold 100 samples and dump hourly. This uses a relatively small amount of RAM and stores frequently enough to avoid any sizable data loss due to mis-operation.

Note that once data are recorded onto the cartridge, no battery is required to retain the data indefinitely. The cartridge uses a load leveling algorithm to equalize wear on individual data sectors. Cartridge life is in excess of 100,000 write operations to any data cell. The cartridge may be inserted or removed at any time, but can result in loss of data if the cartridge is removed while data is being written.

Inputs That Must be Constants:

- Flash disk board #…integer indicating location in card cage of flash disk board (1-8).

Inputs That Must be Pointers:

- Logger where data saved…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.
- File name string (DOS)…string with file name the module is to append to on the cartridge. If file does not exist on the cartridge, it will be created. Name must be at most 8 characters with no extension.
- 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 storage of each sample on the cartridge. Since samples are stored as floating point numbers, full precision is available in the data log.
- 1=clear log after dump…status input that if true will cause this module to erase all samples and time tags in the log after the data are dumped to the cartridge.
- Header string…string to be placed in log before each storage event, can be any length.

Primary Outputs:

- Done trigger…Name.DoneTrg…trigger issued at the completion of each data dumping event.
- File size…Name.Size…integer file size in bytes of the dumped file, after the last storage event.

Outputs for Internal Use: None

Limitations: None

Expected Applications:

Use to provide removable, permanent bulk storage of logged data files.

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, specify the beginning index in the buffer and number if samples to dump, specify the data format, and then trigger the dump. 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:

11/15/1999 13:14:36,1.15,1.56,2.78,4.76,4.77,4.79,5.1,5,5.01,5.12
11/15/1999 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 Board #…integer indicating location in card cage of board to be used board (0-8).
- Send to Port #…integer indicating port to use (1-3).

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.
- First sample to send…integer index in log of first sample to send. Set to 1 to start at beginning (oldest sample) of log.
- # of samples to send…integer indicating number of samples to send, total. If larger than log, or larger than number of logged samples, dumping will end with the last sample stored.

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.
- Sample count…Name.Cnt…integer count of samples sent

Outputs for Internal Use:

- Sequencer…Name.Seq…integer internal state sequencer
- Next index…Name.Nxt…integer internal index counter
- Pointer…Name.Ptr…integer pointer to next sample to send

Limitations: None

Expected Applications:

Use to send formatted data log to serial port or printer.

EncodeBinaryMessage

When triggered, the EncodeBinaryMessage module creates a binary string from inputs and sends to the designated port. Each field in the emitted message has a mode designator and an input value. Mode entries and inputs can be fixed values or entries obtained from the databases. Numeric inputs are converted to series of bytes based on the mode designator for each field: Mode=1 byte; 2=2 byte integer; 3=3 byte integer; 4=4 byte integer; 5=4 byte IEEE float. The first blank or zero mode terminates the message. Flags set certain characteristics: bit 0=blank or zero sets MS...LS byte order; bit 0=1 sets LS...MS byte order for integers.

Inputs That Must be Constants:

- Board #…integer indicating location in card cage of board to be used board (0-8).
- Port #…integer indicating port to use (1-3).

Other Inputs:

- Trigger input…status trigger input that initiates the string conversion and sending sequence.
- Flags (byte order)…integer or status that sets the byte order: LSB=0 sets MS…LS byte order; LSB=1 sets LS…MS byte order.
- Input N mode…integer that sets the conversion length and type for each input: 1=1 byte; 2=2 byte integer; 3=3 byte integer; 4=4 byte integer; 5=4 byte IEEE float.
- Input N…status, integer or floating point inputs to be converted and included in the outgoing message.

Primary Outputs:

- String send/buffer empty…Name.SentTrg…trigger issued at end of each dumped line after buffer is empty.
- Trigger image…Name.Img…copy of prior input trigger

Outputs for Internal Use: None
Limitations: None

Expected Applications:
Use to send converted values in binary form to a port.

ForwardPortSwitch

The ForwardPortSwitch module designates that background store and forwarding software switch ports after receiving a message to be forwarded before the message is retransmitted. This module is not required in systems performing store and forward operations wherein port switching is not required. In other words, all units will perform store and forwarding without this module; but this module is required if you also require the unit to switch ports as well. You would use this module to tie together two sets of RTU's having differing communications media. For example, an RTU with this module and two modem boards could act as the gateway between a set of field RTU's communicating over radio at 1200 baud, and a set of in plant RTU's communicating over RS485 at 9600 baud. One of this unit's modems would be set for 1200 baud, 4-wire operation; the other would be set to use RS232 at 9600 baud.

This module has three modes of operation depending upon address settings:

- Mode 1) Both addresses left blank…any message received on either designated port and needing to be forwarded, will be retransmitted on the other designated port; i.e., all forwarded messages will be retransmitted with port switching.

- Mode 2) Lowest address of range input=0…messages having source and destination addresses both above the highest address of range input, or both below the highest address of range input will be forwarded without port switching. Messages having source and destination address on opposite sides of the highest address of range input will be forwarded with port switching.
- Mode 3) Lowest address of range input>0…messages having both source and destination addresses within or including the lowest and highest address inputs will be forwarded with port switching. Messages with either source or destination address outside the lowest and highest address range inputs will be forwarded without port switching.

Inputs That Must be Constants:

- First board # (1-8)…integer designating one board to be involved in store and forwarding communications.
- First port # (1-3)…integer designating which port on first board to be involved in store and forwarding communications.
- Second board # (1-8)…integer designating second board to be involved in store and forwarding communications.
- Second port # (1-3)… integer designating which port on second board to be involved in store and forwarding.

Other Inputs:

- Lowest address of range…integer setting operating mode as described above, and, for third mode, setting lowest address of range to cause port switching in store and forwarding operations.
- Highest address of range…integer setting threshold across which port switching will occur in second mode above, or setting the highest address to cause port switching in third mode above.

Primary Outputs:

- Executed forward trigger…Name.FwdTrg…status trigger output that will be true immediately following a store and forwarding action.
- Last forwarded src addr…Name.LastSrc…integer indicating ultimate source address of last forwarded message.
- Last forwarded dest addr…Name.LastDest…integer indicating ultimate destination address of last forwarded message.

Outputs for Internal Use: None

Limitations: None

Expected Applications:

Use this module to setup a unit as a gateway between two groups of RTU's communicating over differing media, or using differing communications methods.

GetDistantLogMany

The GetDistantLogMany module controls retrieval of data from a distant LogMany logger over the telemetry channel with the retrieved data deposited in an identical LogMany module in the local RTU. ID numbers designate which LogMany modules are involved. Once initiated, the transfer continues until the distant log is empty, or local local log is full. Link from RX array input should be 4 byte integer entry in the RX array that begins the area used in the RX array for logged data. Enough subsequent 32 bit integer cells should be allocated to reserve space for data records. Link to the TX array output should be dragged into the TX array for retrieval by the distant site. The reset input will stop the transfer and return the module sequencer to idle.

It is important to calculate the size of each record to be transferred in order to assure that the distant transmit array and local receive array have enough 32 bit integer cells to contain the number of records in each transmission. Each record consists of a 32 bit time tag, zero or more floating point samples, zero or one packed 32 bit status words. The number of 32 bit integers necessary for each transmission is equal to:

For logged data sets having NO status inputs:

1+(number of records in each transmission * (1+number of non-status inputs in data set)).

If the data set has one or more status inputs, number of 32 bit integers is equal to:

1+(number of records in each transmission * (2+number of non-status inputs in data set)).

Inputs That Must be Constants: None

Other Inputs:

- Trigger start transfer…status trigger input that will reset the module's sequencer, clear the local LogMany log, and commence commanding the remote site to dump its data.
- # records per message…integer that specifies how many records are to be transferred in each transmission from the remote site; maximum is 31.
- Link from RX array…integer input that is dragged from a 32 bit integer in the RX array that is to be used to transfer file information. The location in the array must be the beginning of a series of 32 bit integers that constitute the area used for logger data transfers.
- ID of distant logger (1-127)…integer identifier that specifies which LogMany module in the distant RTU is to be the source of the transferred data.
- ID of local logger (1-127)…integer identifier that specifies which LogMany module in the local RTU is to be the recipient of the transferred data.
- Reset/stop transfer…status input that will immediately terminate the data transfer I progress.

Primary Outputs:

- Link to TX array…Name.Link…integer output that should be dragged into the TX array in a 32 bit integer cell. This will be the command link to the distant RTU to specify which logger and records are to be transferred in response to each poll.
- Trigger done with xfer…Name.TrgDone…status trigger that will be true for one scan at the end of transfer of the last record.

Outputs for Internal Use:

- Next data record…Name.Next…integer output that holds the record to be transmitted on the next poll.
- Ptr to local logger...Name.Ptr…integer pointer to the local logger to get the next data transferred.

Limitations:

- Both local and distant LogMany loggers must be identical in numbers of inputs designated, as they specify how many words constitute one record.

Expected Applications:

Use this module to accept characters from a port for use by other modules.

GetStrFromPort

This module captures a string from a port, up to a CRLF and makes it 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 or line feed character, or 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 6, 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.
- Board #…integer in range of 0 to 8 specifying with which board in the card cage this module is to work.
- Port #…integer in range of 1 to 3 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.
- String length bytes…Name.Length…integer indicating length in bytes of output string.
- New character trigger…Name.CharTrg…status trigger output true when a character has been received.

Outputs for Internal Use:

- Temporary string accum…Name.StrTemp…string output to hold portion of string received from port.
- Temp string index…Name.IdxTemp…integer index into temporary string accumulator.

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. Start index offset specifies starting field #...5==> start with 5th field. If header string specified, starts parsing only if 1st field matches header.

Inputs That Must be Constants:

- Output string max chars…integer setting the size of each output string.

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 strings and, if possible, floating point values.
- Delim char 44=comma…integer decimal equivalent of ASCII character that designates start of next field. If the blank character (32) is specified as the delimiter then multiple blanks in succession will be regarded as a single delimiter.

- 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 used when delimiter character is blank to specify that an implied delimiter is present every X characters starting from the first character.

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. Start index offset specifies starting field #...5==> start with 5th field. If header string specified, starts parsing only if 1st field matches header.

Inputs That Must be Constants:
- Output string max chars…integer setting the size of each output string.

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. If the blank character (32) is specified as the delimiter then multiple blanks in succession will be regarded as a single delimiter.
- 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 used when delimiter character is blank to specify that an implied delimiter is present every X characters starting from the first character.

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:
Used to pull floating point values from serial strings.

ParseStringToInt

The ParseStringToInt module reads characters from input string and parses fields into individual integer outputs. Start index offset specifies starting field #...5==> start with 5th field. If header string specified, starts parsing only if 1st field matches header.

Inputs That Must be Constants:

- Output string max chars…integer setting the size of each output string.

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. If the blank character (32) is specified as the delimiter then multiple blanks in succession will be regarded as a single delimiter.
- 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 used when delimiter character is blank to specify that an implied delimiter is present every X characters starting from the first character.

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:

Used to pull integer values from serial strings.

ParseStringToStatus

The ParseStringToStatus module reads characters from input string and parses fields into individual status outputs. Start index offset specifies starting field #...5==> start with 5th field. If header string specified, starts parsing only if 1st field matches header.

Inputs That Must be Constants:

- Output string max chars…integer setting the size of each output string.

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. If the blank character (32) is specified as the delimiter then multiple blanks in succession will be regarded as a single delimiter.
- 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 used when delimiter character is blank to specify that an implied delimiter is present every X characters starting from the first character.

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:

Used 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 designated 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. This module works for RUG6 and RUG9 polling only. For Modbus polling, use the **PollModbus** 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.

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. The word selections work differently for RUG6 and RUG9 protocol. Each word is a 16 bit integer. In the case of the RUG9, 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.

In the case of the RUG6 protocol, that protocol supports a maximum of 32 words per message each direction. The "First word to send" entry is ignored; only the "Number of words to send" entry is used. If the "Number of words to send" entry is less than 33, then the transmission will start with word 1 and include the number of words specified. If the "Number of words to send" exceeds 32, then the first word that will be sent will be word (number of words to send – 31), and 32 words will be sent. For example, if the "Number of words to send" is set to 45, then the message would contain words 14 through 45. The same methodology applies to the reply message word specification.

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:

- Board #…integer in range of 1 to 8 specifying with which board in the card cage this module is to work.
- Port #…integer in range of 1 to 3 specifying with which port on the board this module is to work.

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. Ignored in RUG6 protocol. Range is 0 through 65535.
- Number of words to send…integer that, for RUG9 protocol, specifies number of words to transmit from transmit array. Max value for RUG9 is 125. For RUG6 protocol, specifies highest numbered word to send. Max value for RUG6 is 126.
- First word to reply…integer that, for RUG9 protocol, specifies first word in transmit array at destination to send in reply. Ignored in RUG6 protocol. 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 RUG9 is 125. For RUG6 protocol, specifies highest numbered word to send in reply. Max value for RUG6 is 126.
- 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:

- Old trigger status…Name.Old…status image of trigger input to detect rising edge.

Limitations:

- Not to be used for Modbus polling.

Expected Applications:

Used 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.

Message Types Supported

The Modbus protocol uses a separate function code for each type of message. The RUG9 **PollModbus** module supports the following function codes/message types:

Table 12 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:

- Board #…integer in range of 1 to 8 specifying with which board in the card cage this module is to work.
- Port #…integer in range of 1 to 3 specifying with which port on the board this module is to work.

Other Inputs:

- Trigger input…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 65535.
- 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.
- Coil/input item offset…integer that is added to the coil address before transmission of polls referencing coils. Range is +/- 65535.
- Register item number offset…integer that is added to the register address before transmission of polls referencing registers. Range is +/- 65535.

Primary Outputs: None

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

Limitations: None

Expected Applications:
Used to perform details of polling. Usually used in conjunction with **SequenPoll** module.

PrnSetupWatch

You must have one, and only one, of these modules for each printer port in your unit. This module establishes the print spooler, or output buffer, to be used with each printer. Typical spooler size is 4000 bytes. The larger the spooler, the less likely the possibility of software losing characters due to the printer's relative slowness. This module also monitors the printer's health, providing indications of out of paper, busy, or error. This module initializes the printer port in a manner similar to the way the **ComSetup** module initializes serial ports.

Inputs That Must be Constants:
- Card # (1-8)…integer in range of 1 to 7 specifying with which board in the card cage this module is to work. Note that the printer port cannot be used in slot 8.
- Print spooler bytes…integer in range of 320 through available memory size that defines the size of the printer port's output spooler. Larger is better to avoid losing characters.

Other Inputs: None

Primary Outputs:
- Printer busy…Name.Busy…status output that, when true, indicates the printer is busy.
- Out of paper…Name.Paper…status output that, when true, indicates the printer is out of paper.
- Error…Name.Error…status output that, when true, indicates that printer has some sort of error, including disconnected cable, paper jam, off line, etc.

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Must be used if printer port is to be used.

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 trigger output is asserted 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.
- 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 time 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 change…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.

Limitations: None

Expected Applications:

Use to control polling in remote sites that must report by exception.

SendAlertData

When triggered, the SendAlertData module sends ID/data combination pairs using the ALERT format. ID range is 0-8191, data value range is 0-2047. Each ID/Data pair results in a 4 byte ALERT standard message being sent to the designated port. Up to 10 ID/data pairs can be defined to constitute one transmission. The module will install data in the transmit buffer for transmission until it encounters a blank ID or hits the end of the input list.

Inputs That Must be Constants:

- Board #…integer in range of 1 to 8 specifying with which board in the card cage this module is to work.
- Port #…integer in range of 1 to 3 specifying with which port on the board this module is to work.

Other Inputs:

- Trigger input…status trigger input that, when true, will cause the module to create a new ALERT message.
- ID #N…integer in the range of 0-8191 that sets the ID number in the ALERT system of one data point.
- Data #N…status, integer or floating point value in range of 0-2047 to be sent with its corresponding ID number.

Primary Outputs: None

Outputs for Internal Use: None

Limitations: None

Expected Applications:

Use this module to prepare messages for transmission to ALERT master receivers.

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 board and port number are specified, the string is sent to the designated port for transmission, in addition to being emitted from this module as a string. If the delimiter/make binary character is set to <0, then string entries are converted from space-delimited hex ASCII to binary. I.e., ... F3 4C ... would be converted to binary 1111001101001100. A negative value for the End Character/add CRC field sets the CRC mode as follows: Mode=-1: append DNP3 CRC to each line; Mode=-2: append RUG9 CRC to end of message. If the output string max characters input is blank or zero, no output string will be saved in the string database. If board & port are specified, then the module sends the created string to the designated port.

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.
- Board #…integer in range of 1 to 8 specifying with which board in the card cage this module is to work.
- Port #…integer in range of 1 to 3 specifying with which port on the board this module is to work.

Other Inputs:

- Trigger input…status trigger input that, when true, will cause the module to perform a new string conversion.
- Delim char/make binary…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. If value is <0 then string entries are converted to hex ASCII.
- 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/add CRC…integer form of the desired terminating character. A value of 10 gives the CRLF character pair. Leave blank if no terminating character is desired. A negative value sets the CRC mode as follows: Mode=-1: append DNP3 CRC to each line; Mode=-2: append RUG9 CRC to end of message.
- 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.

SequenATPoll

The SequenATPoll sequencer issues AT-type commands to an external cell modem with correct timing for round robin or single remote polling. At the start of a cycle, the module triggers ComSetup for ASCII (mode 7) mode, sends commands specified in the command string to transfer the modem to command mode, sends the remote IP address (usually from a table), and the port address, then commands the modem to data mode and triggers ComSetup to mode for data communications (usually RUG9 protocol). At that point the next actions depend upon the Mode input:
1=poll the same address repeatedly without restarting the modem,
2=advance the address counter and restart the modem before each poll,
3=poll the existing address once then remain in receive mode,
4=skip polling and advance to receive mode.
If polling, after polling the sequencer will wait for a trigger on receive up to the receive delay. Whether it receives a receive trigger or not, it will then wait for the poll delay before progressing to the next address/poll. Enable true will start/continue the process. Enable false will halt the process in the receive state. For modes that stop in the receive state, the Trigger new cycle input will restart and run one cycle.
Sequencer states: 1-12: setup modem, 13=poll, 14=waiting for receive trigger, 15=waiting (poll delay) after receive trigger or timeout, 16 advancing address counter to next online address.
Command string specifies the ATDT-type commands necessary to set up the external modem. It consists of short fields separated by time delays delimited with underscore characters such as this string compatible with the Sena modem:

"+++_10_ATH_+2_ATZ_+2_ATDT" The time delays (10 and +2 in this case) specify the number of tenths of a second delay after each string. The plus (+) symbol specifies that a CRLF is to follow the preceding string. The IP address and port strings use the identical format. For the Sena, this string format works:
IP address:"192168000012".
Port string="4000_+2_".
This specifies address 192.168.000.012 on port 4000 to be followed with a CRLF and then a 0.2 second delay before entering receive mode.

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 is 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 to write com fail and other com statistics to tables.
- Go to next cycle.

Inputs That Must be Constants:

- Board #…integer in range of 1 to 8 specifying with which board in the card cage this module is to work.
- Port #…integer in range of 1 to 3 specifying with which port on the board this module is to work.

Other Inputs:

- Mode (1-4)…integer: 1=poll the same address repeatedly without restarting the modem, 2=advance the address counter and restart the modem before each poll, 3=poll the existing address once then remain in receive mode, 4=skip polling and advance to receive mode.
- Enable…status input that, if blank or true, enables polling. If set to zero, polling is halted.
- Trigger new cycle…When true, starts a new cycle.
- 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 address count…integer setting the highest state for this module's address counter. After the sequencer hits this state, it will transition to the min address count, below.
- Min address count…integer setting the lowest or beginning count of the module's address counter.
- Receive delay sec…integer number of seconds the module is to wait for a reception before the next poll
- Poll delay sec…integer number of seconds the module is to wait after a reception before the next poll
- Trig on Rcv input…status trigger input indicating that a reception has occurred.
- Command string…string such as: "+++_10_ATH_+2_ATZ_+2_ATDT" that commands the external modem to prepare to transmit.
- IP address string…string such as :"192168000012" that specifies the IP address.
- Port string…string such as: "4000_+2_" that specifies the modem's port.

Primary Outputs:

- Address counter…Name.AddrCount…integer address of destination station. This can be sent to the **Poll** module as the station address, or this is the column read selector of any tables used by this module.
- Poll trigger…Name.PolTrg…status trigger output sent to the **Poll** module to issue one poll.

- Com install ASCII trigger…Name.InstallASCIITrig…status trigger output used to trigger installation of the serial port parameters for ASCII tramsnission (triggers **ComSetup** module for ASCII protocol).
- Com install data trigger…Name.InstallDataTrig…status trigger output used to trigger installation of the serial port parameters for data tramsnission (triggers **ComSetup** module for R9 protocol).

Outputs for Internal Use:

- Sequencer…Name.Seq…integer state counter used to control module sequencing for each poll.
- Timer…Name.Tmr…timer for internal sequencing, waiting for reception, etc.
- Parse index…Name.Idx…integer index for message parsing.

Limitations: Only used for external AT type modem control.

Expected Applications:

Use in all applications where the unit needs to control an external modem that accepts AT commands for configuration.

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. This module is usually coupled with one or more tables that hold station addresses, number of registers to send and receive, com fail counts, com fail flags, etc. 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, or as specified by the entries in a table. In a typical application, the module's state counter output would be used to select a column of a table. The table's outputs would then be used by the sequencer to provide necessary data to control polling, for example, each station's on-line flag. By placing addresses and station on-line flags in a table, the operator can control the order of polling and can enable/disable individual stations from the RUG9's LCD display.

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 to write com fail and other com statistics to tables.
- 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 from a table. This module will increment or clear this value and write it to the table depending upon whether it receives a reply to the last poll. It also uses this count to establish whether the station should have a communications failure declared.
- 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.

Primary Outputs:

- Sequencer state…Name.State…integer state of main sequencer. This can be sent to the **Poll** module as the station address, or, more commonly, this is the column read selector of any tables used by this module.
- Poll trigger…Name.PolTrg…status trigger output sent to the **Poll** module to issue one poll.
- Write table trigger…Name.Wtrig…status trigger output used to trigger a column write of tables used by this module.
- 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.
- Com fail flag…Name.Fail…status: 0=com ok, 1=com failed.

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.

SetDisplay

This module enables you to force a particular realtime display to be presented as the present default screen. To do this, you simply supply the display number and port number, then trigger the module. After that, the specified display will become the new realtime display. Note that the LCD is port 0, the programming port is port 1 and, for serial ports on the boards, the port number is calculated as PORT=BOARD*3+(PORT ON BOARD)-1.

Inputs That Must be Constants: None

Other Inputs:

- Display #…integer display number among displays designated for this port.
- Port #…integer port number as defined above.
- Trigger to install…status trigger input that, when true, causes the display to be set to the display defined by this module.

Primary Outputs:
- Present display number…Name.Dsp…integer display number presently being shown.

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Used to force display to a particular display that may present new or critical information.

SpeechAccess

The SpeechAccess module provides low level speech interface for user-designed dialing sequencers. This module should not be used with any other speech modules. Phone state: 0=idle, 1=tone, 2=dialing, 3=ringback, 4=busy. Speech state: 0=idle, 1=playback, 2=recording, 3=deleting. TT (TouchTone) sampling control: 0=off, 1=on, 2 or blank=enabled when off hook.

Inputs That Must be Constants:
- Board #…integer in range of 1 to 8 specifying with which board in the card cage this module is to work.

Other Inputs:
- Trigger on hook…status trigger that will cause the dialer to go off hook (start call).
- Trigger on hook…status trigger that will cause the dialer to go onhook (terminate call)
- Trigger dial phone number…trigger to cause the dialer to commence dialing.
- Trigger playback messager…status trigger input that will commence playback of the selected message.
- Trigger record message…status trigger input that will commence recording of the selected message.
- Trigger stop audio…status trigger input that will terminate recording or playback.
- Trigger delete single message…status trigger input that will delete the selected message from the flash memory.
- Trigger delete all messages…status trigger input that will delete all message in the speech flash memory.
- Trigger start aux tone…begin warbling tone.
- Trigger clear TT buffer…erase touchtone buffer
- Trigger dial pager string…send pager string
- TT sampling (0,1,2)…integer: : 0=off, 1=on, 2 or blank=enabled when off hook.
- Message # to rec/play/del…integer number of the message that will become the next message to act upon.
- Touchtone string to dial…string to send as touchtones when trigger to dial asserted
- Touchtone string to page…string to send when page trigger received.
- Aux tone to send …warbling alert tone to send (1-6)

Primary Outputs:
- Phone line tone state…Name.Pstate…integer indicating call progress (see above).
- Received TT digit trigger…Name.DigitTrg…status that new touchtone digit was received
- Last TT digit received…Name.Digit…integer of last touchtone digit received
- Received TT value trigger…Name.ValTrg….trigger status that new value received from user.
- Last TT value received…Name.Val…floating point value last receive from user.
- Ring count…Name.Rings…integer count of rings received.
- Process done trigger…Name.DunTrg…status trigger output indicating the current event is finished.
- Speech vacant sectors…Name.Vacant…integer indication of number of sectors remaining to be recorded. Updated after each record event.

Outputs for Internal Use:
- Speech state…Name.Sstate…internal speech sequencer

Limitations: None

Expected Applications:
Used to interface to dialer board to record/playback/delete messages.

SpeechDialAnswer

The **SpeechDial/Answer** module takes care of the details of dialing, answering, and providing call progress detection associated with the dialer board and the dial up phone system. This module should be used with the **SPSequenDial** module and associated tables to accomplish speech autodialing. See the autodialer example.

Inputs That Must be Constants:
- Board #…integer in range of 1 to 8 specifying with which board in the card cage this module is to work.

Other Inputs:
- Trigger hangup/key off…status trigger input unconditionally hangs up the phone, turns off the key line, and resets the module's sequencer to the idle state. This usually comes from **SpSequenDial** module. This, therefore constitutes a brute force alarm acknowledgement.
- Trigger offhook & dial…status trigger input starts the dialing process if the sequencer is idle. If the dialer is connected to a user, a trigger here will cause the dialer to issue the designated touchtone string (e.g., to a pager). This usually comes from **SpSequenDial** module.
- Ttone string to send…string of touchtone digits to send when triggered. A [-] minus character will cause the dialer to insert a two second delay in the dialing timing. This input usually comes from a table of operator information.
- Ttone Integer to send…integer value of touchtone digits to send. You should use the string form if you anticipate phone numbers in excess of 7 digits, or if you need to imbed minus [-] characters to implement delays.
- Rings to answer…integer sets how many rings must be detected before the unit will autoanswer the phone line. Setting to zero or leaving blank will disable autoanswering.
- Trigger send pager string…status trigger input to cause pager string to be issued. This usually comes from **SpSequenDial** module.
- Pager TT string to send…string of digits to send a pager. This usually comes from a table of operator information.

Primary Outputs:
- Call progress…Name.Prog…integer indicating call status as follows:
  0=idle
  1=tone present
  2=dialing
  3=ringback detected (distant phone is ringing)
  4=busy signal detected.
  5=connected with user (i.e., no tones present following one or more ringback cycles)
- Connected with user…Name.Trg…status trigger indicating that a user answered. Used by other modules to start a verbal menu.
- Received TT digit…Name.DigitTrg…status trigger indicating that a touchtone digit has been received. Used by **SpSequendial** module to detect menu choices.

- Last TT digit rcvd…Name.Digit…integer indicating what digit has been received (0-9, 11=*, 12=# key).
- Rcvd TT value trigger…Name.MsgTrg…status trigger indicating that a floating point value was received in the form of a touch tone character sequence. This trigger is issued whenever a [#] character is received, signifying the end of a character sequence. Used to tell the **SpSequenDial** module that the user has entered a value for use as a new setpoint.
- Last TT value received…Name.Msg…floating point value, the last value received in the form of a touch tone character sequence terminated by the [#] key. The [*] key is regarded as a decimal point. Used to pass to the **SpSequenDial** module a new value for use as a new setpoint.
- Rings received…Name.Rings…Integer count of rings received. This will be cleared 10 seconds after no further rings are received.

Outputs for Internal Use:

- Dialing sequencer…Name.Seq…integer internal sequencer

Limitations: None

Expected Applications:

Use with the **SpSequendial** module to implement autodial/autoanswer capability.

SpeechRecPlayDel

This module provides the sequencing, timing and interface to the dialer board to accomplish speech message recording, playback and deleting. In a way, it makes the dialer board behave like a tape recorder, except that messages are addressable, and require no rewind or fast forward time. On the dialer board, messages are stored in a flash memory having 512 sectors. No battery is required to retain recorded speech. Each addressable message uses at least one sector. Address range is 1 to 254. Messages can be recorded, played back and erased without affecting other messages. Messages can also be of any length up to the remaining capacity of the flash IC. Each recorded message will require at least one sector, even if the recorded message is shorter that 1.42 second. Total recording capacity is 12 minutes (510 sectors of 1.42 seconds each). (Some sectors are used for message indexing.) To use this module, first set the message number to take action upon; then trigger the action. For recording, you must also trigger the termination of recording. It is most convenient to set up a LCD screen to prompt for all these functions, and then set up a key for each function.

Inputs That Must be Constants:

- Board #…integer in range of 1 to 8 specifying with which board in the card cage this module is to work.

Other Inputs:

- Preset message #…integer number of the message that will become the next message to act upon when the preset trigger below is true.
- Preset trigger…status trigger input that will install the preset message number above as the next message number to take action upon.
- Begin recording trigger…status trigger input that will commence recording of the selected message.
- Begin playback trigger…status trigger input that will commence playback of the selected message.
- Stop record/playback…status trigger input that will terminate recording or playback.
- Delete message trigger…status trigger input that will delete the selected message from the flash memory.
- Delete ALL messages trigger…status trigger input that will delete all message in the speech flash memory.
- Increment message trigger…status trigger input that will increment the selected message number.

- Decrement message trigger…status trigger input that will decrement the selected message number.
- Record mode…integer specifying: 0=standard; 1=playback immediately after recording finished.
- Auto increment…integer specifying: 0=no increment; 1=increment message immediately after recording or playback event in preparation for the next event.

Primary Outputs:
- Next message…Name.Nxt…integer indicating the next message to be effected.
- Process done trigger…Name.DunTrg…status trigger output indicating the current event is finished.
- Vacant sectors…Name.Vacant…integer indication of number of sectors remaining to be recorded. Updated after each record event.
- Recording now…Name.Rcd…status that will be true during recording
- Playback now…Name.Play…status that will be true during playback

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Used to interface to dialer board to record/playback/delete messages.

SpSequenDial

The **SpSequenDial** module has the job of pulling together the dialing module, the speech reports and a table of operator phone numbers and other information to implement a complete speech autodialer. Its job is to commence dialing when triggered, issue the phone number to the **SpeechDialAnswer** module and trigger it to start dialing, detect call progress, and either advance to the next operator or issue a verbal menu. Once connected, the module must capture user touchtone keystrokes and act accordingly, call into operation the proper verbal report, and hang up the phone when done. It also similarly handles the autoanswer sequence. In addition, it can implement the ability for an operator to control equipment and change setpoints from his touch tone phone. It can also be made to issue reports independently of the phone for local annunciation. In the process of dialing and answering, this module will ask the called operator for a security code if he selects that he wishes to acknowledge alarms, or wants to alter a setpoint. If the operator has been called by the dialer, he would use his personal security code to acknowledge alarms or alter setpoints. If he called into the dialer, he would use the security code of the first operator. See the autodialer example for details of a complete autodialer.

Inputs That Must be Constants:
- Board number…board's location in base card cage (1-8).

Other Inputs:
- Trigger Start Sequence…status trigger to command the sequencer to begin dialing the first operator in the list. This indicates that a new alarm exists. The dialer will dial until a stop and reset trigger is received or until an operator acknowledges.
- Trigger Stop and Reset…status trigger that unconditionally stops the dialing sequence and resets back to idle.
- Min operator number…integer that specifies which operator in table is to be dialed first, usually 1.
- Max operator number…integer that specifies the last operator in table to be dialed. After this operator is dialed, the sequencer will go back to the minimum operator number.
- 0=oper., val=sec before page…integer that selects whether the called number is an operator or a pager. If a pager ('val'=non-zero), the sequencer will command the **SpeechDialAnswer** module to issue a pager touch tone sequence a number of seconds (set by 'val') after it obtains a connection. If 'val' is set to a negative number, then paging tones will commence 'val' seconds after the end of dialing, regardless of call progress. It will then hang up afterwards.

- Operator online flag/override delay…integer: 0= the sequencer will skip this operator; 1= the operator will be dialed and talking will commence after connection established, 2 or more= talking will commence that many seconds after dialing regardless of call progress.
- Delay before next dial…integer number of seconds to delay after an unsuccessful call sequence before calling the next operator.
- Wait sec for user response…integer number of seconds the sequencer will wait for the operator to respond to a menu or to supply a security code before repeating the prompt.
- Trigger connected to user…trigger status input indicating that the call connected to a user. Generally taken from the **SpeechDialAnswer** module.
- Expected ACK code…integer expected security code from the user to acknowledge alarms. Generally taken from the operator table, and used to compare against the operator's entered ACK code.
- Received ACK code…integer code entered by the operator to acknowledge alarms. Generally, an integer value from **SpeechDialAnswer** module.
- Trigger have ACK code…Trigger indicating that an ACK code is present on the input above. When this trigger is received, the sequencer will compare the expected code with the received code and, if they match, will regard alarms as acknowledged and will refrain from further dialing after termination of this call. This is also used to validate that an operator is authorized to modify setpoints.
- Received user key…integer user's response to a menu. Used to select report to present, or exit sequence.
- Trigger have user key…status trigger that new key stroke has been received.
- Trigger done with report…status trigger from report indicating that the report or line of a menu list is done.
- Retries before abandon…integer number of retries of any menu item before abandoning and either proceeding or exiting.
- Report before dial seconds…integer number of seconds the dialer will present the first report verbally before dialing. This is to give any local operator time to acknowledge alarms before dialing commences.

Primary Outputs:

- Operator number (state)…Name.OpNum…integer indicating which operator number the dialer is calling this time. Sent to table to select column of operator information.
- Dial trigger…Name.DialTrg…status trigger to tell **SpeechDialAnswer** module to start the dialing sequence.
- Hangup trigger…Name.HangTrg…status trigger to tell **SpeechDialAnswer** module to hang up the phone.
- Single line select…Name.SglSel…integer to select a single line from the menu speech report for issuing prompts. This value is passed to the menu report's single line selector. The lines in the menu speech report should be:
  Line 1="This is the RUGID autodialer. (Your welcome message)
  Line 2="Hit key 1 for alarms, 2 for report2,….* to acknowledge alarms, # to exit."
  Line 3="Please enter your security code followed by the # key."
  Line 4="Your security code is accepted…alarms are acknowledged."
  Line 5="Your security code is rejected."
  Line 6="Thank you…goodbye."
  Line 7=prompt for up to 9 categories of setpoints to change
  …
  Line 10=prompt for up to 9 setpoints in category 1
  Line 11=prompt for first setpoint to change in category 1
  Line 12=prompt for second setpoint to change in category 1
  …
  Line 19=prompt for ninth setpoint to change in category 1
  Line 20= prompt for up to 9 setpoints in category 2
  Line 21=prompt for first setpoint to change in category 2
  …
- Single line trigger…Name.SglTrg…status that triggers the menu report to say one line and stop.

- Report select…Name.RptSel…integer to select which report to start talking. Usually sent to a sequencer expander's state input to route the actual trigger to the proper report.
- Report trigger…Name.Rpttrg…status trigger that triggers the selected report to start talking. Usually sent to a sequencer expander's enable input.
- Dial state…Name.State…integer indicating dialing status: 0=idle, 1=have unacknowledged alarm
- Trigger send pager tones…Name.PgTrg…status trigger output to signal the **SpeechDialAnswer** module to issue pager tones

Outputs for Internal Use:
- Timer…Name.Tmr…integer internal delay timer to time sequence, not externally usable
- Subsequencer 1…Name.Sub1…integer internal subsequencer, not externally usable
- Retry counter…Name.Retry…integer internal counter to count retries.

Limitations: None

Expected Applications:
Use to perform the logic and sequencing for common autodial/autoanswer applications.

StringConvert

The StringConvert module reads characters from input string and converts from incoming HEX/ASCII representation to an integer value. If the shift input is non-blank, the decoded value will be shifted right or left before being written to the output. Positive shift values result in shift left by the number of bits specified; negative shift values result in shift right by the number of bits specified. Bits vacated by the shift will be zero-filled.

Inputs That Must be Constants: None

Other Inputs:
- Trigger to convert…input status that triggers conversion of input string.
- Mode…integer presently unused.
- String to convert…input string to convert to integer value.
- Shift value…integer specifying number of bits to shift result (positive value shifts left).

Primary Outputs:
- Conversion done…Name.DunTrg…status trigger output indicating that a message was converted.
- Result…Name.Int…integer result of conversion.

Outputs for Internal Use: None

Limitations: None

Expected Applications:
Used to pull integer values from serial HEX/ASCII strings.

StringLeftMidRight

The StringLeftMidRight module reads characters from an input string and provides the equivalent of LEFT$, MID$, RIGHT$, LEN and ASC functions in basic languages. It divides the string into left, middle and right string fields as specified in input properties that specify where the string is to be partitioned. The Len output is length in chars of incoming string. ASC outputs are ASCII decimal equiv. of 1st char of each output field. The leftmost character in the incoming string is regarded as character 0.

Inputs That Must be Constants:

- Output string max chars…integer that defines the RAM to be allocated for the output strings. Should be longer than the longest string anticipated to be created by this module.

Other Inputs:

- Trigger to convert…input status that triggers conversion of input string.
- String to parse…input string to process.
- 1st char # of MID field…integer character number in incoming string of 1st character of extracted MID string
- 1st char # of RIGHT field…integer character number in incoming string of 1st character of extracted RIGHT string

Primary Outputs:

- Conversion done…Name.DunTrg…status trigger output indicating that a message was converted.
- Length of string…Name.Len…integer indicating length in characters of incoming string
- Left string…Name.Left…leftmost partition of incoming string
- Mid string…Name.Mid…middle partition of incoming string
- Right string…Name.Right…rightmost partition of incoming string
- ASC left…Name.ASCL…decimal equivalent of first character of leftmost string
- ASC mid…Name.ASLM…decimal equivalent of first character of middle string
- ASC right…Name.ASLR…decimal equivalent of first character of right string

Outputs for Internal Use: None

Limitations: None

Expected Applications:

Used to pull apart strings for subsequent processing, such as to remove leading or trailing fields.

TriggerOnMBWrite

This module provides triggers when a modbus message writes to individual registers or a range of registers. You could use this module to detect, in a master RUG9, when a SCADA system has changed a setpoint or control bit that needs to be sent to an RTU. The module provides 10 individual registers to which writing by Modbus can be detected; and a range entry so the module can issue a trigger if any register in the range has been written.

Inputs That Must be Constants:

- Board #…integer in range of 1 to 8 specifying with which board in the card cage this module is to work.
- Port #…integer in range of 1 to 3 specifying with which port on the board this module is to work.

Other Inputs:

- Start of register/coil range…integer register or coil number beginning the range the module is to watch for changes. Note this must be a register number, not a signal name in a register.
- End of register/coil range…integer register or coil number ending the range the module is to watch for changes. Note this must be a register number, not a signal name in a register.
- Register/coil 1…10 number…integer individual register or coil number the module is to watch for changes. Note this must be a register number, not a signal name in a register.

Primary Outputs:

- Write to register range…Name.RangeTrg…status trigger indicating that a modbus message has written to one or more of the registers/coils in the specified range.
- Write to register/coil 1…10…Name.Trg1…Trg10…status trigger indicating that a modbus message has written to an individual register.

Outputs for Internal Use: None

Limitations: None

Expected Applications:

Used to detect modbus writing so new values can be sent to RTU's.

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 over written by another reception. They are not dependent upon the source and destination address input specifications of this module.

Inputs That Must be Constants:

- Board to RCV…integer in range of 1 to 8 specifying with which board in the card cage this module is to work.
- Port to RCV…integer in range of 1 to 3 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.

CHAPTER 6…DISPLAYS, REPORTS, LADDER

Display/Report Definition

Displays and reports for the RUG5/9 are defined a screen or page at a time by typing them in the Display/Report Editor page within the R9SETUPD program. Displays and reports are defined in an identical manner, so the following sections will simply refer to them as displays. Along with the display text, you will enter a name for the display that will become a menu entry when run on the RUG5/9 so an operator can access the display. You will also define to which port the display applies (the LCD display, the CPU's serial port, or a port on one of the base card cage's expansion cards). Finally, where you wish the RUG5/9 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 or report. 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 R9 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, Reports, Tables, Ladder

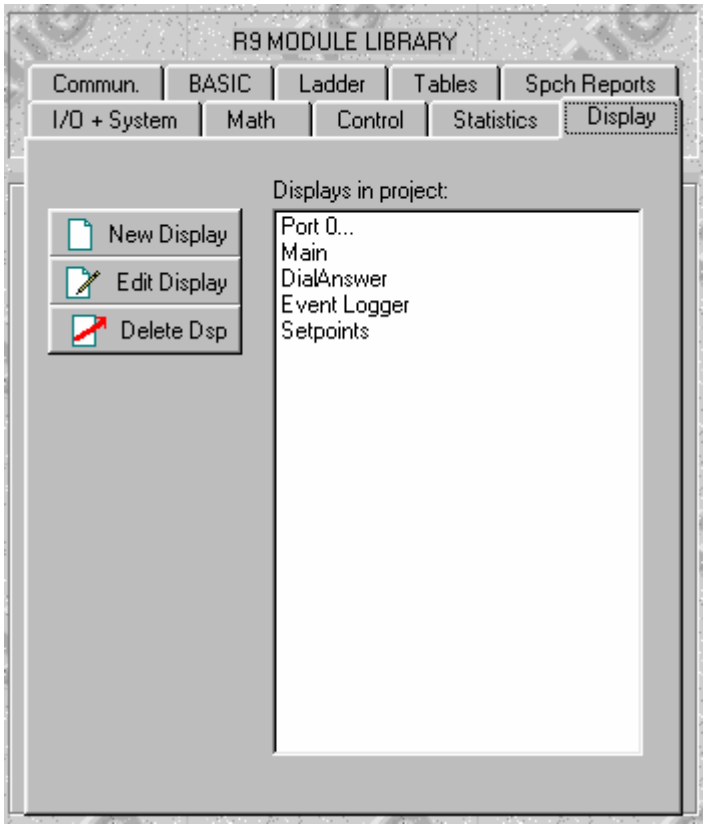


Figure 86 Display Selection Panel

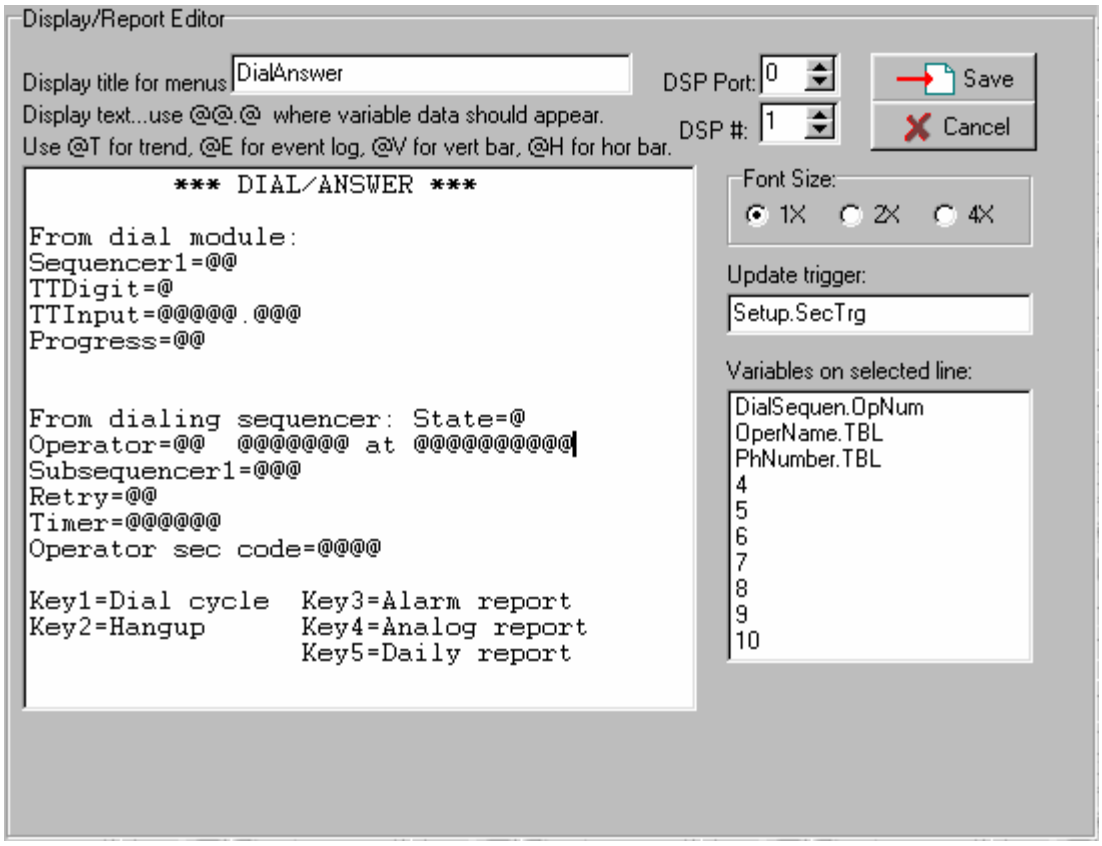


Figure 87 Display/Report Definition Panel

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 'DialAnswer'. The name 'DialAnswer' will appear in the RUG5/9'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 R9SETUPD 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 RUG5/9 [ENTER] key.

Setting Display Port Assignment

The port number specifies which port in the RUG5/9 the display will use. In the RUG5/9, 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 RUG5/9. Note that displays will not be sent to any port not in ASCII mode. Also, only boards in the base card cage can have ports.

Table 13 RUG5/9 Port Assignments

Port 0	LCD Display
Port 1	CPU serial port
Port 3,4,5	Board #1 ports
Port 6,7,8	Board #2 ports
Port 9,10,11	Board #3 ports
Port 12,13,14	Board #4 ports
Port 15,16,17	Board #5 ports
Port 18,19,20	Board #6 ports
Port 21,22,23	Board #7 ports
Port 24,25,26	Board #8 ports

Setting Font Size

The font size radio buttons select font size on the LCD display for the particular display you're working on. Your font size selection applies to all characters displayed; you cannot mix font sizes on the LCD. Also, font size selection only applies to the 320 X 240 LCD display; no provision is made to adjust font or font size for other ports. The table below indicates the numbers of characters and rows available at each font size.

Table 14 Font Size vs Characters on LCD

FONT SIZE SELECTION	CHARACTERS ON LCD
1X	20 lines by 40 characters
2X	10 lines by 20 characters
4X	5 lines by 10 characters

You should use the 1X font size if your display will contain a trend plot or bar graph. If you specify more characters on a line than can be displayed on the LCD, the extra characters at the right end of the line will simply not be displayed.

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 trigger, simply drag the entry from the status database and drop it into the Trigger field. In the above example, 'Setup.SecTrg' is the trigger. It causes the display to update once per second. It's best to use triggers here rather than statuses that may be true constantly because as long as a status is true, it could 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 a 5 second update works well. Finally, for issuing reports to printer ports, you will want a trigger that sends the report to the printer when desired; for example, once per day, or in response to a keystroke for manual report triggering.

Entering the Display Text

The large window to the left of the trigger window is the area wherein you type the text you wish to display on the RUG5/9's LCD, or to transmit to a serial port or to a printer. 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. The large display window is sized to approximately represent the character layout of the RUG5/9's LCD display. In the case of sending displays to serial ports for display on monitors, and for printed reports, 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 RUG5/9 display.

Specifying Variable Data Fields

Wherever you wish the RUG5/9 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 RUG5/9 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 12, which is "Operator=@@ @@@@@@@@ at @@@@@@@@@@@". This line has three variable fields, one for the operator ('@@'), one for the operator's name ('@@@@@@@@') and one for his phone number ('@@@@@@@@@@@'). 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 'DialSequen.OpNum', 'OperName.TBL' and 'PhNumber.TBL'. 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 whose values are too large to display 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.

Specifying Trend Plots

You can specify that a trend plot be placed on the RUG5/9 LCD display by placing the field '@T' where you wish the lower left hand corner of the trend to appear on the LCD. The trend will then occupy the LCD from that point up and to the right except that the top line remains reserved for your title. To specify the data that are to be trended, you must specify one logger module's '.Log' output for each trace you expect the LCD to present. The scale values and time ticks generated by the first specified logger output will be presented. Any other modules specified will simply have their data presented graphically on the same graph as the first but with scales as specified in their own logger modules. The example below shows the display setup for a display with two trends on the same scale. Note that the scales, time ticks, sample intervals, and other details are specified in the logger modules. Also, you should leave at least one line blank below the trend so the trend software can insert time tick values. You should also leave several characters blank to the left of the trend for vertical axis scale values, which are presented automatically if space permits. In the example setup shown below, there are two '@T' fields, specifying that two loggers are to be trended on the same screen. For each, there must be a variable identified so the display can find the logged data. In the example below, the cursor is placed on the line with the '@T@T' entries, and the logger references of 'LvlLog.Log' and 'FlowHistory.Log' are dragged from the floating point database and dropped into the variable list. That's all there is to it. Note that an update trigger of once each five seconds is used to trigger the display update. This is slower than once per second that we commonly use, for the reason that a trend update is time consuming, and usually does not change as frequently as once per second.

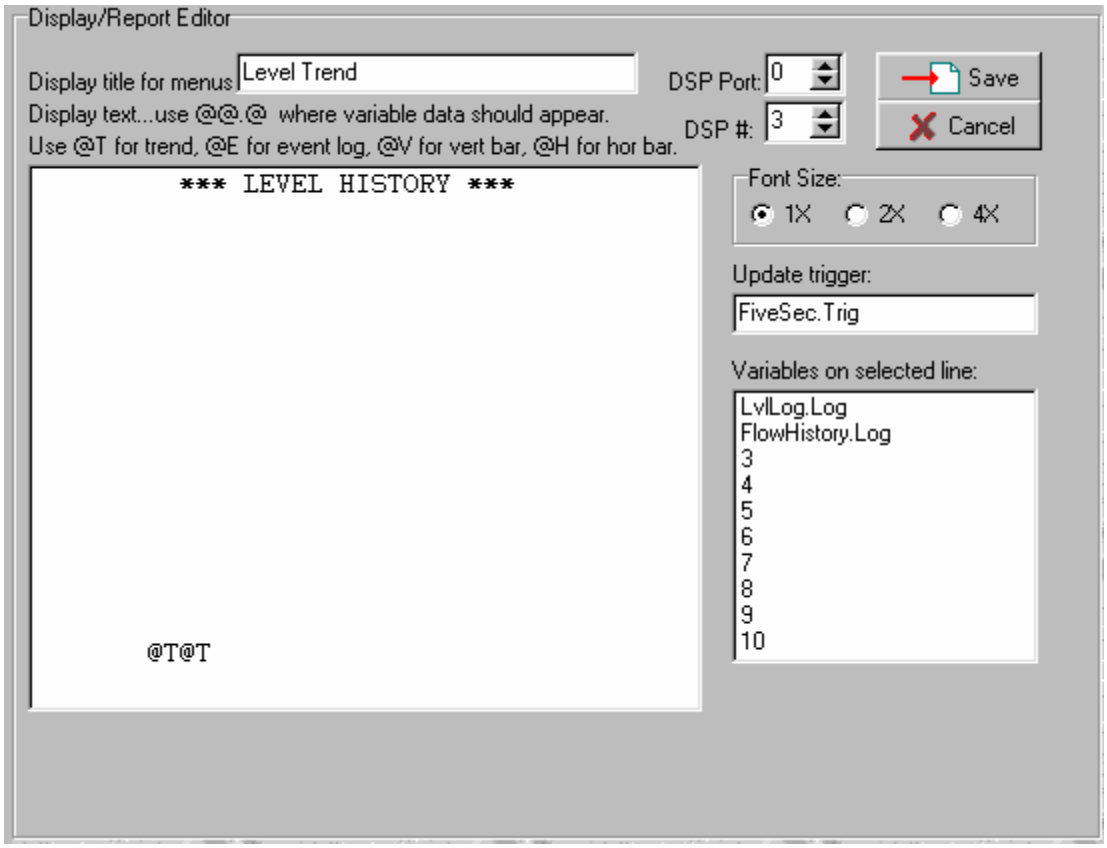


Figure 88 Typical Trend Display Setup

Specifying Bar Graphs

You can specify horizontal bar graphs on any LCD display by inserting the '@H' field on the display where you wish the left end of the bar to be located. For each, you must drag into the 'Variables on Selected Line' box, the output of a **Bargraph** module where the scale values and actual data values are defined. When displayed on the RUG5/9's display, the bargraph will extend from the '@H' specification, to the right edge of the display. You can specify only one horizontal bar per line.

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 15 List of '@' Field Uses

Field in Display	Meaning	Where Applicable
@@@@.@@@	Show value with that format	LCD, serial port, printer port, speech report
@T	Show trend	LCD
@F	Insert form feed	Serial port, printer port
@E	Show event log	LCD, serial port, printer port
@H	Show horizontal bar graph	LCD

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.

Tables

Tables constitute the RUG5/9's way to give your program access to larger lists of data than can be housed in the individual modules. You will use tables to hold lists of, for example, operator phone numbers to be autodialed; speech messages for alarm statuses in a system; RTU addresses, names, com enable bits, etc.; lookup tables of pump control setpoints; lookup tables of instrument linearization constants, etc. Tables are also useful for holding lists of things that involve occasional exceptions. For example, you might use a simple counter to control polling a series of RTU addresses, but how do you handle occasionally declaring a station off line and not to be polled? Or, how do you request more data from some stations than others? Tables enable you to do all of these things conveniently. See the application note on polling, and the configuration file 'R9POLLER' for an example of using tables in a master polling application.

How Tables Work

Each table in the RUG5/9 consist of a two dimensional array of entries in a row and column format. You specify the numbers of rows and columns in the table at design time, as well as the data type (integer, floating point, etc.) for each row, and each row's name. You can install constants or variables in individual cells at design time. To install a constant, you select the cell and then type in the value or string. To install a variable, simply drag it into the cell from a database. If you install a constant or variable into a cell at design time, the cell contents will not be editable from the RUG5/9's LCD. Alternatively, you could leave the cell blank, in which case, the compiler assigns RAM to the cell, making the cell editable from the RUG5/9's display/serial port.

At run time, tables are accessed one column at a time by controlling a 'Column Read Selector' which selects one column of a table. The table constantly places the cell contents of the selected column on its corresponding row outputs. They appear in the databases with the names you assigned the rows, followed by the '.TBL' extension. Usually, the column selector will be driven by a sequencer, setpoint or counter. For example, a sequencer can increment through its states and, at each state, the table will supply outputs that setup time delays, select pumps, select setpoints, etc. A polling sequencer might use a table to supply station address, retry delay, on/off line flag, baud rate, station type (RUG6 vs RUG5/9 for example), etc. as it cycles through its states. Table entries can also be written at run time one column at a time. The program does this by setting a 'Column Write Selector' and then issuing a 'Column Write Trigger'. When that happens, cells that you have left blank at design time in the selected column are written, with the values present in the data base variables designated as write inputs to the rows as the source of data to be written.

Designing a Table

First, to get at the RUG5/9's tables you must click on the 'Tables' tab in the module library. You will see the following initial panel:

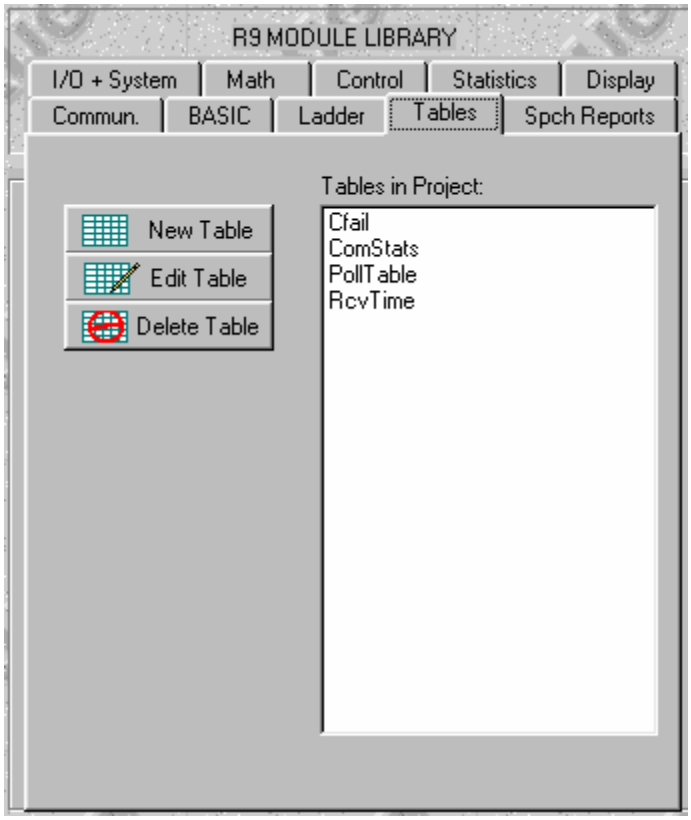


Figure 89 Table Selection Tab

Adding a New Table

To create a new table, click on the 'New Table' button. A blank table editing panel will appear as shown in the next figure.

Deleting a Table

To delete a table from your project, select the table name from the list of tables in the project, then click on the 'Delete Table' button. Once the table is deleted, it cannot be recovered.

Editing a Table

To edit a table that already exists in the project, select the table from the list of tables in the project and then click on 'Edit Table' button. The table will appear in the table design panel, shown below. The table shown below is from the R9Poller application.

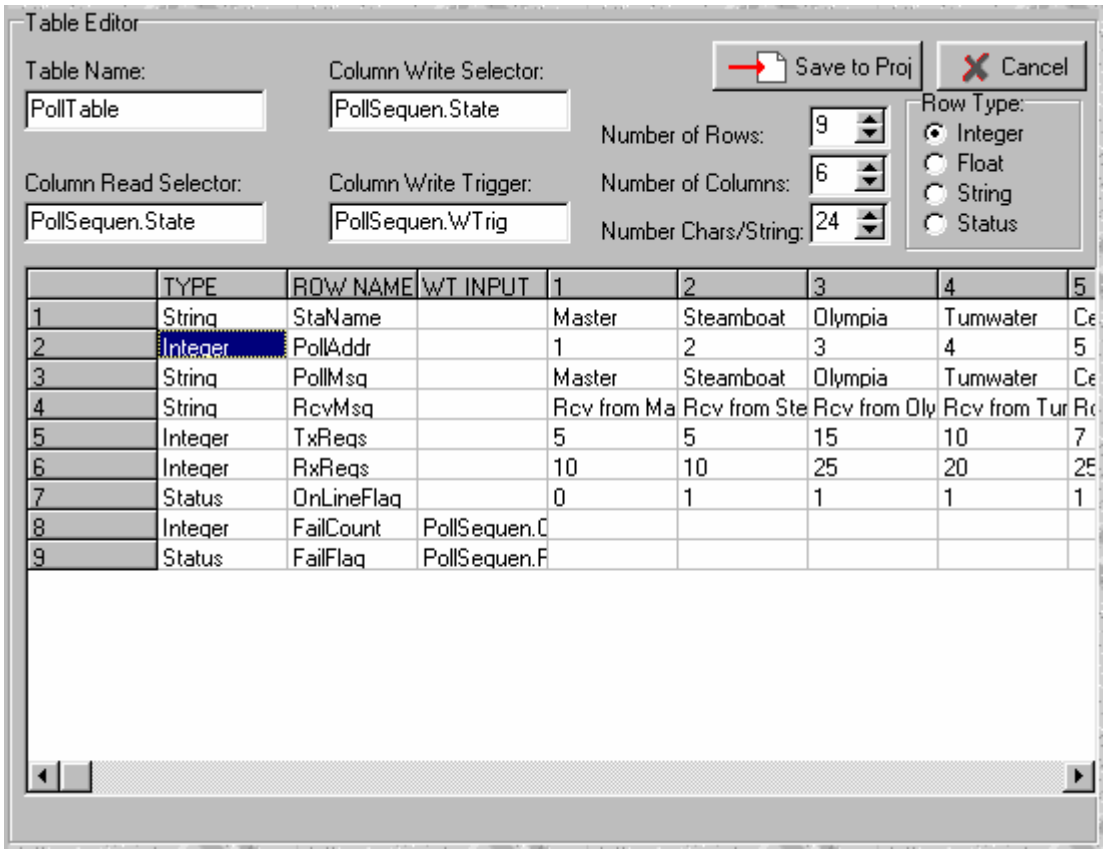


Figure 90 Table Editor Panel

Naming the Table

The first thing you should do after entering this panel when defining a new table, is to name the table by typing a name into the 'Table Name' box in the upper left corner of the panel. Any further references to this table will use this name. The name's limit is 16 characters.

Increasing/Decreasing Number of Rows and Columns, and String Size

The table is initialized with 5 rows and 5 columns. You can easily expand or contract the table using the up and down spin buttons for number of rows and columns in the upper right hand corner of the panel. The only limit on table size is the memory in the RUG5/9. Each cell requires 4 bytes except for cells on rows designated as being string rows in which case each cell requires the number of bytes designated in the 'Number Chars/string' spin button which is initialized at 20 bytes. This sets the number of bytes the compiler allocates for each string in the table, if any.

Setting Row Type and Name

You must give each row a type (integer, string, float, or status) that tells the RUG5/9 how to store and read the values you enter into each cell. It also establishes to which database the output of each row will be saved. To change the row type from integer as shown above, simply click on one of the radio buttons in the box titled 'Row type', and then click on the cell in the table column labeled 'TYPE' on the row you wish to alter. You must also give each row a name, which you enter into the cells in the column labeled 'ROW NAME'. The names you give the rows become the names of the outputs of the table. If you look at the table's rightmost column, labeled 'OUTPUT', you will see the row names you entered with '.TBL' appended. These outputs of the table will be placed in the databases when you save the table to the project, where they can be used as inputs to other modules, displays, telemetry arrays, etc. The row names also become the prompt the RUG5/9 presents the operator at run time when he accesses the table column by column to alter cell contents.

Entering Values Into the Table at Design Time

You can enter data into the table simply by selecting a cell and typing a value or string into it. When the table is compiled, the value you enter will be saved in flash memory at that location in the table. At run time, when that column is selected, the value you entered into the cell will become the output of that row of the table. You can also drag an entry from a database into a cell. The compiler will save a pointer to the data base location in the cell so that at run time, when that row is selected, the signal you dragged into the cell will become the output of the row. Finally, you can leave the cell blank. If you do this, the cell is assigned a RAM location and can be written as well as read at run time.

Reading the Table at Run Time…Column Read Selector

When the tables are loaded into the RUG5/9 and the RUG5/9 is running its program, tables are accessed once per program scan the same as other modules. When this happens, the RUG5/9 will read the 'Column Read Selector' to obtain which column to read. It will then place the contents of that column on the table's outputs in the databases. For example, consider the example table above. The column read selector for this example is 'PollSequen.State', the state output of a polling sequencer. When the program is running and this state has a value of 2, the table outputs are:

StaName.TBL="Steamboat"
PollAddr.TBL=2
PollMsg.TBL="Steamboat"
RcvMsg.TBL="RcvfromMaster"
TxRegs.TBL=5
RxRegs.TBL=10
OnLineFlag.TBL=1
FailCount.TBL= as defined by program
FailFlag.TBL= as defined by program

If the state had a value of 3, then the outputs would be:

StaName.TBL="Olympia"
PollAddr.TBL=3
PollMsg.TBL="Olympia"
RcvMsg.TBL="RcvfromMaster"
TxRegs.TBL=15
RxRegs.TBL=25
OnLineFlag.TBL=1

FailCount.TBL= as defined by program
FailFlag.TBL= as defined by program

Writing to a Table…WT Input, Column Write Selector and Column Write Trigger

If you leave a table's cells blank, the compiler will assign RAM locations to them and they will be alterable at run time. You can mix blank cells and non-blank cells in any way you wish on the table; only the blank cells will be alterable. To be able to write to a cell, you must specify a write input source (WT INPUT) for any row you wish to write. This is where the table will get its data for each row when it is triggered to write. You must also specify a 'Column Write Selector' to specify which column of the table will be written when triggered. This is usually the state of a sequencer or counter. Finally, you must specify a 'Column Write Trigger'. This triggers the writing event, causing all the values in the write input variables to be saved in the corresponding cells at the column specified. Note that any cells that are not left blank by you at design time will not be written at run time, nor will any rows that don't have a write input source. The following example illustrates the process.

In the 9 X 6 table above, row 1 is a read-only row of site names (Master, Steamboat, Olympia, etc.). Rows 2 through 7 are also read only because they have been given contents at design time. Rows 8 and 9 have blank cells in all columns so they can be written at run time, enabling the sequencer to keep track of communications successes and failures, and establish a communications fail status for each remote station. The column write selector (PollSequen.State) is the output of a polling sequencer that needs to write into rows 8 and 9. The column write trigger is the '.Wtrig' output of the PollSequen module. Each time the sequencer issues a trigger on 'Wtrig', its com count and com fail flags are recorded into the table on rows 8 and 9. Using the table to store the communications fail count and fail flag enables the polling sequencer to provide polling for a virtually unlimited number of remote sites. Note that the column read selector is completely independent of the column write selector.

Ladder Logic

The RUG5/9 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 R9 Module Library. The following design page would then appear.

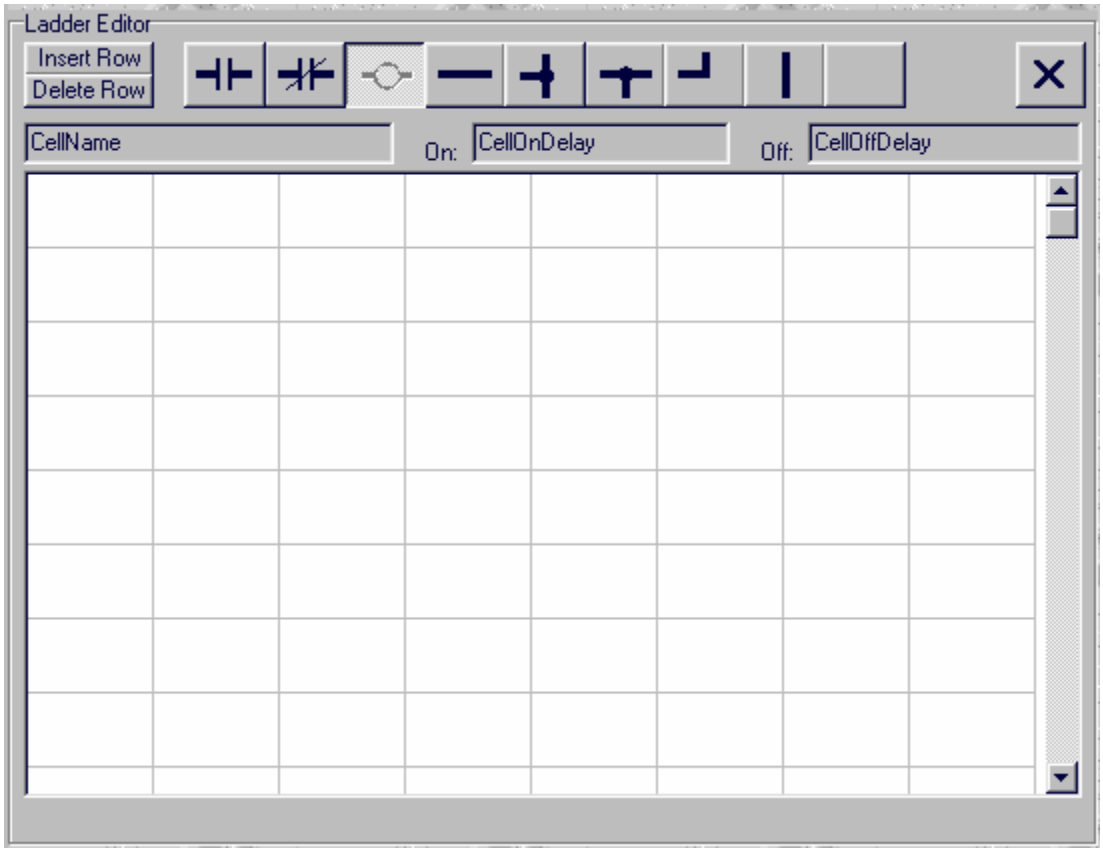
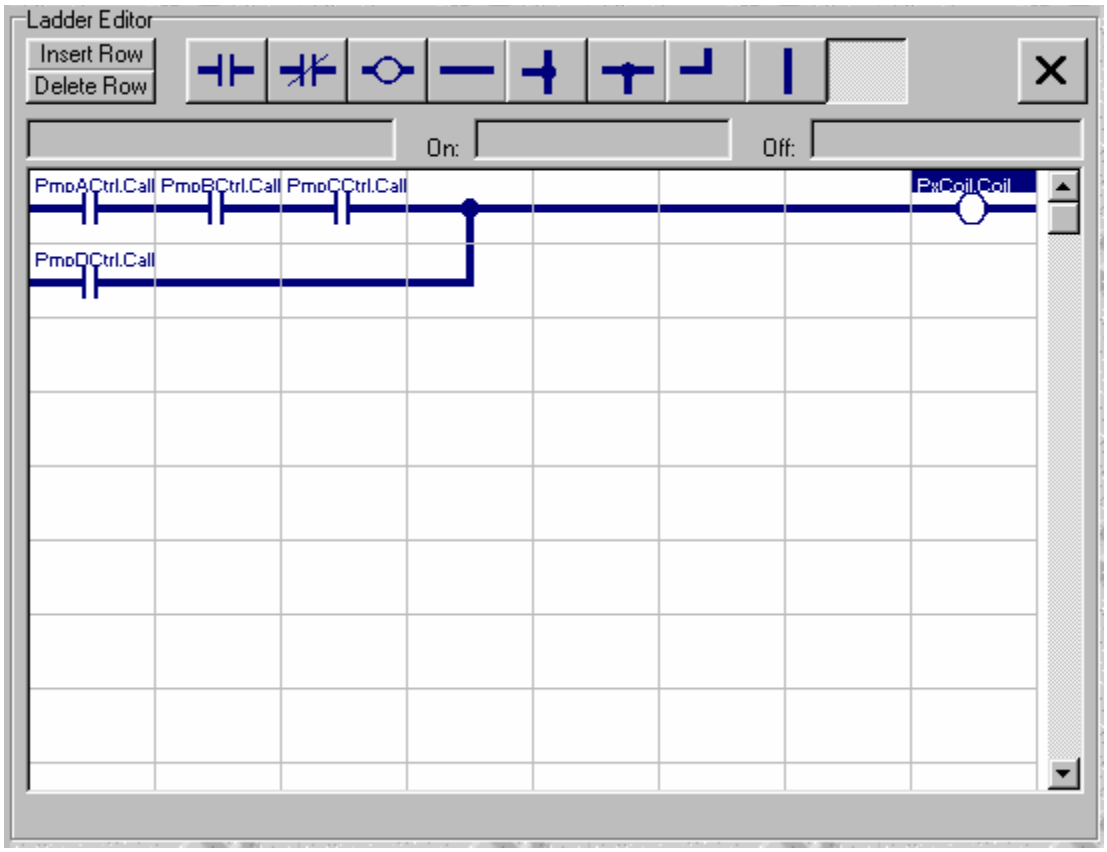


Figure 91 Ladder Editor Initial Panel

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 other 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.

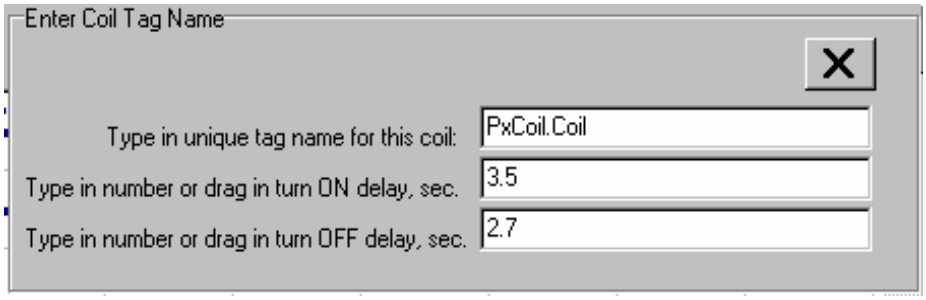


Figure 92 Coil Name and Delay Specification Page

Generally, you will leave the delays at zero, but you can have both call and off delays with ranges of 4,294,767 seconds and resolutions of 1/10 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 from a database for the time delays.

Speech Reports

The RUG5/9 operating system supports speech autodial/autoanswer operations. This section describes how you set up speech reports. Speech reports enable you to define a list of phrases and then connect those phrases together into sentences to report alarms, statuses, values, etc. You can define as many speech reports as flash memory can hold. As you define your reports, you will also define new phrases the report is to speak as it reports. After you finish your report definition, you will load your configuration file into the RUG5/9 and start it running. At that time, and before the RUG5/9 is asked to issue a speech report, you must record the proper phrases onto the dialer board by speaking into its microphone so it can record your phrases with the correct phrase number. The board can record up to 254 phrases into flash memory, which is capable of holding a total of up to 12 minutes of speech. Each phrase can be any length up to the maximum capacity of the board. At run time, any speech report can be triggered, at which time the RUG5/9 will begin speaking the report until it reaches the end of the report, or until a trigger is received by the report to stop talking. Speech reports constitute one part of the RUG5/9's overall speech capability. In order to make use of speech in real applications, you will also need one or more of the following modules:

SpeechRecPlayDel…controls recording, playback and deleting individual phrases
SpeechDial/Answer…controls dialing and answering by the dialer board
SpSequenDial…controls sequencing of dialing, reporting, menuing, security codes, etc.
Table…holds list of phone numbers, operator security codes, names, etc.
OrGate…used to insert multiple triggers for speech functions

In the sections below, the AutoDialer example project is used to illustrate various aspects of speech application setup.

Defining a Speech Report

For a simple speech report, you can store a complete sentence for each line of the report and then specify the order the lines are to be stated when spoken by the RUG5/9. Alternatively, you can define a collection of phrases that can be appended together to form complete sentences. We will illustrate both techniques. To define a speech report, click the "SpchReports" tab in the module library, then press the "New Table" button. The following screen should appear:

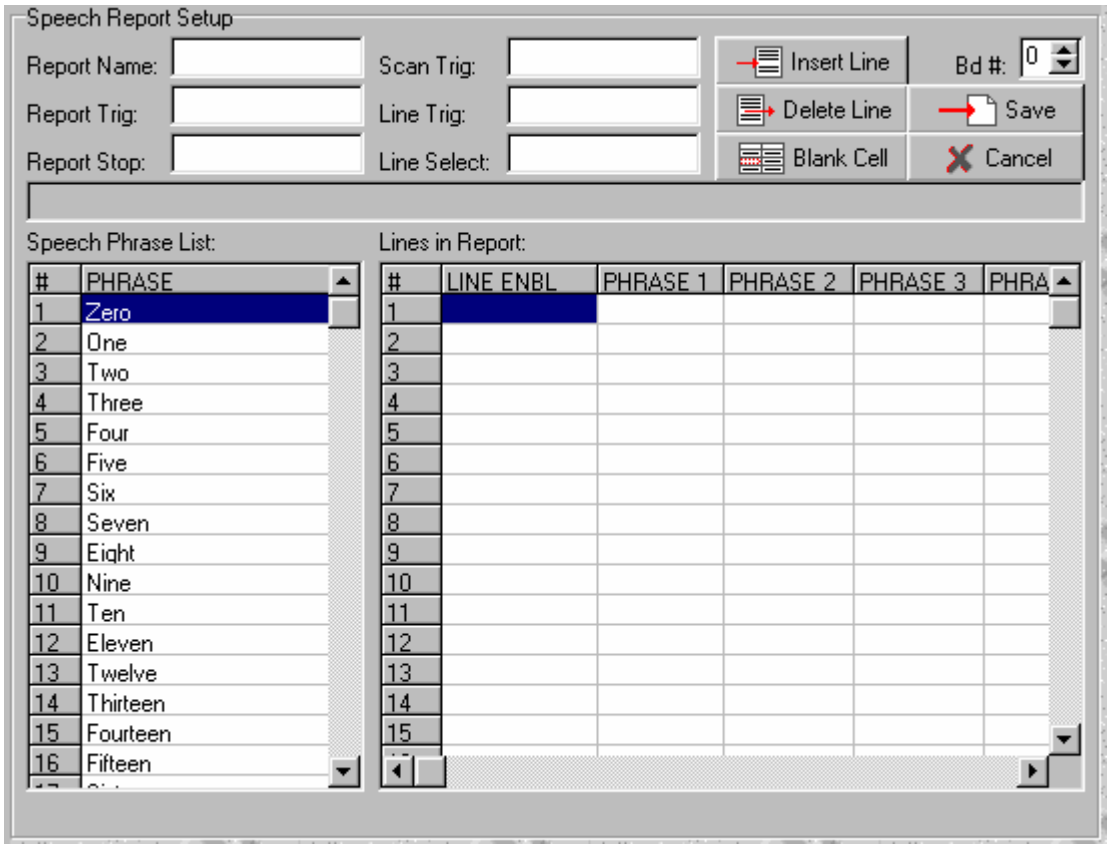


Figure 93 Speech Report Setup Panel

This report definition screen supports reports of any length, with individual lines of up to 14 phrases. In the phrase list to the left, phrases 1 through 34 are already defined by the system to support presentation of analog values verbally. If your application needs to report analog values, then these phrases must be recorded exactly as listed in the phrase list. If analog values are not needed, then those phrases can be omitted. The phrase list can contain up to 254 items. These are the phrases you must record later into the dialer board's flash memory.

The table titled "Lines in Report", which is presently blank, will contain the specific lines of your report. Note that the leftmost column of the large report grid is titled "LINE ENBL". This column is used to hold statuses that will enable or disable the speaking of the lines to their right. If a line's "LINE ENBL" cell is left blank, the line will always be spoken in the report. The rest of the columns will contain the speech phrases that define what each line is to say. Those phrases will be dragged from the phrase list to the left and dropped onto the 'Lines in Report' grid. In this manner, up to 14 phrases can be appended on a line to make up a single line of speech.

Speech Report Inputs and Outputs

Just like the software modules in Chapter 5, speech reports have inputs and outputs. The inputs are dropped into the edit boxes at the top of the report panel shown above. Outputs are not shown in the report panel but will appear in the status database. All I/O are defined below:

Inputs That Must be Constants:

- Bd #…integer board number, where dialer board resides in card cage (1-8).

Other Inputs:

- Report Trig…status trigger that triggers presenting entire report, beginning with the report's first line and continuing through the report's last line.
- Report Stop…status trigger that will cause the report to stop at the end of the present line.
- Scan trig…status trigger that will cause the report to scan all statuses in the "Line enable" column (where alarms reside on alarm reports) to determine if there are any new alarms, or any alarms at all.
- Line trig…status trigger that will cause a single line, addressed by the 'Line Select' input below, to be spoken.
- Line select…integer value that selects a single line from the report to be spoken when the 'Line trig' input trigger is issued.

Primary Outputs:

- Done with report…Name.DunTrg…status trigger output indicating that the report is done talking. This is commonly used as an input to the **SpSequenDial** module to signal that a report is done.
- Change trigger…Name.ChgTrg…status trigger output indicating that a new alarm is present in the alarm input list. This will only be issued at the end of a scan triggered by the scan trigger input.
- Have alarm…Name.HavAlrm…status output indicating that one or more alarm inputs are ON. This will only be issued at the end of a scan triggered by the scan trigger input.

Outputs for Internal Use: None

Limitations: None

Naming and Triggering the Report

The "Report Name" field holds the name of the report, which will be used later to reference this report in the list of all reports in the project. The report's name must be typed into the "Report Name" field. To give this report the name "DailyReport", simply type that name into the Report Name field. Triggers to start and stop the report must be dragged in from the status database. For this daily report, the output of an OR gate is used to trigger the report. Drag the DailyRptTrig.OrOut status from the status database, and drop it into the "Report trig" edit box. This will trigger the report whenever the OR gate output is true, which is whenever the sequencer triggers it, or when a key is pressed by the operator.

Adding New Phrases to Phrase List

As you design your report, you will need to add phrases or sentences to the report. You do this by selecting an unused cell in the phrase list on the left side of the speech report design panel, and then typing in the phrase. Don't worry if the phrase is longer than will fit within the phrase list box. If you need to overwrite an existing phrase, simply select that phrase in the phrase list and type in a new phrase, or edit the one that is there.

Entering Report Title Lines

Let's assume our report needs to say the following beginning and ending title lines:

"This is the daily report."
.
.
.
"End of report"

To enter these into the report, first scroll down to a blank cell in the phrase list (cell 35), click on it and type into it "This is the daily report". Similarly, type "End of report" into cell 36. Now, drag the contents of phrase list cell 35 into the PHRASE1 cell of the first line of our report. Similarly, drag the contents of phrase list cell 36 into the PHRASE1 cell of line 5 or our report

Reporting Alarms

Let's add a line to this report to say an alarm if it is present, and say nothing if the alarm is not on. To do that, we drag the alarm status from the status data base into the LINE ENBL cell for the line we wish to enable, and then drag into the phrase 1 cell the speech phrase we wish the alarm to say when it is active. The status in the LINE ENBL field will control whether the phrases on its line will be stated. If we wish the report to state "AC power has failed" when the status "ACFail.DIGIN" is true, then type the phrase "AC power has failed" into the phrase list, and drag it into the phrase 1 cell of line 2 of our report. Then drag the variable "ACFail.DIGIN" from the status data base into the LINE ENBL cell for line 2. (You may have to create the variable first by naming a digital input as "ACFail"). The alarm line will look like line 2 of the panel below.

Reporting Analog Values

In order to report analog values, an introductory phrase should be entered into the phrase 1 cell, followed in the phrase 2 cell by a reference to the data base entry to be reported. That would usually be followed by the units of the value (feet, GPM, etc.) in phrase 3. For example, suppose we wish to report "Tank level is 12.34 feet" in line 3 of our report. We would drag the phrase "Tank level is" from the phrase list to the phrase 1 cell of line 3. Then drag the variable "TankLvl.Out" from the floating point data base to cell 2 of line 3. Finally, drag the phrase "feet" from the phrase list to phrase 3 cell of line 3. The result would look like line 3 below. When the report is executed, line three will report the tank level with a format of the form @@@.@@ by default. If you wish to change the format to something else, simply click on the cell containing the tank level data base entry (TankLvl.Out) and a panel will appear enabling you to edit the @@@.@@ designation, as shown below. You can specify any format using the @@.@@@ method with the format limited to 9 digits before or after the decimal point. In our report, line 3 will be reported each time the report is presented, since there is no status in the LINE ENBL cell for line 3. You should not report more than one analog value per line in order to avoid over running the speech processor's input buffer.

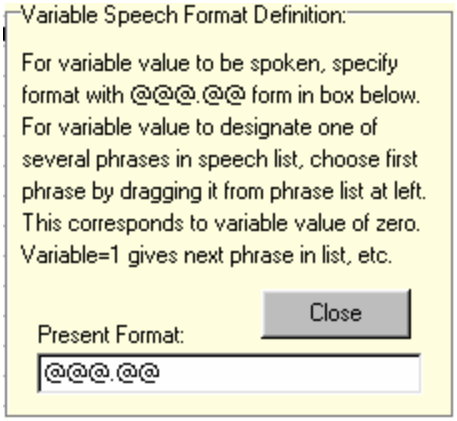


Figure 94 Speech Format Setting Panel for Analog Values

Reporting Sequencer States

The report generator can issue a different phrase for each state of a variable. For example, suppose you have a pump with one of the following five conditions present at any time: OFF/ON/AUTO/FAIL/OUT OF SERVICE. If the variable "Pump1.State" has a value of 0 to 4 corresponding to the five conditions, then you can have the pump's condition stated in a spoken report. First, enter the following phrases into the phrase list in consecutive phrase locations:

Pump 1 operating status is
Off
On
Auto
Failed
Out of service

To report the pump status in the form "Pump 1 operating status is… failed" on line 4 of our report, drag the phrase "Pump 1 operating status is" from the speech phrase list to the phrase 1 cell of line 4 of our report. Then, drag the variable "Pump1.State" from the integer database to the phrase 2 cell of line 4. Now, click on line 4's "Pump1.State" entry in its phrase 2 cell. This will bring up the number format editing panel. This time, instead of editing the @@@.@@ field, simply drag the "Off" word from the phrase list, and drop it onto the @@@.@@ field of the formatting panel. This tells the RUG5/9 that it is to use the value of the variable "Pump1.State" to select a phrase from the phrase list starting with the phrase "Off". If the variable has a value of zero, then "Off" will be reported; if the variable has a value of 1, then "On" will be reported, etc. Note that for this to work properly, the phrases associated with the variable must be in the order corresponding to the value of the integer part of the variable. The report should look like the one below.

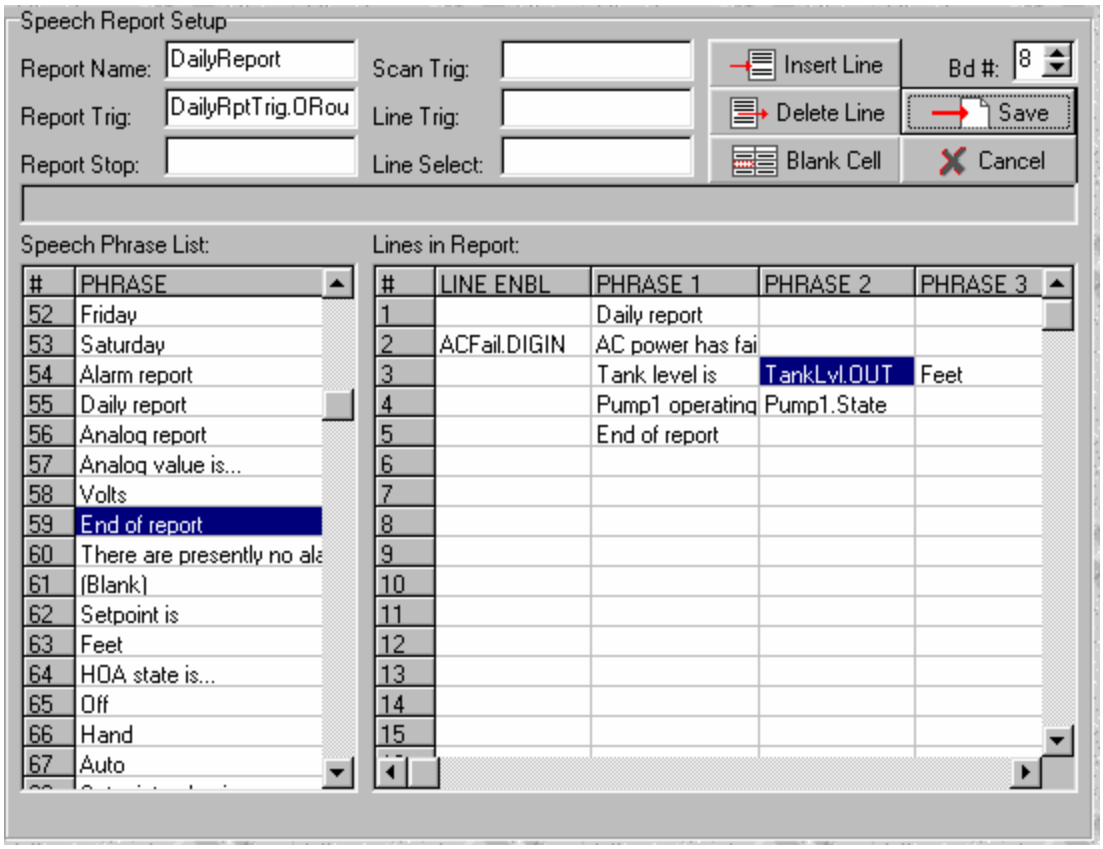


Figure 95 Example Daily Report Setup

Issuing a Single Line of a Report

A report can be made to say its lines one at a time by using the Line Trigger and Line Select inputs on the report definition page above. Whenever a report receives a line trigger input, it will issue the line selected by the Line Select input. This is used by the speech report sequencer to issue prompts to an operator. Typically, the line trigger input is a trigger output from another module, and the line select input is an integer.

Using Speech Report to Detect New Alarms

In most autodialer applications, at least one report will be used to report existing alarms. Some means is needed to detect the occurrence of new alarms to trigger dialing. Since an alarm report will contain references to all its alarms in its LINE ENBL column, the RUG5/9 has been set up to scan the LINE ENBL column to detect any entry transitioning from off to on. It does this by keeping a copy of the last state of its alarms and comparing them against the new states. If it detects an alarm turning on, it will turn on its change trigger output. So, for example, if you have a report titled "AlarmReport", then the status variable named "AlarmReport.ChgTrg" will turn on whenever a new alarm is detected by the report. The report will perform the scan whenever its Alarm Scan Trigger input is true. Therefore, for reports that you wish to provide indication of new alarms, you must drag a status into the Alarm Scan Trigger window at the top of the speech report definition page. This trigger should be either a frequently occurring trigger such as a once per second trigger, or one related to the occurrence of alarms, such as a TriggerOnRcv, which would cause the report to scan the alarms after each reception.

Using Speech Report to Determine if Any Alarm is Present

In a manner similar to the change detection above, the speech report has an output to indicate if any alarm in its list is on. If the report is named "AlarmReport", then the status named "AlarmReport.HavAlrm" will be turned on after an alarm scan whenever any entry in its LINE ENBL column is on. You can use this to terminate dialing if an alarm caused dialing to commence and later turned off before any operator acknowledged the alarm. This enables you to avoid calling an operator in the middle of the night and reporting that there are no alarms. You can also use this to present the sentence "There are presently no alarms" or similar statement to summarize the alarm condition. To do this, add the sentence "There are presently no alarms" to the phrase list, followed by a line that is to remain blank. You might want to type "(blank)" into the phrase list to remind you to leave it blank. Then, at the end of your alarm report, simply drag "AlarmReport.HavAlrm" into the phrase 1 cell. Now, click on that cell to bring up the format panel. Into its editing window, drag the phrase "There are presently no alarms" from the phrase list and close the panel. What this does is to treat the variable "AlarmReport.HavAlrm" as a variable whose state we wish to announce. If it is OFF, the unit will say "There are presently no alarms". If it is ON, the unit will say the next phrase in the list, which you intentionally have left blank.

Detecting End of Report Speaking

Each report has an output trigger that will turn on when the report is done. This works for both single line reports and long multi-line reports. If the report is named "AlarmReport" then the end of report output will be named "AlarmReport.DunTrg". It is typically used to pace the speech report sequencer.

Recording, Playing Back and Deleting Speech Phrases

Speech phrases are recorded by speaking into the microphone on the front of the dialer board at the appropriate time. The **SpeechRecPlayDel** module, in the communications tab of the module library, controls message recording, playback and deleting. As illustrated in the typical setup depicted in the figure below, most of its inputs are usually taken from keys on the keyboard.

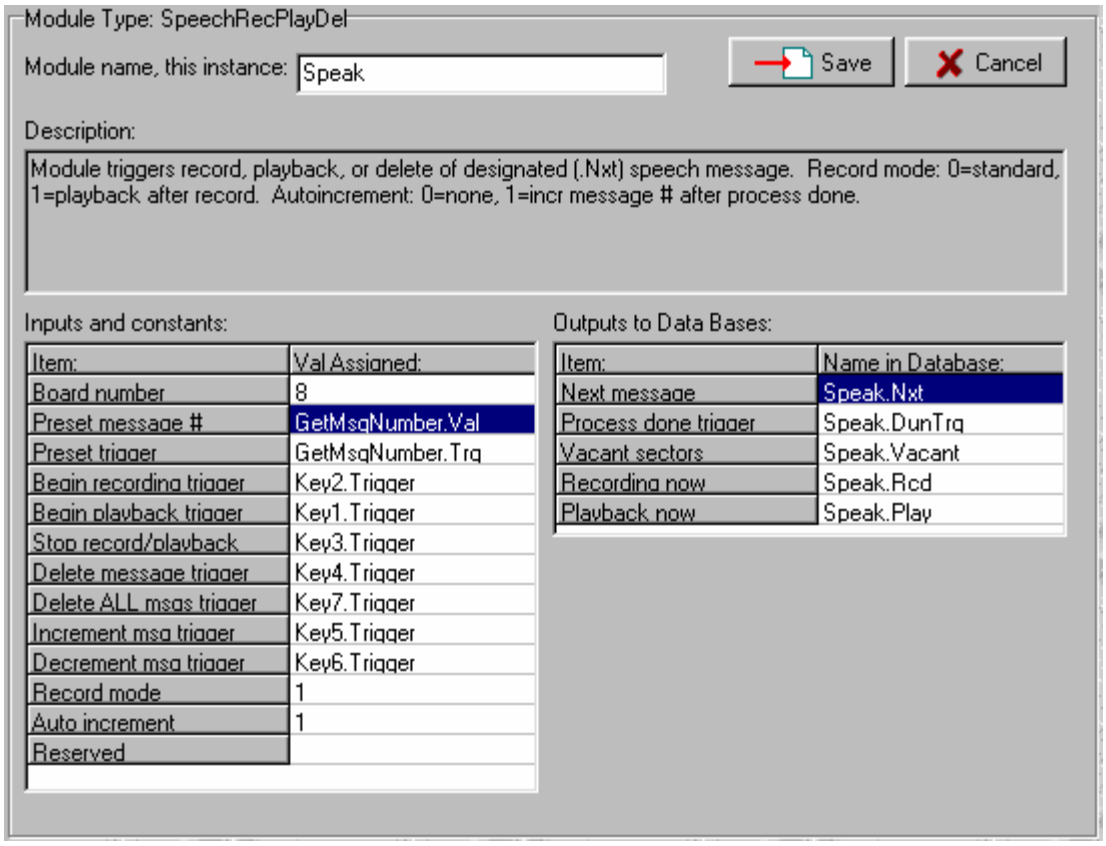


Figure 96 Typical SpeechRecPlayDel Module Setup

A screen to support this process should be built into each autodialer application, although, once the speech phrases are recorded, there is no necessity to keep the record/playback/delete capability in the project. A typical screen to support record/playback/delete is presented below and corresponds to the use of keys in the **SpeechRecPlayDel** module setup above.

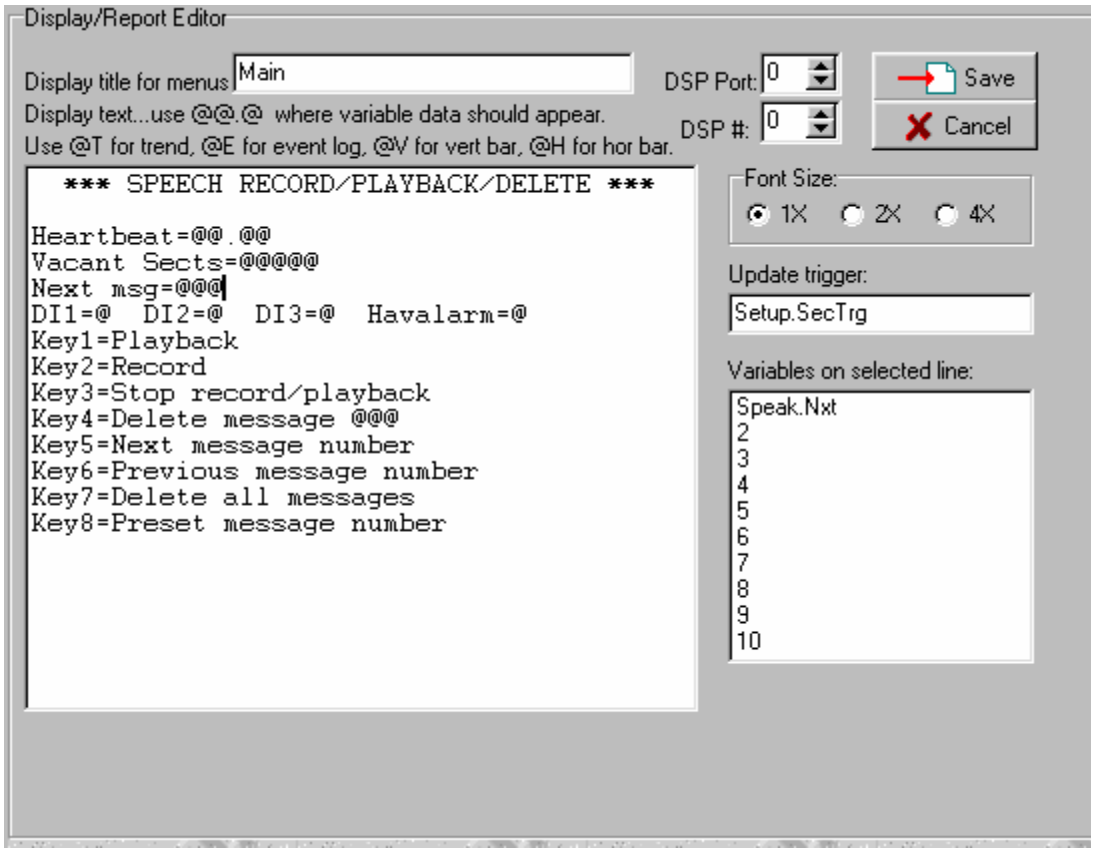


Figure 97 Typical Speech Record/Playback/Delete LCD Screen

Note that the **SpeechRecPlayDel** module maintains an integer value for the next message to be effected. That integer (Speak.Nxt in the example above) can be incremented, decremented or preset to a value. Whenever a command to record, playback, or delete a single message is given, the message addressed by Speak.Nxt will be the one upon which the action is taken. Also, the command to delete all messages is very convenient for starting over, but is also dangerous in that all messages can be erased by accident using it. Finally, note that the display above shows the number of vacant sectors on the speech board. The sector is the minimum erasable chunk of flash memory on the dialer board and can hold 1.42 seconds of speech. Therefore, the board can hold a total of 12 minutes of speech (511 sectors * 1.42 sec.). Each message, no matter how short will use up at least 1.42 seconds of storage. You might want to multiply the vacant sectors by 1.42 to display the seconds of speech remaining unrecorded instead of the number of vacant sectors.

BASIC COMPILER

INTRODUCTION

Beginning with OS revision 4.96 the BASIC compiler is available for use in projects. It enables you to create modules of your own design that you can incorporate into your projects in the same manner that you use the software modules built into the operating system. After you create your module, it becomes part of your BASIC module library and can be used on multiple projects as well as multiple times in a single project. This section describes the method of creating a BASIC module, compiling it, and using it in an example project. In a nutshell, you create your module (e.g., MODULE.BAS) as ASCII text using an editor (either ours or one of you own choice). You then compile your module using our text editor, which, if there are no errors, will create an object file (e.g., MODULE.Obj) that converts your code to a linkable form. You then call your module into a project as you do any other modules. You name your module in the project and drag or type inputs into its input list. When you tell the software to save your module to your project, your module's outputs become entries in the databases just as with other modules. When you compile your project, the compiler will link your BASIC module's object file to the rest of the modules in your project. When you download your project into an RTU, your object code will be downloaded and will be executed each time a module is encountered in the module list referencing your code. The module we've included as an example is named LeadLag4Pump.BAS. The example project is named BAS4PumpTest. Both are included in releases 4.96 and later. The LeadLag4Pump.BAS module is a complete pump up 4 pump controller that is more complete than our leadlag sequencer module in that it includes the equivalent of four pump up controllers, four HOA switches, four pump fail logic blocks with fail delays, backspin/call delay timer, setpoint consistency test and alarm output, and pump lockout and reset logic.

CREATING A MODULE

To create a BASIC module, you must click the BASIC tab in the module library tab set to open the BASIC module library as shown below. Then select one of the sample modules to begin your design. DEMO1.BAS is a minimal template; Info.BAS is a template with a set of comments defining variable naming conventions and characters needed to define storage class. All modules in the list will have the '.BAS' extension to indicate that they are BASIC source files. Other files can exist in the folder, such as the '.Obj' object files produced by the compiler, but they will not be shown.

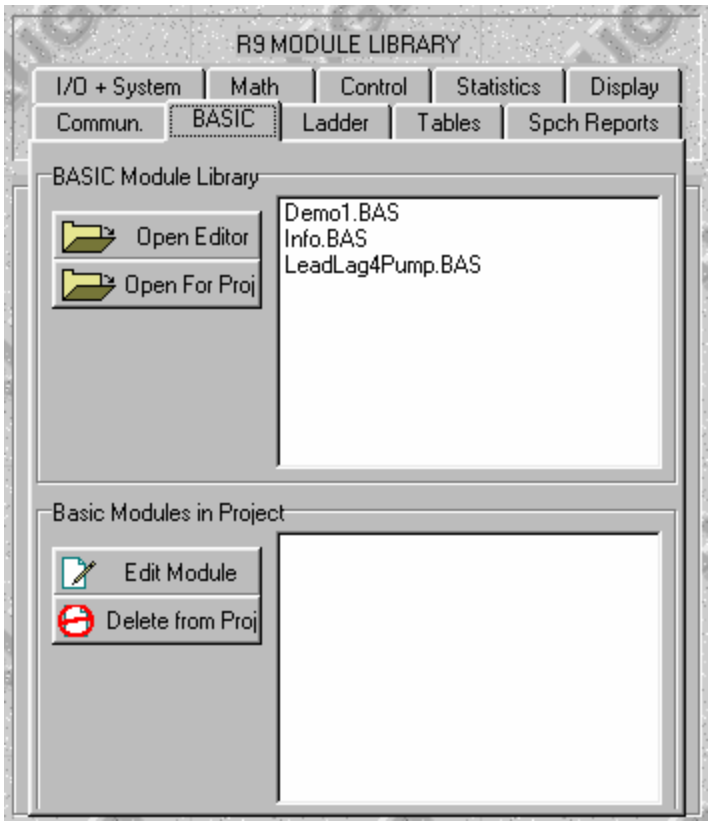


Figure 98 BASIC Module Library Panel

Three BASIC files are indicated as present in the library shown above. They are:

- Demo1.BAS minimal template for defining a module
- Info.BAS more extensive template containing definition of variable types
- LeadLag4Pump.BAS example of a 4 pump lead lag pump up controller written in BASIC.

As you design your own modules, they will appear in the library above for you to use in your projects. To begin creating a module, click on the **Demo1.BAS** module in the upper pane and click on the **Open Editor** button. This will launch an editor that you can use to edit or enter your module's I/O definition and code. The BASIC files in the library are simple ASCII text files so can be created and edited using other programmer's editors. However, you must open your module using our editor in order to compile the module in advance of using it in a project.

Defining Inputs, Outputs, and Comments

Once you open the editor, you will see the following text, which is a minimal template for BASIC programs in the RUG9/RUG5 system:

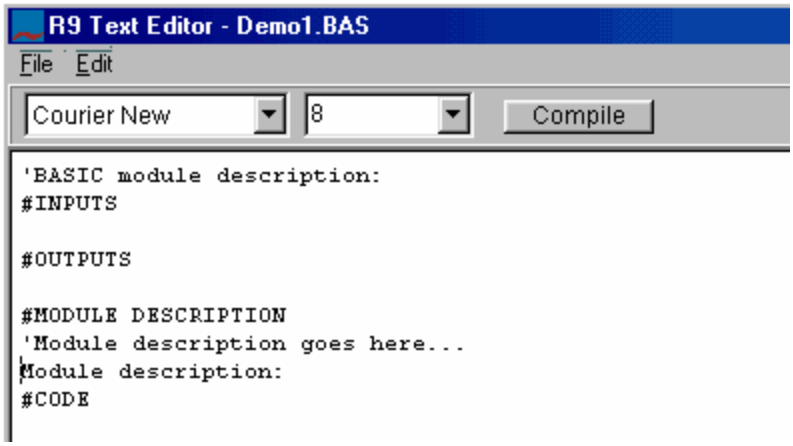


Figure 99 BASIC Module Template

Notice two things here. A few lines begin with a single apostrophe. These are comments and will be ignored by the compiler. A comment can begin anywhere on a line; whenever the compiler encounters an apostrophe, it will ignore the rest of the line. You would be wise to use comments liberally in your code to remind you what that section or line of code is doing. The other item to notice is that there are four lines that begin with the '#' symbol. These are directives to the compiler defining what section of the file follows and they are required. As indicated, the four sections are: #INPUTS, #OUTPUTS, #MODULE DESCRIPTION, and #CODE.

Before writing any code, it is usually most efficient to first define the module's inputs, outputs and associated descriptions. This helps you organize your thinking about the module's functions and establishes the I/O variable names for use in the code. As you write your code, you may easily change, add or delete I/O points later as you discover they are needed simply by editing. Each input point can accept variables from a project's database; and each output point will constitute a global variable in your project's databases. When you go to install your module in a project, the I/O points will appear along with a short description in your module's setup panel.

Inputs are defined in the lines immediately following the #INPUTS directive; outputs are defined immediately following the #OUTPUTS directive. Each input or output must have the following form:

DESCRIPTION:VARIABLE_NAME 'Optional comment

The description can be up to 20 characters and will be used to prompt you or anyone who uses your module as to the intended function of the I/O point. The variable name immediately follows the colon character on the above line and is the name used later in the BASIC code to reference the I/O point. Some typical input definitions are the following:

```
Level/Pressure input:Level 'Input tank level/pressure to control pumps
Pump A call SP:PMPACallSP 'Pump A (1st pump) call setpoint in feet/PSI
Enable lockouts:LockEnable! '0=Resets/disables lockouts, 1=enable lockout on fail
Lead pump:LeadPump% '0=rotate pumps, 1-4 designates lead pump
Pump switch delay:Delay% 'Setpoint sets delay seconds between any successive pump switch
events
```

Some typical outputs are similar:

```
P1Call:P1Call! 'Call to pump 1
P2Call:P2Call! 'Call to pump 2
Fail timer 4:FailTimer4%% 'Pump 4 fail timer
Lead pump:LeadNow% 'Present lead pump
```

When you go to install this module in a project, the module definition panel would appear as shown below:

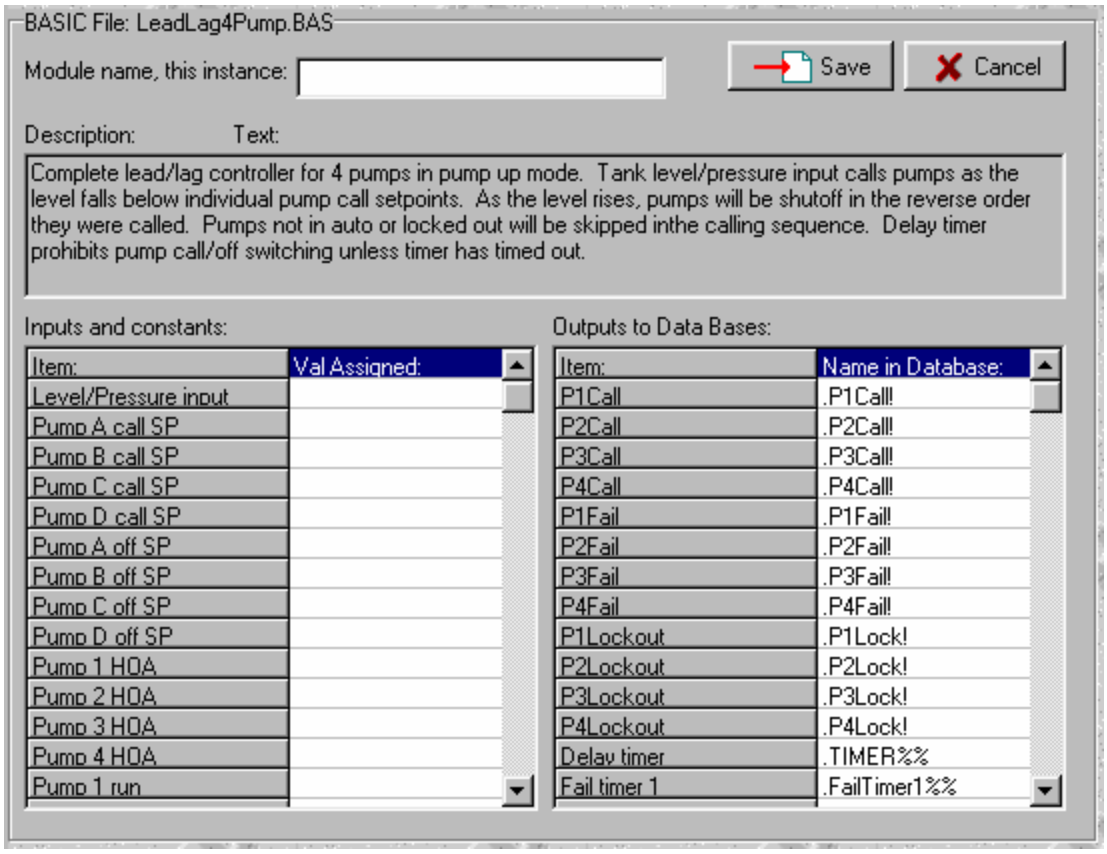


Figure 100 Example Module Definition Panel for BASIC Module

It is important to notice that the description entered to the left of the colon in the module's text file I/O definition becomes the prompt in the panel above for each I/O point. For example, in the I/O definition for the Level/Pressure input in the list above:

```
Level/Pressure input:Level 'Input tank level/pressure to control pumps
```

"Level/Pressure input" becomes the prompt for the first input point in the panel above. You can find others in the panel. The input variable name does not appear in the above panel, but the output variable name is used as the extension to the database item defined by this module. For example, consider the second output for this module as defined in the #OUTPUTS section of the BASIC program:

```
P2Call:P2Call! 'Call to pump 2
```

The description ('P2CALL', part to the left of the colon) appears in the first cell of the output list in the above panel. The variable name ('P2CALL!', part to the right of the colon) is used as the extension of the database entry. If this module were to be named 'CONTROL', then the second output would appear in the database with the name 'CONTROL.P2CALL!'.

Variable Names and Types

If we examine another typical I/O definition line, the character field to the right of the colon constitutes the variable name for that I/O point as it's to be used in the program code to be written. In the following lines:

```
Level/Pressure input:Level 'Input tank level/pressure to control pumps
Enable lockouts:LockEnable! '0=Resets/disables lockouts, 1=enable lockout on fail
```

the word 'Level' is the variable name on the first line, and the word 'LockEnable!' is the variable name for the second line. These variable names must be used by the BASIC code exactly as defined in the I/O lists or the compiler will complain with error messages. The one exception is that the compiler doesn't care whether your variable names and BASIC tokens use upper or lower case characters or some mix. The compiler would regard the name 'Level' as the same as 'LEVEL'. Notice that the 'LockEnable!' name includes an appended exclamation point. This designates the variable as a status, whereas a name with no appended special character will be treated as a floating point value. This is an important distinction. The names 'Level', 'Level%', and 'Level!' would be regarded as different variables. Here are the rules regarding variable names:

- Variable names can be any length, must start with an alpha character and can be composed of alpha, numeric and underscore characters.
- Leading or trailing blanks will be ignored.
- A one or two character extension to the variable name defines storage class:
- Variables without extensions such as ALPHA, X, I, etc. are floating point
- Variables with % after the variable name such as ENABLE%, Y%, etc. are 32 bit integers
- Variables with $ after the variable name such as A$, BB$, NAME$, etc. are strings
- Peculiar to RUG9 BASIC:

-Variables with ! after the variable name such as AA!, ENABLE!, etc. are statuses
-Integers with two % signs are 0.1 second countdown timers, e.g., TIMER1%%, X%%.
When timer outputs are nonzero, background software decrements them every 0.1 sec.
-Statuses with two ! signs are triggers, e.g., Trig1!!, START!!.
Triggers remain TRUE for one program scan, then background software turns them FALSE.

Entering Module Description

If you examine the example module definition panel above, you will notice some descriptive text in a panel labeled 'Description'. That text comes from the '#MODULE DESCRIPTION' section of your program after the input and output definitions and before the #CODE section. It helps you or other users of your module understand the purpose and operation of your module. The following text leads to the description you see in the module definition panel above:

```
#MODULE DESCRIPTION
Complete lead/lag controller for 4 pumps in pump up mode.  Tank level/pressure input
calls pumps
as the level falls below individual pump call setpoints.  As the level rises, pumps will
be shut
off in the reverse order they were called.  Pumps not in auto or locked out will be
skipped in
the calling sequence.  Delay timer prohibits pump call/off switching unless timer has
timed out.
```

Entering Code

Text following the '#CODE' directive is your actual program. It is executed from top to bottom doing calculations and taking branches you specify. You enter the code using our editor or one of your choosing. The details of our implementation of the BASIC language are discussed in the **BASIC LANGUAGE** section below. We'll use the code below to illustrate a few important programming points. Please refer to the code immediately following the #CODE directive in the listing below.

The first line starts with the word 'START'. This is not a variable, it is a label. A label must be alpha or alphanumeric and can only appear at the start of a line. It is usually used as the target of a GOTO or GOSUB statement, as the labels 'CALL' and 'OFF' are near the bottom of the listing. For example, if you want your program to return to the start of this program, you could include the statement 'goto START' , and it would return to the START label.

Following the START label is the assignment statement 'F10%=FAILDELAY%*10'. The variable 'F10%' is not in our I/O list for this module. Instead, it is a temporary local variable that the compiler will create when the code begins executing and then destroy a fraction of a second later when it is

finished with this module. In this program segment, the assignment statement 'F10%=FAILDELAY%*10' calculates the value FAILDELAY%*10 and then assigns it to the variable 'F10%'. It remains there unless changed later for use down in the program, but is invisible outside this program. If you wanted this variable to be usable by other modules, or to retain its value from one program scan to the next, you would need to assign it as an output of this module so the compiler would assign it a permanent location in a database. The colon character separates the two assignment statements on the first line. In general you can have as many statements on a line as you wish; simply separate them with colon characters. The third line of this segment has four assignment statements on the same line separated by colons.

The second line of the example code segment is an 'if' statement. There are several forms of this statement as discussed in detail below. The 'if' statement basically gives the program decision making or branching control. The conditional part of the statement is calculated first (in this case 'ResetAll!>0'). If the conditional part is true, then all the statements following the 'then' token will be executed until the program hits an 'else' token or an 'endif' token. If the conditional part is false, the program will skip the 'then' part and proceed with the statements after the 'endif' token. Finally, notice the 'return' token. The 'return' token tells the program to return from wherever it came from. If it hits the 'return' token without having first executed a gosub statement, it will return from your module. If it executed a 'gosub LABEL' first, then it will return to the first statement following the gosub statement. Here is the entire text file for the four pump, pump up sequencer:

```
#INPUTS
Level/Pressure input:Level 'Input tank level/pressure to control pumps
Pump A call SP:PMPACallSP 'Pump A (1st pump) call setpoint in feet/PSI
Pump B call SP:PMPBCallSP 'Pump B (1st pump) call setpoint in feet/PSI
Pump C call SP:PMPCCallSP 'Pump C (1st pump) call setpoint in feet/PSI
Pump D call SP:PMPDCallSP 'Pump D (1st pump) call setpoint in feet/PSI

Pump A off SP:PMPAOffSP 'Pump A (1st pump) off setpoint in feet/PSI
Pump B off SP:PMPBOffSP 'Pump B (1st pump) off setpoint in feet/PSI
Pump C off SP:PMPCOffSP 'Pump C (1st pump) off setpoint in feet/PSI
Pump D off SP:PMPDOffSP 'Pump D (1st pump) off setpoint in feet/PSI

Pump 1 HOA:P1HOA% 'Pump 1 HOA switch...0=off, 1=ON, 2=AUTO
Pump 2 HOA:P2HOA% 'Pump 2 HOA switch...0=off, 1=ON, 2=AUTO
Pump 3 HOA:P3HOA% 'Pump 3 HOA switch...0=off, 1=ON, 2=AUTO
Pump 4 HOA:P4HOA% 'Pump 4 HOA switch...0=off, 1=ON, 2=AUTO

Pump 1 run:P1Run! 'Pump 1 run contact for fail test
Pump 2 run:P2Run! 'Pump 2 run contact for fail test
Pump 3 run:P3Run! 'Pump 3 run contact for fail test
Pump 4 run:P4Run! 'Pump 4 run contact for fail test

Enable lockouts:LockEnable! '0=Resets/disables lockouts, 1=enable lockout on fail
Lead pump:LeadPump% '0=rotate pumps, 1-4 designates lead pump
Pump switch delay:Delay% 'Setpoint sets delay seconds between any successive pump switch
events
Pump fail delay:FailDelay% 'Setpoint sets delay seconds before pump fail declared
Reset all pumps:ResetAll! 'Trigger to reset pumps on bootup (use System.BootTrg here)

#OUTPUTS
P1Call:P1Call! 'Call to pump 1
P2Call:P2Call! 'Call to pump 2
P3Call:P3Call! 'Call to pump 3
P4Call:P4Call! 'Call to pump 4

P1Fail:P1Fail! 'Pump 1 fail
P2Fail:P2Fail! 'Pump 2 fail
P3Fail:P3Fail! 'Pump 3 fail
P4Fail:P4Fail! 'Pump 4 fail

P1Lockout:P1Lock! 'Pump 1 lockout
P2Lockout:P2Lock! 'Pump 2 lockout
P3Lockout:P3Lock! 'Pump 3 lockout
P4Lockout:P4Lock! 'Pump 4 lockout

Delay timer:TIMER%% 'Delay timer for pump switching
Fail timer 1:FailTimer1%% 'Pump 1 fail timer
```

```
Fail timer 2:FailTimer2%% 'Pump 2 fail timer
Fail timer 3:FailTimer3%% 'Pump 3 fail timer
Fail timer 4:FailTimer4%% 'Pump 4 fail timer
Lead pump:LeadNow% 'Present lead pump
Setpoint error:SP_Error! 'Call/off relationship not consistent with pump up control
Pump calls:Calls% '!!!!

P1 Status String:P1String$[40] 'HAND CALL RUN LOCK status strings
P2 Status String:P2String$[40]
P3 Status String:P3String$[40]
P4 Status String:P4String$[40]
Setpoint error string:SpError$[40]

#MODULE DESCRIPTION
Complete lead/lag controller for 4 pumps in pump up mode.  Tank level/pressure input
calls pumps
as the level falls below individual pump call setpoints.  As the level rises, pumps will
be shut
off in the reverse order they were called.  Pumps not in auto or locked out will be
skipped in
the calling sequence.  Delay timer prohibits pump call/off switching unless timer has
timed out.

#CODE

START F10%=FAILDELAY%*10:D10%=DELAY%*10
  if ResetALL!>0 then 'need to turn off all pumps?
    P1CALL!=0:P2CALL!=0:P3CALL!=0:P4CALL!=0 'turn all pumps off on bootup
    FAILTIMER1%%=F10%:FAILTIMER2%%=F10% 'restart timers just in case
    FAILTIMER3%%=F10%:FAILTIMER4%%=F10%
    TIMER%%=D10%                'delay 1st call by delay SP
    return                           'exit after setup
  endif

'RESET LOCKOUTS...
  if LOCKENABLE!=0 then P1LOCK!=0:P2LOCK!=0:P3LOCK!=0:P4LOCK!=0 'Reset lockouts?

'ALARMS...
  if P1Call!=P1RUN! then P1Fail!=0 'Call & run match?
  else
    if FAILTIMER1%%=0 then  'Pump 1 failed?
      P1FAIL!=1 'Pump 1 failed
      if LOCKENABLE!>0 then P1LOCK!=1 'Set lockout if enabled
    endif
  endif
  if P2Call!=P2RUN! then P2Fail!=0 'Call & run match?
  else
    if FAILTIMER2%%=0 then  'Pump 2 failed?
      P2FAIL!=1 'Pump 2 failed
      if LOCKENABLE!>0 then P2LOCK!=1 'Set lockout if enabled
    endif
  endif
  if P3Call!=P3RUN! then P3Fail!=0 'Call & run match?
  else
    if FAILTIMER3%%=0 then  'Pump 3 failed?
      P3FAIL!=1 'Pump 3 failed
      if LOCKENABLE!>0 then P3LOCK!=1 'Set lockout if enabled
    endif
  endif
  if P4CALL!=P4RUN! then P4Fail!=0 'Call & run match?
  else
    if FAILTIMER4%%=0 then  'Pump 4 failed?
      P4FAIL!=1 'Pump 4 failed
      if LOCKENABLE!>0 then P4LOCK!=1 'Set lockout if enabled
    endif
  endif

'Setpoint reasonableness...
  SP_Error!=0
  if (PMPAOffSP<PMPACallSP) or (PMPACallSP<PMPBOffSP) then SP_Error!=1
  if (PMPBOffSP<PMPBCallSP) or (PMPBCallSP<PMPCOffSP) then SP_Error!=1
```

```
  if (PMPCOffSP<PMPCCallSP) or (PMPCCallSP<PMPDOffSP) then SP_Error!=1
  if (PMPDOffSP<PMPDCallSP) then SP_Error!=1
  if SP_Error!=0 then SpError$="OK         "
  else SpError$="INCONSISTENT"

'PUMP CONTROLS...

'Lead pump rotation if any...
  if LeadPump%>0 then LeadNow%=LeadPump% 'User specifies lead pump
  if (LeadNow%<1) or (LeadNow%>4) then LeadNow%=1 'just in case
  C%=P1CALL!+P2CALL!+P3CALL!+P4CALL! 'number of pumps called now
  if (C%=0) and (Calls%<>0) and (LeadPump%=0) then 'Need to rotate lead pump?
    LeadNow%=LeadNow%+1
    if LeadNow%>4 then LeadNow%=1 'Range limit to 4 pumps
  endif
  Calls%=C% 'Output calls to watch for change to zero calls

'String outputs:
  P1String$="AUTO ":P2String$="AUTO ":P3String$="AUTO ":P4String$="AUTO "
  if P1HOA%=1 then P1String$="HAND " 'HOA states
  if P1HOA%=0 then P1String$="OFF  "
  if P2HOA%=1 then P2String$="HAND "
  if P2HOA%=0 then P2String$="OFF  "
  if P3HOA%=1 then P3String$="HAND "
  if P3HOA%=0 then P3String$="OFF  "
  if P4HOA%=1 then P4String$="HAND "
  if P4HOA%=0 then P4String$="OFF  "
  if P1Call!=0 then P1String$=P1String$+"OFF  " 'Call statuses
  else P1String$=P1String$+"CALL "
  if P2Call!=0 then P2String$=P2String$+"OFF  " 'Call statuses
  else P2String$=P2String$+"CALL "
  if P3Call!=0 then P3String$=P3String$+"OFF  " 'Call statuses
  else P3String$=P3String$+"CALL "
  if P4Call!=0 then P4String$=P4String$+"OFF  " 'Call statuses
  else P4String$=P4String$+"CALL "
  if P1Run!=1 then P1String$=P1String$+"RUN " 'Run statuses
  else P1String$=P1String$+"OFF "
  if P2Run!=1 then P2String$=P2String$+"RUN " 'Run statuses
  else P2String$=P2String$+"OFF "
  if P3Run!=1 then P3String$=P3String$+"RUN " 'Run statuses
  else P3String$=P3String$+"OFF "
  if P4Run!=1 then P4String$=P4String$+"RUN " 'Run statuses
  else P4String$=P4String$+"OFF "
  if P1Lock!=1 then P1String$=P1String$+"LOCKOUT" 'Alarm status
  else
    if P1Fail!=1 then P1String$=P1String$+"FAIL   "
    else P1String$=P1String$+"OK     "
  endif
  if P2Lock!=1 then P2String$=P2String$+"LOCKOUT" 'Alarm status
  else
    if P2Fail!=1 then P2String$=P2String$+"FAIL   "
    else P2String$=P2String$+"OK     "
  endif
  if P3Lock!=1 then P3String$=P3String$+"LOCKOUT" 'Alarm status
  else
    if P3Fail!=1 then P3String$=P3String$+"FAIL   "
    else P3String$=P3String$+"OK     "
  endif
  if P4Lock!=1 then P4String$=P4String$+"LOCKOUT" 'Alarm status
  else
    if P4Fail!=1 then P4String$=P4String$+"FAIL   "
    else P4String$=P4String$+"OK     "
  endif

'See if timer ready for another control...

  if (TIMER%%<0) or (TIMER%%>D10%) then TIMER%%=0 'range check timer
  if TIMER%%>0 then return 'no controls if not timed out

'Turn off pumps that have lockouts...
  if (P1CALL!=1) and (P1LOCK!=1) then P1CALL!=0:TIMER%%=D10%:FAILTIMER1%%=F10%:return
```

```
  if (P2CALL!=1) and (P2LOCK!=1) then P2CALL!=0:TIMER%%=D10%:FAILTIMER2%%=F10%:return
  if (P3CALL!=1) and (P3LOCK!=1) then P3CALL!=0:TIMER%%=D10%:FAILTIMER3%%=F10%:return
  if (P4CALL!=1) and (P4LOCK!=1) then P4CALL!=0:TIMER%%=D10%:FAILTIMER4%%=F10%:return

'Handle HOA's...
  if (P1HOA%<2) and (P1CALL!<>P1HOA%) then
P1CALL!=P1HOA%:TIMER%%=D10%:FAILTIMER1%%=F10%:return 'P1?
  if (P2HOA%<2) and (P2CALL!<>P2HOA%) then
P2CALL!=P2HOA%:TIMER%%=D10%:FAILTIMER2%%=F10%:return 'P2?
  if (P3HOA%<2) and (P3CALL!<>P3HOA%) then
P3CALL!=P3HOA%:TIMER%%=D10%:FAILTIMER3%%=F10%:return 'P3?
  if (P4HOA%<2) and (P4CALL!<>P4HOA%) then
P4CALL!=P4HOA%:TIMER%%=D10%:FAILTIMER4%%=F10%:return 'P4?

'See if we need to call another pump..
  if (C%=0) and (Level<PMPACallSP) then gosub CALL:return
  if (C%=1) and (Level<PMPBCallSP) then gosub CALL:return
  if (C%=2) and (Level<PMPCCallSP) then gosub CALL:return
  if (C%=3) and (Level<PMPDCallSP) then gosub CALL:return

'See if we need to turn a pump off...
  if (C%=4) and (Level>PMPDOffSP) then gosub OFF:return
  if (C%=3) and (Level>PMPCOffSP) then gosub OFF:return
  if (C%=2) and (Level>PMPBOffSP) then gosub OFF:return
  if (C%=1) and (Level>PMPAOffSP) then gosub OFF:return
  return 'end of program

CALL if (LeadNow%<1) or (LeadNow%>4) then LeadNow%=1 'just in case
  if LeadNow%=1 then goto LEAD1
  if LeadNow%=2 then goto LEAD2
  if LeadNow%=3 then goto LEAD3
  if LeadNow%=4 then goto LEAD4
  return 'error
LEAD1 if (P1HOA%=2) and (P1Call!=0) and (P1Lock!=0) then
P1Call!=1:TIMER%%=D10%:FAILTIMER1%%=F10%:return
LEAD2 if (P2HOA%=2) and (P2Call!=0) and (P2Lock!=0) then
P2Call!=1:TIMER%%=D10%:FAILTIMER2%%=F10%:return
LEAD3 if (P3HOA%=2) and (P3Call!=0) and (P3Lock!=0) then
P3Call!=1:TIMER%%=D10%:FAILTIMER3%%=F10%:return
LEAD4 if (P4HOA%=2) and (P4Call!=0) and (P4Lock!=0) then
P4Call!=1:TIMER%%=D10%:FAILTIMER4%%=F10%:return
      if (P1HOA%=2) and (P1Call!=0) and (P1Lock!=0) then
P1Call!=1:TIMER%%=D10%:FAILTIMER1%%=F10%:return
      if (P2HOA%=2) and (P2Call!=0) and (P2Lock!=0) then
P2Call!=1:TIMER%%=D10%:FAILTIMER2%%=F10%:return
      if (P3HOA%=2) and (P3Call!=0) and (P3Lock!=0) then
P3Call!=1:TIMER%%=D10%:FAILTIMER3%%=F10%:return
  return 'leave if none can be turned on

OFF if LeadNow%=1 then goto OFF4 'start turning off from bottom
  if LeadNow%=2 then goto OFF1
  if LeadNow%=3 then goto OFF2
  if LeadNow%=4 then goto OFF3
  return 'error
OFF4 if (P4HOA%=2) and (P4Call!>0) then P4Call!=0:TIMER%%=D10%:FAILTIMER4%%=F10%:return
'turn pump off if auto and on
OFF3 if (P3HOA%=2) and (P3Call!>0) then P3Call!=0:TIMER%%=D10%:FAILTIMER3%%=F10%:return
OFF2 if (P2HOA%=2) and (P2Call!>0) then P2Call!=0:TIMER%%=D10%:FAILTIMER2%%=F10%:return
OFF1 if (P1HOA%=2) and (P1Call!>0) then P1Call!=0:TIMER%%=D10%:FAILTIMER1%%=F10%:return
     if (P4HOA%=2) and (P4Call!>0) then P4Call!=0:TIMER%%=D10%:FAILTIMER4%%=F10%:return
     if (P3HOA%=2) and (P3Call!>0) then P3Call!=0:TIMER%%=D10%:FAILTIMER3%%=F10%:return
     if (P2HOA%=2) and (P2Call!>0) then P2Call!=0:TIMER%%=D10%:FAILTIMER2%%=F10%:return
  return 'exit if none can be turned off
```

Compiling

After you have entered your program, click the 'File' menu item at the top of the editor page and select either 'Save as' if you wish to save under a different name, or hit 'Save' to save with the same name. Then hit the 'Compile' button to compile the program. The entire program will become highlighted while it is compiling. When done, the program will return to normal appearance. If there are no errors, the status panel at the bottom of the editor form will say 'There are no errors'. If an error is found, as in the panel

below, the status panel at the bottom will indicate that there are one or more errors, and the compiler will highlight in red, any lines containing errors.

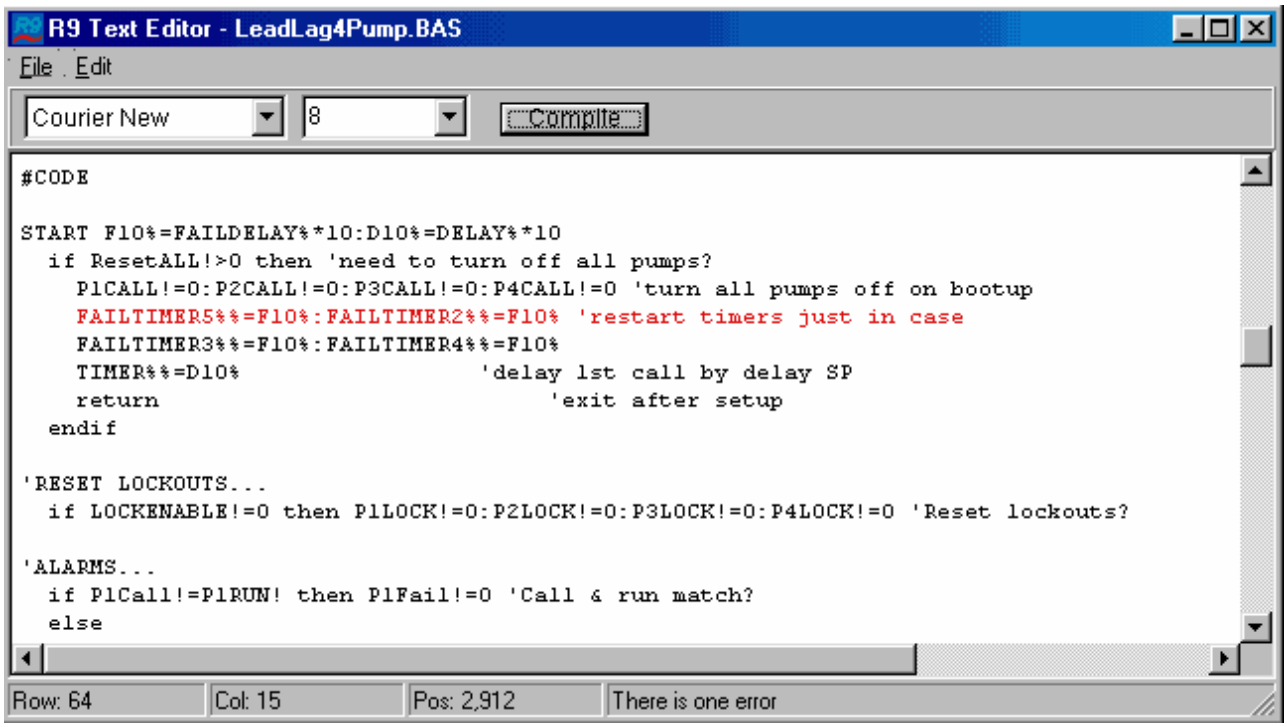


Figure 101 Compiler Result With an Error

Note that the fourth line of the program references a variable 'FAILTIMER5%%' which has not been assigned as an input or output. That by itself is not an error…the compiler simply regards it as a local variable. The error is that the local variable 'FAILTIMER5%%' is not used anywhere else in the program. If you click on any red line in the program, the status bar will tell you what error exists on that line. Clicking on black lines will give the indication 'OK' on the status bar, indicating that the line has no detected error. You would then edit the red lines to correct the errors and recompile. If the compile is successful with no errors, the status bar will indicate that there are no errors and there will be no red lines in the text. In that case the compiler will generate the 'FileName.Obj' file that can then be used in a project.

Debugging

Once your module compiles you must then include it in a project, download it to a RUG9 or RUG5, and observe its operation. If it does not appear to execute your code as you expect, you can observe individual outputs using the unit's display or watch window as with any other module. Sometimes, the module does not have enough of the correct outputs to enable you to observe execution within the module. In that case, you should install some temporary outputs placed in strategic locations in the code to enable you to ascertain the problem, again, either by displaying the temporary outputs on a display, or using the watch window.

Using a Module in a Project

You install your module in a project the same way you use any of the modules that are part of the operating system except that you obtain them from the BASIC tab in the module library panel. When you click the BASIC tab, you will see the following panel:

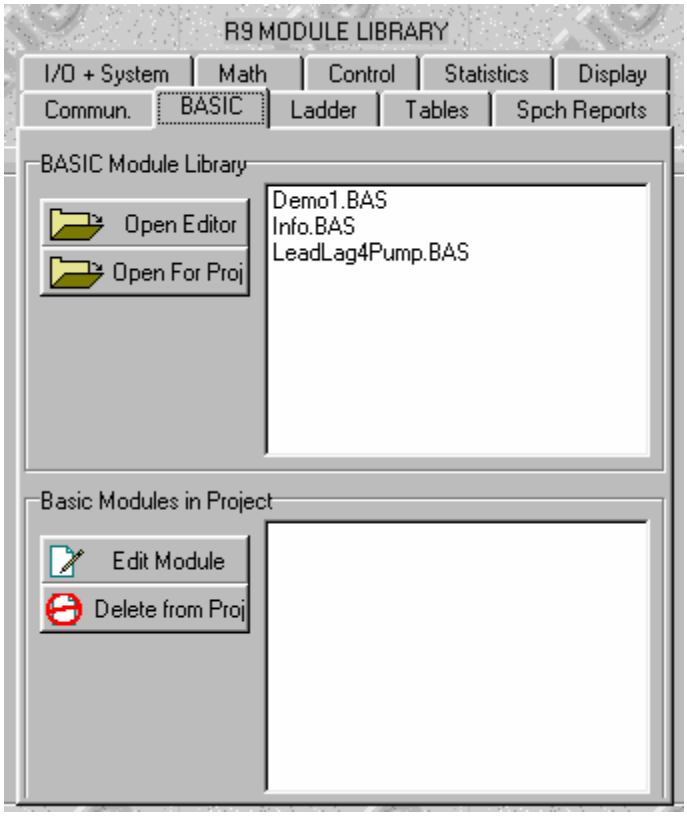


Figure 102 Basic Module Selection for Inclusion in Project

To bring the sample module 'LeadLag4Pump.BAS' into your project, select that module from the BASIC module library list and click the 'Open For Proj' button. The system will then open a new instance of the selected module so that you can name it and drag in or type in its inputs. The completed module in the example project is presented below.

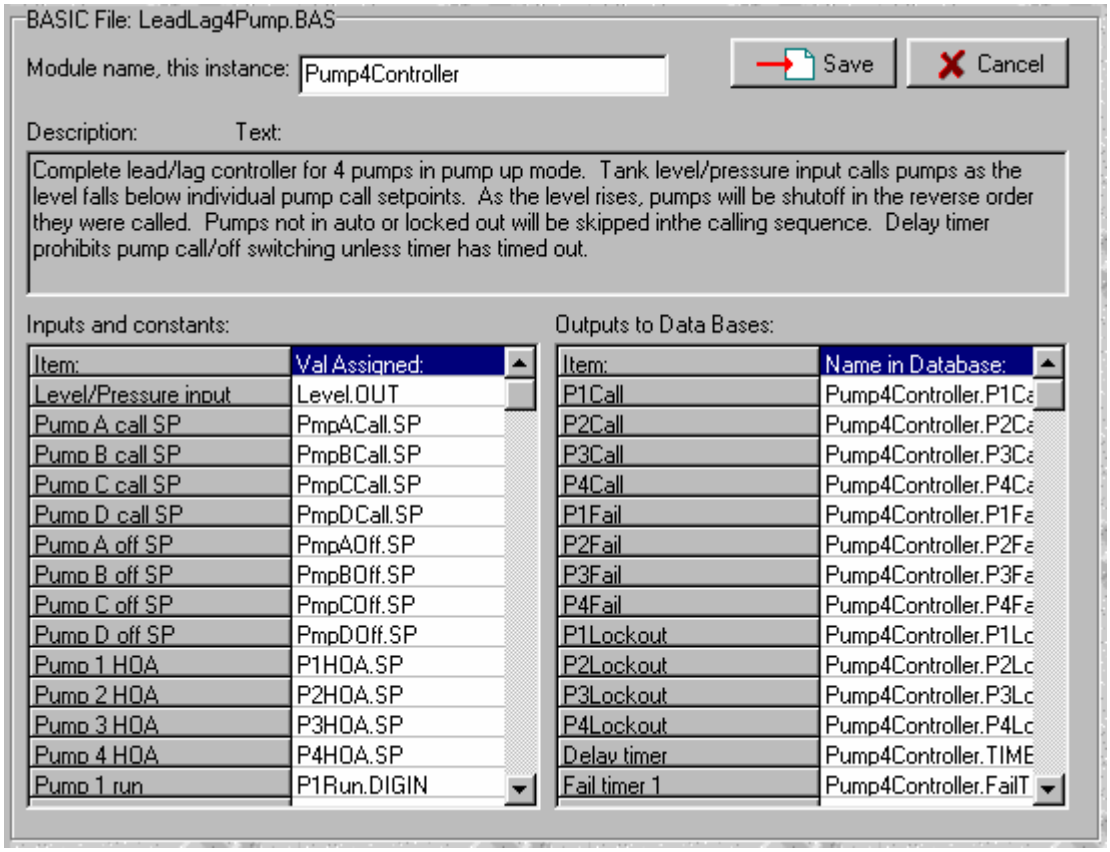


Figure 103 Example Module to be Installed in Project

Note that we have dragged in all inputs to the module from the databases. Note also that we have named the module 'Pump4Controller', so all the module's outputs have been given that name followed by the variable name we gave each output in the #OUTPUTS section of our code. If we wish to edit the inputs to the module after we have installed it, then we simply click on the module's name in the 'BASIC Modules in Project' panel above and click on 'Edit Module'. Similarly, to delete the module from the project, simply select the module's name in the 'BASIC Modules in Project' panel and then click on the 'Delete from Proj' button.

Distributing Projects With BASIC Modules in Them

In order to compile and download a project to a RUG9 or RUG5, the system needs the project file and all the BASIC source and/or object files used in the project. Project files need to be installed in the 'R9Projct' folder; and BASIC files in the R9BASIC folder. Files that must be included are one of two combinations as follows:

- Project source file plus the complete BASIC source files. BASIC modules would then be compiled by user to produce object files. This is recommended to be compatible with RUGID's open source policy for application programs.
- Project source file, plus BASIC object files, plus BASIC source files down to #CODE section. In this case, BASIC will not compile, but will install using delivered object files.

BASIC LANGUAGE

We chose to include BASIC to give the RUG9 and RUG5 units some general purpose procedural programming capability. We could have chosen C, Java, Pascal, Forth, or some other language, but BASIC

supports existing users who are familiar with our earlier RUG2 through RUG8 models which were programmable in BASIC. In addition, BASIC is easy to learn for non-programmers and has fairly forgiving syntax. This section presents descriptions of the BASIC language elements included in our compiler. As of this writing, the BASIC is incomplete in that it does not yet support indexed variables and arrays, the For/Next loop structure, or strings. However, as you should be able to observe from the included example lead lag sequencer, it is quite functional for even involved control strategies.

Language Elements

This section lists all BASIC language elements included in the RUG9/RUG5 support software as of this writing. Examples for most elements are provided. Elements familiar to you but not listed here will probably result in compiler errors and should not be used. All elements are presented in upper case for emphasis, but can be used as either upper or lower case.

Arithmetic Precedence

BASIC provides a complete mathematics package. The following operators support both floating point and integer arithmetic:

Symbol	Use
+	Addition
-	Subtraction, negation
*	Multiplication
/	Division
=	Assigns a value to a variable

The above operators can be used to perform mathematical computations and will be executed in the order of precedence beginning with operations of highest precedence and working down to operations of lowest precedence, as listed below. This means that division and multiplication will be performed before addition and subtraction. For example, 17 + 21 / 7 equals 20, not 5.43… When the compiler encounters operations of equal precedence, they execute from left to right. Therefore, 5 – 7 + 8 equals 6, not –10. You can use parentheses to explicitly specify the order of calculation if you wish. For example, (17 + 20) / 7 equals 5.43… As the compiler scans each expression it will decide whether the expression is to be computed as floating point or integer. If all elements are integer, then integer math will be used. However, if any element in the expression is floating point, then all integer elements will be converted to floating point, and the entire expression will be calculated using floating point math. The following list begins with operations of highest precedence and ends with those with lowest precedence. Operations on the same line have equal precedence.

Precedence List

1) () Expressions in parentheses are evaluated first
2) Negation –Z, where Z can be an expression
3) * and / multiplication and division
4) + and – addition and subtraction
5) relational:
   All below have equal precedence:
   = equal
   <> not equal
   < less than
   > greater than
   =< or <= less than or equal to
   => or >= greater than or equal to

6) NOT logical and bitwise NOT as in negation
7) AND logical and bitwise AND
8) OR logical and bitwise OR

Arithmetic Functions

ABS Computes the absolute value of an expression
Example:
X1=abs(Y)

COS, SIN, TAN
Computes the cosine, sine and tangent of an expression, where the expression is in radians.
Example:
X1=cos(Y)
X2=sin(Y)
X3=tan(Y)

ARCCOS, ARCSIN, ARCTAN
Computes the angle in radians whose cosine, sine or tangent are given by the expression.
Example:
Angle1=arccos(X1)
Angle2=arcsin(X2)
Angle3=arctan(X3)

EXP Computes the exponential of the expression, i.e., 2.71828… to the power of the expression.
Example:
X=exp(Y)

INT Returns the integer value of the expression without rounding.
Example:
X%=int(Y)

LN Returns the natural log of the expression.
Example:
X=ln(Y)

LOG10 Returns the base 10 log of the expression
Example:
X=log10(Y)

RND Returns a random number in the range of 0 to 1.0
Example:
X=rnd(0) ‘ the zero argument is ignored

SQR Returns the square root of the argument
Example:
X=sqr(Y) ‘ Y cannot be negative

Logical Operators

AND Performs relational as well as bitwise logical AND operations.
Example relational AND:
If (X>3) and (Y<1) then…

Example logical AND:
Y%=Integer1% and Integer2%

OR Performs relational as well as bitwise logical OR operations.
Example relational OR:

```
        If (X>3) or (Y<1) then…
```

Example logical OR:

```
        Y%=Integer1% or Integer2%
```

NOT Used to invert a logical expression.

```
Example: if not X% then… 'negate logic for IF test
        X%=not Y%       'if Y% true, then X% false
```

Program Execution Control and Branching

Since the program is constitutes one of many modules in a typical RUG9/RUG5 project, it will be invoked several times per second, so no command such as RUN is needed to initiate its execution.

- GOTO LABEL

  Sends execution to the label that follows the GOTO token. Since the compiler supports the IF…THEN…ELSE construct, the use of GOTO is generally unnecessary.

  Example:

```
            If X>3 then Y=X*5:goto REST
            Y=X*7
            J=3
    REST    Z%=7
```

- GOSUB LABEL

  Sends execution to the label that follows the GOSUB token. After executing the code at that location, execution returns to the location following the GOSUB LABEL statement. Use this to call a subroutine that needs to be called from several places in your code and you wish not to write it more than one time. You can also call a subroutine from within a subroutine. In the example below, the main program calls the subroutine labeled REST then returns to the statement Z=Y+9 after executing the subroutine.

  Example:

```
        .
        .
        X=5:Y=6+X:gosub REST:Z=Y+9   'Main program
        .
        .
REST if Y>6 then X=sqr(Y)       'Body of subroutine, line 1
        else X=Y*Y              'Body of subroutine, line 2
        Return                  'Return from subroutine
```

- END Stops the program and returns to the rest of the executing system. RETURN does the same thing.

- IF…THEN…ELSE…ENDIF

  Branching control that branches depending on the result of the test of an expression. This can have one of six forms:

  Simple one line IF. If the test expression is true, then the Y=5 is executed. Otherwise, execution proceeds to the next line.

```
        Example: if X>2 then Y=5 'all on one line, no else part
```

Simple one line IF followed by one line ELSE. If the test expression is true, then the THEN part is executed and the ELSE part is skipped. If the expression is false, then only the ELSE part is executed.

```
Example: if X>2 then Y=5
         Else Y=6
         X=3…
```

Block THEN, no else part. If the test expression is true, then the THEN block is executed, otherwise execution proceeds with the program after the ENDIF token. The ENDIF token is necessary to mark the end of the THEN part.

```
Example: if X>2 then      'no statements after the THEN on this line
         Y=6              '1st line of THEN block
         Z=3              '2nd line of THEN block
         Endif            'end of THEN block
         X=4              'where execution goes if X>2 is false
```

Block THEN part, single line ELSE part. If the test expression is true, then the THEN block is executed, otherwise execution proceeds with the ELSE part. No ENDIF is needed since ELSE marks the end of the THEN block.

```
Example: if X>2 then      'no statements after the THEN on this line
         Y=6              '1st line of THEN block
         Z=3              '2nd line of THEN block
         Else Y=3         'Where execution goes if X>2 is false
         X=4              'where execution goes after THEN block or ELSE line
```

Single line THEN part, block ELSE part. If the test expression is true, then the THEN line is executed, otherwise execution proceeds with the ELSE block. An ENDIF is required to mark the end of the ELSE block.

```
Example: if X>2 then Y=3      'Single line THEN part
         Else                 'Start of ELSE block
         Y=4                  '1st line of ELSE block
         Z=3                  '2nd line of ELSE block
         Endif                'marks end of ELSE block
         X=4                  'where execution goes after either THEN line or
ELSE block
```

Block THEN part, block ELSE part. If the test expression is true, then the THEN block is executed, otherwise, the ELSE block is executed. An ENDIF is required to mark the end of the ELSE part.

```
Example: if X>2 then
         Y=3                  '1st line of THEN block
         Z=4                  '2nd line of THEN block
         Else                 'Start of ELSE block
         Y=4                  '1st line of ELSE block
         Z=3                  '2nd line of ELSE block
         Endif                'marks end of ELSE block
         X=4                  'where execution goes after either THEN block or
ELSE block
```

Block THEN, no else part. If the test expression is true, then the THEN block is executed, otherwise execution proceeds with the program after the ENDIF token. The ENDIF token is necessary to mark the end of the THEN part.

```
Example: if X>2 then      'no statements after the THEN on this line
         Y=6              '1st line of THEN block
         Z=3              '2nd line of THEN block
         Endif            'end of THEN block
         X=4              'where execution goes if X>2 is false
```

Strings

String functions enable you to work with ASCII text so that you can create messages under program control, pick strings apart to isolate needed substrings, convert from numeric to string and string to numeric, and other functions involving strings. A string is a collection of characters enclosed in double quotes. For example, "Hello", "Tuesday", and "This is a string!" are strings. A string can only be assigned to a string variable; and string variables can only accept data in string form. A string variable is any variable name with an appended dollar sign ($). Thus, A$, Input$, and Array$[3] are all string variables. Maximum allowed string length is 255 characters. Note that functions that have dollar signs ($) after them such as MID$ and CHR$ return strings. Functions without appended dollar signs such as LEN and VAL return numeric values.

- **ASSIGNMENT**

  To assign a string to a variable, simply set the variable equal to the string. For example, the following are valid assignments:

```
A$="Tank level"      'OK
Boy$=" Fred"         'OK
```

  These are invalid and will result in error messages:

```
A="Tank level"       'invalid…A is not a string variable
Boy$=Fred            'invalid…Fred must be enclosed in double quotes
```

- **CONCATENATION, +**

  You can combine strings to create new strings by using the plus (+) token to concatenate them. The following are valid concatenations:

```
C$="String1"+"String2"       'Result: C$="String1String2"
D$=A$+" is too high"         'Result: D$="Tank level is too high"
E5$=D$+Boy$+"…Bye"           'Result: E5$="Tank level is too high Fred…Bye"
```

- **ASC(string)**

  The ASC function returns the ASCII equivalent of the first character of the string argument. For example, the following statement will set X% to the ASCII equivalent for the letter "B", which is 66:

```
X%=asc("Bellingham")         'Result: X%=66
```

- **CHR$(value)**

  The CHR$ function returns the one character string equivalent of the numeric value in its argument. The expression in parentheses must evaluate to a value from 0 through 255. This function is used mostly to include control codes in strings to be sent to a port. In the first example below, the variable C3$ is set to a one character string equal to the ASCII code for a carriage return.

```
C3$=chr$(13)                    'Carriage return
Bell$=chr$(7)                   'Bell character
```

- **LEFT$(string,length)**
  LEFT$ returns the left most characters of the string argument. The number of characters is specified by the length numeric value. Examples:

```
A$=left$("RUGID Computer",7)    'Result: A$="RUGID C"
B$=left$("software program",4)  'Result: B$="soft"
```

- **LEN(string)**
  The LEN function returns the number of characters in the string argument. It is useful for testing that a string has some characters before proceeding with further processing. Examples:

```
X%=len("RUGID Computer")        'Result: X%=14
Y%=len(B$)                      'if B$="soft" then result: Y%=4
```

- **MID$(string, start, length)**
  The MID$ function enables you to pick a string from the middle of another string. It has three arguments: the string which is to be picked apart; the start location within the string where the first character is character #1; and the length or number of characters that are to be extracted. Examples:

```
A$=mid$("RUGID Computer",7,5)   'Result: A$="Compu"
B$=mid$(A$,2,3)                 'Result: B$="omp"
```

- **RIGHT$(string,length)**
  The RIGHT$ function returns the rightmost characters from the argument string, where the number of characters to be returned is specified by the length argument. Examples:

```
C$=right$("RUGID Computer",8)   'Result: C$="Computer"
D$=right$(C$,5)                 'Result: D$="puter"
```

- **STR$(numeric expression)**
  STR$ converts the numeric value in its argument to a string. Example:

```
Xfloat=34.567
A$=str$(Xfloat)                 'Result: A$="34.567"
```

- **VAL**
  The VAL function returns the value of the string in its argument. This is used to convert an incoming string that contains a value from an ASCII output instrument to a numeric form that can be used in calculations. Example:

```
A$="95.345"
Xfloat=val(A$)                  'Result: Xfloat=95.345, floating point
```

Error Messages

During execution in the RUG9/RUG5, there are no runtime error messages provided; the program simply exits the BASIC code and proceeds with executing the rest of its modules. During compilation, however, extensive error detection is provided. One of these messages will be shown on the status line of the BASIC editor after you click on a red line of text having an error.

ERROR MESSAGE **MEANING**

- **Cannot find ENDIF destination**…The compiler cannot unambiguously identify the end of an IF..THEN…ELSE statement series. Even if in the case of an If without an ELSE, the compiler must identify a place for control to continue if the test expression is false. Possibly an ENDIF is missing.

- **Cannot find label…LABEL** …A destination label referenced by a GOSUB or GOTO cannot be found.

- **Comparing numeric to string** …a string and a numeric variable or expression cannot be compared in an IF statement. Either compare two strings or compare two numeric values.

- **Duplicate label...LABEL** …The named label appears more than once in the program. Labels must be unique.

- **Duplicate variable name...VARNAME** The variable name for an I/O point has already been used.

- **ELSE must be first token on line…**The ELSE token must start a line, it cannot appear to the right of any other tokens.

- **Error in variable name...VARNAME** Variable name must be alphanumeric optionally followed by one or two storage class identifiers (%,!). The variable noted probably contains some other character.

- **Expect THEN token after test expression**…The test expression of an IF statement is not followed by a THEN token.

- **Expected = in assignment statement**…An expression that starts with a variable name does not have an '=' token. The compiler may be misinterpreting the preceding field as a variable name.

- **Expected comma between function arguments** …Function arguments must separated using commas.

- **Expected end of dimension**…some other character than a closing parenthesis, bracket or comma follows a dimension.

- **Invalid expression**… syntax error or unrecognized token in expression. Possible misspelling.

- **Invalid label for GOSUB...LABEL**…The target destination name of a GOSUB statement is invalid or cannot be found in the program.

- **Invalid label for GOTO...LABEL**… The target destination name of a GOTO statement is invalid or cannot be found in the program.

- **Invalid math operation on string**… You cannot mix strings and numeric values in an expression. Use the VAL function to convert strings to numeric values before use in a math expression; or use the STR$ function to convert numeric values to strings before use in a string expression.

Displays, Reports, Tables, Ladder

- **Invalid operation on string**…The only operator that can be used on a string is the concatenation operator (+).

- **Invalid argument type for function +OpCode**…Numeric functions cannot have string arguments, you must convert first using the VAL function. String functions must have strings as their first argument, and numeric arguments for subsequent arguments.

- **Invalid section specifier** Allowed sections: #INPUTS, #OUTPUTS, #MODULE DESCRIPTION, or #CODE. The compiler encountered a # symbol followed by some other word.

- **Local variable..VARNAME..never assigned value** …The named local variable is defined but is not assigned a value.

- **Local variable..VARNAME..not used in expression** …the named local variable is defined but is never used in an expression.

- **Missing ELSE destination**…The compiler cannot unambiguously identify the ELSE portion of an IF..THEN…ELSE statement series. Even if in the case of an If without an ELSE, the compiler must identify a place for control to continue if the test expression is false. Possibly an ENDIF is missing.

- **Missing parenthesis above**… One or more parentheses is missing, possibly in a statement above.

- **Missing parenthesis before assignment statement** … One or more parentheses is missing, possibly in a statement above.

- **Must have string argument for '+OpCode** … String functions must have strings as their first argument, and numeric arguments for subsequent arguments.

- **Output variable..VARNAME..never assigned value** …The named output variable is not assigned a value in the program.

- **String variable missing dimension** …A dimension in brackets defining the length of each string is missing.

- **Type conversion error in assignment statement** …Compiler cannot determine unambiguously what type of calculation class (floating or integer) to use for expression.

- **Type mismatch in expression: string and numeric** …You cannot mix strings and numeric values in an expression. Use the VAL function to convert strings to numeric values before use in a math expression; or use the STR$ function to convert numeric values to strings before use in a string expression.

- **Unrecognizable statement initiator**…A statement starts with an unrecognized token.

- **Variable def not of form>> Description:Variable name** Variables in the #INPUT or #OUTPUT sections must consist of a description followed by a colon, followed by the variable name.

- **Variable dimension must be numeric**…The dimensions of an array must be numeric.

- **Wrong argument type for +OpCode** …Function argument is of wrong type, string vs numeric

CHAPTER 7…COMMUNICATIONS

INTRODUCTION

The RUG5/9 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. RUG5/9 communications hardware and software are compatible with both radio and phone line channels; serial channels such as RS232 and RS485; leased line, customer-owned lines, cell phones, or dial up phone lines; and virtually all types of radios including audio types, radios with built in modems, and spread spectrum radios. R9SETUPD enables you to define telemetry formats as 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 RUG5/9, 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 RUG5/9 hardware. Finally, choose the software modules and array setups that will accomplish the required communications over the channel.

Sample Applications

RUG5/9 Protocol Basics

Normally, for unit to unit communications, you will use the RUG5/9 protocol. It is presented in detail later in this chapter, but some explanation is necessary before we proceed with the following sections. The RUG5/9 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 is 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 RUG5/9 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 the 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 and the RTU's must repeat their polls. For this reason, the RUG5/9 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.

Sample Applications

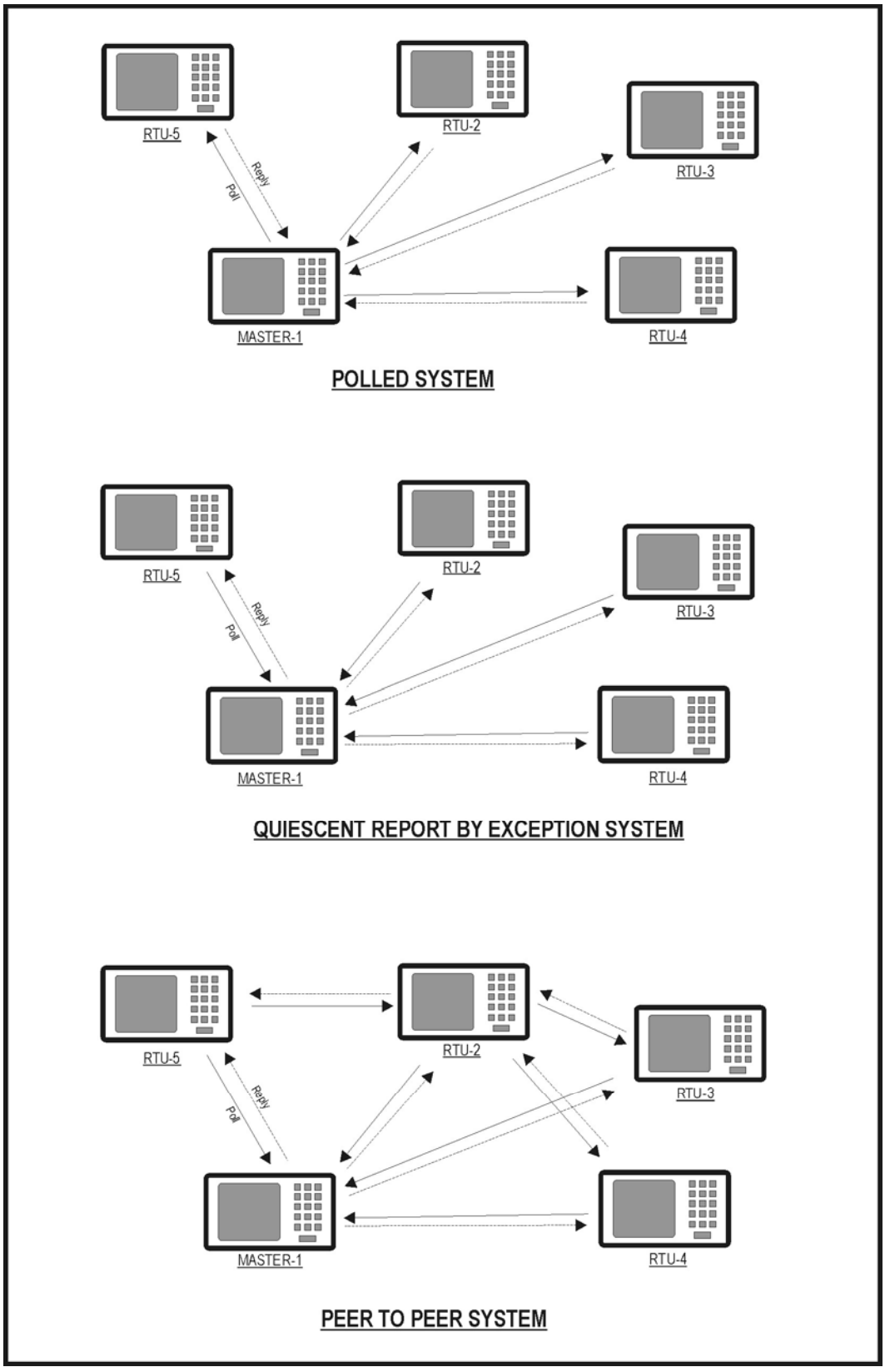


Figure 104 Illustration of Three Main Communication System Types

Sample Applications

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 our 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:

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.
- Total cycle time: 5.21 sec.

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.

We can speed this up by using 1200 baud instead of 300 baud to give a cycle time of 123 seconds. That's still over 2 minutes to report an alarm. 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. In that case, polls from the master can be routed through RTU-2. In the RUG5/9 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.

Sample Applications

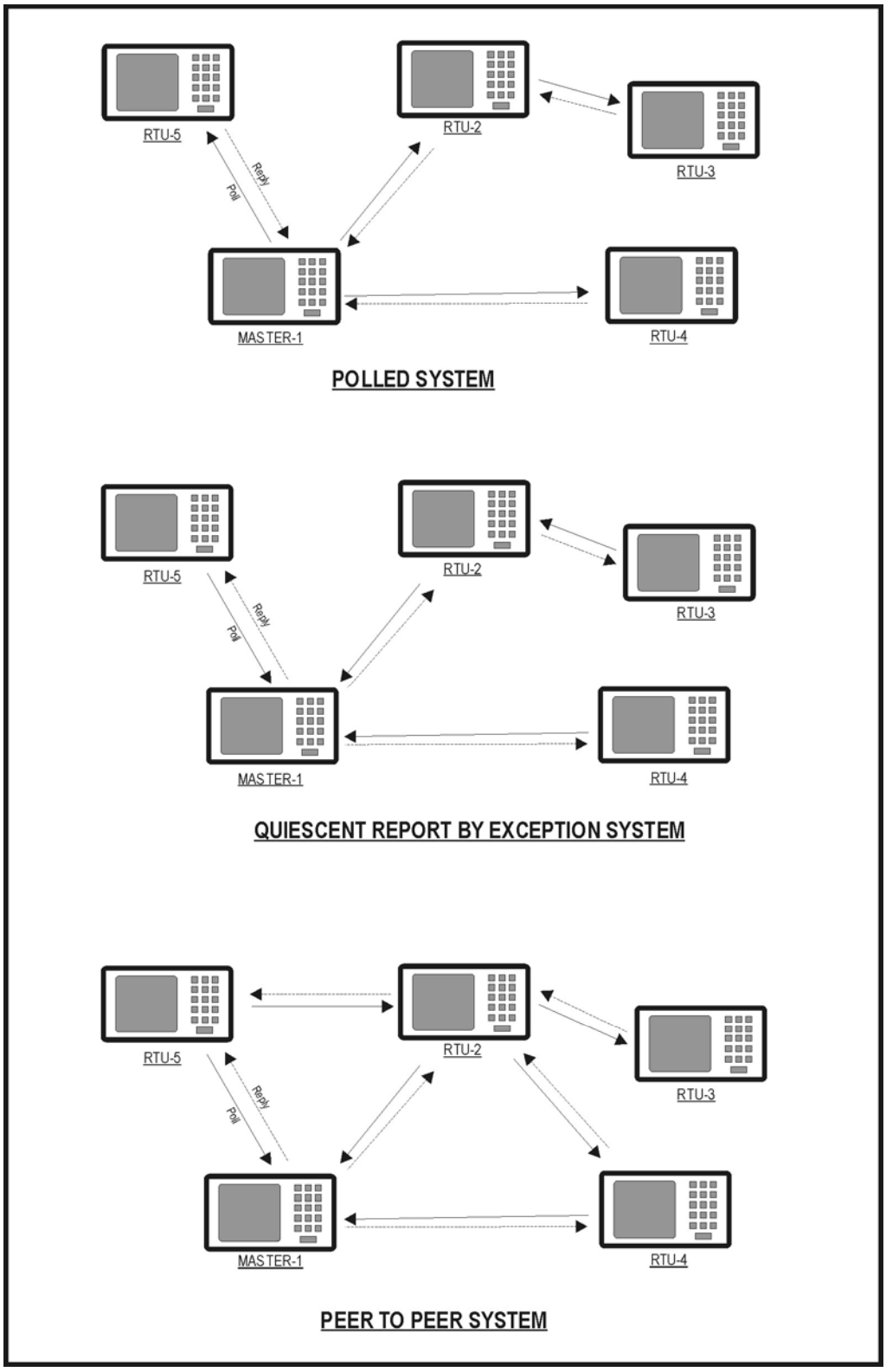


Figure 105 System Types Using Store and Forwarding to RTU 3

Sample Applications

PROTOCOLS SUPPORTED

The RUG5/9 operating system provides complete communications services for any port present in the base card cage. All communications are secured with appended CRC16 codes except for ASCII, which uses no security. The RUG5/9 supports the following protocols:

Table 16 Protocols Supported

PROTOCOL	WHEN TO USE
ASCII	Use for ASCII com when you want RUG5/9 menu services supported
RUG6,7,8	Use to communicate with RUG6,7,8 units (also works with RUG5/9)
RUG5/9	Use to communicate with other RUG5/9's
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
MODBUS MASTER	Use to poll other Modbus devices
ASCII 2	Use for ASCII com when you don't want RUG5/9 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 employ a poll and reply dialogue whereby one station transmits a poll and the addressed station replies. The RUG5/9 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 RUG5/9 also supports store and forward operation when using RUG6 and RUG5/9 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 RUG6 format supports store and forward using up to 14 intermediary stations; the RUG5/9 supports up to 3 intermediary stations.

SETTING UP PORTS AND TLM 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 card and port designators, which must be constants. 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 RUG5/9 operating system.

Sample Applications

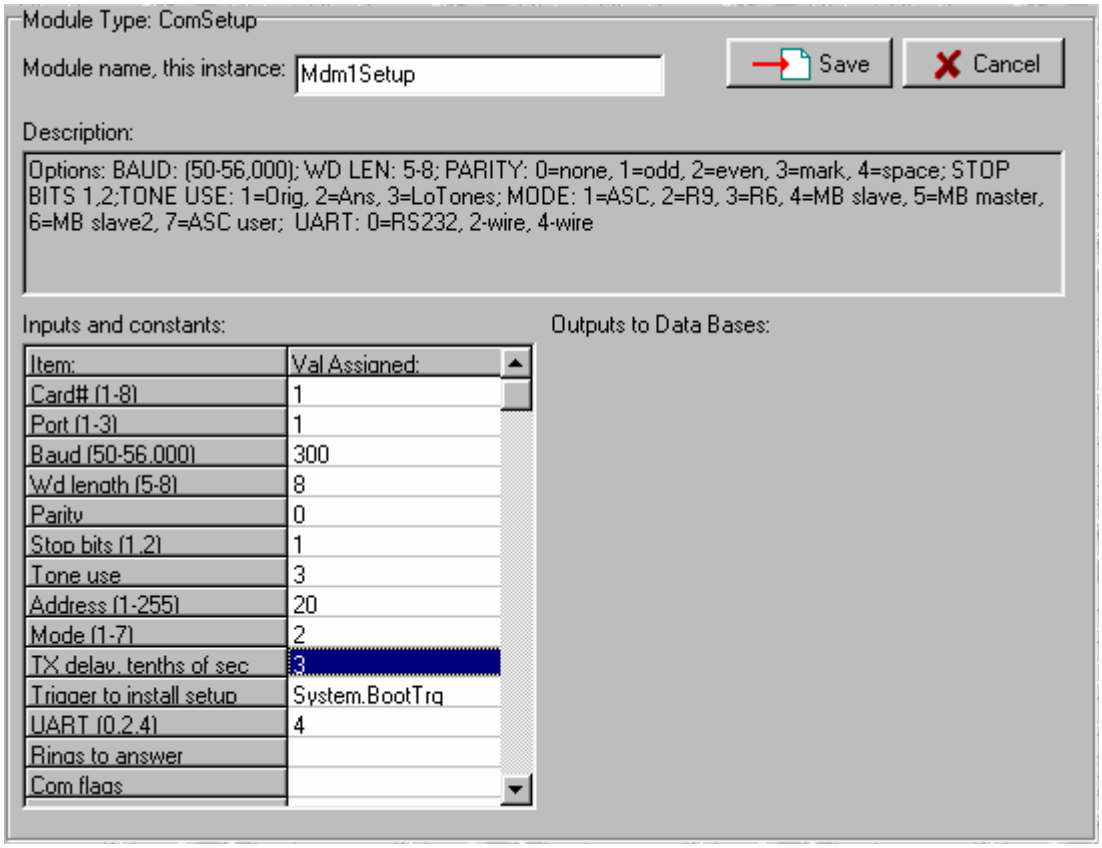


Figure 106 Typical Modem Setup for Radio Applications

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 R9 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 RTU's 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 board, 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 board, port and destination address. If you have previously defined a format for a selected combination of board, 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 using port 1 on board number 1.

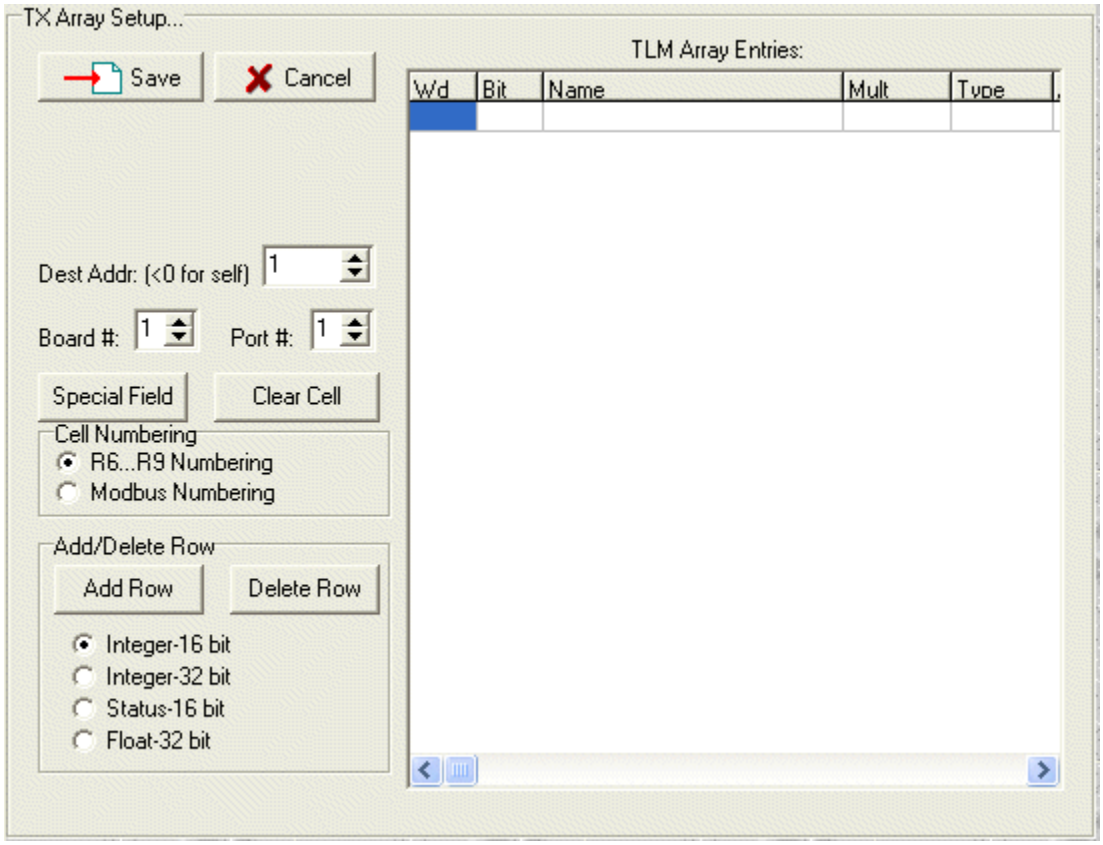


Figure 107 Initial TX Array Setup Screen

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, integer-32 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. In the screen presented below, the variable "TankHiAlrm.SP" 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 RUG5/9 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.

Sample Applications

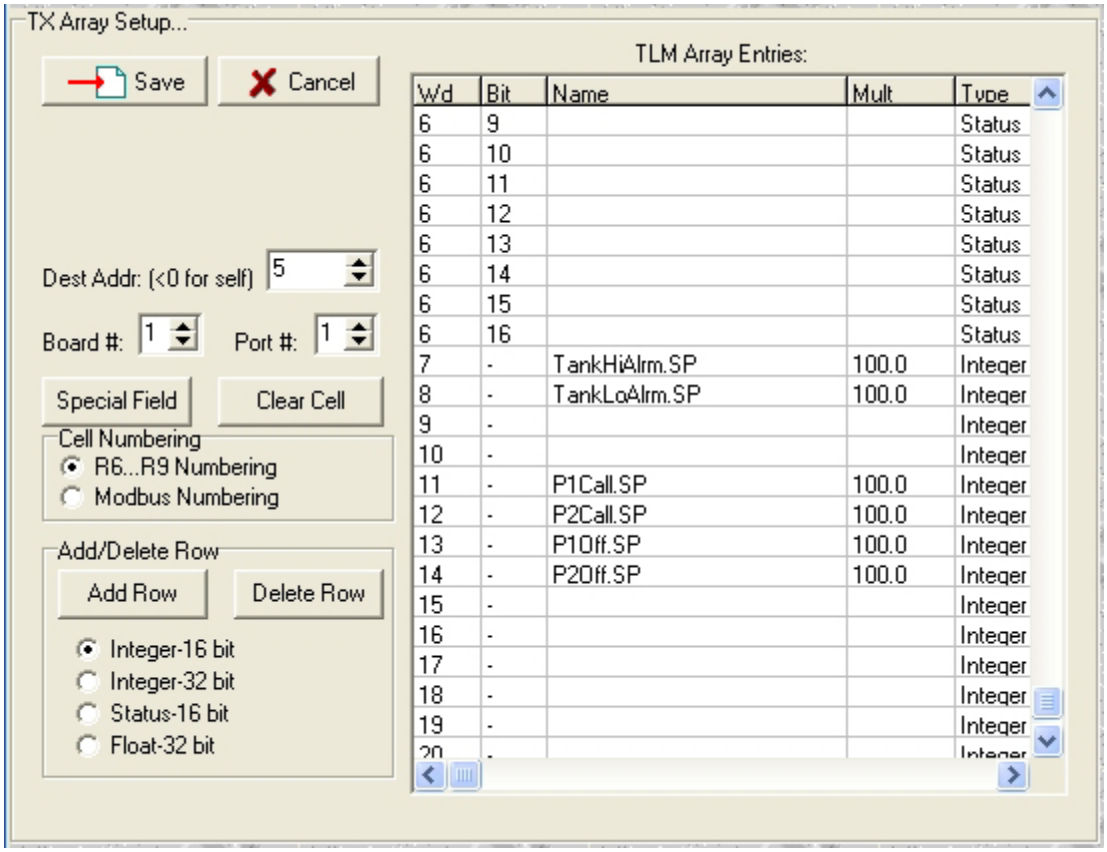


Figure 108 Example TX Array with Variables Installed

Sample Applications

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 as an integer 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 multiplied by the multiplier 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 RUG5/9 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.

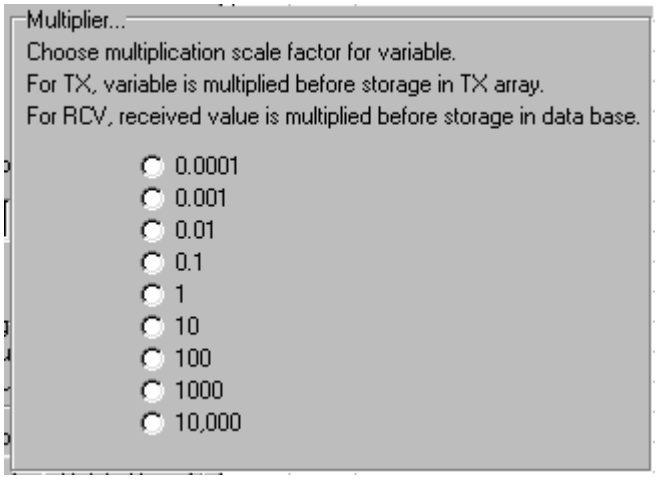


Figure 109 Panel to Select Telemetry Multiplier

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 board, port number, 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 board, port, source and destination address. If you have previously defined a format for a selected combination of board, port and 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 receiving from station address 1 using port 1 on board number 1. Note that the destination address is specified as –1. 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.

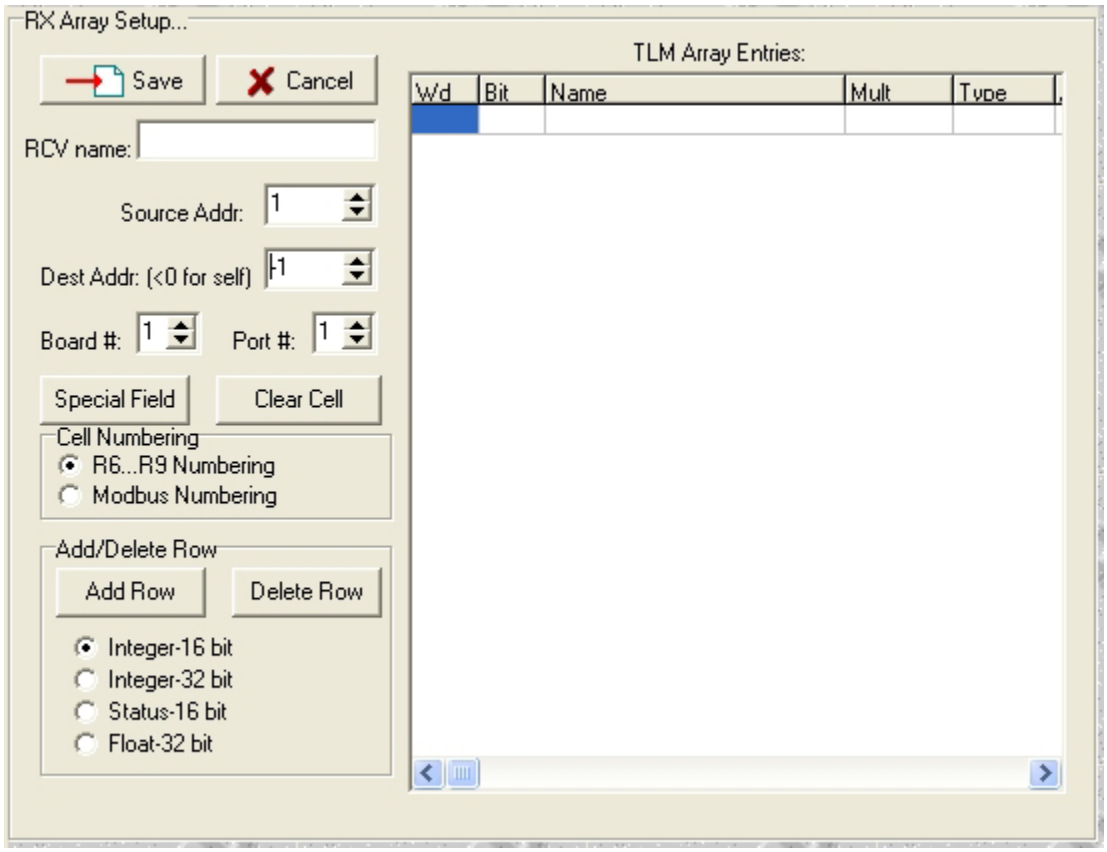


Figure 110 Initial Receive Array Setup Panel

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, integer-32 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 "T1Level2" was entered into the name edit box for the line 5 of the format. It is saved for future reference as "T1Level2.1" so that you can easily see that the variable originated at site address 1. Variables in the RCV database can be used as the inputs to any other modules.

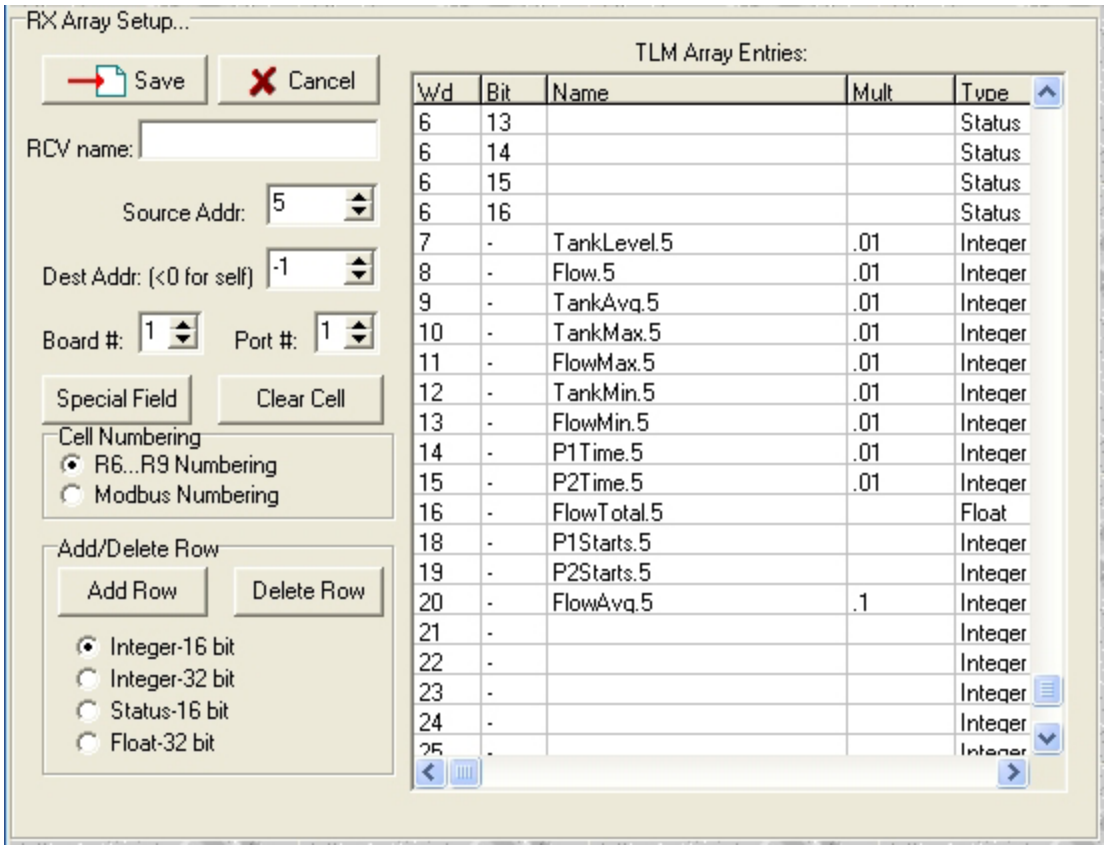


Figure 111 Receive Array Signal Naming

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 RCV database with an appendix equal to the station address from where the data is being received.

SPECIAL FIELDS…GlobalRTC, GlobalSP

To support special functions of realtime clock synchronization and global setpoints, 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.
- GlobalSP…designates that a 4 byte word contain a global setpoint value as 16 bit integer, multiplier, index in system, and age code. When transmitting, the system will read the next global setpoint from the transmit list and pack the global setpoint's value along with its index, multiplier and age code into

the outgoing message. On reception, the system will place the 4 byte word at the top of the global setpoint receive list for installation by the individual GlobalSetpoint modules on the next scan. The global setpoint transmit and receive lists are maintained by the GlobalSetpointSetup module.

To use these, you must first install a 32 bit integer 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 below, the broadcast array has a setpoint that is local to the master (SystemMode.SP), a GlobalRTC field to hold the realtime clock, and four GlobalSP entries where the master can transfer up to four setpoints received from RTU's in a single message. There is no advantage to having more than one GlobalRTC entry in any transmission. Multiple GlobalSP fields allow multiple global setpoint values to be sent in a single transmission if more than one exist in the transmit list. Note that you must have identical formats in both transmit arrays and corresponding receive arrays for this to work properly.

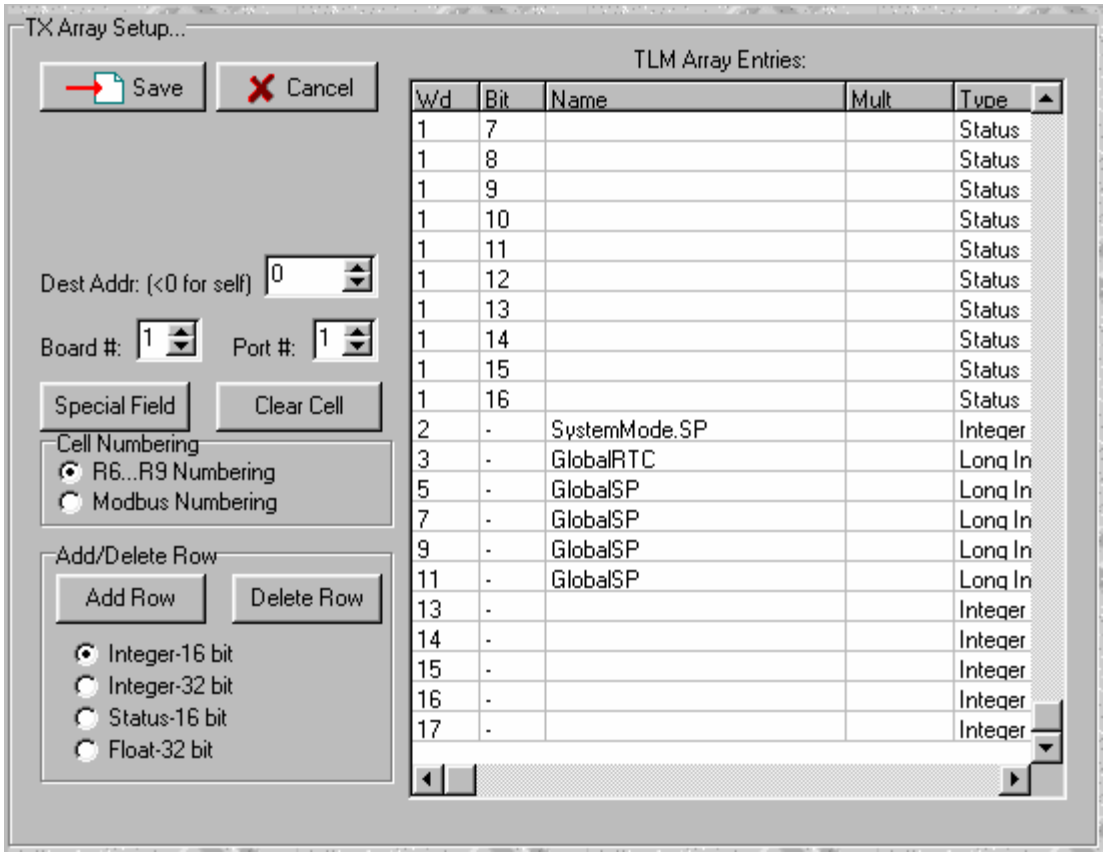


Figure 112 Special Field Installation Into RX/TX Array

BROADCAST MODE

The RUG5/9 system reserves address zero as the broadcast address. Obviously, with the RUG5/9 eavesdrop mode, any address could be regarded as a broadcast address. The advantage of using address zero is that the RUG5/9 unit will accept the transmission regardless of the source address…any reception destined to address zero from any station will be accepted by any station with a receive array having a source address of –1 (Us) and a destination address of zero. The receiving station will not reply to the message. You would use the broadcast mode to send data to be used by many other stations, such as

master setpoints, global setpoints, clock synchronization, or other values and statuses intended to be used by other sites.

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 integer 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.

HOW TO SETUP GLOBAL SETPOINTS

When you need a setpoint that can be entered anywhere in a system, such as at a remote site as well as at the master site, it is better to use the global setpoint rather than the standard setpoint. This is because the global setpoint has an associated index number and age code enabling the system to automatically handle new entries without undue special programming by you. To use global setpoints, you will need a single GlobalSetpointSetup module at each site that will use any global setpoints. The GlobalSetpointSetup module maintains a list of global setpoints that need to be transmitted. It also maintains a list of received setpoints to be installed into the individual GlobalSetpont modules after reception if age codes indicate that the new values are more recent than those already in use. The figure below illustrates a typical setup. The setpoint buffer size must be large enough to hold as many setpoints as may change between transmissions. Each buffer entry uses 8 bytes of RAM.

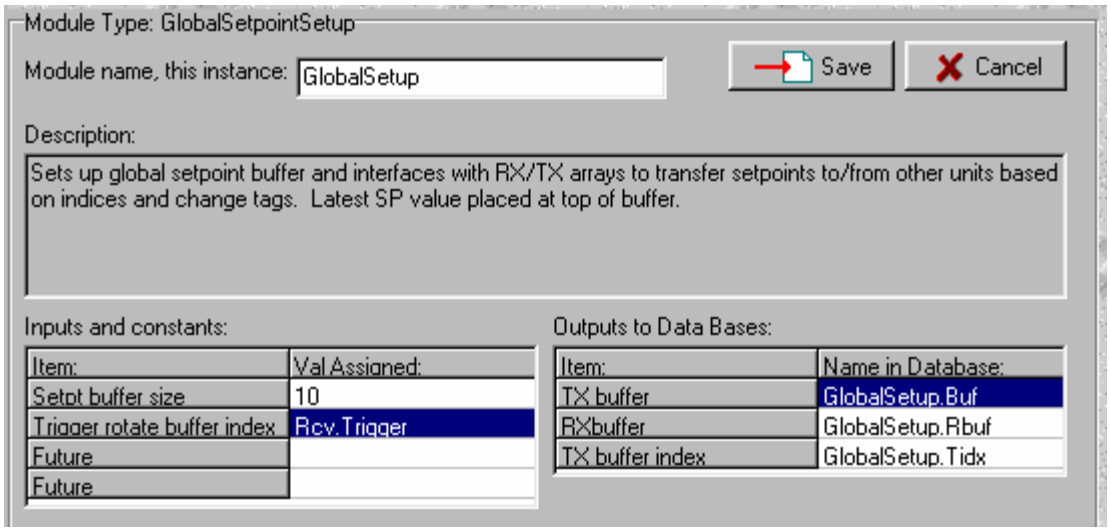


Figure 113 GlobalSetpointSetup Typical Installation

You will also need a GlobalSetpoint module at each site for each setpoint that will be used or entered at the site. The GlobalSetpoint module is similar to the standard local Setpoint module except that it adds a global index, holds the age code, holds the transmit multiplier, and does range limiting. At the site, global setpoints appear in the setpoint list the same as standard setpoints, so they can be entered by the user and protected by the logon code just as with standard setpoints. A typical GlobalSetpoint module setup is illustrated below.

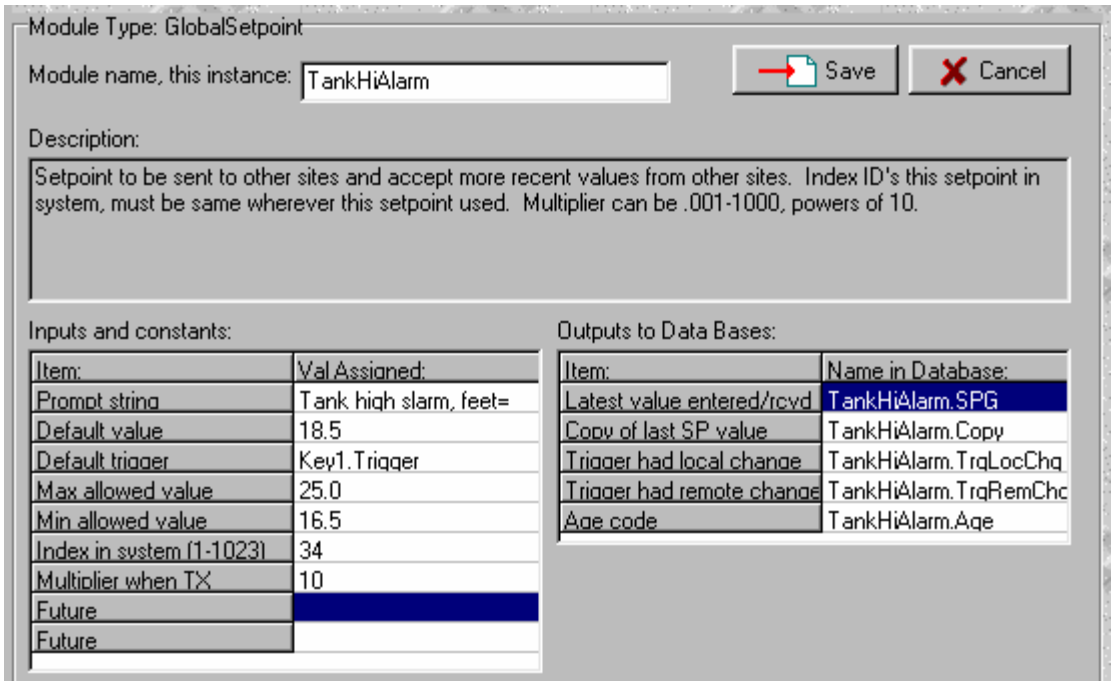


Figure 114 Typical GlobalSetpoint Setup

Notice that this module's index in the system is set to 34. The index is a system-wide numbering system that identifies each global setpoint in the system. Wherever this setpoint is to be used in the system, you must install a GlobalSetpoint module with this same index. This way, whenever an RTU receives a setpoint, it will know to install the new value in the correct location. The index range is 1 through 1023, so up to 1023 different global setpoints can exist in any system. Also, since no board or port number is specified, the global setpoint indexing system applies to all ports and boards in an RTU. Therefore, an RTU involved in communications over multiple networks will accept and can send global setpoints to all ports. This module also has a transmit multiplier for its setpoint so that the setpoint value can be sent as an integer while preserving fractional values. The unit will multiply the setpoint's value by the multiplier before transmission and divide by the same value upon reception. The floating point value of this setpoint is its first output with the '.SPG' suffix. Whenever a new value is typed in by a local operator, or poked in by a poke module, the local change trigger will be asserted to signal that a change has occurred. This can be used to trigger a transmission or other function. Whenever this module receives a value from telemetry, the remote change trigger will be asserted. Finally, the age code is used to discern if an incoming value over telemetry is newer than the existing value. It has a range of 0 to 7 and rolls back to zero when it tries to count past 7. The module will accept any setpoint whose age code is at least one count greater the setpoint's current age code.

HOW GLOBAL SETPOINTS WORK

Let's follow a setpoint entered at an RTU to illustrate how the system sends it from site to site. Assume it's a polled system and the process starts when an operator changes a global setpoint at an RTU. When that happens, the GlobalSetpoint module will install the setpoint at the top of the transmit list maintained by the GlobalSetpointSetup module. On the next master poll, the setpoint would be contained in the reply to the master and would be installed in the master's corresponding global setpoint. This would constitute a remote change to the master's setpoint, so the master's global setpoint module would install the setpoint in its transmit list. At the end of the current polling cycle, the master would send a broadcast poll containing this setpoint and any other setpoints received during the last cycle, thereby sending the setpoints to all stations at once. When the RTU that originated the change receives the setpoint, it will ignore it since the age code would indicate that the setpoint is no newer than the RTU's current value, except that the

Sample Applications

GlobalSetpointSetup module would remove the setpoint from its transmit setpoint list. Other RTU's would accept the setpoint as a new remote change.

In a quiescent system, the RTU would trigger a poll immediately and the poll would contain at least one GlobalSP entry in its transmit array. If the poll were to the master site, the global setpoint would be accepted at the master; if it were broadcast, it would be accepted by all RTU's and the master at that time. The choice of whether to poll a master or to broadcast depends upon visibility of the RTU by other stations. If all stations that need the setpoint have visibility of the RTU, then broadcast is the best choice. Otherwise, the setpoint should be sent to a station that can send it to the others, usually a master site.

COMMUNICATIONS FORMATS

RUG6,7,8 FORMATS

The RUG5/9 supports the RUG6 bidirectional transfer formats as well as the RUG5/9 formats described in the next section. You can specify that the RUG5/9 use the RUG6 formats by setting the mode entry to 2 in the **ComSetup** module described above. Be sure not to use the floating point cell format in the format definition phase of your setup since the RUG6 telemetry format does not support floating point transfers. The RUG6 format is presented below.

Table 17 RUG6 TRANSMIT FORMAT

	HEADER						
Byte	0	1	2	3	4	5	6
Function	ID	Source Address	Destination Address	Message Type	High TX Index	High Reply Index	Msg Length Bytes
Example	$83	$03	$04	$63	$0D	$25	$1A

	Forward Path	DATA FIELD	Security
Byte	7-9	10-23	24,25
Function	Fwd Path	Words 1 through 7	CRC-16
Example	3,4,5	00,02,00,02,00,03,00,04,00,05,00,06,00,07	CRC-16

Table 18 RUG6 REPLY FORMAT

	HEADER						
Byte	0	1	2	3	4	5	6
Function	ID	Source Address	Destination Address	Message Type	High TX Index	Error Reply	Msg Length Bytes
Example	$83	$07	$05	$E4	$25	$00	$4C

	Fwd Path	DATA FIELD	Security
Byte	7,8,9	10-73	74,75
Function	Fwd Path	Destination's words 5 through 37	CRC-16
Example	3,4,5	00,05,00,06,00,07…..00,37	CRC-16

RUG5/9 FORMATS

The RUG5/9 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.

Table 19 RUG5/9 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=reserved 3=special commands 4=reserved 5=reserved 6=reserved 7=flash load 8=get status 9=pgm load	# bytes in fwd path	Place in fwd path			

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 20 RUG5/9 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 21 RUG5/9 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	CRC-16

SPECIAL COMMAND FORMAT

The special command is used to control flash access, arm flash erasure, etc.

Table 22 RUG5/9 SPECIAL COMMAND FORMAT

Byte #	0-4	5		6
Function	HEADER	COMMAND		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 1 flash sector 5=Install pgm security code 6=xfer to boot blk code/arm flash	Data, 0-15	

Table 23 RUG5/9 STATUS 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

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 24 RUG5/9 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 25 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

ETHERNET COMMUNICATIONS

The RUG5/9 units can communicate over the Ethernet using an external serial to Ethernet converter. Please refer to our web site for a detailed application note describing the process. The web address is:

www.rugidcomputer.com/Downloads/AN0802_RUGID_RTUs_Communicating_Over_Ethernet.pdf

CHAPTER 8…SAMPLE APPLICATIONS

INTRODUCTION

This chapter presents a number of application examples to illustrate how to apply the RUG5/9, set up the cards, configure the software modules, set up displays, use ladder logic, etc. All examples assume the use of standard RUG5/9 board configurations (RUG9C, RUG9D, etc.) even though some I/O boards may not be necessary in some of them and could be deleted to save RTU cost. The idea is that you could use one of the application examples as the beginning point for your application by adding necessary functions without having to move boards and thereby invalidate the module set up in the example. 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. 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. Finally, we have assumed the following telemetry word arrangement for all examples so that, for most applications, the first several words of the telemetry arrays would stay the same; any substantial additions would simply tack on to the end of the arrays. This arrangement matches that used in most of our past RUG6 projects.

Table 26 Standard Telemetry Format For Examples

WORD ASSIGNMENT	FUNCTION
1	Spare, MSB set for RUG6 compatibility
2	General purpose statuses 1-16
3	General purpose statuses 17-32
4	General purpose statuses 33-48
5	Alarm statuses 1-16
6	Alarm statuses 17-32
7	Integer analog value #1
8	Integer analog value #2
9…	Integer analog values #3…

Sample Applications

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, R9DTANK1

The following application uses a RUG9D 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. A second LCD screen displays a trend of the tank's level. Two other screens present tank level in 2X and 4X size characters. High and low alarm setpoints are entered locally. The following diagram presents the necessary wiring. Configuration file for this application is R9DTANK1.

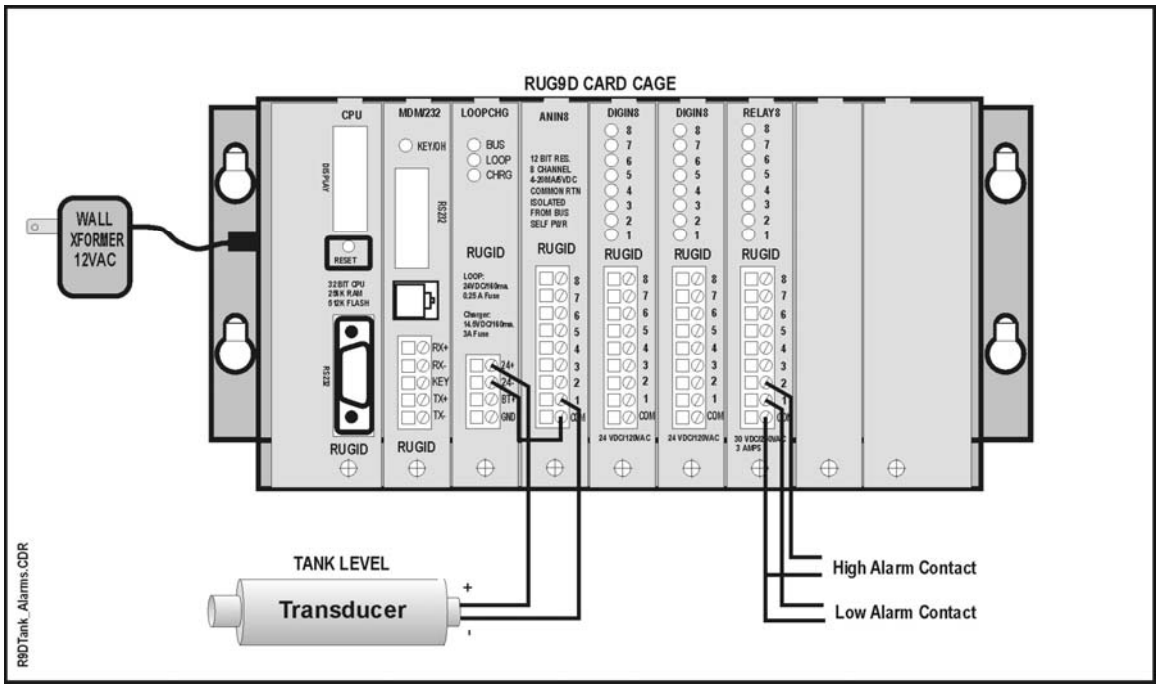


Figure 115 Tank Level Monitor Application

The following modules are used in this example:

Sample Applications

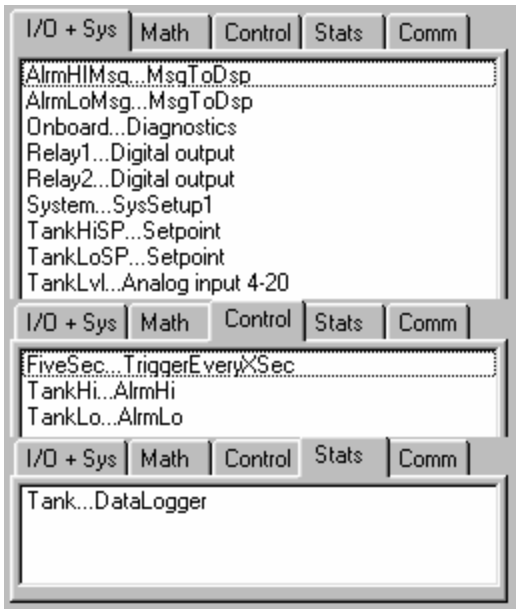


Figure 116 Modules in R9DTank1 Application

Here is a list of what each module does:

AlrmHiMsg…MsgToDSP and **AlrmLoMsg…MsgToDSP**:

These modules print a message on the display (Hi Alarm and Lo Alarm, respectively) whenever a high or low alarm is present. These messages are printed just to the right of the tank level on the main screen.

FiveSec…TriggerEveryXSec and **OneSec…TriggerEveryXSec**:

These modules issue triggers on a fixed time interval. The FiveSec module issues a trigger every five seconds which is used to trigger a display update when the trend display is presented. The one second trigger is used as the update trigger of each of the other displays

Onboard…Diagnostics:

This module reads the Loop/Charger board's A/D converter and puts the unit temperature, bus voltage and battery voltage in the database, from where it is displayed.

Relay1…Digital Output and **Relay2…Digital Output**:

These modules are used to connect the alarm statuses for low alarm and high alarm, respectively, to the first two relays on the relay output board. If we had used Alarm Output modules instead of Digital Output modules here, the outputs would have flashed on and off automatically when their corresponding alarms were declared.

Setup…SysSetup:

This module sets the display backlight blanking interval, the logoff interval, and sets up the logon security code for setpoint access.

Tank…DataLogger:

This data logger module allocates 300 words for trend storage and sets the trend sample interval, lower and upper scale values, and horizontal and vertical tick mark settings.

Sample Applications

TankHi…AlrmHi and **TankLo…AlrmLo**:

These modules compare the high and low alarm setpoints with the tank level and issue output statuses when alarm conditions exist. They delay the declaration of an alarm by 5 seconds using timers.

TankHiSP…Setpoint and **TankLoSP…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.

Data Logger Setup

The data logger module is a little tricky, so we'll examine its setup, which is presented below:

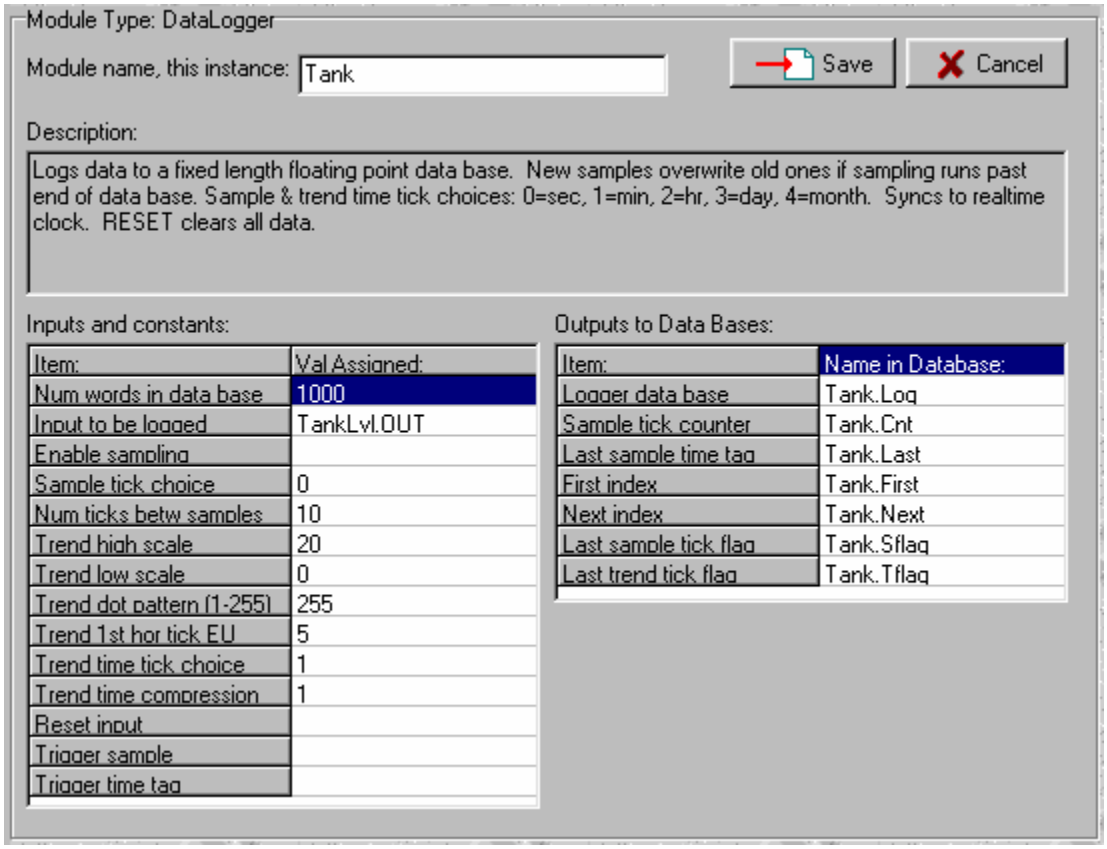


Figure 117 R9DTank1 Logger Setup

Here, we have told the system to allocate space for 300 samples which is a few more than can be shown on our LCD at one time from where we have told the display to begin trending. We have also identified the variable TankLvl.OUT, our tank level, as the variable we wish to log. The choice of sample tick choice and ticks between samples can be confusing. In this application, we have chosen zero as the sample tick

choice, which means that the sampler will have a basic sampling time tick of once per second. The next property, number of ticks between samples, tells the system we wish to take a sample every 10 of our ticks, or, in other words, every ten seconds. Trend high and low scale properties set the scales of our trend plot on the LCD, in engineering units. Note that the RUG5/9 saves all samples in floating point, so the scales can be changed at any time to expand or compress the vertical scale. The trend dot pattern, with a range of 0 to 255, controls the appearance of the trend line on the LCD. It simply sets the bit pattern of an 8 bit byte that is used to write the line. A value of 255 gives a solid line. A value of zero would erase the line. Values in between would give varying dot patterns. For example; 127 would give an almost solid dashed line; 85 would turn on every other dot, etc. The Trend 1st hor tick EU property establishes where a horizontal tick line will be located. It then calculates the location of the rest of the ticks to give them uniform spacing. In this example, choosing 5 sets the first horizontal tick at 5 feet. The system then calculates the rest to be at 10 and 15 feet based on the first. If we had chosen 4 for the first horizontal tick, the system would have calculated 8, 12, and 16 for the rest. Finally, the trend time tick choice establishes at what interval the system will insert a time tick in the sample data base so vertical time reference markers can be presented on the LCD. Choosing 2 sets the time tick at once per hour. Therefore, at each hour, the system will show a vertical bar on the trend display.

Main Display Setup

We'll examine the main display and the trend display in this example. The main display is presented below.

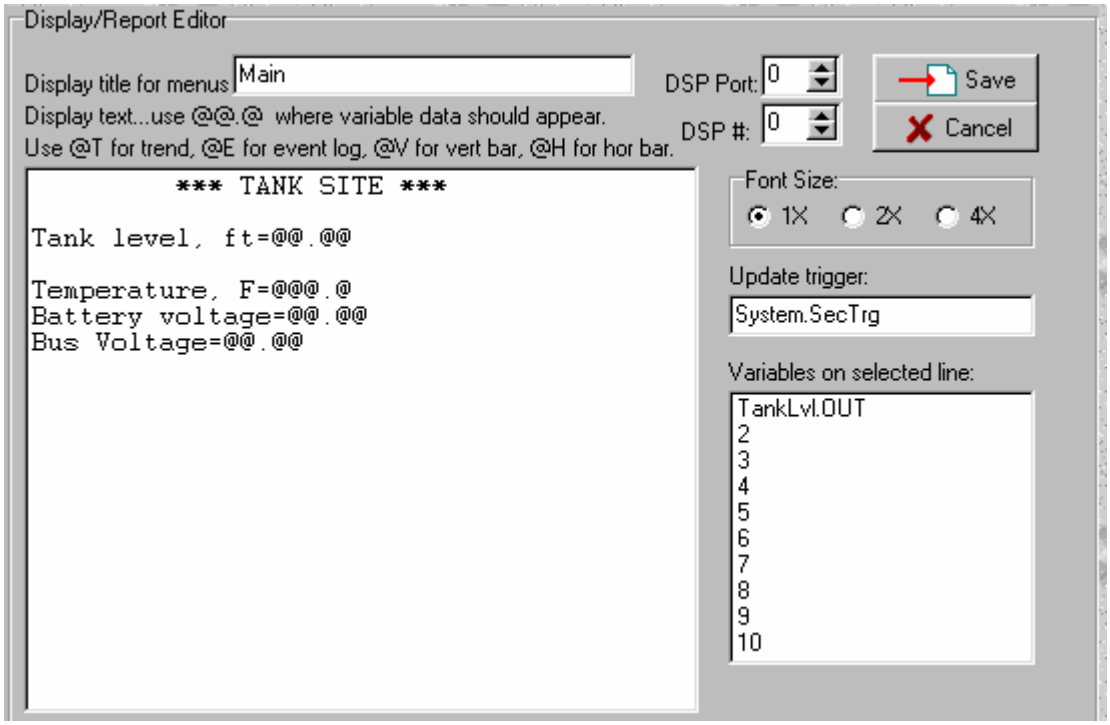


Figure 118 R9DTank1 Main Display Setup

Notice that the display title for menus is given as "Main". This means that this display will be referred to as "Main" in the display list box of the R9Setup program, and in the RUG5/9 display menu when the RUG5/9 is running. The display port choice of zero means that this display will be presented on the RUG5/9's LCD. Display number zero means that this display will always be listed first in menus. Font size of 1X is the default for the RUG5/9, and provides the most room for presenting information. Other displays in this application use 2X and 4X to illustrate their appearance. The update trigger of OneSec.Trig

was dragged into the trigger list box from the status data base. When running on the RUG5/9, that trigger will cause the display to rewrite once per second. Finally, notice that the cursor is resting on the line that presents the tank level in the large display editing window. At the same time, the list in the lower right corner 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 RUG5/9 hits the "@@.@@" field on the third line of the display, it will go get the tank level and place it on the LCD in place of the "@@.@@" field.

Trend Display Setup

The trend display setup is similar to that above. Notable differences are the 5 second trigger update interval and the fact that the large display definition window only has a title line and the symbols "@T" near the lower left corner. The "@T" character pair tells the RUG5/9 that it is to install a trend plot beginning at that location. When the cursor rests on the line with the "@T", notice that the variables list has the variable "Tank.Log" on the top line. This tells the RUG5/9 where to get the data values to be plotted along with other trend information such as scales, tick mark definition, etc. The setup screen is presented below.

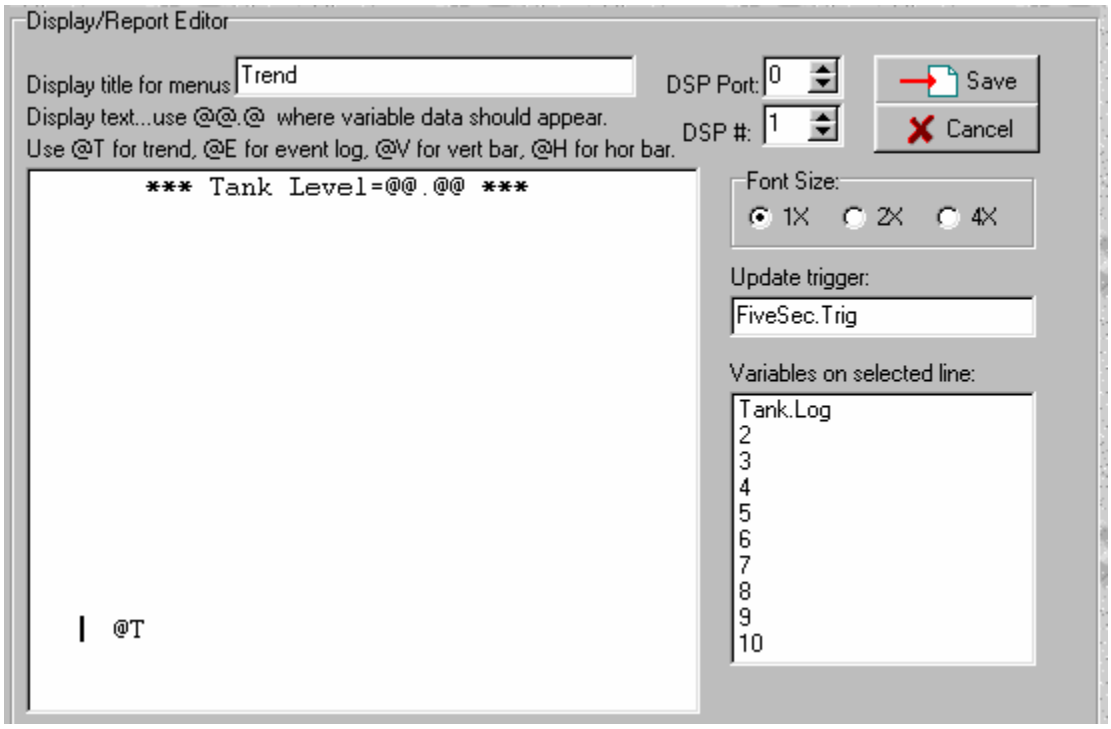


Figure 119 R9DTank1 Trend Setup Screen

AP NOTE #2: STAND ALONE 4 PUMP UP CONTROLLER, PUMPCTRL4

This example implements a controller for four pumps to maintain a tank's level using a local tank level measurement. It is based on the example file "PUMPCTRL4". In it, we wish to illustrate use of the following modules:

- **LeadLagSeq**…lead lag sequencer for rotating the lead pump and including backspin delay
- **PumpUpCtrl**…pump up controller
- **AlrmMismatch**…used to detect pump failure
- **Digital alarm output**…alarm annunciator form of digital output module
- **DataLogger**…data logger used to support trending on the LCD
- **EventLogSetup** and **EventLogger**…implements event logging to LCD
- **StringSwitch**…used to present meaningful messages to LCD
- 2X and 4X font…shows how to implement larger fonts on LCD

General Operation

The pump control function is accomplished by four **PumpUpCtrl** 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 four controllers are routed to a **LeadLagSeq** sequencer. Its job is to call as many of its outputs as its inputs demand, beginning with the lead pump. It must rotate the lead pump to equalize wear on the pumps, and delay pump switching so that a minimum delay occurs between successive pump switch actions. Sequencer outputs are routed to four relays that would be used to call pump starters. This logic is presented in the following diagram. Additional logic, consisting of **AlrmMismatch** modules and **Digital alarm output** 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 triggered. Finally, a **DataLogger** module is used to capture tank level trend over time; and an event logger is setup to record alarm occurrences, alarm acknowledgements and setpoint changes in a time tagged list. Aspects of the use of these modules in this application are discussed below.

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.

For pump down applications, substitute the **PumpDnCtrl** modules for the **PumpUpCtrl** modules and alter the pump CALL and OFF setpoints accordingly.

Sample Applications

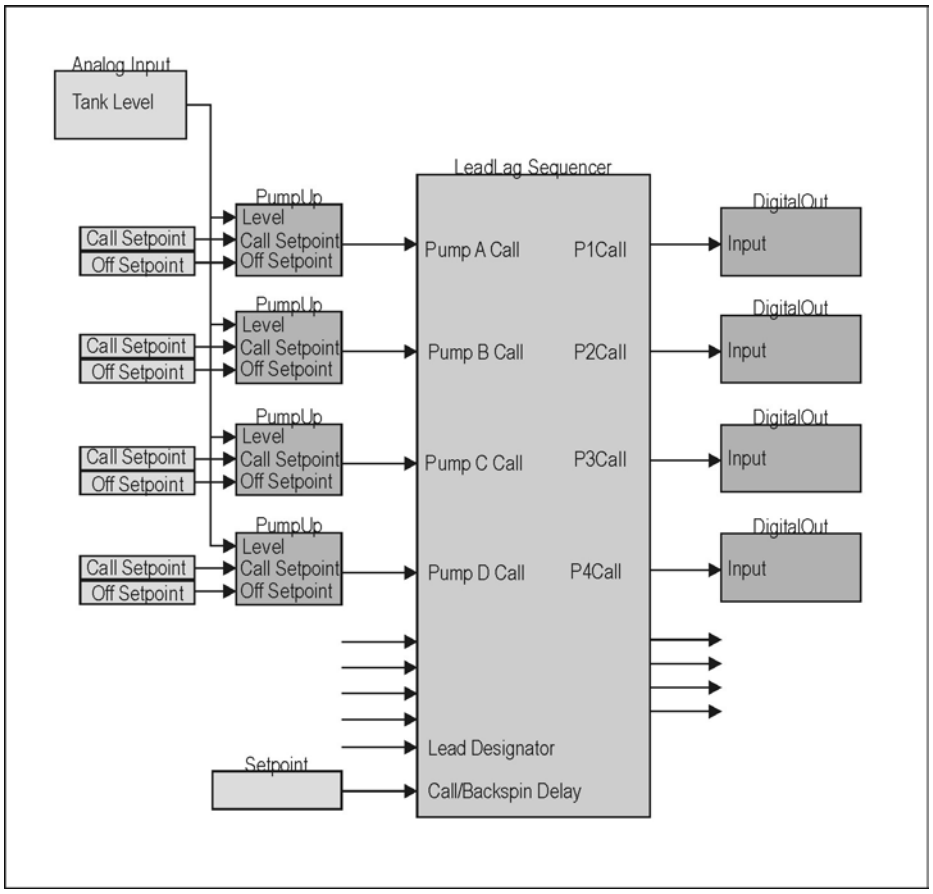


Figure 120 Four Pump Control Logic Diagram

LeadLagSeq Module

In this application, the **LeadLagSeq** module receives pump call demands from the **PumpUpCtrl** 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 **LeadLagSeq** module's call inputs. The sequencer determines how many pumps to control based on how many call inputs have been configured.

DataLogger/Trending Setup

The **DataLogger** module performs data logging for LCD trending, and also establishes trend characteristics for the LCD. DataLogger inputs are listed below:

Inputs That Must be Constants:

- Number of words in database…integer sets number of floating point RAM locations to be set aside for use by module, set to 1000. This setting will give approximately three screens worth of trend data.

Other Inputs:

- Input to be logged…floating point data base point whose value is to be logged at the designated time interval, set to log tank level.

Sample Applications

- Enable sampling…status input that enables sample storage when true, disables when false, left blank to enable trending constantly.
- Sample tick choice…integer sets choice of time base type for sample storage…0=seconds, 1=minutes, 2=hours, 3=days, 4=months, set to 0 for demo purposes, showing roughly 5 minutes across the LCD screen. For a practical application, set this to one to give a once per minute basic time tick.
- Number of time ticks between samples…integer to set sample interval, set to one. For practical application, set this to 1, giving one sample every minute, or showing approximately 5 hours across the LCD.
- Trend high scale…floating point value that sets engineering units of trend plot top of scale, set to 20 (feet). Set to your maximum tank level.
- Trend low scale…floating point value that sets engineering units of trend bottom horizontal axis, set to zero.
- Trend dot pattern…integer in range of 0 to 255, set to 255 to give solid trend line.
- Trend 1st horizontal tick EU…floating point value that sets the engineering units level of the first horizontal dotted line. Others up to 10 will be at an even interval from the first, set to 5. Should be set to one fourth or one fifth of your trend high scale.
- Trend time tick choice…integer that sets choice of time tick for vertical dotted lines so trend can be related to the real time clock, set to 1, giving a time tick per minute. Practical application: set to 2 giving a time tick each hour.
- Trend time compression…integer that sets trend decimation, set to 1 to show every point.
- Reset input…status trigger input: true erases the data base and resets the indices to the beginning of storage, set to clear trend with a keystroke.
- Trigger sample…status trigger input used in lieu of the sample tick choices above to externally trigger data sampling, left blank to let internal time base take samples.
- Trigger time tag…status trigger input used in lieu of trend time tick choice above to externally trigger storage of a time tick in the data, left blank to let internal time base install time ticks.

The following figure shows the display setup to implement trending:

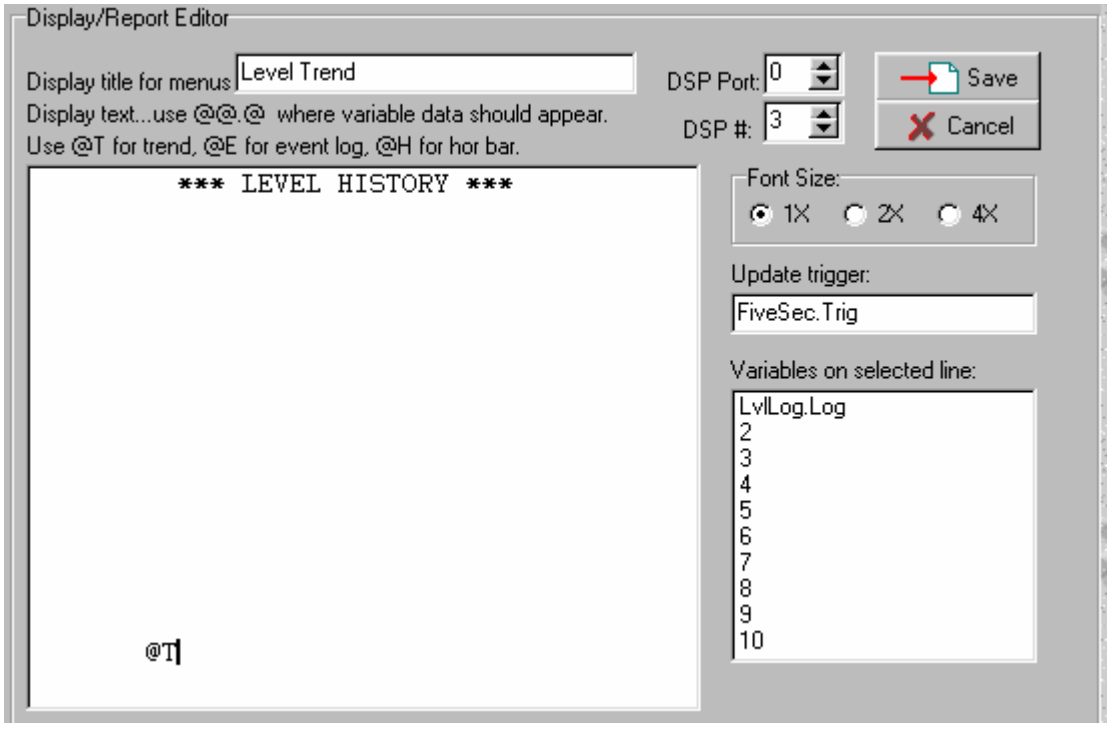


Figure 121 LCD Trend Screen Setup

Note that the '@T' character pair is located one line above the bottom of the screen to leave room for the time tick numbers. It is also located 6 characters to the right of the left side of the screen to leave room for the vertical tank level scale values. In the above figure, the cursor is on the line with the '@T' characters. Therefore, the panel titled 'Variables on selected line' is indicating that the variable 'LvlLog.Log has been dropped from the floating point data base to tell the display software where to get the trend data. Finally, the update trigger is set to trigger the display update every 5 seconds.

Event Logger Setup

This application uses the event logger to log alarms, alarm acknowledgements and setpoint changes. To do this, an EventLogSetup module, an EventLogger module, and a LCD display are required. Here is what they do:

- **EventLogSetup**…establishes how much RAM is reserved for log (100 events in this application), gives a way to clear the log (key 2 here), and enables logging of setpoint changes. Other features such as dumping the log to the serial port are not enabled.
- **EventLogger**…accepts statuses from the status database and stores them in the log. You can have as many EventLogger modules as you need feeding events to the same logger. In this application, the logger this module uses is set to 'Events.Log", the **EventLogSetup** module above. This has been dragged from the integer database. Mode is set to 1 to enable logging of both alarm occurrences, and alarm return to normal events. Events being logged are 1) alarm acknowledgements, 2-5) pump fail alarms, 6) tank high alarm, and 7) tank low alarm. Note that for the tank high and low alarms, we are logging tank level in the logged messages. Also, instead of letting the logger append the default ON/OFF states to the alarm messages, we have included the field {ON/NORMAL} in the event strings to cause the module to append 'ON' when the alarm occurs, and 'NORMAL' when the alarm turns off.

The following screen shows the proper setup for displaying the event log:

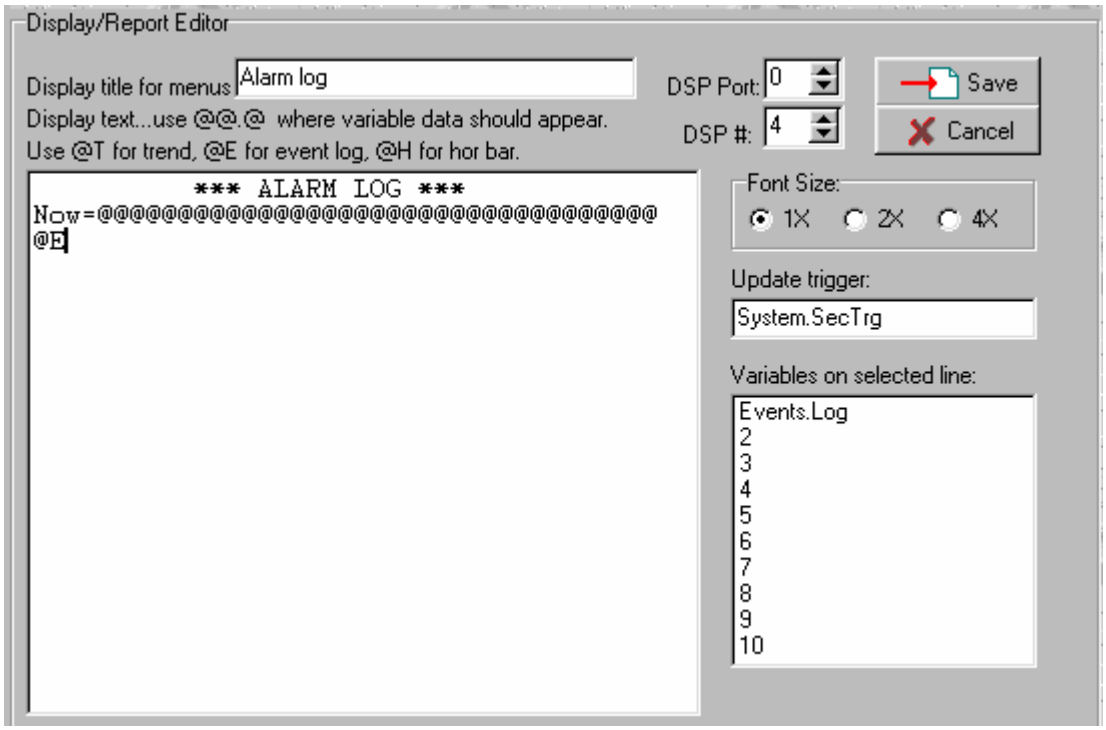


Figure 122 Event Log Display Screen

Sample Applications

The event log uses the center 16 lines of the display, leaving the top two and bottom two lines for our use. Here we use the top line for the title, and the second line to display the realtime clock time and date string, so the user knows if the clock is set properly, and can reference it in relation to the logged entries. The '@E' character pair triggers the LCD software to present the event log on this display. Note that the cursor is resting on the second line and the top line in the 'Variables on selected line' list is 'Events.Log', which was dragged from the integer database. This tells the LCD where to get the log entries. You could have multiple displays referencing different logs.

AP NOTE #3: TELEMETERING TANK SITE, TANKTLM

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 "TankTlm". 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
- **LatchFloat**…latches a floating point value when triggered
- **ValueTest**…compares two values and outputs status results
- **PokeMany**…pokes several values to several destinations
- **LatchString**…latches a string when triggered
- **ReadRTC**…reads the realtime clock/calendar
- **Diagnostics**…provides on board power fail, battery voltage and temperature measurements.

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 power fail, and low battery voltage alarms, and measures battery voltage and on board temperature to send to the master.

This program is compatible with a polling master program, such as the PollMaster program in an application note below. It 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:

- Baud (50-56,000)…integer specifying any baud rate from 50 to 56,000 baud. Set to 300 baud for radio or phone line use. Could also use 1200 baud here for more speed. Higher baud rates would require an external modem or use of a radio with internal modem.
- Wd length (5-8)…integer specifying word length. Set to 8 bits.
- Parity…integer specifying parity choice: 0=none, 1=odd, 2=even, 3=mark, 4=space. Set to 0=none.
- Stop bits (1,2)…integer specifying number of stop bits per word. Set to 1.
- Tone use…integer designating which tone set the modem is to use: 1=originate, 2=answer, 3=low tones. Set to 3 for low tones, best for R9 to R9 communications over leased lines or radio.

Sample Applications

- Address (1-255)…integer address for this port on the network. Set to the output of the **LatchFloat** module as described below.
- Mode (1-7)…integer to select communication protocol. Set to 2, RUG5/9 protocol.
- TX delay tenths of sec…integer to set delay between the time the RUG5/9 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.
- 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.
- UART (0,2,4)…integer to establish connection between the UART and other hardware on the modem board. Set to mode 4 to connect to audio radios or to 4-wire leased line phone systems.
- Rings to answer…integer sets the number of rings that must be counted on the 2-wire channel before the unit answers the line (goes off hook). Left blank.
- Com flags…integer to set hardware handshaking: 0=no handshaking, 1=use RTS/CTS handshaking. Left blank.

Detecting and Displaying 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:

- 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. 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 address held by **Address.SP**. See address setting below.

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 capture the realtime clock's value, being read by the **ReadRTC** module, at the time of reception so we can present it to the operator. We do this using a **LatchString** module to capture the realtime clock's RTC.Str output string so we can display it on the LCD. This shows the time and date of 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 logic detailed in the figure below, where the address will be installed into the system (**ComSetup** will be triggered) when either the unit boots up, or the operator changes the station address setpoint:

Sample Applications

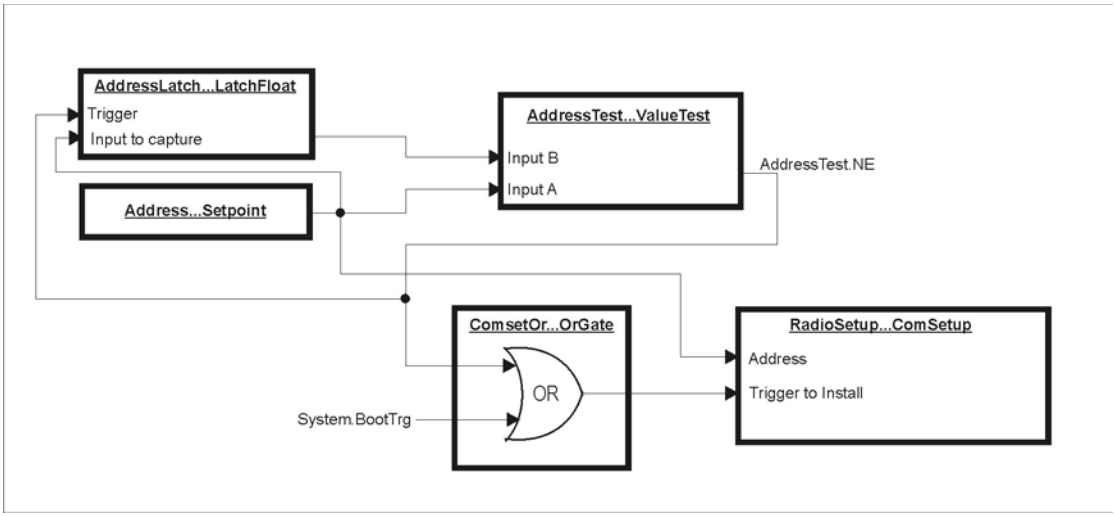


Figure 123 Address Setting Logic

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

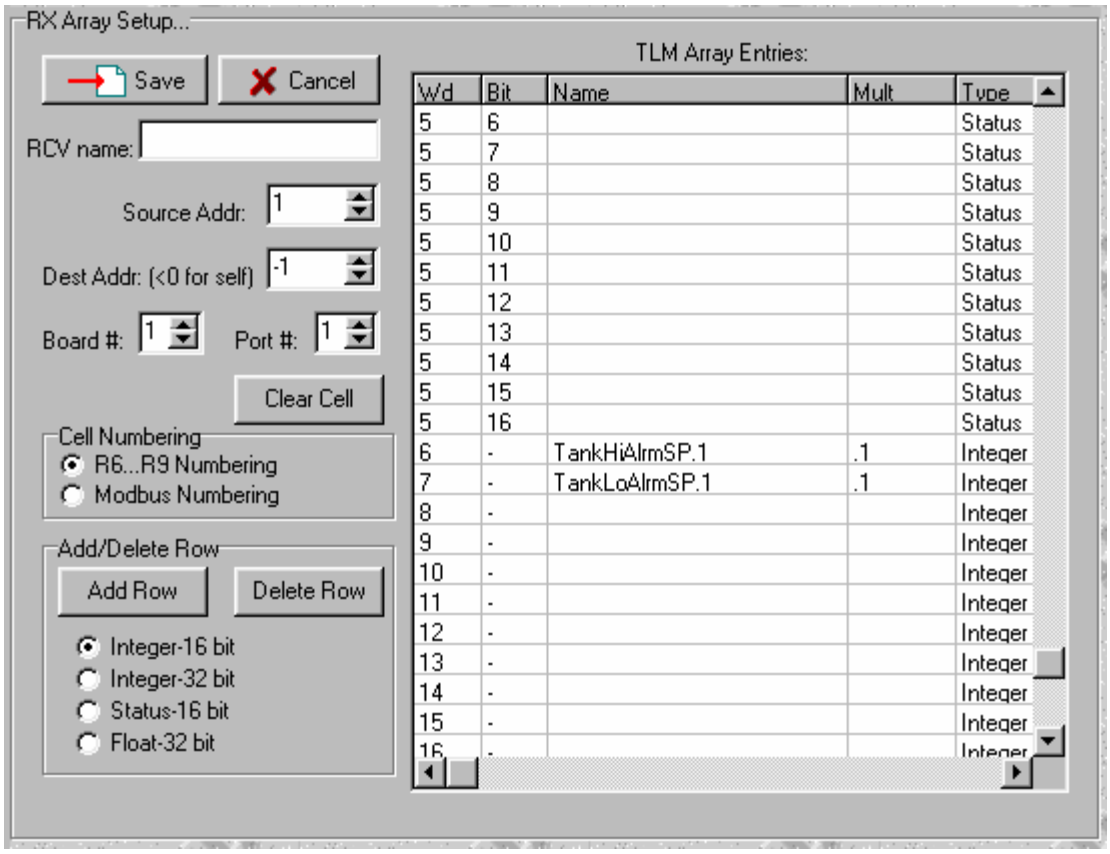


Figure 124 Receive Array Setup

Notice that this array references source address=1, destination address=-1, port=1, board=1. Board and port numbers refer to board number 1 in the card cage, where our modem board resides, and port number 1 on that board. Source address=1 means that this array will accept data from address number 1, the master site. Destination address=-1 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

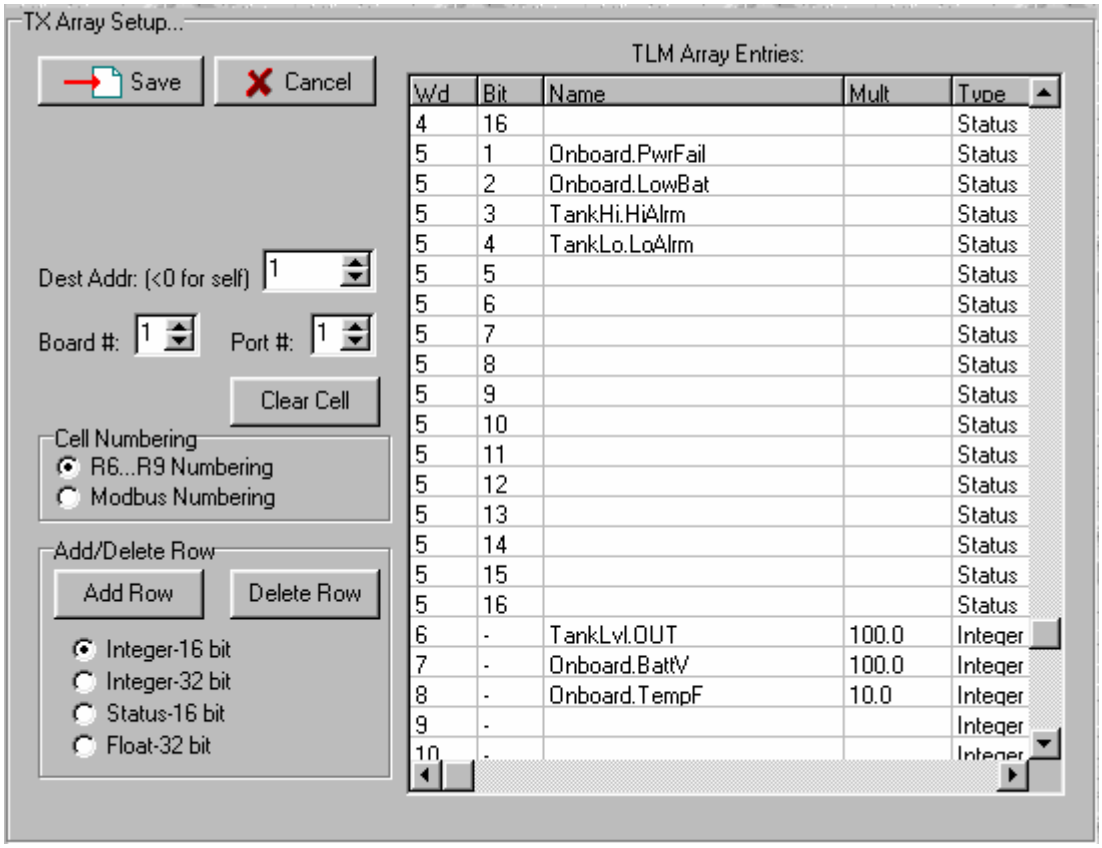


Figure 125 Transmit Array Setup

Here, there is no source address, since we are the source. Board number and port number refer to the modem board in card slot 1, and port 1 on that board. 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 four 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: TLM 4 PUMP CONTROLLER, PUMPCTRL4TLM

In this application note we take the four pump controller above and add the ability to control it, and to obtain operational status from it over the modem channel. Instead of local setpoints for pump call and off levels, we now use setpoints obtained form the telemetry channel. We also now obtain the tank level from the telemetry channel instead of from a local tank. We changed the pump up control modules to **PumpUpDn** modules so the design can apply to both pump up and pump down applications. Finally, we added **HOA2** modules between the **LeadLagSeq** module and the relay output modules, to give operators at the master site HOA type control over the pumps. A block diagram of the control strategy is presented below:

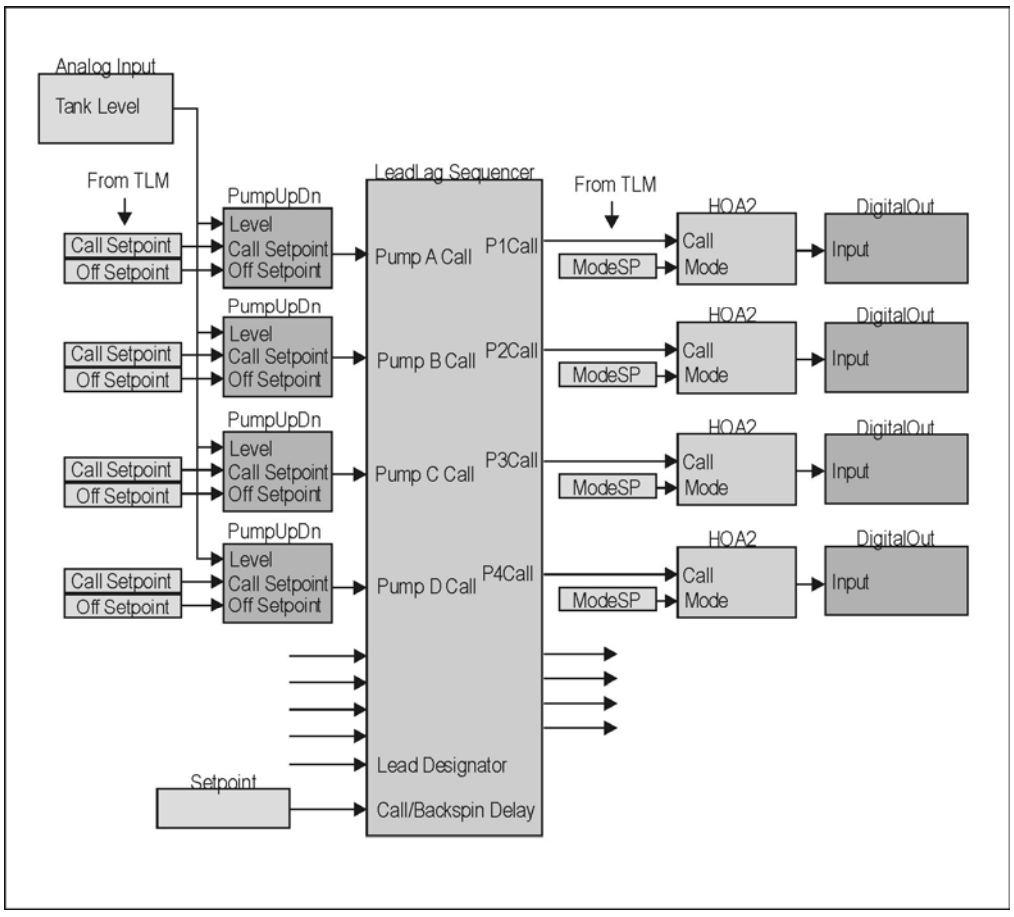


Figure 126 Pump Control Strategy

In this application note, we wish the illustrate the setup of the following modules:

- **ComSetup**…establishes modem communications parameters
- RX and TX arrays…set the communications formats
- **PumpUpDn**…issues pump call based on level comparison and up/down selection
- **HOA2**…enables operator control of pump HAND/OFF/AUTO setting

General Operation

This program reads a tank level from the receive telemetry array obtained from the master site (RX), and compares it with call and off setpoints for four pumps that it has also received from the master site, using four **PumpUpDn** modules. It controls up to four pumps based upon the results of those comparisons. It uses the **LeadLagSeq** module to perform lead pump rotation and call and backspin delays. The outputs of the sequencer are fed to **HOA2** modules, whose HAND/Off/AUTO states are obtained from the RX array. Their outputs provide final output to the **Digital output** modules (relays). Those outputs are also compared, using AlrmMismatch modules, to pump run statuses to generate pump fail statuses, which are placed in the TX array for transmission to the master site in response to master polls. The program also generates power fail, and low battery voltage alarms; and measures battery voltage and on board temperature to send to the master.

This program is compatible with a polling master program, such as the PollMaster program in an application note below. It 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 set in the **ComSetup** module to a value of 3.

Controls Setup

PumpUpDn Module Setup

The PumpUpDn module enables this example to be used for both pump up and pump down applications. Inputs are set as follows:

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:

- 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; from telemetry.
- Mode: 0=pump up, 1 down…integer choice of: 0=pump up, 1=pump down, taken from local setpoint.
- Call setpoint…floating point level to cause pump to switch on; from telemetry
- Off setpoint…floating point level to cause pump to switch off from telemetry.
- Call steady delay sec…floating point time in seconds that pump call condition must remain true before the pump will be called. Set to one second.
- Off steady delay sec…floating point time in seconds that pump off condition must remain true before the pump will be turned off. Set to one second.

Sample Applications

LeadLagSeq Module

In this application, the **LeadLagSeq** 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 **LeadLagSeq** module's call inputs. The sequencer determines how many pumps to control based on how many call inputs have been configured.

HOA2 Module

The **HOA2** module provides the functionality of a hardware hand/off/auto switch with a lockout function. The lockout feature is unused in this example. A single numeric value controls the position of the switch as follows:

- State=0 (off): output is off
- State=1 (hand): output is on
- State=2 (auto): call input is passed to call output

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. Taken from the output of the LeadLag sequencer.
- State input…integer input: 0=off, 1=hand, 2=auto, taken from telemetry words.
- Lockout input…status input that when on will turn off the output independently of any other input. Unused.

Communications Setup

The modules used to setup communications are the **ComSetup** module, which initializes the modem; and the RX and TX arrays, which hold received data and data to be transmitted, respectively. These modules are discussed below:

ComSetup

ComSetup module input properties for this application are presented below (this is a repetition of the setup of the above application note):

Inputs:

- Baud (50-56,000)…integer specifying any baud rate from 50 to 56,000 baud. Set to 300 baud for radio or phone line use. Could also use 1200 baud here for more speed. Higher baud rates would require an external modem or use of a radio with internal modem.
- Wd length (5-8)…integer specifying word length. Set to 8 bits.
- Parity…integer specifying parity choice: 0=none, 1=odd, 2=even, 3=mark, 4=space. Set to 0=none.
- Stop bits (1,2)…integer specifying number of stop bits per word. Set to 1.
- Tone use…integer designating which tone set the modem is to use: 1=originate, 2=answer, 3=low tones. Set to 3 for low tones, best for R9 to R9 communications over leased lines or radio.
- Address (1-255)…integer address for this port on the network. Set to 3.
- Mode (1-7)…integer to select communication protocol. Set to 2, RUG9 protocol.

- TX delay tenths of sec…integer to set delay between the time the RUG5/9 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.
- Trigger to install setup…status trigger input. This input TRUE causes this module's setup to be installed in the designated port. Set to install on boot up.
- UART (0,2,4)…integer to establish connection between the UART and other hardware on the modem board. Set to mode 4 to connect to audio radios or to 4-wire leased line phone systems.
- Rings to answer…integer sets the number of rings that must be counted on the 2-wire channel before the unit answers the line (goes off hook). Left blank.
- Com flags…integer to set hardware handshaking: 0=no handshaking, 1=use RTS/CTS handshaking. Left blank.

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.

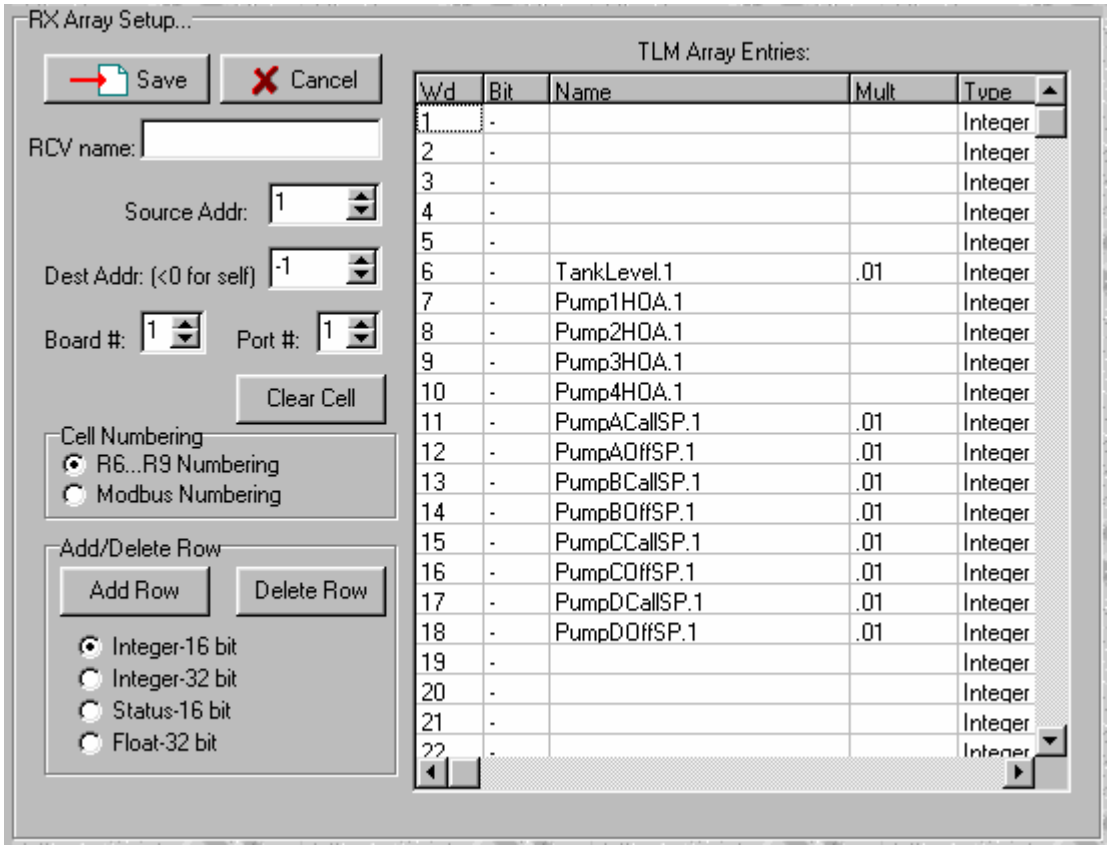


Figure 127 Receive (RX) Array Setup

Notice that this array references source address=1, destination address=-1, port=1, board=1. Board and port numbers refer to board number 1 in the card cage, where our modem board resides, and port number 1 on that board. Source address=1 means that this array will accept data from address number 1, the master site. Destination address=-1 means that this array is to accept messages destined for us, no matter what our address is set to. Even though this application has a different address than that of the tank site, above, the destination address stays set to –1. The actual address of this site is set in the **ComSetup** module. Notice that there are 13 entries in this array. 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 master site. The multiplier of .01 for nine of these variables, and the designation of integer for all of them, indicate that the system is to read each of these as an integer from the receive buffer and multiply by .01 for certain measurements before storing the result in the receive database. At the master, before transmission, each setpoint is multiplied by 100.0 before being sent to this site. The multiplier preserves two places 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.

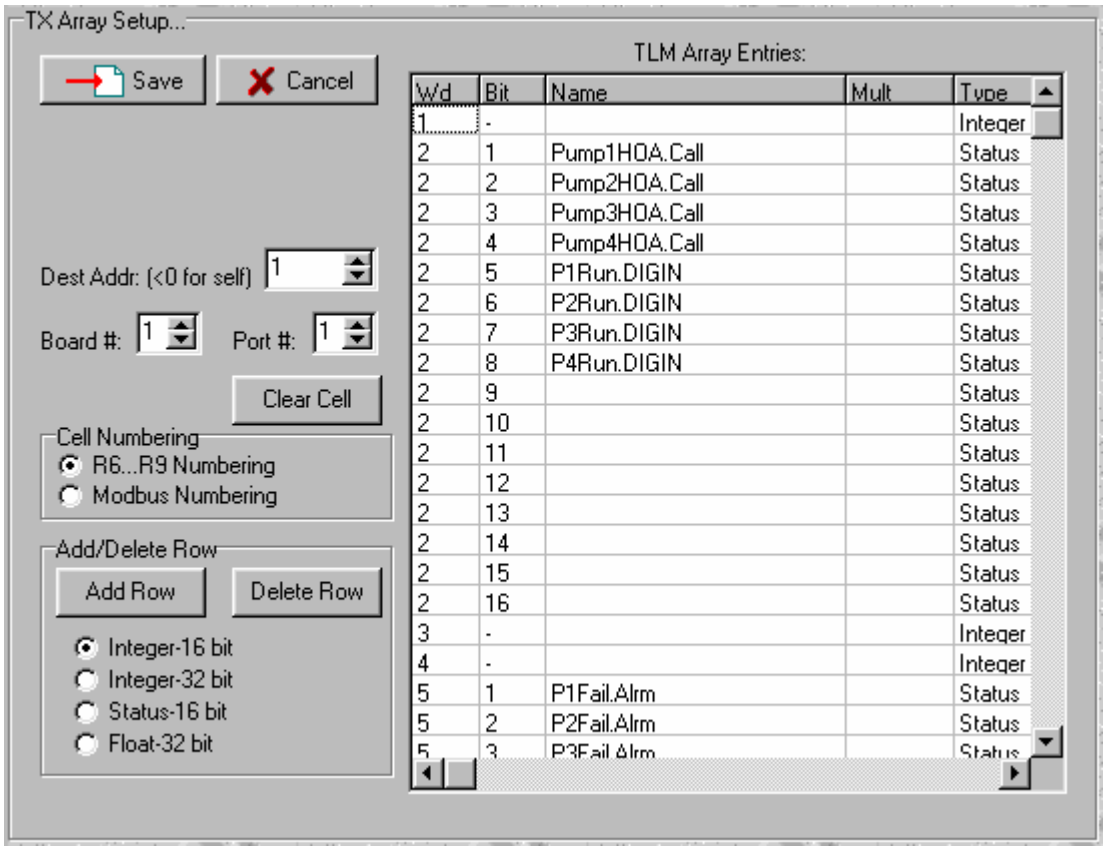


Figure 128 Transmit Array Pump Status Setup

Here, there is no source address, since we are the source. Board number and port number refer to the modem board in card slot 1, and port 1 on that board. 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 eight call and run statuses in word two, and six alarm statuses in word five. These were all dragged in from the status database and dropped into the array. The figure below presents the floating point section of the transmit array:

Sample Applications

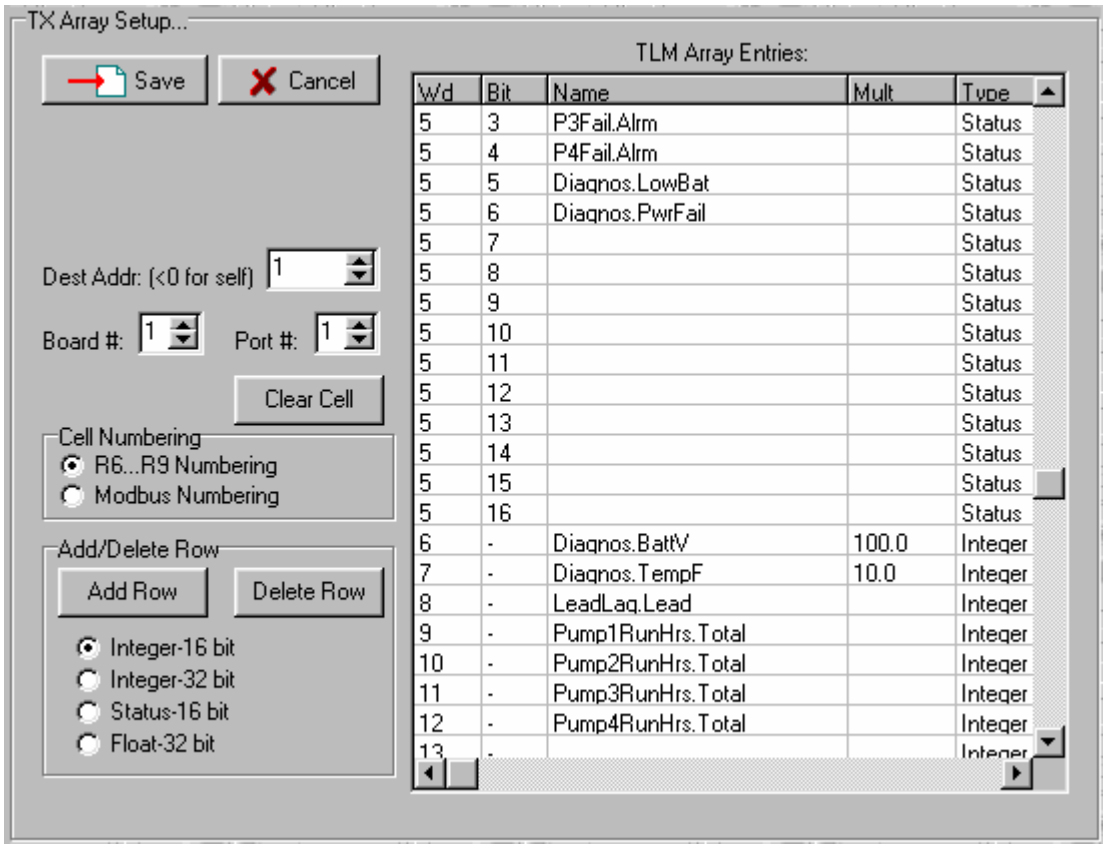


Figure 129 Transmit Array Floating Point Entries

Here, there are seven values we want to send the master. Two values are designated to be multiplied by a factor (100.0 or 10.0), and then be sent as integers. The rest are sent as integers, even though the pump running times are obtained 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 battery voltage above, then the value that we can report is limited to the range of –327.67 to +327.6. We chose not to multiply the pump running times by factors so that we can get the full 32,768 hour range for pump running times.

AP NOTE #5: POLLING MASTER, POLLMASTER

Introduction

This application note describes the design and operation of the PollMaster program, which implements the polling master site of a three-site system consisting of the master, a tank site, and a pump station. In addition to polling the two remote sites, this master also interfaces with a master SCADA program running on a PC doing Modbus polling. The tank site and pump station programs are described in the two preceding application notes. The diagram below presents the system layout.

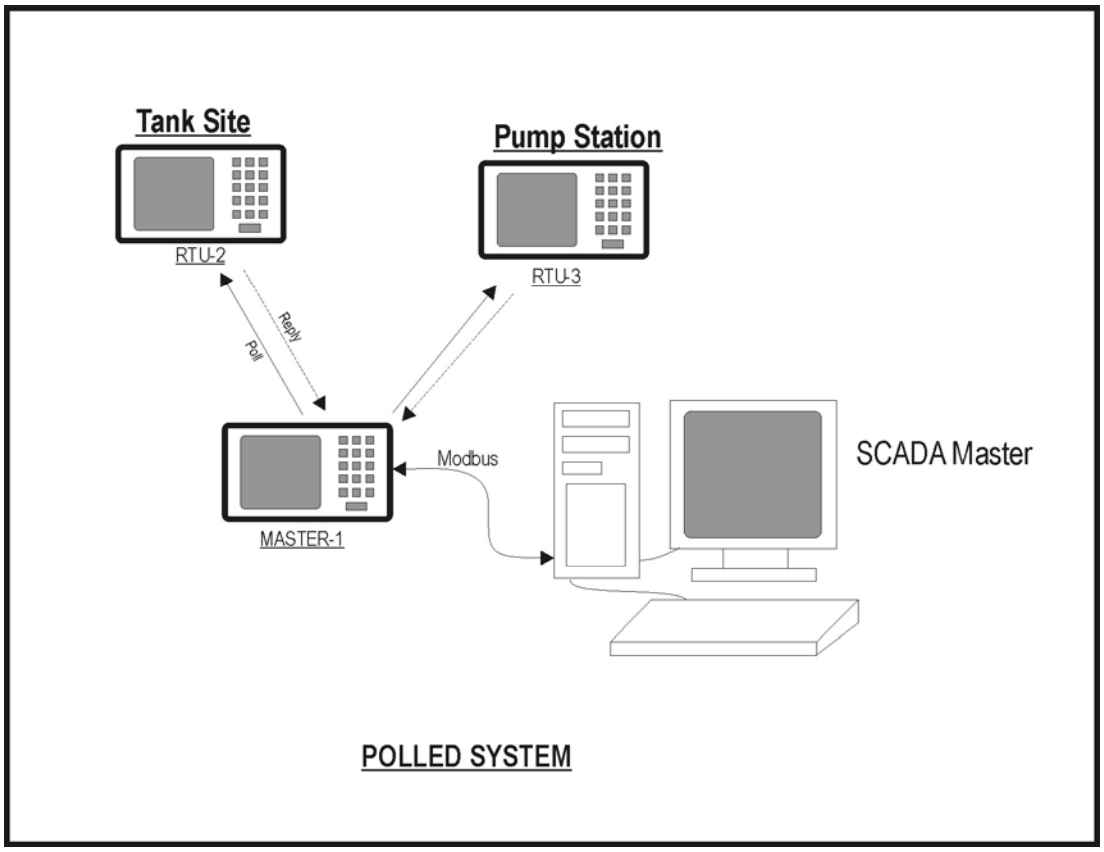


Figure 130 Polling Master System Diagram

In this application note, we wish to illustrate the following modules:

- **Poll**…formats and issues polls to other sites
- **SequenPoll**…controls poll timing and does some statistics gathering
- **ComSetup**…establishes serial port communications settings
- **WriteTableRow**…writes values to a row of a table
- **ReadTableRowInt**…reads one row of a table
- **Tables**…used to hold arrays of data for many uses. Used here to hold polling information, and to hold outgoing setpoints in lieu of modbus received data.
- **CounterStack**…stack of 5 counters, used here to count communications activity

Sample Applications

- **FlipFlop**…two state logic storage/toggle element, used here to select whether modbus or local data are to be sent to the remotes.

Operation

This unit's main job is to deal with communications involving the RTU's and the SCADA master. Two polling operations occur simultaneously…the master RUG5/9 polls the two field RTU's to gather field data; and the master SCADA system polls the master RUG5/9 to gather data that it has collected from the field. Other I/O present in this unit could be used but is not engaged in this application note.

RTU polling by the master RUG5/9 uses the onboard modem in slot 1at 300 baud, so it is compatible with phone lines or audio radios. Polling occurs at approximately one poll each 5 seconds and is controlled by the **SequenPoll** module. Information to set polling data that varies from station to station is contained in the PollTable table, and used by the **SequenPoll** module. Polls and receptions are counted by the **CounterStack** module and saved in the ComStats table. The ratio of receptions divided by polls is used to provide communications statistics. Using tables for these functions enables the polling system to be expanded to a large number of RTU's simply by adding columns to the tables and adjusting a few index properties.

The master SCADA system polls the RUG5/9's RS232 port in slot 7 at 9600 baud independently of and asynchronous to RUG5/9 polling of RTU's. The SCADA polls either request field data, or send commands to the master RUG5/9, which the RUG5/9 then sends to the two RTU's to control pumps and set control and alarm setpoints.

In order to enable the master RUG5/9 to operate independently of the SCADA system, and to illustrate the use of tables, setpoints received by the modbus channel are routed through one column of a table so that a **FlipFlop** module can be used to select whether that column or the other is used as the source of setpoints to the RTU's. This implements a local/remote capability enabling the operator to select either modbus-received setpoints or locally entered setpoints to be sent to the RTU's. (Actually, three tables…PumpStaCtrls, PumpStaHOAs, and TankSiteSetpts are used for this function.)

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.

Sample Applications

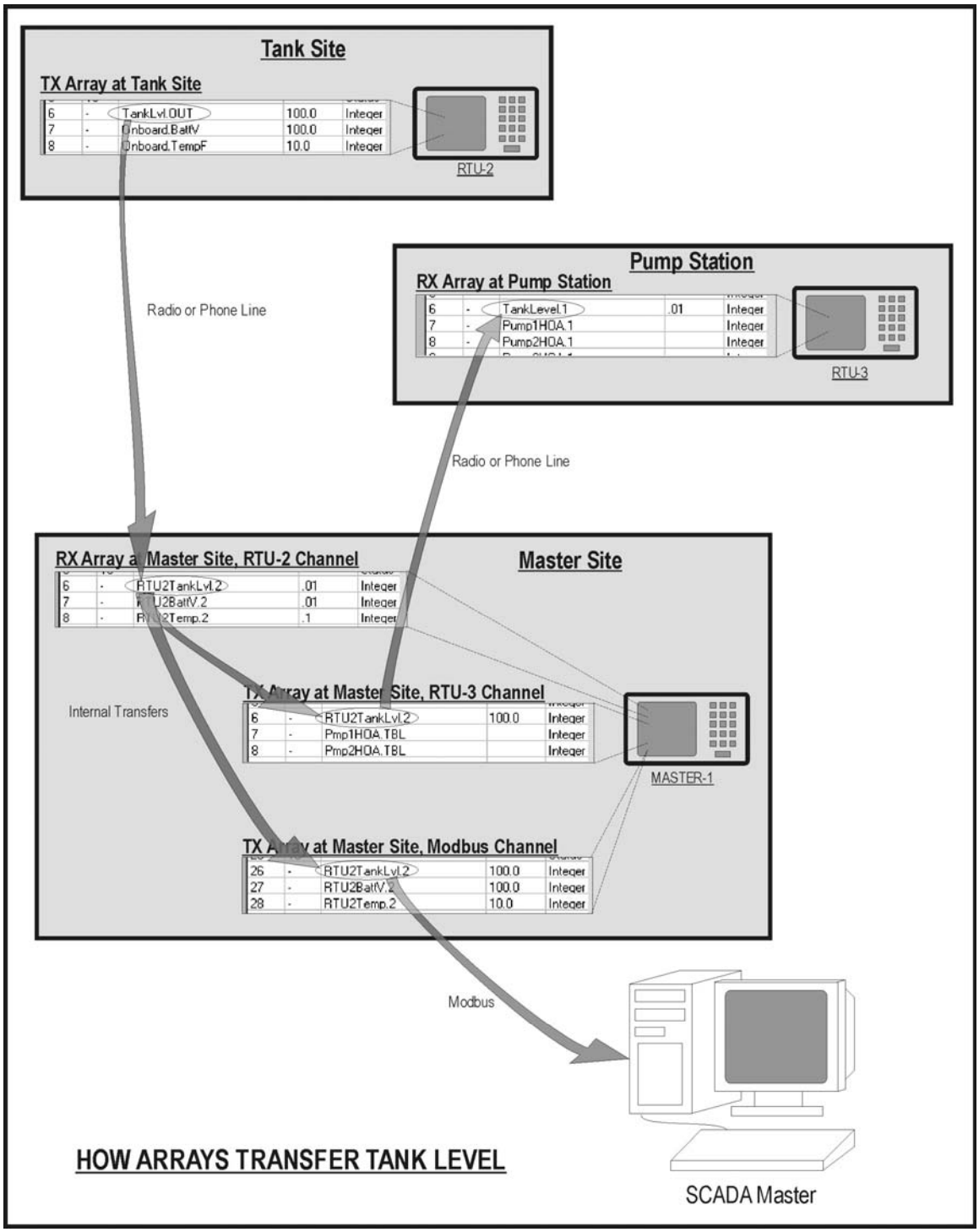


Figure 131 Passing Tank Level Among Sites

Communications Setup for RTU Channel

The modules used to setup communications are the **ComSetup** modules, which initialize the modem; and the RX and TX arrays, which hold received data and data to be transmitted, respectively. These modules are discussed below:

ComSetup for RTU Polling Channel

ComSetup module input properties for the RGU polling channel of this application are presented below:

Inputs:

- Baud (50-56,000)…integer specifying any baud rate from 50 to 56,000 baud. Set to 300 baud for radio or phone line use. Could also use 1200 baud here for more speed. Higher baud rates would require an external modem or use of a radio with internal modem.
- Wd length (5-8)…integer specifying word length. Set to 8 bits.
- Parity…integer specifying parity choice: 0=none, 1=odd, 2=even, 3=mark, 4=space. Set to 0=none.
- Stop bits (1,2)…integer specifying number of stop bits per word. Set to 1.
- Tone use…integer designating which tone set the modem is to use: 1=originate, 2=answer, 3=low tones. Set to 3 for low tones, best for R9 to R9 communications over leased lines or radio.
- Address (1-255)…integer address for this port on the network. Set to 1.
- Mode (1-7)…integer to select communication protocol. Set to 2, RUG5/9 protocol.
- TX delay tenths of sec…integer to set delay between the time the RUG5/9 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.
- Trigger to install setup…status trigger input. Set to install when commanded by the **PollSequen** module.
- UART (0,2,4)…integer to establish connection between the UART and other hardware on the modem board. Set to mode 4 to connect to audio radios or to 4-wire leased line phone systems.
- Rings to answer…integer sets the number of rings that must be counted on the 2-wire channel before the unit answers the line (goes off hook). Left blank.
- Com flags…integer to set hardware handshaking: 0=no handshaking, 1=use RTS/CTS handshaking. Left blank.

RX Array Setup

The figure below presents the receive array setup for data to be received from the tank site, address 2. Notice that the source address is 2, the tank site's address. Destination address is set to –1, to designate that data received from site 2 is to be placed in this array no matter what address this unit has established for this channel. If you compare this array with the transmit array in the tank site program, you will see that they are functionally identical. The names in this array were entered by hand, with the compiler adding the ".2" appendage to designate the data's source address. Multipliers are the reciprocal of those transmitted by the tank site to undo its multipliers. For example, at the tank site, before transmission, the tank level is multiplied by the value 100. Here, we must multiply by 0.01 to recover the original measurement.

Sample Applications

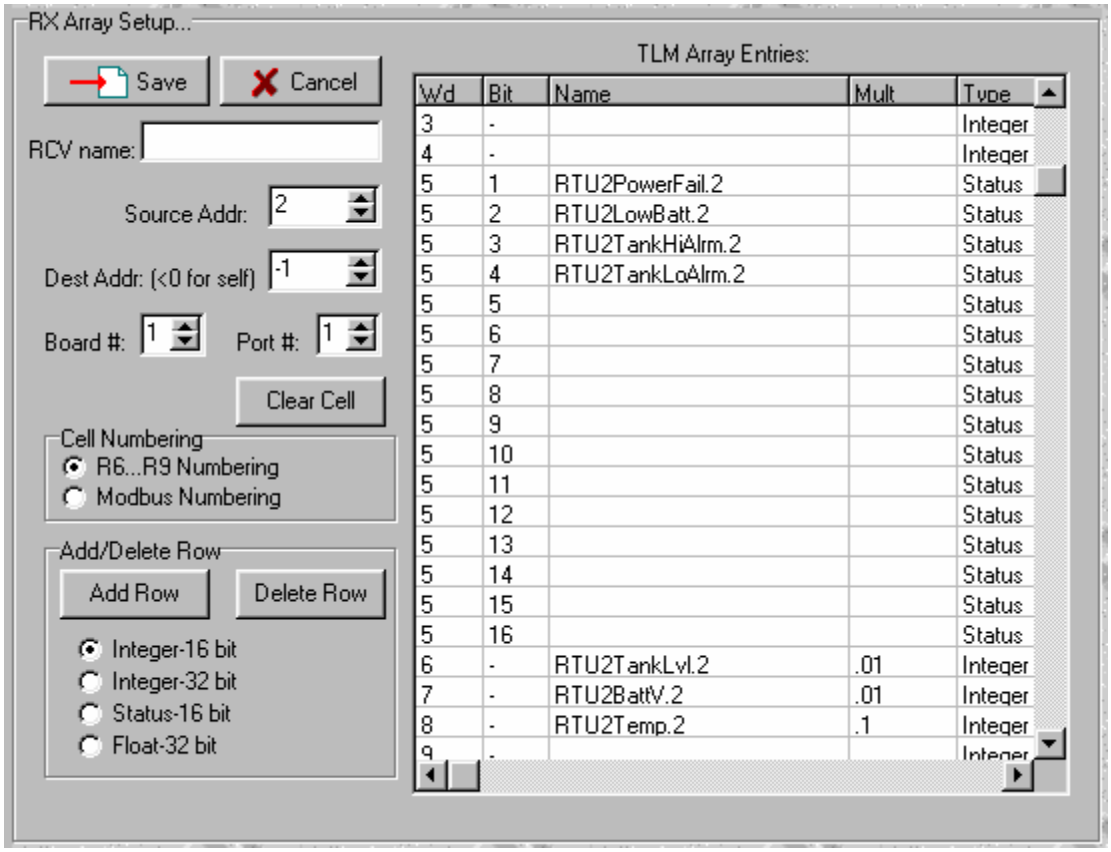


Figure 132 Receive Array Setup for Data from Tank RTU

Similarly, the receive array for data to be received from the pump station must be designed in close correspondence with the transmit array present in that site's configuration file.

TX Array Setup

The transmit array setup for data destined to the tank site is presented below. Here, the destination address is set to 2 to designate the address of the tank site RTU. Board and port number set to 1 refer to the only port on the modem board in I/O slot 1. The two entries in this table are tank high and low alarm setpoints taken from the table named TankSiteSetpts by dragging from the floating point database. If you compare this table with the tank site's receive array, you will see that they are functionally equivalent, except that the names are slightly different and the multipliers are the reciprocal of each other. For example, the high alarm setpoint in word 6 of this table is multiplied by 10.0 to preserve one place to the right of the decimal point before transmission as an integer. Upon reception at the tank site, it is multiplied by 0.1 to recover the original setpoint.

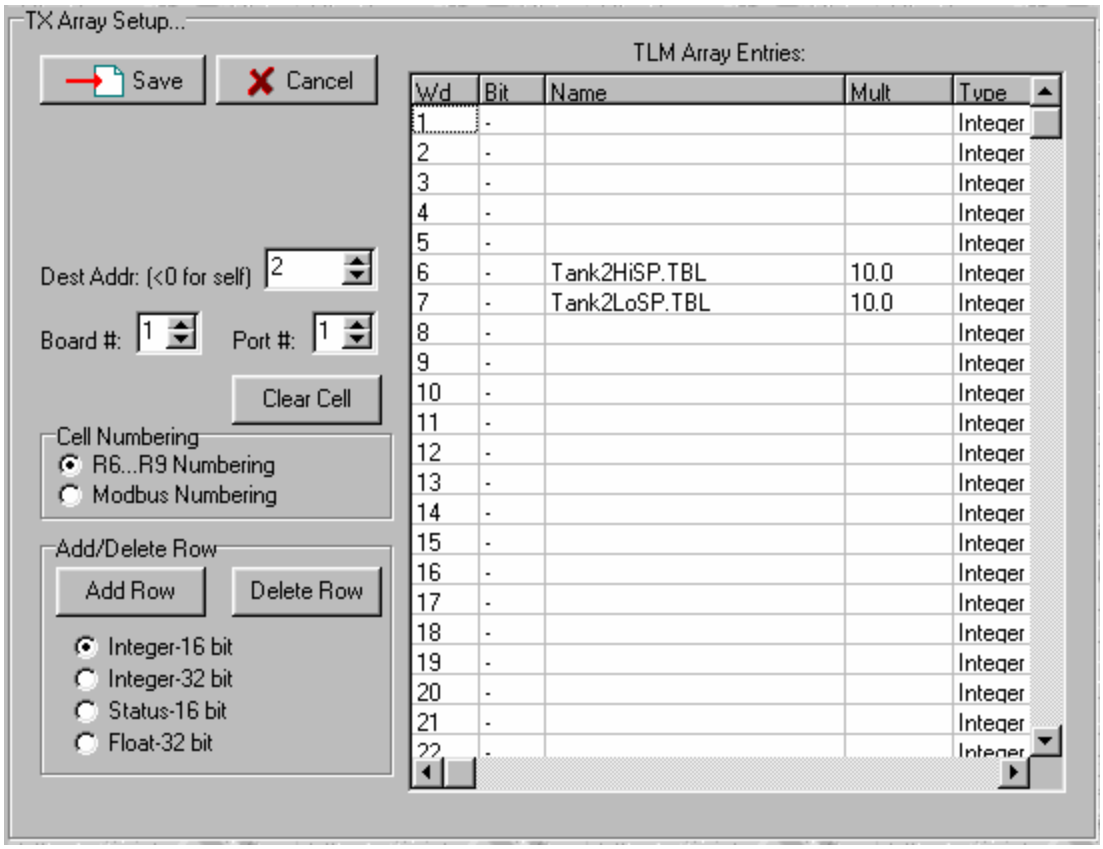


Figure 133 Transmit Array Setup for Data destined to the Tank Site

Setup of the transmit array destined for the pump site similarly corresponds to the entries in the receive array at the pump site, with multipliers at each site the reciprocal of those at the other.

Communications Setup for Modbus Channel

Communications for the Modbus channel is completely independent of the RUG5/9 polling channel and so requires its own ComSetup module and RX and TX arrays.

ComSetup for Modbus Channel

ComSetup module input properties for the modbus channel of this application are presented below:

Inputs:

- Baud (50-56,000)…integer specifying any baud rate from 50 to 56,000 baud. Set to 9600 baud for RS232 use.
- Wd length (5-8)…integer specifying word length. Set to 8 bits.
- Parity…integer specifying parity choice: 0=none, 1=odd, 2=even, 3=mark, 4=space. Set to 0=none.
- Stop bits (1,2)…integer specifying number of stop bits per word. Set to 1.
- Tone use…integer designating which tone set the modem is to use: 1=originate, 2=answer, 3=low tones. Set to 3.
- Address (1-255)…integer address for this port on the network. Set to 1, this unit's address as a slave in the Modbus network.
- Mode (1-7)…integer to select communication protocol. Set to 6, Modbus slave2 protocol.

Sample Applications

- TX delay tenths of sec…integer to set delay between the time the RUG5/9 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 0, no delay.
- Trigger to install setup…status trigger input. This input TRUE causes this module's setup to be installed in the designated port. Set to install on boot up.
- UART (0,2,4)…integer to establish connection between the UART and other hardware on the modem board. Set to mode 0 for RS232 channel.
- Rings to answer…integer sets the number of rings that must be counted on the 2-wire channel before the unit answers the line (goes off hook). Left blank.
- Com flags…integer to set hardware handshaking: 0=no handshaking, 1=use RTS/CTS handshaking. Left blank.

Modbus Communications Array Setup

A single transmit array and a single receive array are necessary to support the Modbus channel, i.e., to enable SCADA master software to poll the RUG5/9 for field data, and to enable the SCADA software to send commands and setpoints to the RUG5/9 for later transmission to the remote RTU's. Signals from the remote RTU receive arrays must be dragged into the Modbus transmit array; and signals in the Modbus receive array must be dragged into the RTU transmit arrays. In this example, we have allocated 20 words per RTU in the Modbus array, and spaced them evenly. This is arbitrary…there is no requirement to keep RTU data together in the Modbus array, or to allocate space in the array evenly among the RTU's. The figure below presents the correspondence between the remote RTU receive arrays and the Modbus TX array for this example. Notice that the data points can be kept in order as in the tank site data, or data can be rearranged as in the case of the pump station data. Notice also the reciprocal relationship between the RX array multipliers and the TX array multipliers. With the **ComSetup** module configured as listed above and the RX and TX arrays defined, a Modbus master SCADA system could poll the RUG5/9 on the serial port in slot 7 and obtain the data measurements and statuses contained in the TX array.

Sample Applications

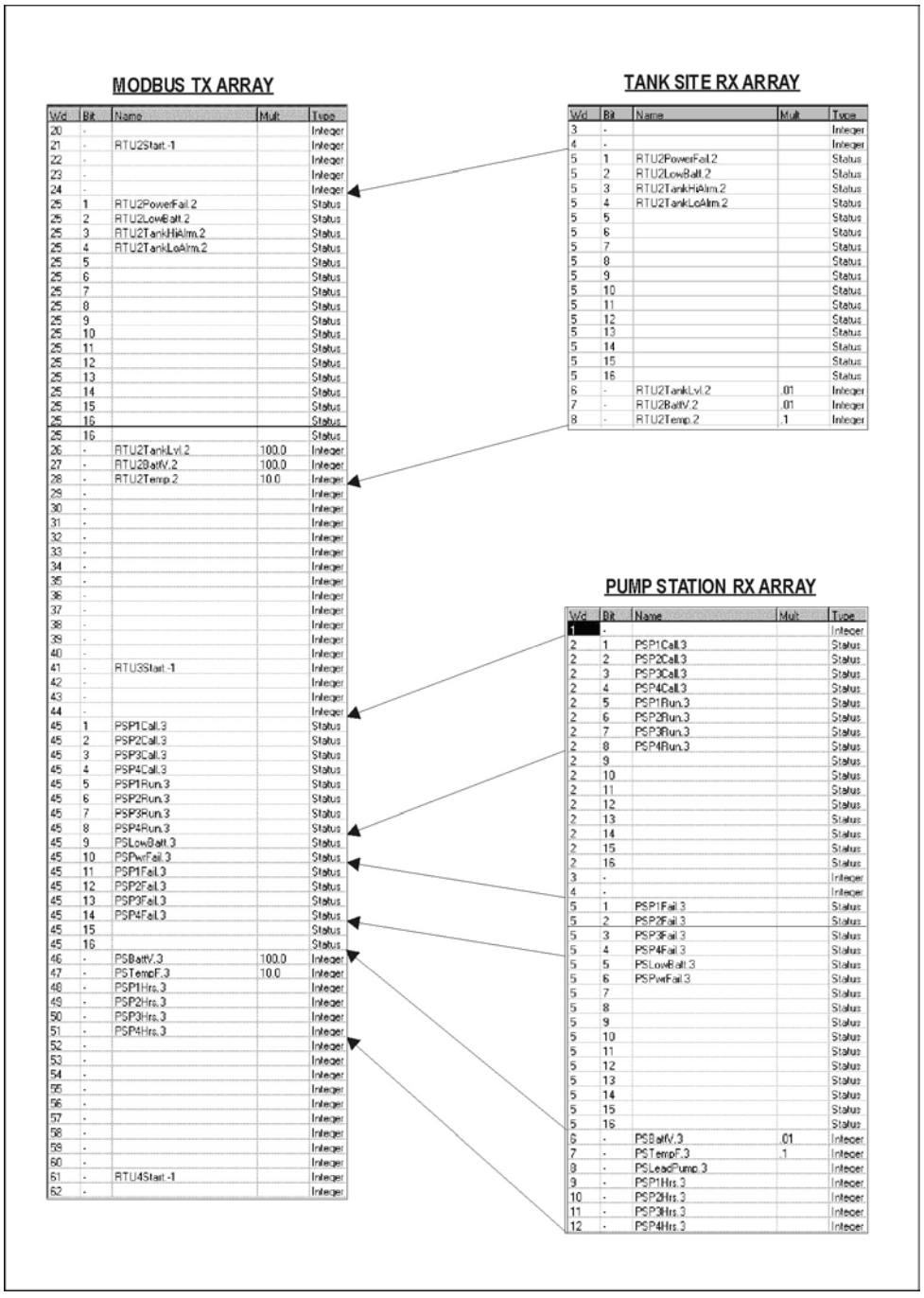


Figure 134 Dragging Data from RTU RX Arrays to Modbus Array

Communications Statistics Setup

In this application, we use some tables to hold communications statistics; and we use some **ReadTableRow** modules to read some rows of those tables. The RUG5/9 table structure enables you to write selected cells of a single column at a time, and to read all cells of a single column at a time. But, the tables provide no means by which you can read an entire row at once. That's what the **ReadTableRow** module does, up to 10 cells from one row. The figure below presents a block diagram of the

communications system for RTU polling, including the modules that read the tables to make various rows available to the display.

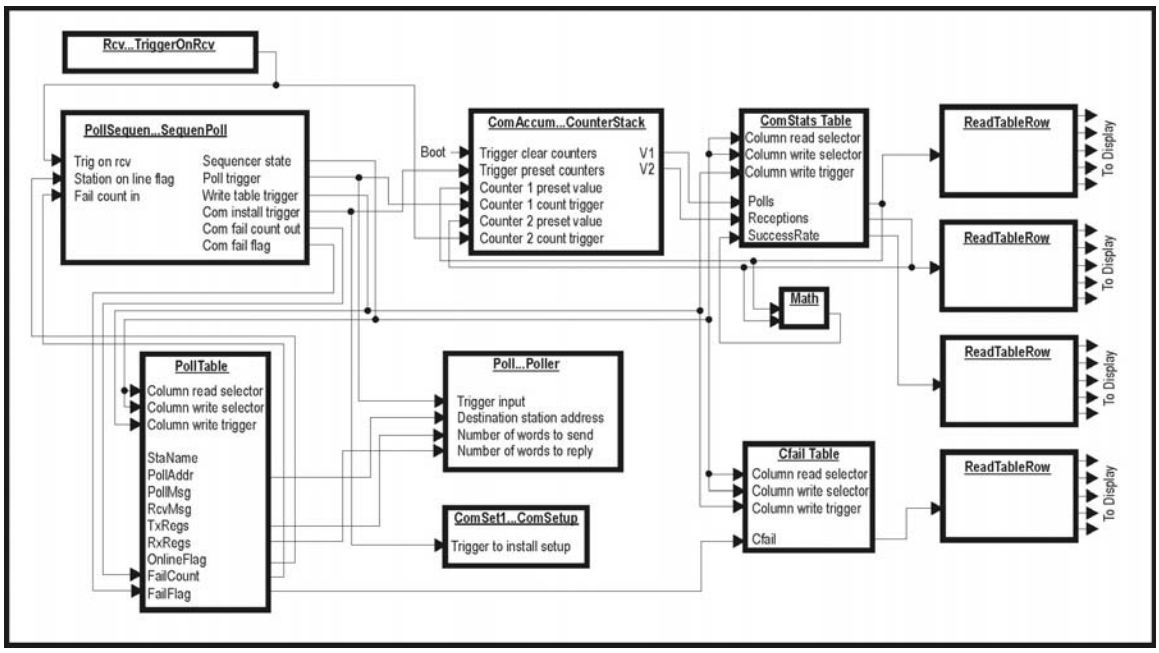


Figure 135 Communications Control and Statistics Block Diagram

Basically, in this example, the **PollSequen** module controls the other modules and obtains some RTU-specific data from the table named PollTable. As it prepares to poll a site, it sets its state output to select a column on all tables associated with polling and poll statistics. On the next scan, all tables will have asserted their outputs corresponding to the selected column. The **PollSequen** module then triggers the **ComSetup** module to install its setup into the modem board. At the same time, the counter stack presets its counters to the outputs from the ComStats table. On the following scan, the **PollSequen** module triggers the **Poll** module to send the poll. Inputs to the **Poll** module are obtained from the PollTable. This same trigger is counted by the first counter in the **CounterStack** to keep track of the poll count for the RTU being polled. The system then waits for the polled RTU to reply as indicated by a trigger out of the **TriggerOnRcv** module. If the RTU replies, the **PollSequen** module clears the com fail flag and resets its com fail counter. Also, the **CounterStack's** second counter increments to keep track of number of receptions. If no trigger is received by the time the **PollSequen** module's reception timer times out, then the **PollSequen** module decrements its com fail counter and tests to see if a com fail should be declared. On the next scan, the **PollSequen** module issues its table write trigger to make all tables write any new values associated with the last poll. At that time, the PollTable, ComStats and ComFail tables will record their entries for the station that was just polled. On the next scan, the process will start over for the next station in the polling cycle. Independently of the polling process, the **ReadTableRow** modules continuously read the ComStats and Cfail tables to make the statistics and com fail flags held by those tables available to the LCD display.

Implementing Local/Remote Setpoint Entry

In order that this application program can be used without a SCADA master program, it implements a way by which the operator can select for setpoints destined for the RTU's to come from either the entries in the Modbus RX array, for remote control, or from tables in the master, for local control. We accomplish this by using a **FlipFlop** module to give the operator a local/remote status he can toggle using the Key1.trigger signal. The key here is to route the Flipflop's inverted output back to its data input,

so its data input is always the inverse of its present state. That way, each time its clocked by the clock input, it will toggle its state. The **FlipFlop** module setup panel is shown below.

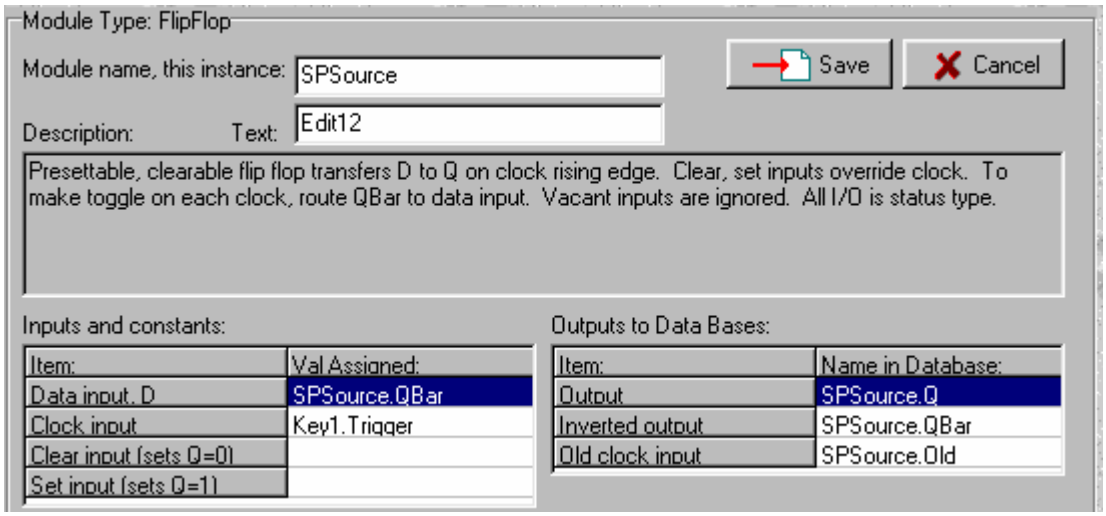


Figure 136 FlipFlop Module Setup for Toggling

Since the **FlipFlop**'s output value range is 0 or 1, and we need a range of 1 to 2 to control a table's read column, we add a value of 1 to the **FlipFlop**'s output using a Y=A+B module. That module's output, named SPSourceAdder.Out, is used to select either a local or remote data column in three tables, named PumpStaCtrls, PumpStaHOAs, and TankSiteSetpts. The table for selecting pump station call and off setpoints is presented below. Notice that we have dragged entries into the Modbus column from the Modbus RX array, and have left the corresponding entries in the local column blank. By doing this, the local entries remain editable from the RUG5/9's LCD display/keyboard module, whereas the Modbus entries are taken from the Modbus RX array. Therefore, the outputs of this table, which are dragged into the pump station's TX array, are either taken from the locally entered column, or are obtained from the Modbus RX array. This table simply acts as a big selector switch. The other two tables act in the same manner.

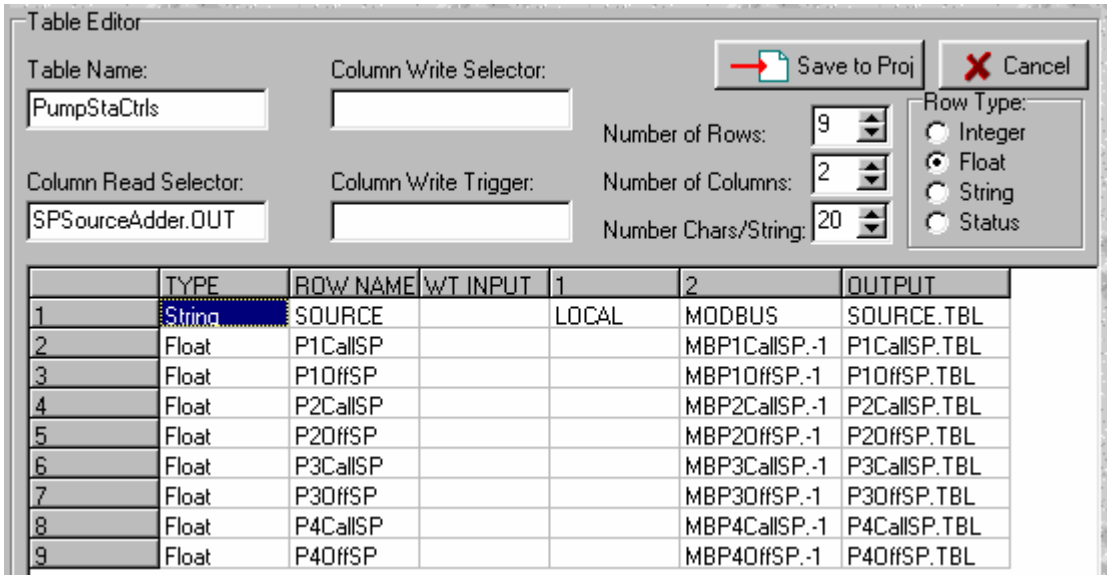


Figure 137 Local/Remote Setpoint Selection Table

AP NOTE #6: STAND ALONE AUTODIALER

Introduction

The RUG5/9 operating system supports speech autodial/autoanswer operations using the dialer card. The dialer card can hold up to 254 messages or phrases, with a total speech storage capacity of 12 minutes. This section describes how you set up a stand alone speech autodialer application. This application refers to the file named "Autodialer", which performs the following functions:

- Dials from a list of operators when an alarm occurs
- Includes pagers in dialing list
- Reports one alarm list and three other spoken reports
- Answers incoming calls
- Supports remotely changing setpoints

Module interconnection for speech applications is somewhat involved. The following diagram illustrates speech module interconnections and connections to the speech reports. For this application note, we have included illustrations of how to say the day of week, to show how to present a speech message selected by an integer, and have simply used three digital inputs as the alarm sources. Also, the two non-alarm reports are, in effect, place holders with meaningless data. In order to produce a practical application, you will need to make the following changes:

- Expand operator list to include all operators/pagers
- Produce practical alarms and include them in the alarm report
- Add to or delete analog reports
- Modify analog reports to present real data
- Add to or delete field changeable setpoints

This application, as well as virtually all speech autodialer applications, uses the following modules:

- **SpSequenDial**…controls sequencing of dialing, reporting, menuing, security codes, etc.
- **SpeechRecPlayDel**…controls recording, playback and deleting individual phrases
- **SpeechDial/Answer**…controls dialing and answering by the dialer board
- Table…holds list of phone numbers, operator security codes, names, etc.
- **Or Gate**…used to insert multiple triggers for speech functions, and to consolidate the end of report signals into the speech sequencer.
- **SequenOut**…used by the **SpSequenDial** module to select one of several reports to trigger.

Sample Applications

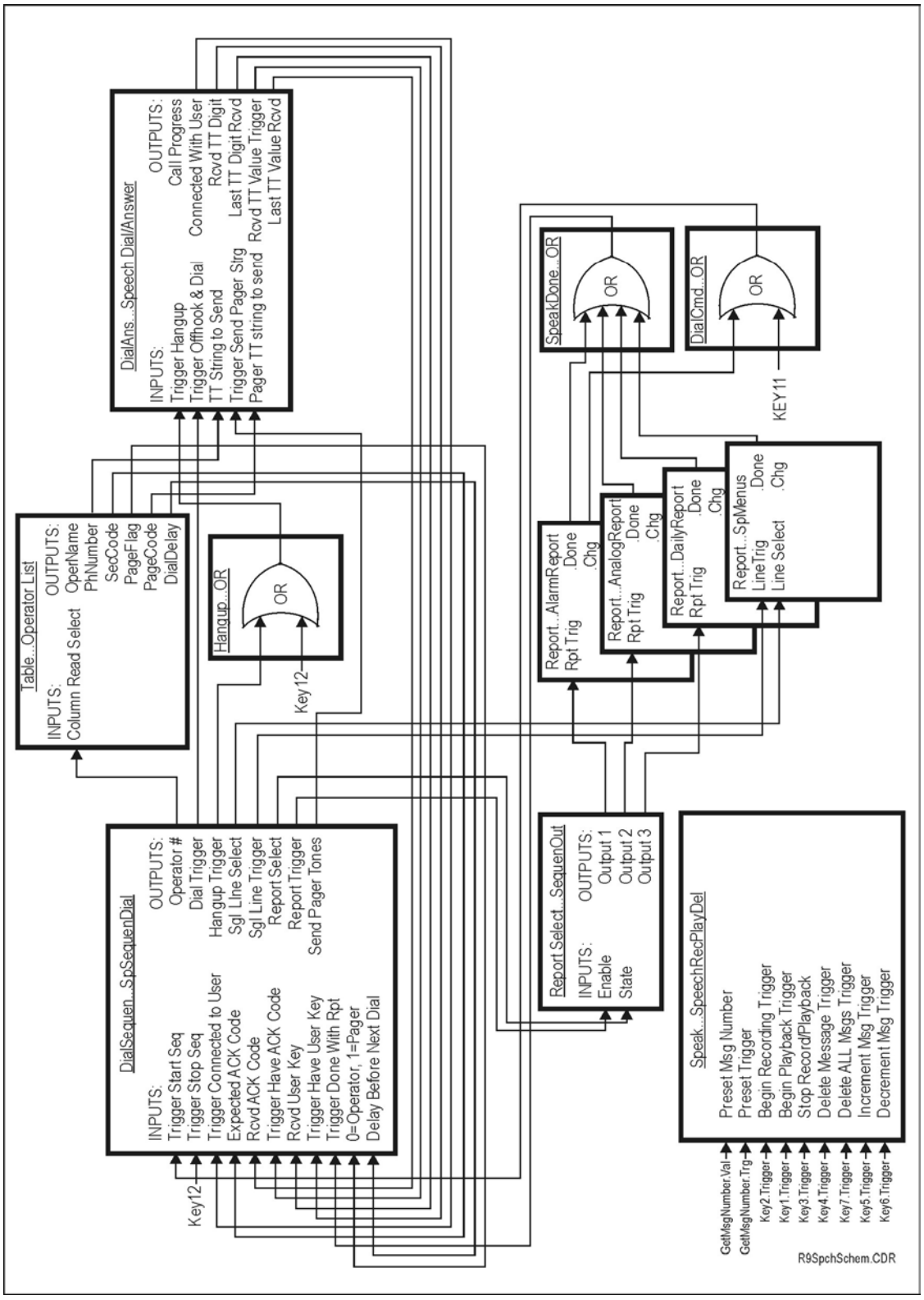


Figure 138 Speech Autodialer Module Interconnection

How the Dialer Functions

The dialer can either dial an operator, whose phone number is obtained from the OperatorList table, or it can answer an incoming call. Once an operator is on line with the dialer, dialer functions are virtually the same whether he called the dialer or it called him. The first message he will hear is "This is the RUGID autodialer" (your welcome message), followed by "Hit key 1 for alarms, 2 for analog report, 3 for daily report, 9 to adjust setpoints, star to acknowledge or pound to exit" (the keystroke prompt). The dialer will then wait for a keystroke. What happens next depends on the keystroke he hits:

- No keystroke…unit will repeat the keystroke prompt until the operator hits a key or until the retry count is exceeded. If no keystroke, the unit will say good bye and hang up.
- Key 1-8…unit will trigger one of 8 reports and start it talking. When the report finishes, the unit will repeat the keystroke prompt.
- Key 9…unit will enter setpoint changing procedure below. When done, the unit will repeat the keystroke prompt.
- Key *…unit will enter alarm acknowledgment sequence below. When done, the unit will repeat the keystroke prompt. The star key is also used in the setpoint entry procedure to enter a decimal point.
- Key #…unit will say goodbye and hang up. The pound key is generally used as the equivalent of the [ENTER] key to terminate an entry.

Setpoint Changing

If the operator hits key 9 from the keystroke prompt, he will be prompted to enter his security code. If the unit called him, the code it will expect will be his personal code. If he called in, the unit will expect the code assigned to the first operator in the operator table. He must enter the expected code followed by the pound '#' key. If he enters an unacceptable code, he will be returned to the keystroke prompt. After entering the correct security code, he will be prompted to select from one of up to 9 setpoint categories. (His keystroke times 10 will be the message selected from the menu report to prompt him for up to 9 setpoints in that category, see lines 10, 20, 30). The next prompt he will hear will prompt him to select one of up to 9 setpoints from the category he selected. After he selects a setpoint, the particular setpoint prompt will be issued. (See lines 11,12, 21, 22, 31, 32, 33.) The prompt must include the setpoint's present value, since that reference is used as the destination of his new value. After he receives the setpoint prompt, he must either hit the '#' key to leave the setpoint unchanged, or enter the new setpoint value using his keypad. To enter a floating point value, he must use the '*' key as the decimal point, and the '#' key as the ENTER key. There is no provision for entering negative setpoint values.

Alarm Acknowledge Sequence

If the operator hits the star '*' key from the keystroke prompt, he will enter the alarm acknowledgement sequence. He will first be prompted to enter his security code. If the unit called him, the code it will expect will be his personal code. If he called in, the unit will expect the code assigned to the first operator in the operator table. He must enter the expected code followed by the pound '#' key. If he enters an unacceptable code, he will be returned to the keystroke prompt. After entering the correct security code, he will be told his code is accepted, alarms will be acknowledged, and he will be returned to the keystroke prompt.

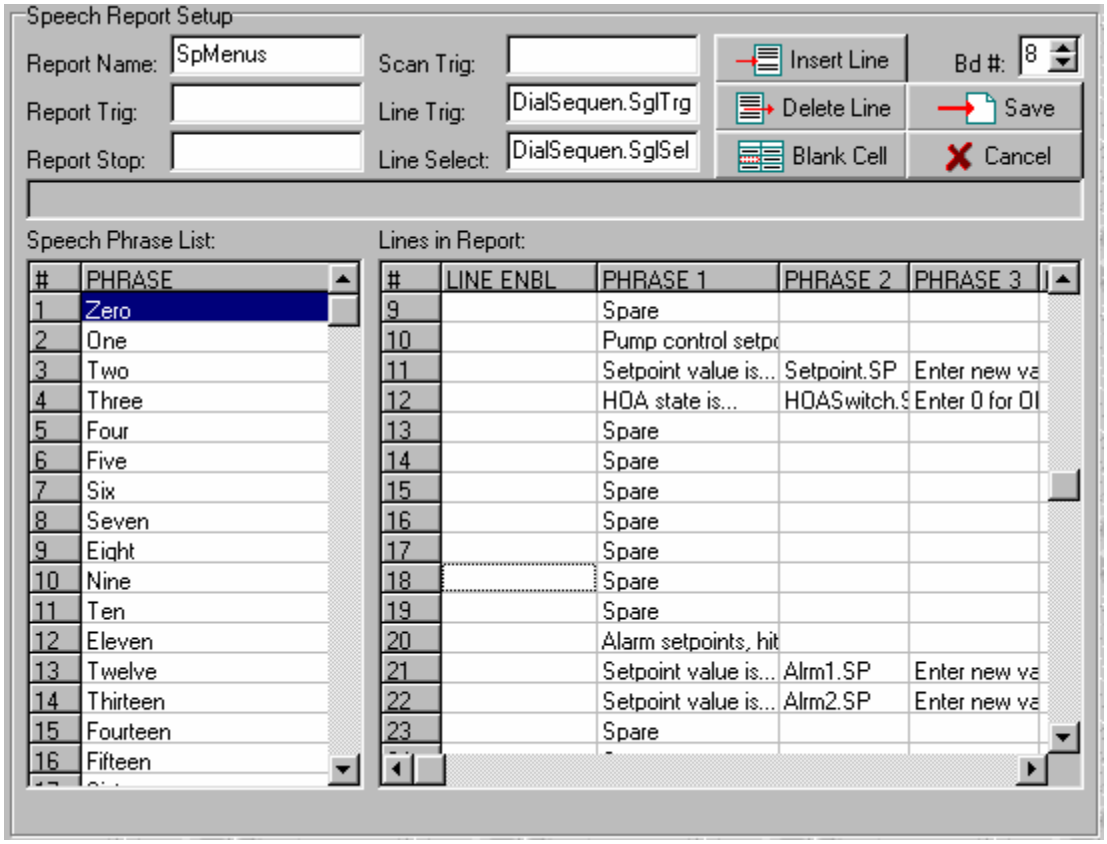


Figure 139 Speech Menu Setup for Entering Setpoints

Speech Reports

Speech reports enable you to define a list of phrases and then connect those phrases together into sentences to report alarms, statuses, values, etc. You can define as many speech reports as flash memory can hold, however, you can only prompt the user verbally for up to 8 reports. Each line of a speech report constitutes the shortest set of speech phrases the RUG5/9 can be made to emit. In practical terms, each line should be a sentence. A report can be made to emit speech in either of two ways. If the report trigger input is triggered, then the entire report will be emitted. If the line trigger input is triggered, then only the designated line will be emitted. In this application, there are four speech reports defined: an alarm report, two miscellaneous reports, and one menu report. The alarm and miscellaneous reports use the report trigger to trigger execution of the entire report. The choice of which report is triggered is determined by the **DialSequen** module, which uses the **SequenOut** module to select one of 10 outputs to enable based on the value of the **DialSequen** module state output. The **DialSequen** module's state output is determined by what key the operator enters after the initial prompt. Therefore, you will want your prompt to the operator to correspond to the order in which reports are connected to the **SequenOut** module. We have chosen the prompt "Press key 1 for alarm report, key 2 for analog report, key 3 for daily report"; so **SequenOut** outputs S1, S2 and S3 must be connected to reports in this order: AlarmReport, AnalogReport and DailyReport. The menu report is a repository of speech menu items that the speech sequencer triggers one line at a time, using the report's line trigger input.

Alarm Report

In this application note, we have defined one alarm report, as shown below.

Sample Applications

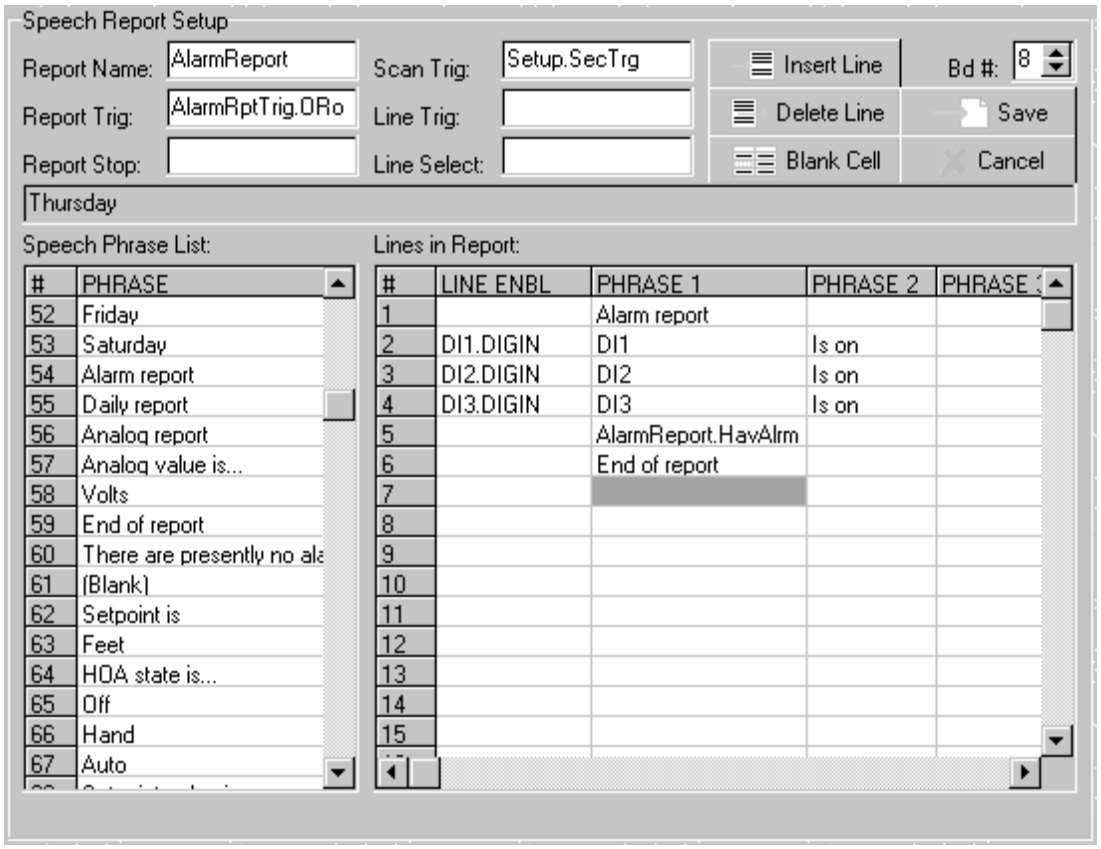


Figure 140 Speech Alarm Report

Notice that the alarms are dragged into the **LINE ENBL** cells at the start of each line. Basically, in any report, if there is an entry in the **LINE ENBL** cell preceding a line, then the line will only be emitted if that item is true at the time of speaking the report. Therefore, in the case of the alarm report below, if the digital input is true, as it is in lines 2, 3, and 4, then that line will be emitted. Otherwise, that line will be skipped. Items in the **LINE ENBL** cells can be any items from the status database. Any line without a **LINE ENBL** entry will be emitted unconditionally. This is the case with lines 1 and 6, which constitute fixed information. In the case of line 5, we are covering the case of presenting an alarm report when there are no alarms active. For example, if an operator calls in and requests to hear the alarm report, and there are no alarms, we would like to tell him "There are presently no alarms". Line 5 does this. It references the **AlarmReport.HavAlrm** status that will be true if any alarm exists, and false if there are no alarms. It uses that status to select between one of two phrases in the phrase list, phrases 60 and 61. Phrase 60, "There are presently no alarms" will be stated if **AlarmReport.HavAlrm** is false (value of 0); phrase 61, (blank) will be stated if it is true (value of 1), i.e., if any alarm is present. To set up that function, you would first drag the **AlarmReport.HavAlrm** status from the status database and drop it into the phrase 1 cell of line 5. Then click on that cell to bring up the format definition page presented below. When first brought up, the **Present Format** entry would show @@@.@@, the default format for variables to be presented. To make it select from a list of phrases in the phrase list, simply drag the phrase corresponding to a value of zero (phrase 60 in this case) from the phrase list and drop it into the **Present Format** box, as shown below.

Sample Applications

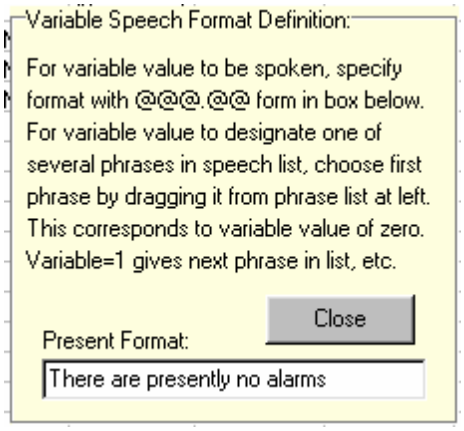


Figure 141 Line 5 Variable Speech Format Setup

In your application, you will probably want to include many more alarms in your list. Simply drag them in to the LINE ENBL cells, enter new speech phrases for them into the phrase list, and then drag the phrases from the phrase list to the alarm report. You can also have more than one alarm report; either by triggering them separately from the SequenOut module, or by triggering each alarm report from Report.Dun output of the previous report. Notice that in the above alarm report, we are using a once per second trigger to trigger the report's alarm scanning. This causes the report to scan its LINE ENBL column once per second, looking for occurrences of new alarms, which it uses at the end of each scan to set its AlarmReport.ChgTrg trigger, and to set or clear its AlarmReport.HavAlrm output.

Miscellaneous Reports

This application has two miscellaneous reports: AnalogReport and DailyReport. There is nothing unusual about these reports; they simply present information to the operator. The analog report presents a pair of values to the user along with phrases identifying the values and engineering units. The daily report, on the other hand, uses a value to select a speech message from the speech phrase list (one of 7 days of the week) to present based on the value of the variable (RTC.DayofWk).

Menu Report

The menu report is used to hold single line messages to be triggered by the speech sequencer during the dialog with the operator. Order is important here. The lines in the menu speech report should be similar in meaning to:

Line 1="This is the RUGID autodialer. (Your welcome message)
Line 2="Hit key 1 for alarms, 2 for analog report, 3 for daily report,…star to acknowledge alarms, poind to exit."
Line 3="Please enter your security code followed by the pound key."
Line 4="Your security code is accepted."
Line 5="Your security code is rejected."
Line 6="Thank you…goodbye."
Line 7=prompt for up to 9 categories of setpoints to change
…
Line 10=prompt for up to 9 setpoints in category 1
Line 11=prompt for first setpoint to change in category 1
Line 12=prompt for second setpoint to change in category 1
…
Line 19=prompt for ninth setpoint to change in category 1
Line 20= prompt for up to 9 setpoints in category 2
Line 21=prompt for first setpoint to change in category 2

…

If your application does not require for the operator to change setpoints from his phone, then messages 7 and above can be omitted. If setpoint changing is required, then you will need message 7, the prompt introducing any categories you need; messages 10, 20, 30…, the prompts for up to 9 setpoints in each of categories 1,2,3… respectively; and finally, messages 11, 12, 13…21, 22, 23… which are the specific setpoint prompts including the present setpoint values. Any messages that you will not use must have a place holder in at least one cell on the line, otherwise the compiler will delete the unused line(s).

Recording, Playing Back and Deleting Speech Phrases

Speech phrases are recorded by speaking into the microphone on the front of the dialer board at the appropriate time. The **SpeechRecPlayDel** module, in the communications tab of the module library, controls message recording, playback and deleting. As illustrated in the typical setup depicted in the figure below, most of its inputs in this application are taken from keys on the keyboard.

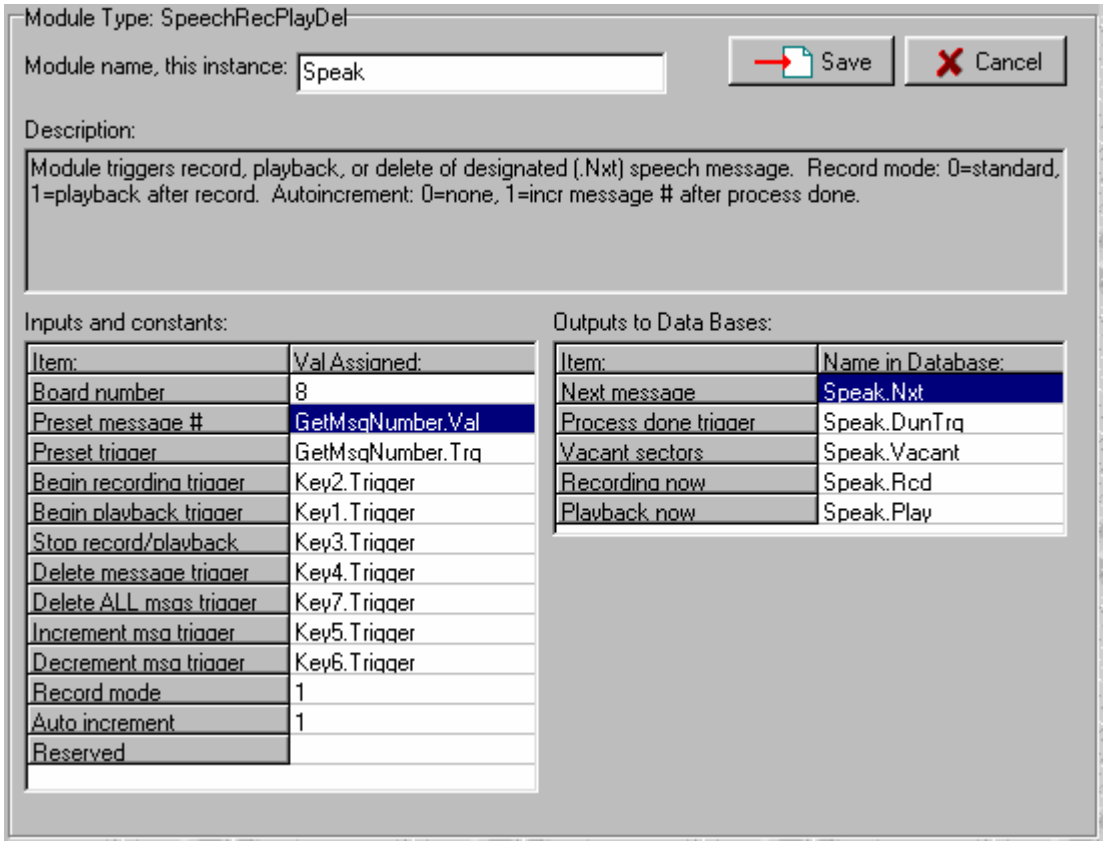


Figure 142 Typical SpeechRecPlayDel Module Setup

The screen called “Main” supports this process in this application, although, once the speech phrases are recorded, there is no necessity to keep the record/playback/delete capability in the project. It is presented below and corresponds to the use of keys in the **SpeechRecPlayDel** module setup above.

Sample Applications

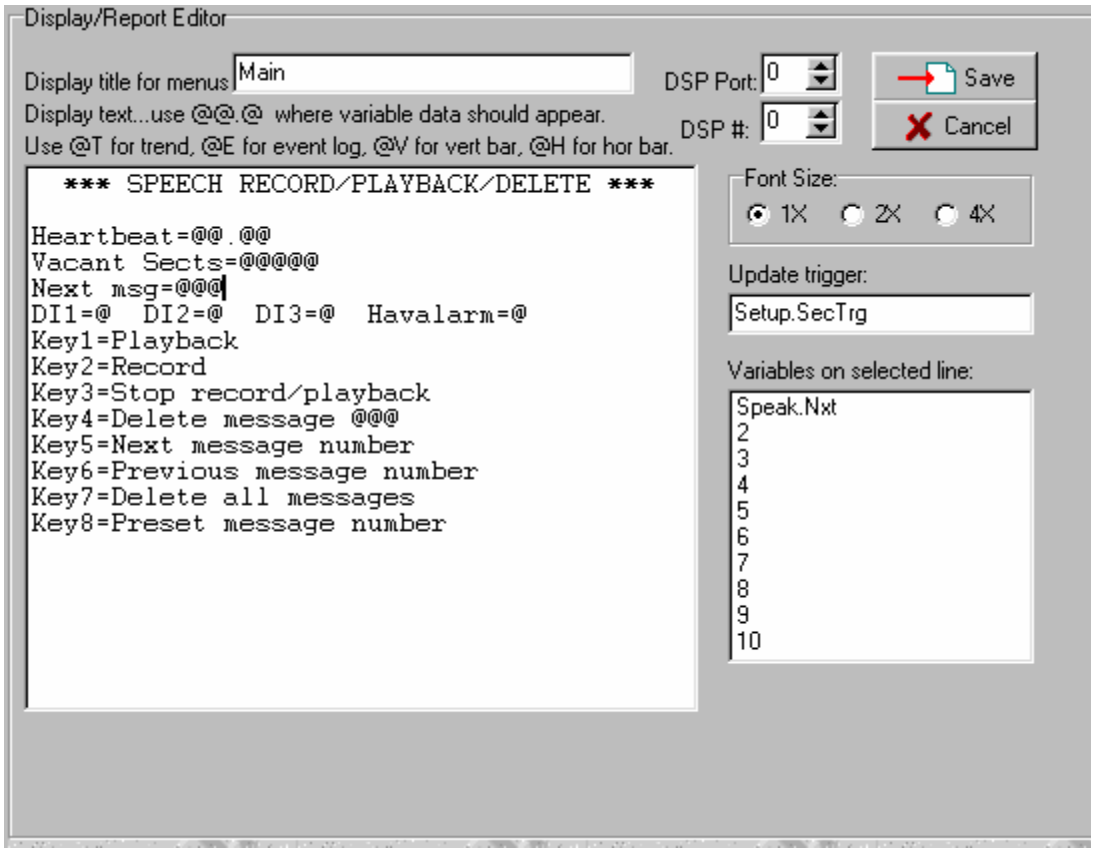


Figure 143 Typical Speech Record/Playback/Delete LCD Screen

Note that the **SpeechRecPlayDel** module maintains an integer value for the next message to be effected. That integer (Speak.Nxt in the example above) can be incremented, decremented or preset to a value. Whenever a command to record, playback, or delete a single message is given, the message addressed by Speak.Nxt will be the one upon which the action is taken. Also, the command to delete all messages is very convenient for starting over, but is also dangerous in that all messages can be erased by accident using it. Finally, note that the display above shows the number of vacant sectors on the speech board. The sector is the minimum erasable chunk of flash memory on the dialer board and can hold 1.42 seconds of speech. Therefore, the board can hold a total of 12 minutes of speech (511 sectors * 1.42 sec.). Each message, no matter how short will use up at least 1.42 seconds of storage. You might want to multiply the vacant sectors by 1.42 to display the seconds of speech remaining unrecorded instead of the number of vacant sectors.

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 RUG5/9 use. We will first address the use of the RUG5/9'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 RUG5/9, 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 RUG5/9, 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** columns. Once you do that, R9SETUPD will begin polling the RUG5/9 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.

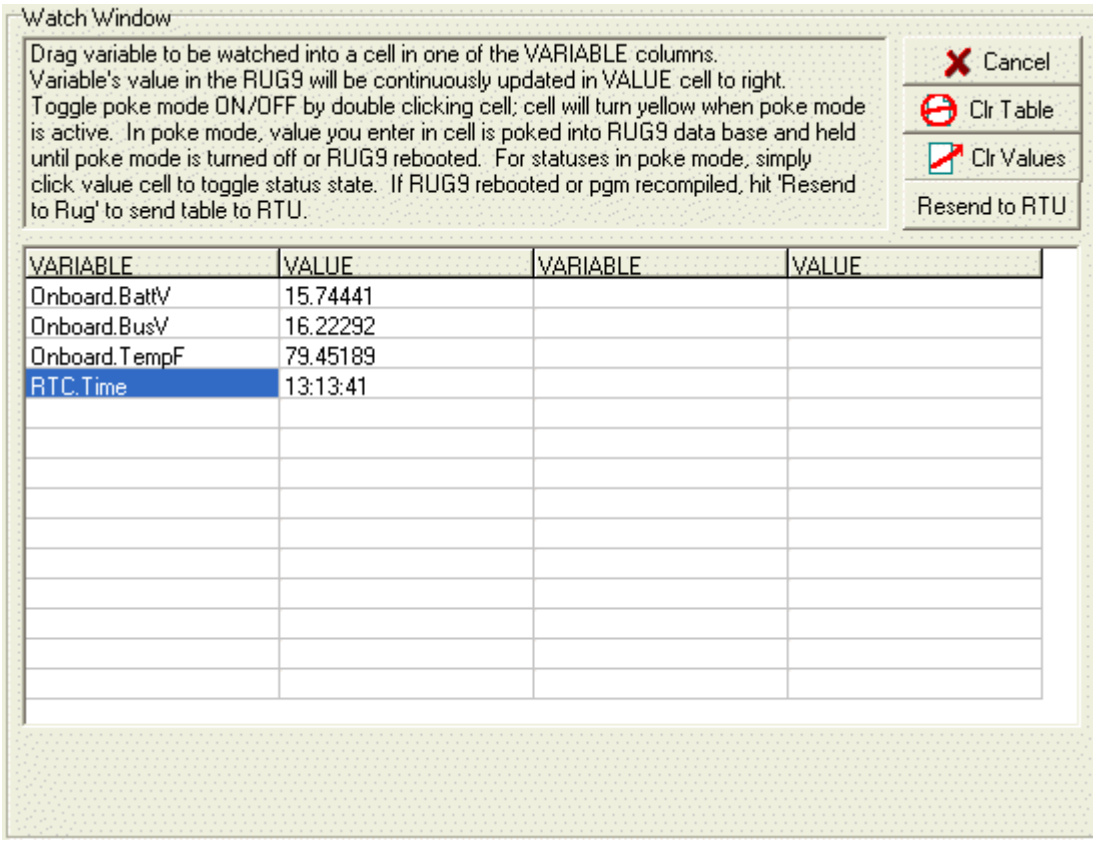


Figure 144 Watch Window

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 RUG5/9 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 R9SETUPD 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 RUG5/9, which can interfere with other serial communications, particularly Modbus 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. If the RUG5/9 has rebooted, it will have erased its watch window table entries. In that case you must either re-drag your variable entries into the watch window above or hit the 'Resend to RTU' button to re-establish the table in the RUG5/9.

TROUBLE DIAGNOSIS

Basic Operation and Program Loading Problems

Unit Appears Dead

- Power up the unit. If unit has illuminated bus LED on loop supply board, then unit main bus power is probably OK. If bus LED is not as bright as usual, check AC power adapter, if used. It should deliver 12 VAC under load. If unit is powered from battery, check that battery voltage is at least 12.0 VDC.
- If power is OK, remove power, connect computer or laptop running R9SETUPD 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 serial port, remove power, disconnect LCD, remove all I/O boards from card cage, not including CPU, and reapply power. If you get the "Testing OS" message, then one of the I/O boards or LCD is interfering with CPU operation. Replace the defective part. If you still get no message on boot up, either the CPU, mother board, serial cable, or your computer has failed. Try swapping known good items to isolate the problem. Return defective RUGID boards to RUGID for repair.

Unit Will Not Respond to Program Load

- While power is applied to the unit, press the recessed reset button on the CPU board. 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 on the CPU for 5 seconds. If you still get no message, remove power and remove the CPU board and all I/O boards from the card cage. Remove the battery from the CPU board and wait 10 minutes for the storage capacitor to discharge. Replace the battery and reinstall just the CPU board into the card cage. Reapply power to the card cage, and try reloading the program. If the program still will not load, either the CPU or the mother board is defective. Return both to RUGID for repair.

Program Appears to Load But Will Not Run

- Operating system in RUG5/9 may be incompatible with R9SETUPD revision. Try reloading the operating system into the RUG5/9, then reload the program. If after loading, program will still not run, try loading and running one of the RUGID supplied programs. 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 RUG5/9 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 CPU board 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 CPU board 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 interface on the CPU card 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 RUG5/9's optical isolation should block most transients that might try to come in through I/O. However, DC connections to the RUG5/9 through the AC power jack, battery terminals, or sleep board power terminals 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 RUG5/9.
- The onboard watchdog timer should restart the program if the program hangs up for more than two seconds. If your CPU board shipped earlier than August 1998, then the watchdog is disabled, and the CPU board must have its boot loader reprogrammed at RUGID. See the topic "Watchdog Timer Does Not Work" below for a way to test the watchdog timer.
- Units with failed or mis-located dialer board or sleep board can stall due to the CPU board waiting for a response from the microprocessor on those boards. Try running a program that does not access those boards. If the program runs correctly, the program may be referencing the board in the wrong card slot, or the board may be defective.

Unit Resets Occasionally

- Resetting is usually caused by local transients from switched inductive loads such as starters, relays, motors and solenoids. The RUG5/9's optical isolation should block most transients that might try to come in through I/O. However, DC connections to the RUG5/9 through the AC power jack, battery terminals, or sleep board power terminals 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 RUG5/9.
- Unit could be spending too much time on some software task. Try shortening any long tasks, or deleting them to solve problem.

Watchdog Timer Does Not Work

- The onboard watchdog timer should restart the program if the program hangs up for more than two seconds. To test the watchdog timer, do the following: Press the reset button on the CPU board and wait for the welcome message. Press key 5 ("Read memory location"). When prompted for a memory location, begin at the upper left hand corner of the RUG5/9's keyboard (the UP arrow) and start typing keys, going across the top row, then across the second row, etc., until you have entered all keys, ending with the [ENTER] key. If the unit stalls at that point, the watchdog is inoperative and must be reprogrammed at RUGID. If the unit reboots, the watchdog is working.

LCD Display Problems

Contrast is Too Dark or Too Light

- LCD contrast is controlled by the RUG5/9's CPU board, and is affected by component values and voltages on the display controller board in the display module. If you can read the LCD, you can set the contrast from the LCD module's keyboard. Otherwise, you must use the terminal window in R9SETUPD. To set LCD contrast, press the recessed reset button on the CPU board. When you get the welcome message, press key [3]. When prompted for the new contrast setting, type in a new contrast setting number in the range of 1 to 511. Numbers in the range of 60 to 200 usually work. If you cannot obtain acceptable contrast, return the CPU board and LCD module to RUGID for repair.

Contrast Changes Right After Program Starts Running

- If your configuration file's SysSetup1 module has a non-zero number in its **Enable temp ctrl of LCD** input, then the unit will attempt to use the temperature measurement from the loop supply board to compensate for temperature changes. Your project must have a loop supply board in the base card cage, and it must be referenced by a **Diagnostic** module in order for the temperature measurement to be available. The temperature measurement will take over contrast adjustment approximately 2 seconds after initiation of the program. If the loop board and **Diagnostic** module are properly in place, then you may simply need to adjust the contrast. Be sure that you can obtain acceptable contrast with

the program halted as in the topic above. Then, with the program running, press the [-] key to get the main system menu. Item 7 should enable you to adjust LCD contrast, and will show the contrast as two values such as "245(150)". The first value is your contrast entry; the second is the value actually used by the CPU after temperature is included. Press key [7] and enter a contrast number in the range of 180 to 280, until you observe acceptable contrast. If you cannot obtain acceptable contrast, the temperature measurement may be in error, or unavailable. Check the loop supply board's temperature value from the watch window, and replace the board if necessary.

Contrast Changes When Nearby Equipment Powers On/Off

- Nearby equipment could be causing surges or sags in main power to the RUG5/9, causing the LCD module, or mother board regulators to be unable to maintain clean power to the LCD controller. Check that the RUG5/9's bus voltage is at least 11 VDC using the loop supply board and **Diagnostic** module. Also, check that main power is 12 VDC, if powered through the loop supply board or sleep board; or that AC power into the AC power jack on the side of the card cage is at least 12 VAC.

Display Gets Random Characters On It

- This problem is probably caused by noise or reflections in the cable between the CPU board and the LCD module. If the cable is running in parallel with other wiring, try separating the cable from the other wiring. If the LCD module is mounted remotely from the card cage, install a ground strap from one of the LCD module mounding studs to one of the card cage mounting screws. If the problem persists, return the LCD module to RUGID for repair.

Display Has Light Characters on Dark Background and Should Be Dark on Light

- The display's normal/reverse setting can be toggled from the welcome menu. Press the reset button to stop the program and get the welcome menu. Then, press the [4] key to toggle normal/reverse. Then restart the program. If the normal/reverse setting is toggling by itself, the unit could be subject to nearby transients. Try installing snubbers on inductive loads such as starters, relays, motors and solenoids.

Display Has a Few Corrupted-Looking Lines

- The RUG5/9 display may appear corrupted if a line is too long or if there are too many characters on a line. Be sure that there are no more than 40 characters per line, and that there are no more than 20 lines in your display definition.
- If your display is presenting strings that may not have been initialized, you can get random characters, including some that appear to be graphic. Make sure that all strings have known contents.

Event Logger Display Has Random Strings

- If an event logger has not been reset, as is usually the case upon first running a program, the event log will have random contents that can cause serious problems for the display. Install a way to trigger the **Erase log** input on the **EventLogger** module.
- If you have changed the size or other property of the event logger, or have installed it over another program, then the events that have been logged may point to the wrong strings for display presentation. You must erase the log to correct the problem as in the item above.

Trend Has Random Values

- If problem is observed just after installation of the program, or after a change to the **DataLogger**'s input properties, then the logger is probably not referencing valid samples but is presenting random data that existed in memory upon first initiation. You must either trigger the **DataLogger** module' **Reset** input to erase the log, or you must wait until enough samples have been taken to clear out the erroneous data. You could also press the "Clear memory" key from the welcome menu to zero all logged data.
- The problem could also be due to inaccurate, or wildly varying values being sampled in relation to the trend's vertical range settings in the **DataLogger** module.

Trend Has Lots of Vertical Bars

- If the trend is showing a lot of vertical bars, then your log has too high a proportion of time tags imbedded in the data. You must adjust the sample timing or time tag timing to reduce the number of time tag entries.

I/O Problems

Analog Input is Inaccurate (4-20 ma)

- Check that the module you are using for the channel in question is the **AnalogInput 4-20** type, and that its offset and span settings are proper for the measurement you are making. If they appear correct, remove power from the RUG5/9 and remove the header on the analog input board. With an ohmmeter, measure the resistance from the analog input channel in question to the common terminal on the analog input board. It should read within 2 ohms of 220 ohms. If it reads an open circuit, remove the analog input board from the unit and verify that the shorting bar for the channel is in place. If missing, install one. If the shorting bar is in place, and if the resistance is out of this range, or, if none of these errors exist, 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. 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.

Analog Input is Inaccurate (0-5V)

- Check that the module you are using for the channel in question is the **AnalogInput 0-5V** type, and that its offset and span settings are proper for the measurement you are making. If they appear correct, remove power from the RUG5/9 and remove the header on the analog input board. With an ohmmeter, measure the resistance from the analog input channel in question to the common terminal on the analog input board. It should read greater than 10Kohms. If it reads low, remove the analog input board from the unit and verify that the shorting bar for the channel is removed from at least one of that channel's pins. If the shorting bar is in place, pull it off and re-install it on only one pin. If the resistance is less than 10K ohms, or, if none of these errors exist, return the board to RUGID for repair.
- Check that another analog input on this same board is not out of range of 0 to 5.0 volts. 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.

Analog Output is Inaccurate

- Check that the module you are using for the channel in question is the **Analog Output** type, and that its engineering units (EU) settings are proper for the measurement you are making. Using the watch window, verify that the database variable being sent to the analog output module, is as you expect. With power applied to the RUG5/9 and to the output loop, measure the voltage across the two analog output pins. If the voltage is less than 12 VDC, your loop supply voltage for the analog output loop may be too low, or the sum of voltage drops in the loop may be too large. If none of these conditions exist, try another channel on the analog output board, or return the board to RUGID for repair.

Loop Supply Board Voltage or Temperature Measurement is Inaccurate

- With an external battery not connected, the battery and bus voltages should be within a few tenths of a volt of each other. You can measure bus voltage by measuring battery terminal voltage at the loop supply board with the battery disconnected. Bus voltage for normal operation should be in the range of 12 to 16 volts. With the battery attached, the battery voltage measurement should match that measured across the battery. If any appear out of range, return the loop supply board to RUGID for repair.
- Board temperature measurement measures the temperature inside the card cage. It will normally read higher than ambient temperature due to heating inside the card cage. If it appears out of range, return to RUGID for repair.

- If the low pass filter time constant is long, the measurement will take a long time to settle to an accurate value.

Loop Board Measures Battery Voltage Even With Battery Detached

- The charging circuitry will attempt to source current to the battery. With no battery attached, the battery terminal voltage will rise to the bus voltage. This is normal.

Analog Output Sends Occasional Glitches

- Older analog output boards can occasionally send spikes to the analog output load due to a design error on the boards. Return to RUGID for replacement.

Relay Output Will Not Turn On

- 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. This can also happen if the board is inserted in the wrong slot. If the module is driven correctly and the board is in the correct slot, then return the relay output board to RUGID for repair.

Relay Output LED Lights But Relay Does Not Power Load

- Relay has failed. Return the relay output board 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 RUG5/9's digital inputs are designed to operate from 24 VDC or 120 VAC. When your signal is turned on, be sure that the voltage across the digital input is at least 24 VDC. Some users are using the digital inputs with 12 VDC applied with success; but we do not guarantee operation below 24 VDC.

AC Digital Input Does Not Correctly Sense Input

- Be sure that you are using a **Digital Input AC** type module to sense the digital input. The RUG5/9's digital inputs are designed to operate from 24 VDC or 120 VAC. When your signal is turned on, be sure that the voltage across the digital input is at least 24 VAC. Operation is not guaranteed below that voltage.

Communications Problems

Modem Will Not Key Radio

- Make sure that you have a **ComSetup** module in your project referencing the modem card, and that it is set for port 1. The modem card has several hookup options (RS232, 2-wire, 4-wire) but only one port. 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 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 has at least the number of words defined as are requested in the poll.

Modem Key LED Lights But Radio Does Not Key

- If modem board jumper J6 is missing, install it. If it is in place, return the modem board to RUGID for repair.

Troubleshooting

Modem Board Makes Clicking Noise With Each Transmission

- Modem board jumper J6, when in place, engages the keying relay with each transmission. Remove the jumper to disconnect the relay without affecting other functions.

Modem Keys But Will Not Transmit

- Make sure that the transmit array referenced by the poll has at least the number of words defined as are requested in the poll, and that it is defined for the correct board, port and address.
- Use a speaker/amplifier to listen to the transmission. A suitable model is the Archer model 277-1008C, available from Radio Shack. If no signal is present, or the signal is distorted, return the modem board to RUGID for repair. If the signal is low in amplitude, adjust potentiometer R2 to increase signal level.

Modem Transmits But Other Receiver Will Not Accept Message

- Use a speaker/amplifier to listen to the transmission. A suitable model is the Archer model 277-1008C, available from Radio Shack. If no signal is present, or the signal is distorted, return the modem board to RUGID for repair. If the signal is low in amplitude, adjust potentiometer R2 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 0.2 seconds. For radios, use 0.7 seconds. Shorter times may work, but be conservative.

Modem Dials But Call Does Not Go Through

- The modem board's signal amplitude may be too low to be detected by the dial up phone system. Adjust potentiometer R2 on the modem board to increase amplitude.
- Make sure modem is delaying long enough after going off hook for dial tone to be present before it dials.
- Make sure that the answer tone time defined in the **DialMdm** module is long enough for the distant modem to acquire and connect.

Modem Answers When Ringing Detected, But Will Not Connect

- Make sure that the answer tone time defined in the DialMdm module is long enough for the distant modem to acquire and connect.

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 0.2 seconds. For radios, use 0.7 seconds. Shorter times may work, but be conservative.
- Use a speaker/amplifier to listen to the transmission. A suitable model is the Archer model 277-1008C, available from Radio Shack. 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.

Troubleshooting

- Make sure that the RUG5/9 transmit and receive arrays are configured for the board and port you are using. If you are using the modem board's RS232 port, you should be using port 1. If you are using the printer/serial port board, you should be using port 1 or 2.
- Make sure that you are using source and destination addresses of –1 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 can either wait for a reception or stop the program to get the welcome message, and then press key [6] to clear all memory.
- 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 RUG5/9 and the radio when it transmits.

Speech Problems

Unit Will Not Record or Playback Speech

- Install the Autodialer program included in the R9SETUPD software, and use it to test the dialer board. Make sure the dialer board is installed in the proper slot as designated in the Autodialer program. If the board will not record or playback, return entire unit to RUGID for repair.

Recording is Excessively Delayed From Record Keystroke

- Program may be running slowly due to large size. Try recording using the Autodialer program, or a pared down version of your program.

Speech Has Click At End of Each Phrase

- If your unit has the LCD module hinged to the card cage, the microphone may be recording the tactile detent at the end of each keystroke. The easiest solution is to detach the LCD module from the card cage to do the recording. Alternatively, you could use a status input to trigger recording start and stop, making sure the that switch you use is not rigidly attached to the RUG5/9 card cage.

Troubleshooting

Dialer Goes Off Hook But Will Not Dial

- The dialer will not dial unless it hears a dial tone. The dial tone may also be weak.
- The **SpeechDial/Answer** module will not dial if it has no phone number in one of its phone number inputs.

Unit Dials But Call Does Not Go Through

- Volume control potentiometer R12 may be adjusted too low. The volume control adjusts both the speech volume on playback, and the touch tone volume.
- Make sure the phone number being dialed is correct.

Unit Dials Wrong Number

- Make sure that the phone number sent to the **SpeechDial/Answer** module is correct. Note that if you are using the integer phone number input, the integer is only valid for up to seven digits. Consider using the string input.

Unit Does Not Respond to Operator Touch Tone Keystrokes

- The unit may not respond to touch tone keystrokes entered while the unit is speaking due to interference from the speech. Wait until the spoken message is finished before entering keystrokes.
- Touch tone keystrokes may be excessively attenuated. Try entering them from another phone or phone line.

Unit Runs Slowly or Frequently Resets When Running Speech Program

- If the CPU is waiting too long for a dialer board response, the watchdog timer will restart the program. This can happen if the software designates the dialer board as located in a slot other than the one in which it is installed. If the board is in the software designated location, the board may be defective. Try the Autodialer program included with R9SETUPD. If the problem persists, return the dialer board to RUGID for repair.

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.

Unit Will Not Log Negative Numbers

- Negative numbers are used to designate time tags. This is normal.

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 RAM. You must trigger the data logger's reset input to erase the log, or wait for samples to replace the erroneous data.

Module Problems

PID Oscillates

- You may have too high a proportional gain or differential gain. Set differential gain to zero and reduce proportional gain. 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.

Troubleshooting

PID Will Not Accurately Control Process Variable

- Set PID properties as suggested above, then increase integrator gain to reduce residual error.
- Apply longer low pass filter to process variable.

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, 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 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, 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 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 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.

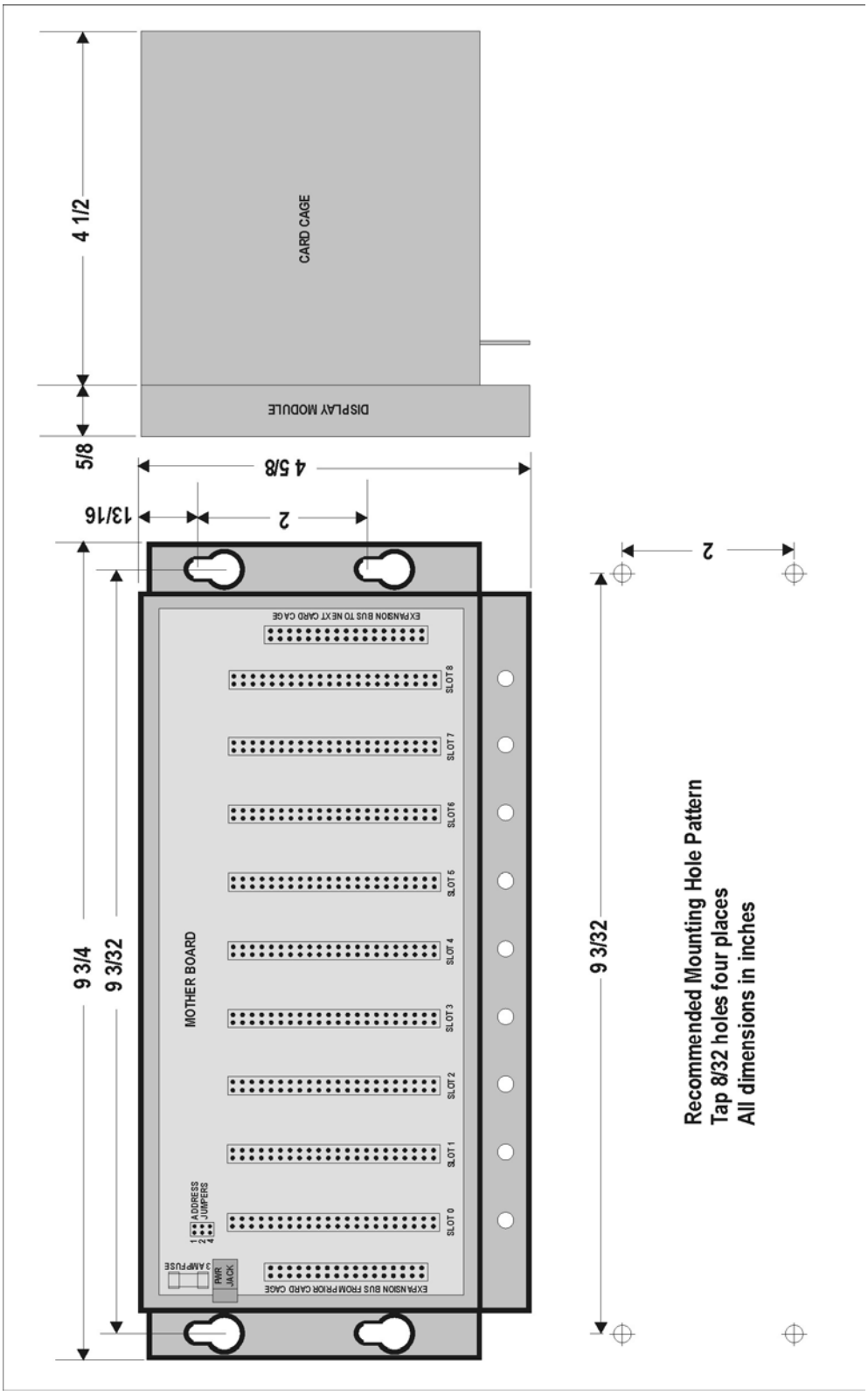


Figure 145 Card Cage Dimensions

Warranty, Dimensions, Specifications

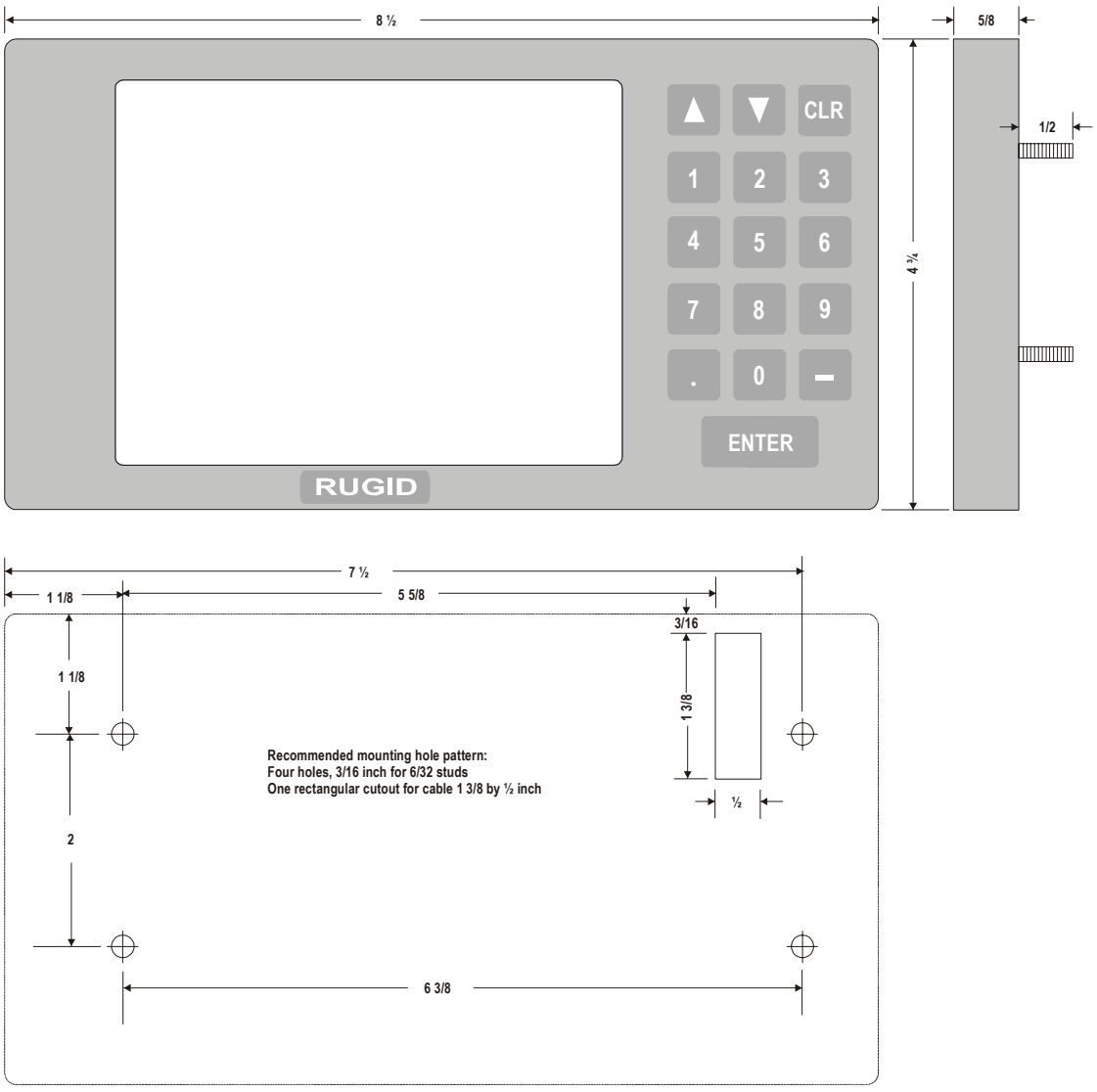


Figure 146 Display Module Dimensions

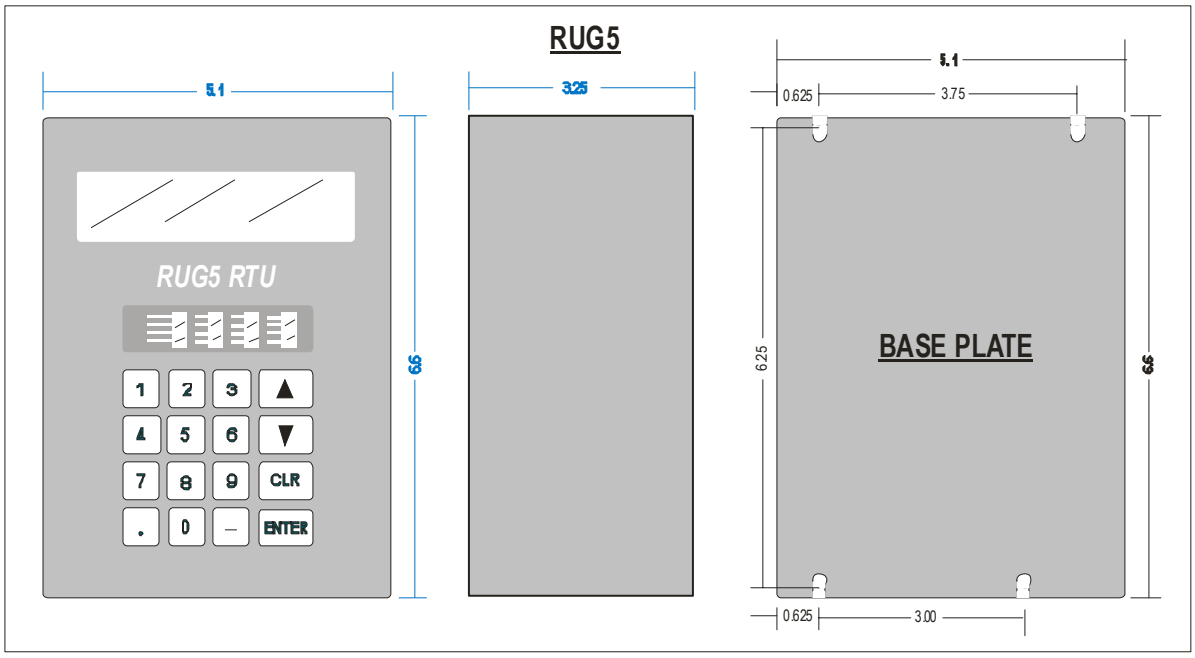


Figure 147 RUG5 Dimensions

RUG9 SPECIFICATION

LOGIC FAMILY
All low power CMOS
MICROPROCESSOR
32-bit 68331, 16 Mhz, 16 bit data bus, 24 bit address bus
MEMORY
RAM-256 Kbytes battery backed low power static RAM
FLASH-512 Kbytes
Battery Backup-Lithium coin cell backs up RAM & realtime clock/calendar min 2 years
MEMORY CARTRIDGE
4 Mbyte to 2 Gbyte removable Compact Flash cartridge
I/O EXPANSION
First card cage can have any I/O, modems, FlashDisk, sleep board, etc., up to 8 cards plus CPU. Up to 7 card cages attach with ribbon cables, can hold I/O up to 64 chan per cage.
DISPLAY
20 line X 40 char (320 by 240 dot) backlit graphic LCD, 6 in diag. detachable from card cage.
Text- All std ASCII chars plus special graphic chars
Trends-Up to 3 traces per page; pages incorp into user defined text pages, as many as will fit in RAM
Bargraphs-Up to 20 horizontal bars to show analog values
KEYBOARD
16 key sealed tactile membrane with interrupt scanning
REALTIME CLOCK
Battery backed, 0.005% crystal accuracy
SPEECH SYNTHESIZER
8 Khz sampling record & playback. Up to 256 messages in 12 minutes total storage.
OPERATION SECURITY
Watchdog Timer-Hardware timer resets unit 0.5 sec. after interrupt fail. Cannot be disabled.
Telemetry Watchdog-Resets rcv buffer if no character received within 1 sec.
Brownout Detector-Halts process if logic voltage falls below 4.75 V, restarts when voltage rises to 5 V
AUTOBOOTING
Unit delays 2 sec., then restarts program on power application
I/O SURGE PROTECTION
All I/O is optically isolated, meets IEEE surge protection rqmts.

ANALOG INPUTS 12 bit
8 ch per board, 12 bit res., successive approx, optically isolated, 4-20 ma or 0-5 v. Factory calibrated.
ANALOG INPUTS 16 bit
4 ch per board, 16 bit res., delta-sigma, optically isolated, 4-20 ma. loop pwred Factory calibrated.
ANALOG OUTPUTS
4 chan per board, 12 bit resolution, optically isolated.
DIGITAL INPUTS
Status- 8 chan/board, optically isolated, 120VAC or 24 VDC compatible.
Pulse Counting-all DI channels in first card cage count 256 PPS
DIGITAL OUTPUTS
8 ch per board, 3 or 10 amp relays
SERIAL PORTS
Up to 8 RS232/modem ports in base card cage, user defined buffers
MODBUS Protocol
Modbus RTU or TCP protocol on any port
MODEM PORT
Bell 103/212 standard
Radio Interface
4-wire audio, adj. gain, xformer isolated + isolated key line. Low tones mode for splinter chan
Phone Interface
2 wire audio adjustable gain, transformer isolated
Autodialing
On/off hook relay, touchtone generate & detect.
Autoanswering
On/off hook relay and ring detector
Touchtone Detection
Standard tones on 2-wire chan.
Communications
Background CRC gen/decode, variable length messages, user defined message length & buffer sizes. Can combine status, int, float, and double prec int in any message.
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
0 to 255
PRINTER PORT
Standard Centronics compatible parallel port option

POWER INTERFACE
Unit requires 12 VAC or 15 VDC +/-20%, 130 ma. to 2 amps max, fused.
Power regulator-
Switching supply, 90% efficient.
Loop Supply
Isolated, regulated 24 VDC +/-5%, fused, 160 ma.
Battery Charger
Temp. compensated, 300 ma., reverse protected, fused.
I/O CONNECTIONS
All I/O uses removable rising cage screw headers in banks of up to 16 each, 14 ga wire
SOFTWARE
Storage-operating system and all user config. and programming stored in nonvolatile FLASH memory.
Flash loader stored in FLASH protected boot block.
Security-parameter voting and memory integrity on boot up, CRC gen/detect on serial ports
Scanning-Builtin software scans all I/O, ports, timers, realtime clock
PROGRAMMING
Modules-most applications will use precompiled modules resident in FLASH where the programmer will interconnect modules and set properties using a supplied Windows program. No procedural programming rqd for most applications.
LADDER LOGIC
Ladder logic is built in to the Windows config. program to handle misc control logic.
VARIABLES
Supports integer, floating point, boolean, strings, arrays
ERROR MESSAGES
Configuration program handles all setup errors. Run time software is self protecting... no run time errors.
ENCLOSURE
16 ga. Steel, blue powder coated, 9 slot card cage with detachable display/keyboard module
Cage: 9.75 X 4.6 X 4.1 in.
DSP: 8.6 X 4.8 X 0.75 in
TEMPERATURE RANGE
-40 to +85 deg. C logic
-20 to +60 C LCD display
DOCUMENTATION
300 page bound manual
WARRANTY
1 year std limited warranty
REPAIR

Warranty, Dimensions, Specifications

Nominal 24 hr turnaround

Index

Index

E

F

G

H

I

J

Index

Q

R

S

Index

T

U

V

W

Y