

---

# Chapter 3

## COMMUNICATIONS

### Edition 3

#### Contents

### Communications Overview

INTRODUCTION .....	3-1
Purpose .....	3-1
Scope .....	3-1
COMMUNICATIONS PRINCIPLES .....	3-2
Master-slave operation .....	3-2
Peer-to-peer .....	3-3
Parameter addressing .....	3-4
COMMUNICATIONS PORTS .....	3-5
Standards .....	3-5
Installing intelligent modules .....	3-6
Cabling devices to ports .....	3-7
Multidropped serial links .....	3-7
Converting RS422 to RS232 operation .....	3-8
DEFAULT COMMUNICATIONS .....	3-10
Ports support .....	3-10
Addressing parameters .....	3-11
Custom applications .....	3-14
COMMUNICATIONS FUNCTION BLOCKS .....	3-15
Driver function blocks .....	3-15
Slave variable function blocks .....	3-18
Remote variable function blocks .....	3-27
Raw communications block .....	3-39
Euro_Panel driver function block .....	3-40

---

Contents (continued)

PROBLEMS AND SOLUTIONS ..... 3-41

**Communications**

EI\_BISYNC\_M..... 3-43

    Functional Description ..... 3-43

    Function Block Attributes ..... 3-43

    Parameter Descriptions ..... 3-44

    Parameter Attributes ..... 3-56

EI\_BISYNC\_S ..... 3-57

    Functional Description ..... 3-57

    Function Block Attributes ..... 3-59

    Parameter Descriptions ..... 3-59

    Parameter Attributes ..... 3-89

RAW\_COMMS ..... 3-90

    Functional Description ..... 3-91

    Function Block Attributes ..... 3-92

    Parameter Descriptions ..... 3-92

    Parameter Attributes ..... 3-112

SIEMENS\_M\_S ..... 3-114

    Functional Description ..... 3-114

    Function Block Attributes ..... 3-115

    Parameter Descriptions ..... 3-115

    Error Reporting ..... 3-135

    Parameter Attributes ..... 3-144

---

Contents (continued)

J_BUS_M.....	3-145
Functional Description .....	3-145
Function Block Attributes .....	3-146
Parameter Descriptions .....	3-146
Error Reporting.....	3-151
Parameter Attributes .....	3-153
J_BUS_S .....	3-154
Functional Description .....	3-154
Function Block Attributes .....	3-155
Parameter Descriptions .....	3-155
Parameter Attributes .....	3-175
TOSHIBA_M .....	3-176
Functional Description .....	3-177
Function Block Attributes .....	3-177
Parameter Descriptions .....	3-178
Error Reporting.....	3-189
Parameter Attributes .....	3-197
EURO_PANEL .....	3-199
Functional Descriptions .....	3-199
Function Block Attributes .....	3-203
EURO_PANEL2 .....	3-204
Functional Description .....	3-205
Function Block Attributes .....	3-205
Parameter Descriptions .....	3-205
Basic Modes of Operation.....	3-209
Mode Selection .....	3-210
OIFL enhancements.....	3-211
Parameter Attributes .....	3-216
Character Codes.....	3-218

---

Contents (continued)

ALLENB_M .....	3-223	
Functional Description .....	3-224	
Function Block Attributes .....	3-231	
Parameter Descriptions .....	3-231	
Configuration and Communication Errors .....	3-236	
APPENDIX A	Glossary of terms .....	3-241
APPENDIX B	Standard Communications	
	Error Codes .....	3-244
APPENDIX C	ASCII Table .....	3-246



## **COMMUNICATIONS OVERVIEW**

### **INTRODUCTION**

#### **Purpose**

The Communication Overview describes the basic principles of serial communications, assists with the selection of suitable communications protocols, defines how to configure a communication system, provides guidance on how to diagnose common faults and gives tips on how communications can be optimised.

This communications overview will be particularly useful to users when configuring a serial communications system for the first time. It is assumed that the reader is familiar with some communications terminology although explanations of more technical terminology will be given. A glossary of terms is provided in Appendix A.

#### **Scope**

This document contains technical information to enable a user to configure a communication links between a PC3000 and other devices using serial communications.

Serial communications can be used to interface with:

- PC3000 Programming Station

- Other Eurotherm instruments (e.g. 905s).

- Motor drive units, thyristor stacks

- Programmable Logic Controllers (e.g. using Siemens 3964 (R) protocol)

- Operator displays such as the Euro Panel, Xycom 12 inch and 9 inch terminals

- Other PC3000s

- Data acquisition systems

- Production cell control systems such as the Eurotherm Production Orchestrator

- Supervisory systems such as Eurotherm ESP package

- Printers for report generation

- Miscellaneous proprietary equipment such as electronic weighing scales, intelligent sensors, etc.

For information on using specific communications protocols, the user should refer to the related protocol on this chapter.

## **COMMUNICATIONS PRINCIPLES**

Serial communications allows information to be exchanged between two or more devices by encoding information contained in a stream of data bytes, into a series of bits. The bits can be transmitted by a variety of techniques depending on the media used for the serial link. For example, if twisted-pair wire is used, the bits are transmitted by switching the different voltage between the wire pair. Usually the information transfer occurs by breaking the information down into a series of messages which are framed with special control characters depending on the protocol used.

The receiving device normally returns a short message to the sending device to acknowledge each message reception. This is detected by the sender and implies that the last message was correctly received and that the sender is free to send the next message.

To ensure that transmission errors are detected, most protocols require that messages are sent using a particular format, i.e. that certain control characters are added to the beginning and end of the message. A special block check byte is usually added to the end of the message, based on the message content. For example, with the EI Bisync protocol, the block check character is based on the longitudinal parity of each byte in the message and is transmitted as the last byte of a message. The software within the remote device can then verify that the message is correct by checking the framing control characters and the block check character. If an error is detected, the remote device will usually return a special control message such as a 'Not Acknowledged', NAK control character to indicate to the sender that the message should be re-transmitted.

### **Master-slave operation**

With most serial communications protocols, it is possible to have many devices connected to one serial link using the Master-Slave mode of operation. One device will be designated as the 'Master' and the rest will be designated as 'Slaves'.

The Master device is usually the only device on the serial link that can send information to, or request information from other devices connected to the serial link. The Slave device, have the complementary function; they can only send information to the Master in response to a request from the Master device, or receive information sent from the Master device. With this mode of operation, it is not possible for a Slave device to send information directly to any other Slave device or to send information when not specifically requested to do so.

This mode is often used when connecting the PC3000 to a number of discrete instruments. For example, with a PC3000 configured as a Master of a EI Bisync serial link, it is possible to poll a number of multi-dropped 900 series instruments to obtain the process values from each instrument.

Even when there are only two devices on a point-to-point serial link, many serial protocols require that one device is designated the Master and the other the Slave.

## Slave address

Each Slave should have a different communications address so that the Master device can direct messages to a selected device. For example, with EI Bisync, each Slave device is given an address consisting of Group Identity (GID) and Unit Identity (UID), which together form the address. As the addressing scheme varies between protocols, care should be taken to ensure that each Slave device has a unique and valid communications address for the particular protocol being used.

Master devices will also need to be configured to address specific Slave device, i.e. the Slave addresses and addresses known to the Master should match. Setting up addresses for use with PC3000 ports in Master or Slave modes is described further in this document. For particular remote devices, such as a PLC using JBus, setting up communications addresses will be defined in the communications manual for the device.

## Peer-to-peer

Using protocols that support the Peer-to-peer mode of operation, allow any device on a network to initiate a request to send information to, or read information from any other device. Generally this mode of operation is associated with token-passing or Ethernet networks, although a few serial protocols such as the Siemens 3964 (R) protocol allow Peer-to-peer operation on a single point-to-point serial link by a moving master technique.

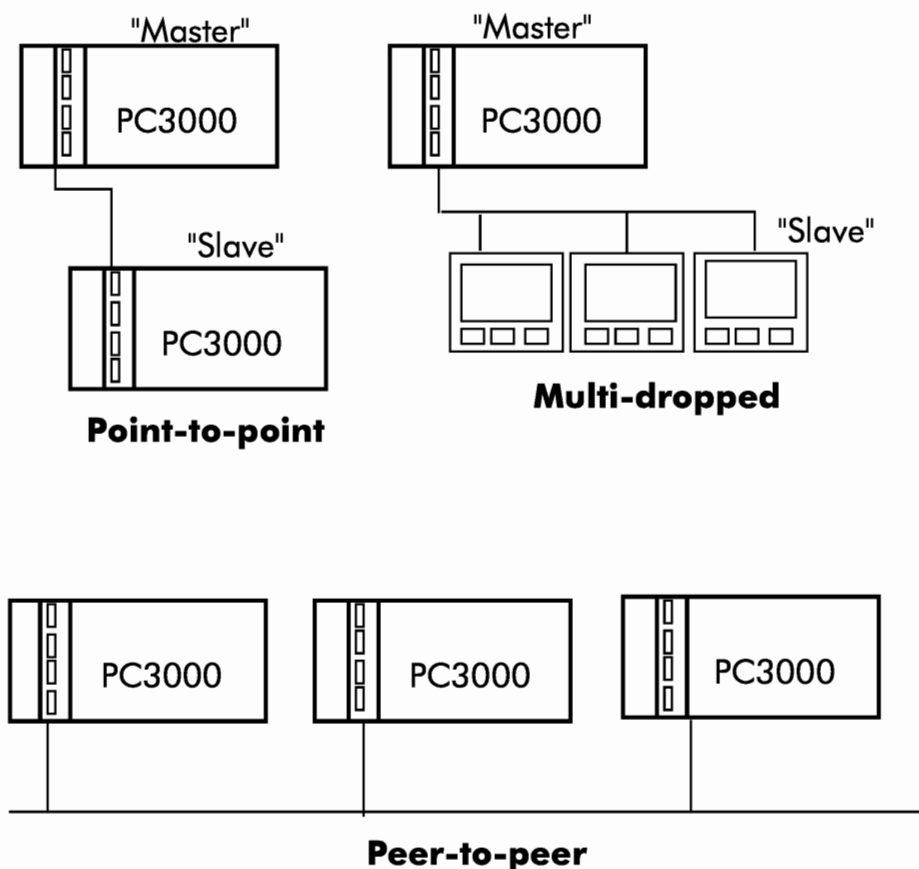


Figure 3-1 Communications configurations

## **Parameter addressing**

To address a particular item, parameter or function within a device, each protocol defines an addressing scheme. This can vary significantly for different protocols but usually allows single or multiple items to be addressed.

With EI Bisync, typically the Group Identity (GID) and Unit Identity (UID) are used to address a particular multi-dropped Slave device. A channel identity (CHID) may be optionally required to address a sub-set of parameters. A two character mnemonic is required to address the required parameters. For further details on addressing parameters within PC3000 refer to the EI Bisync description in this chapter.,

## COMMUNICATIONS PORTS

### Standards

The PC3000 provides a number of communications ports on modules such as the LCM and Intelligent Communications Module (ICM). Each port will use or as is the case with the ICM, can be configured to use, one of the following transmission standards:

**RS232** This standard allows a pair of devices to be connected over a short length of cable, typically lengths less than 15M, using a single wire for transmission and one for reception of data. It should only be used in environments where there are low levels of electrical noise and for transmission speeds less than 20K Baud. It does not allow multi-dropped communications. both Half-duplex and full-duplex operation is possible.

**RS422** This allows one or more Slave devices to be multi-dropped from a single Master device. It should be used in electrically noisy environments in preference to RS232 but requires two pairs of twisted cable for the transmission and reception of data. Unlike RS232, it provides balanced, differential signal transmission and can therefore be used over longer distances and at higher transmission speeds. Both Half-duplex and full-duplex operation is possible, although half-duplex mode is more commonly used.

**RS485** this has similar characteristics to RS422, but allows devices to communicate using peer-to-peer protocols. Only one pair of twisted pair cables are used for both transmission and reception of data. Only Half-duplex operation is possible using this mode.

Both RS422 and RS485 allow a serial link to have a total length of up to 1000m without signal repeaters when using transmission speeds of up to 100K Baud and 24 AWG twisted pair cable.

A brief description of the capabilities of the PC3000 communications ports are summarised in the following table.

<b>Module</b>	<b>Port</b>	<b>Standard</b>	<b>Modes</b>	<b>Speeds(Baud)</b>
LCM	A	RS422	Point-to-point only*	75-38.4K
	B	RS422	Point-to-point only*	75-38.4K
	C	RS422	Master-to-Slave*	75-115.2KK
ICM	A	RS422 or RS485	Master or Slave	300-115.2K
	B	RS422 or RS485	Master or Slave	300-115.2K
	C	RS422 or RS485	Master or Slave	300-115.2K
	D	RS4232	Point-to-point	300-19.2K

All three LCM ports can function in either Master or Slave modes. LCM ports A and B can only be used for point-to-point configurations when configured as Slaves but can be Master of multi-dropped serial links. LCM port C can be used on multi-dropped configurations as a Master or Slave.

Port D on the LCM can only be used for extension racks and should not be connected to any other external device.

Links can be set on the LCM mother-board to set the Slave address which can be used for protocols including EI Bisync and JBus. Similarly a switch is provided on the ICM to set the default Slave Address. The Slave Address parameters on communications, slave driver function blocks should be set to null values for the hardware default addresses to be applied. See 'PC3000 Hardware Reference', for details on setting Slave Addresses on these modules.

## Installing intelligent modules

ICMs should be used when extra communications ports are required or where there is a need to off-load some of the communications activity from the LCM. This will be made evident by PC3000 performance data such as task overruns, see the sections on tasking in the 'PC3000 Real Time Operating System Reference'. To obtain specific metrics on function block performance refer to the function block data in this chapter.

As with LBus modules, ICMs can be only installed in slots 1 to 5 in the first PC3000 rack. It is important to note that only LBus modules including other ICMs should be installed in slots between an ICM and the LCM. In other words, ICMs should be part of a contiguous set of LBus modules next to the LCM.

The 'PC3000 Installation Handbook' should be referenced for further details on the wiring and capabilities of communications modules.

## Cabling devices to ports

A variety of cables are available to connect commonly used devices to the PC3000. It is recommended that these are used where possible. The full list is available in the 'PC3000 Installation Handbook HA022231'.

Between device and Device		Mode
LCM or ICM	261	RS422
ICM Port D	12 inch colour terminal	RS23
261	12 inch colour terminal	RS23
LCM or ICM	9 inch terminal	RS422
LCM or ICM	Euro-Panel	RS422
261	PC COM Port	RS232

When cabling between Master and Slave devices using RS422, the Master transmit pair Tx- and Tx+ should be connected to the Slave Rx- and Rx+, and Slave Tx- and Tx+ connected to the Master Rx- and Rx+. Multiple Slave Tx-, Rx+ and Rx-, Rx+ are ganged together as shown in the figure 'Cabling Multi-drop Slave Devices'.

Note: The ICM provides a link on the board to swap the Rx and Tx connections on port A.

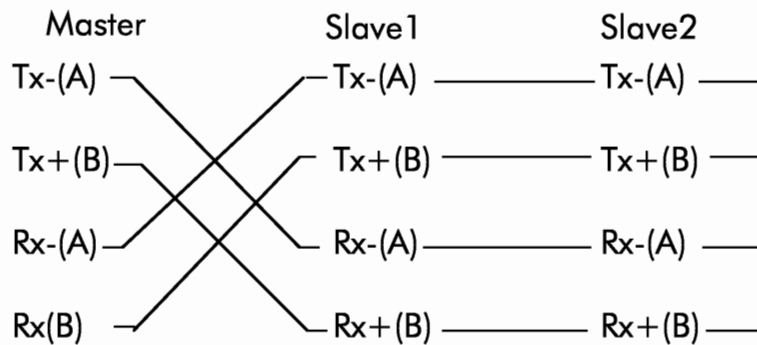


Figure 3-2 Cabling multi-dropped slave devices using RS422

## Multidropped serial links

When multi-dropping a number of Slave devices from a Master device, using RS422, it is recommended that the cable stubs that connect to the 'tee' points should be kept short where possible. 'Tee' points should be positioned near to groups of instruments so that individual stub cable lengths to each instrument is short compared to distances between 'tee' points and the master port.

Typically, inexpensive non-shielded cables will support networks up to about 250m in length at transmission speeds up to 19.2K, for higher speed operation and distances up to 1000 m, good quality shields, twisted pair cables are recommended. Criteria such as the environment specially in regard to electrical noise, position of cables, transmission speed and acceptable error rates should all be considered when defining cabling requirements.

A master device can have up to 10 Slave devices multi-dropped on a single serial link using RS422 and up to 31 using RS485.

Further details in cabling requirements can be found in the 'PC3000 Installation Handbook HA022231'.

## **Converting RS422 to RS232 operation**

Frequently it is necessary to connect a remote device such as a PC which only has RS232 ports, to the PC3000. By connecting a suitable RS422 to RS232 converter, any PC3000 RS422 port can be used.

A recommended device is:

Eurotherm 261 Universal Serial Interface

Order Code:	261/230	-for 230V supply operation
	261/115	-for 115V supply operation

Even with the connection of a 261, a PC3000 port configured for RS422, can continue to function in either Master or Slave mode, where this is normally supported.

To minimise interference from electrical noise, it is recommended that the 261 is positioned close to the remote device, so that the RS232 cable between 261 and remote device is kept short.



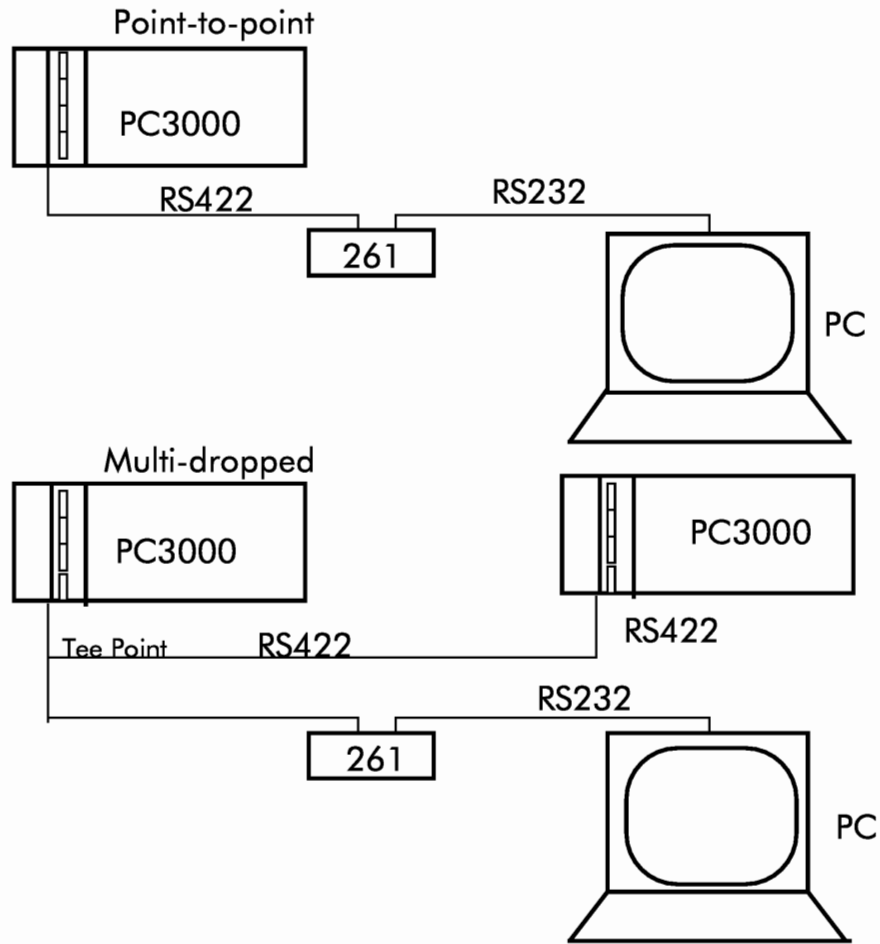


Figure 3-3 Using RS422/RS232 convertors

Further details on the 261 is available in the 'PC3000 Installation Handbook '

The 261 can also be used to convert a PC3000 port such as ICM port D, that only supports RS232 to RS422 if required.

## DEFAULT COMMUNICATIONS

### Ports support

Ports A, B. and C will always support the EI Bisync protocol as default whenever the PC3000 is powered-up, providing LCM ports A, B or C are not allocated to specific communications function blocks within a running user program. This is to ensure that it is always possible for the PC3000 Programming Station to communicate with the PC3000 whenever a user program is not running or is not loaded. (The use of communications function blocks is described in later sections of this document).

By convention, LCM Port B is normally used for the Programming Station, but it is possible to use ports A or C if these are not committed to other communications, applications.

The serial communications supported on LCM ports A, B and C has the following characteristics:

- Baud rate 9.6K
- Slave mode
- EI Bisync Protocol
- Slave address set by LCM Links\*
- One stop bit (EI Bisync Standard)

\*The default Slave address (GID) is selectable by links on the LCM board to have one of 16 addresses in the range '0 through to F'. The factory set address is '7'. However, address selection and multi-dropping PC3000s is not possible on older style issue 2 LCMs. LCM Address is fully described in 'EI Bisync Slave Driver' function block within this chapter. Also refer to the 'PC3000 Installation Handbook' for further details on the LCM and how to identify the issue number

Default communications is primarily designed for use by the PC3000 programming Station, but can also be used for communicating with the Eurotherm Supervisory System (ESP) and the Production Orchestrator cell controller system. For example, a PC running ESP can be connected to ports A, B. or C using the default EI Bisync protocol. However, default communications can be used for a range of applications where 9.6K Baud, EI Bisync and the PC3000 in Slave mode is suitable, providing an implementation of an EI Bisync Master is supported on the remoted device.

A mechanism to enable a remote Master to read and write to, a wide range of parameters and functions within the PC3000 is provided by EI Bisync protocol. This includes all Function Block parameters used in a loaded and running user program, PC3000 system functions and parameters of EI Bisync Slave Variable function blocks.

Using LCM port C, it is also possible to multi-drop a number of PC3000s from a single RS422 serial link. For example, the PC3000 Programming Station can be connected via a 261 converter to a number of PC3000s multi-dropped using port C on the different LCMs; but note that each LCM should be set to a different Slave address (i.e. GID for EI Bisync).

## Addressing parameters

Whenever a user program is created on the PC3000 Programming Station (PS), all the communications addressable parameters of Function Blocks are given unique identities. For the default EI Bisync communications, each parameter in a user program is addressed uniquely by the Unity Identity (UID), Channel Identity (CHID) and mnemonic. A full description is given in the section on Parameter Addressing in the 'EI Bisync Slave Driver' description in this chapter.

It is stressed that there is no need to configure any communications function blocks or set-up any parameters within the user program in order to use the default communications. When the user program is created on the PC3000 Programming Station, the user has an option to produce two files '<name>.CEL' and '<name>.GAT' which define the addressable parameters within the user program. The <name> part of the file name will vary for each user program. Refer to 'PC3000 Program Station User Guide' for further details on how these can be produced.

Both address files contain similar information but in a slightly different format. Each contains a list of function block parameters which exist within a user program and their individual addresses and data types, i.e. floating point (REAL), boolean (BOOL) etc..

The 'CEL' file is used for the Production Orchestrator to create a database for addressing PC3000 parameters. A description of the file format is given in the appendix.

The 'GAT' file is used within ESP and contains a list of PC3000 function block parameters defined as ESP gates., For further information on the 'GAT' file structure refer to ESP documentation.

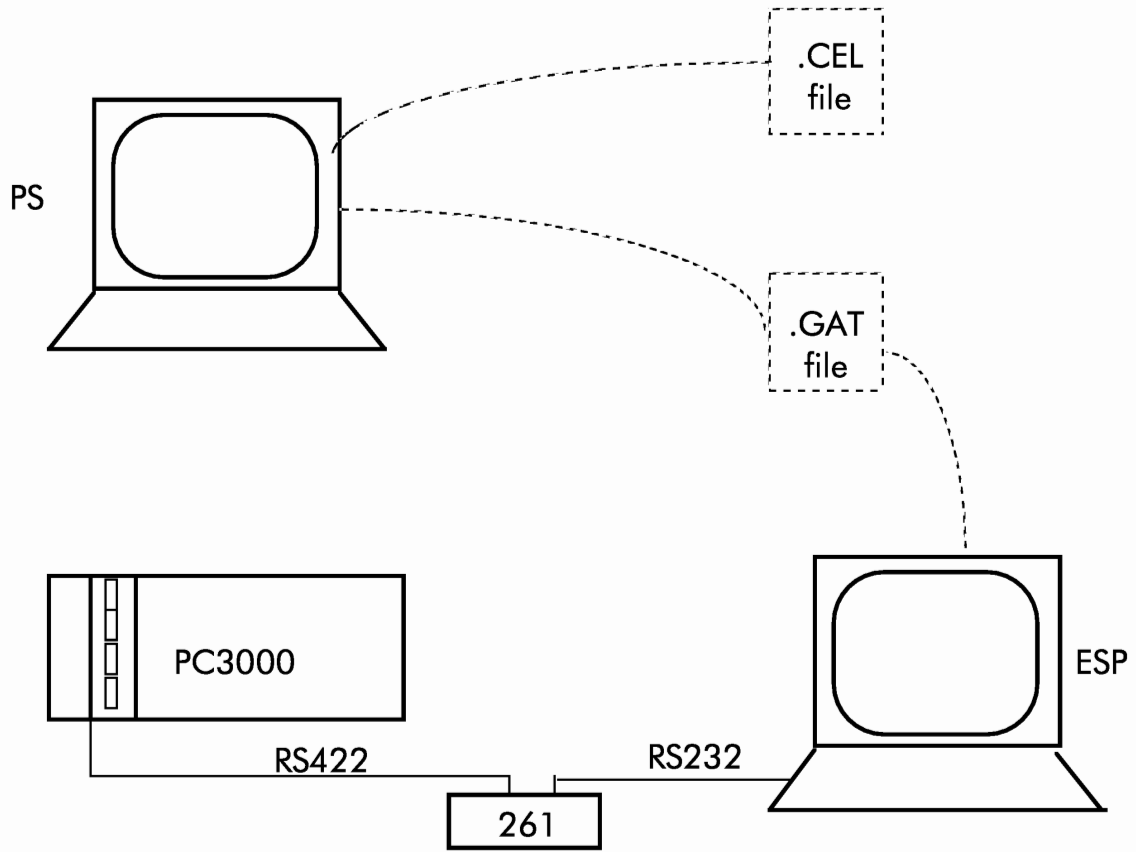


Figure 3-4 Communication with ESP

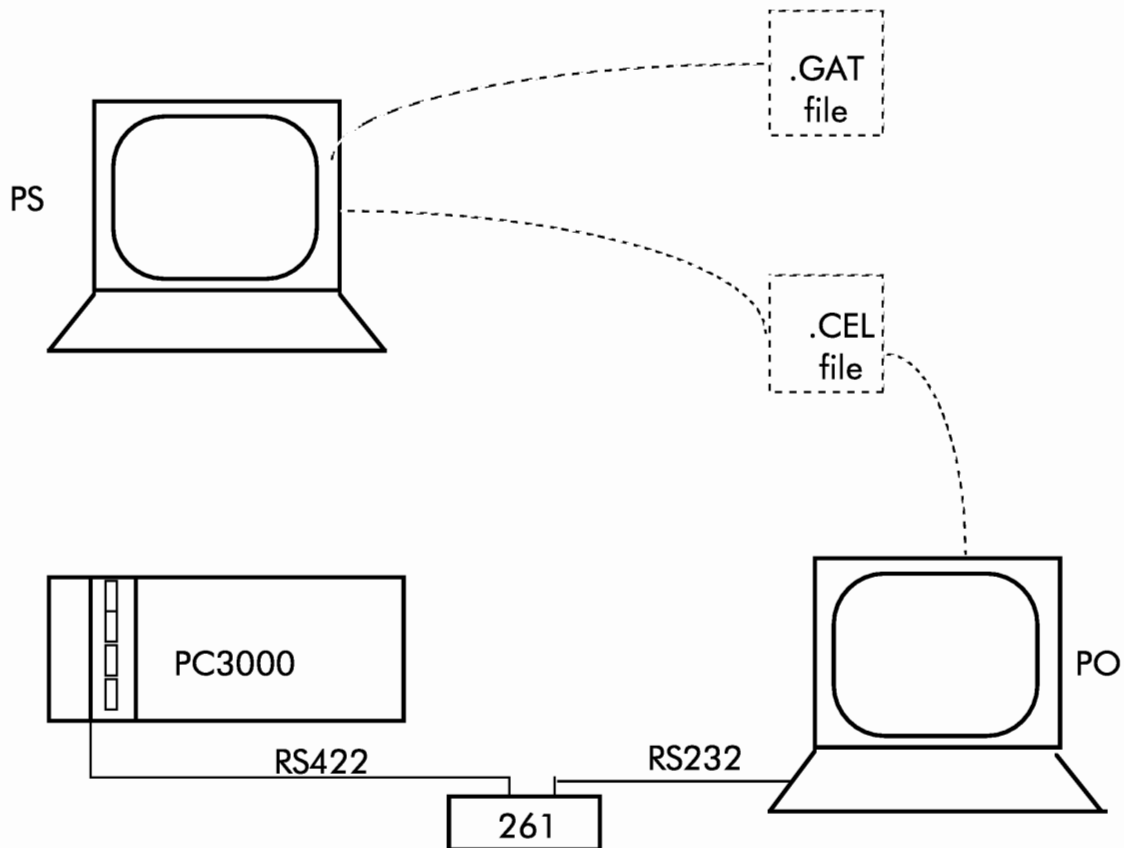


Figure 3-5 Communication with the Production Orchestrator

Normally, to set-up communications between PC3000 and ESP or Production Orchestrator, detailed knowledge of the EI Bisync protocol, or of the 'GAT', or 'CEL' file structure is not required, it is only necessary to:

Select a free LCM port A, B, or C. Port B is normally reserved for the Programming Station.

If the PC (used for ESP or Production Orchestrator) is not equipped with an RS422 interface, use standard cables, which can be ordered from Eurotherm, to connect between the port and a 261 and between the 261 and the PC.

Generate either a GAT (for ESP) or CEL (for Production Orchestrator) file on the Programming Station for the User Program.

Either copy the GAT file to ESP to create ESP screens or copy the CEL file to the Production Orchestrator cell build toolset.

When the user program is loaded and running, ESP or Production Orchestrator will be able to address function block parameters in the user program using addresses supplied by the address file.

Note: Whenever the user program is modified, particularly when function blocks are added or deleted, addresses of parameters of function blocks which are not EI Bisync 'Slave' blocks, may change. It is therefore recommended that the new GAT or CEL file is created and copied to ESP or Production Orchestrator for re-integration. The use of EI Slave Variable function blocks to avoid doing this, is described in a later section.

## **Custom applications**

In some cases, it may be necessary to develop an application program which functions as an EI Bisync Master, to communicate with a PC3000. For example, to create a customised PC based report generator that reads particular parameters from the PC3000. A PC application with an embedded EI Bisync Master driver can be speedily developed using a software library available from Eurotherm. Alternatively, an EI Bisync Master driver can be developed by reference to the 'EI Bisync Handbook'.

## COMMUNICATIONS FUNCTION BLOCKS

There are three types of function block which are associated with communications and are required if other protocols or additional communications facilities are needed. Using the Programming Station, these blocks should be created and configured within a user program in the same manner as any other function block.

**Communications Drivers Function Blocks.** These are used to allocate a protocol and establish various characteristics for designated ports.

**Slave Variable Function Blocks .**Used to provide parameters within a user program, that require predefined communications addresses. Slave parameters are accessible by a designated protocol from remote Master devices.

**Remote Variable Function Blocks.** These are used in association with a port that can operate as a Master, and provide a mechanism to read parameters or write parameters in a remote device.

### Driver function blocks

A port may be designated for use with a particular protocol by creating (instantiating) a protocol specific communications driver function block.

Protocol	Function block type	Comment
El Bisync Master	El_Bisync_M	
El Bisync Slave	El_Bisync_S	
JBus Master	JBus_M	Supports JBus and Modbus
JBus Slave	JBus_S	Supports JBus and Modbus
Siemens 3964(R)	Siemens_M_S	Supports both Master and Slave
Toshiba Master	Toshiba_M	For Toshiba EX250, EX500 and EX2000
Euro Panel	Euro_Panel	Supports connection to a Euro_Panel display
User defined	Raw_Comms	Permits low level control of the serial port

For a full list of communications drivers for the current version of the PC3000 refer to the 'PC Technical Summary HA022230'.

### Assigning a communications function block to a port

Every communications driver function block has a Port parameter which is used to assign the function block to a selected port. The port assignment is defined as a string of two characters, where the first character is a number in the range 0 to 5 representing the PC3000 rack slot and the second character is a letter representing the port within that slot. For example, '3A' would allocate Port A of the module in slot 3 of the main PC3000 rack.

The user should ensure that :

- a) the designated slot contains a hardware module such as an ICM, that can support serial communications before running the user program and
- b) that the particular port has not been allocated to other drivers.

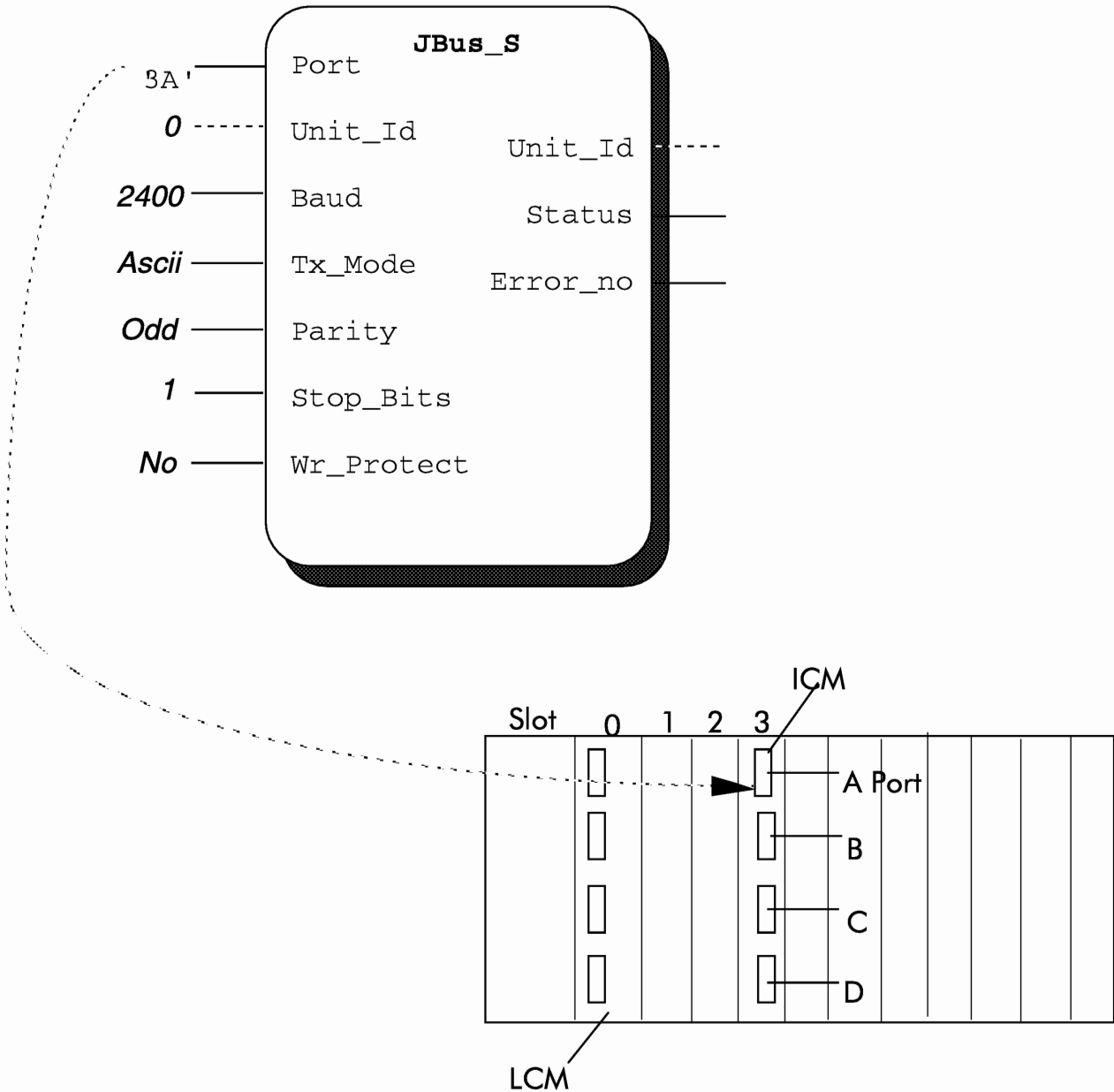


Figure 3-6 Assigning a communications driver function block to a port

### Changing communications driver function block parameters

Most communications driver function blocks have input parameters that can be used to customise the configuration of the allocated port, for example, to change the baud rate, and parity.



Although these parameters can be changed, they only take effect when the user program first begins running. Parameters in this category include:

Baud rate

Stop bits

Parity

A user program cannot dynamically change the communications configuration of a port by changing these types of parameters. However the configuration can be temporarily changed for test purposes via the Programming Station - see the next section. For further information on communications driver configuration parameters, refer to the specific communications driver in this chapter.

### Temporarily changing configuration parameters

Some communications function block parameters such as, the Port assignment, parity, baud rate of a communications driver are normally defined when the function block is first created. They can also be changed at any time via the Programming Station when on-line to the PC3000, but the PC3000 system only takes note of new values of communications function block configuration parameters when the user program begins running (this includes warm and cold start-ups).

To temporary change a parameter such as the port assignment for test purposes, the user program should be halted, the port assignment changed, and then the user program re-run. For permanent changes, parameters such as the port assignment should be modified in the off-mode, an the user program compiled and re-downloaded.

### Communications driver function block error detection

Every Communications Driver Function Block has the following output parameters:

**Status** When the user program is running this parameter will change from 'Go' to 'NoGo', if the communications driver detects an error. It is possible that Status will remain set to 'NoGo' if some configuration parameters are incorrect, such as having a faulty port assignment.

**Error No** This parameter is an integer which is set to a value to reflect the type of error detected. The value used will vary for different protocols. Where possible, error codes have been used to match those normally used with the protocol. If there is no error condition, it is set to 0 i.e. 'OK'. See appendix for a full list of standard error codes.

If the error condition is transitory, the **Status** and **Error-No** will change back to 'Go' and 'OK' respectively, the next time the communications function block is executed, providing the condition has cleared.

## Slave variable function blocks

Slave Variable Function Blocks are used to contain parameter values that can be accessed using specific communications addresses, via ports that support a particular Slave protocol. For protocols including EI Bisync, they allow parameters to be assigned a protocol specific address. Slave Variables are configured to use a designated protocol and can therefore be accessed from any port that supports that protocol. Although any number of Slave Variables can be associated with a particular Slave protocol, they cannot be accessed by different protocols simultaneously.

Each Slave Variable holds a value or set of values which can be read and written to by the user program and are also accessible via a designated protocol for reading and writing by a remote Master device.

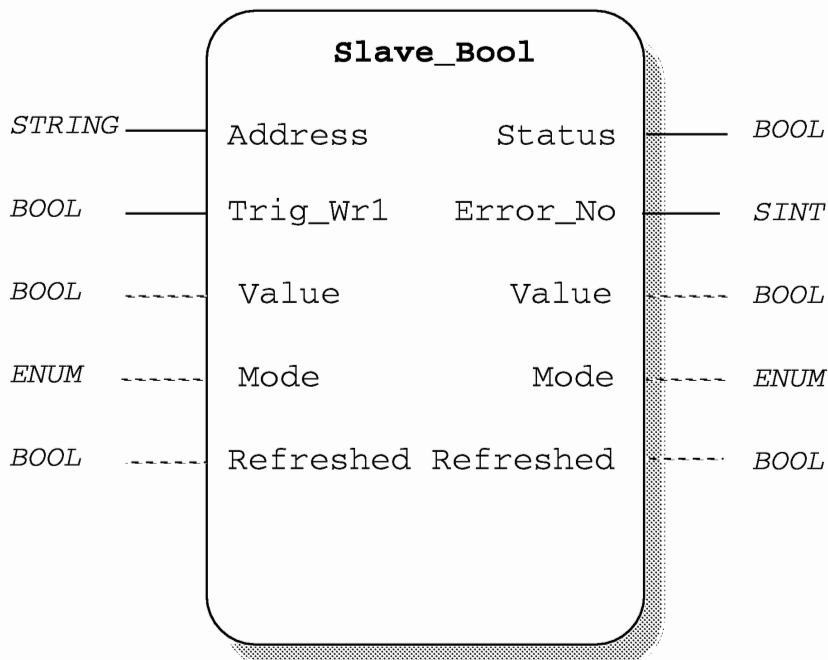


Figure 3-7 Example of a slave variable function block

Addresses of Slave Variables do not change when a user program is modified. In contrast, normal EI Bisync addresses used for function block parameters may change when a user program is modified, particularly when function blocks are added or deleted. Slave Variables are therefore particularly useful when there is a need to minimise the changes to parameter addresses in remote Master devices see chapter 3, section 'Addressing parameters using default communications'.

For example, by using a Slave Variable it is possible to provide a floating point (REAL) parameter that can be accessed by any port that supports EI Bisync Slave. The Slave Variable can be given a particular EI Bisync address and mnemonic such as: Channel='1', Mnemonic='80'.

This parameter can always be accessed using the same address and mnemonic from any port supporting EI Bisync Slave.

Typically, Slave Variables are used to provide parameter values that can be read and written by remote devices and supervisory systems, e.g. ESP, Operator Terminals using JBus, the Euro-Panel.

There are a variety of different Slave Variables to store parameters of different data types. The types supported include:

<b>Usage</b>	<b>Function block type</b>
Single Boolean	Slave_Boolean
Single Real	Slave_Real
Single Integer	Slave_Int
Single Time	Slave_Time
Single String	Slave_Str
Set of 8 Booleans	Slv_Boolean_8
Set of 8 Reals	Slv_Real_8
Set of 8 Integers	Slv_Int_8
Set of 16 Booleans read via communications as single 16 bit integer (Status word)	Slave_SW

### Assigning addresses and protocols to slave variables

Each Slave Variable has a configuration parameter which defines the protocol and communications address for the parameter. The address is defined as a string of characters where the first two characters select the protocol associated with the parameter and the rest of the string defines the protocol specific address.

The protocol selection characters include:

- EB Eurotherm EI Bisync
- EP Euro-Panel display
- SI Siemens 3964(R)
- JB JBus Slave

Refer to the latest 'PC3000 Technical Summary HA022230' for a full list.

Examples of address strings for Slave Variables are:

'EB060' - Parameter to be addressed by the EI Bisync Slave protocol with Channel ID = '0', Mnemonic = '60'

The GID and UID are defined by the EI Bisync communications driver function block.

'EBX1B' - Parameter to be addressed by the EI Bisync Slave protocol with Channel ID = 'X', Mnemonic = '1B'

The GID and UID are defined by the EI Bisync communications driver function block.

'JB0001' - Parameter to be addressed by the JBus Slave protocol as register 0001. The JBus Unit-Id is defined by the JBus communications driver function block.

Refer to the specific communications driver function block descriptions for details on protocol address conventions. In some cases, certain addresses are not permitted. For example, with EI Bisync, a channel should always be given, the first character of a Slave Variable mnemonic should be a digit the range '0' to '9', Slave Variables are always addressed using UID = '0' or 'P' if addressed from ESP.

The Address parameter cannot be changed while the user program is running and in this respect behaves as other communications function block configuration parameters, - see section: 'Temporarily changing configuration parameters'.

Figure 3-8 shows:

- a boolean Slave Variable with channel identity = '0', mnemonic = '60',
- an integer Slave Variable with channel identity = '0', mnemonic = '62',
- both can be addressed via each EI Bisync driver, via port '0A' using GID = '0', UID = '0' and via port '1B' using GID = '1', UID = '0'.
- an integer Slave Variable which can be addressed as register '0001' by a JBus Slave protocol via port '0C'.

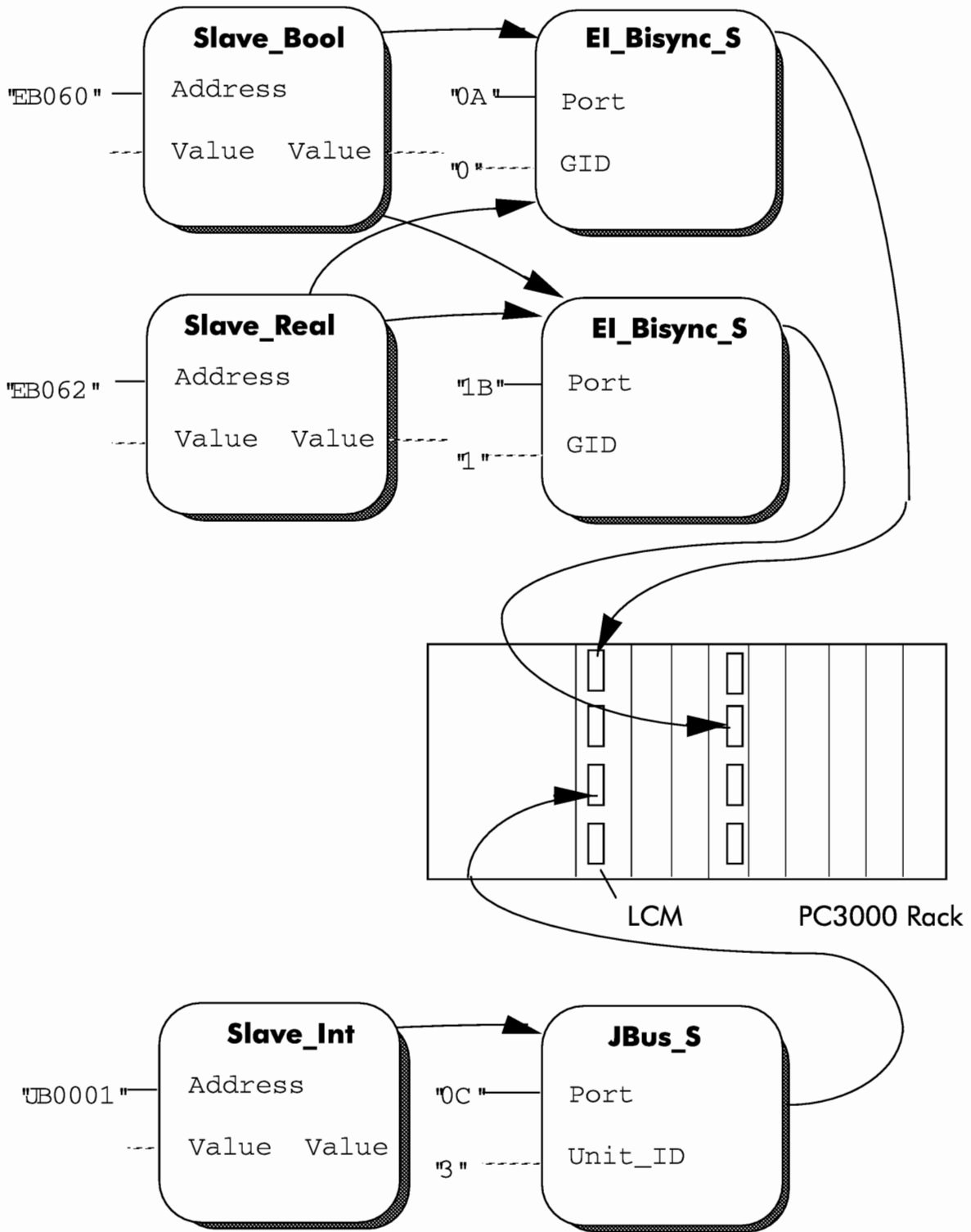


Figure 3-8 Slave variable usage example

## Listing slave variable addresses

The Programming Station provides an option to produce a file containing a list of Slave Variables with communications addresses for a selected protocol. This can be useful for example, when configuring a display terminal such as CRT9M (9 inch monochrome) or CRT12C (12 inch colour) using JBus where there is a large number of Slave Variables with numeric addresses to remember.

The file generated is called '<name>.ads', where '<name>' varies for different user programs.

## Control of slave variable function blocks

All Slave Variable Function blocks have a standard set of parameters which include:

- Mode** This can take the following values: Rd-Wr - which allows the Slave Variable value to be read or written to over communications, Rd-Only, which inhibits remote communications from writing to the Slave Variable and Wr\_Once which allows the Slave Variable to be written once but thereafter further writes are inhibited. Wr\_Once is used to ensure that a remote device will not overwrite a value written to a Slave Variable before the user program has had time to deal with it. The mode changes from Wr\_Once to Rd-Only when a write to the Slave Variable is complete.
- Trig\_Wr1** This is a boolean parameter which on being changed from Off to On causes the Mode parameter to change to Wr\_Once. It should only be driven from a source derived by softwiring, such as a digital I/O input. It should be switched back to Off to be re-used again. This parameter can be used as part of an interlock to regulate writes to the Slave Variable from a remote device.
- Refreshed** This boolean is set to Yes whenever a new value has been written to the Slave Variable by a remote device; it can also be cleared by the user program. The main use is to signal to a user program when a new value has been written to the Slave Variable by a remote device.
- Status** An output boolean parameter that can either be Go or NOGO to indicate that an error has occurred while running with the Slave Variable.
- Error\_No** The output integer parameter changes from 0 (OK) to an error code value whenever the Status is changed to NOGO. Because the error code is set by the associated protocol Communications driver, the Communications Driver Function Block description should be consulted for a full description of error codes. An error code value of 255 indicates that the Slave Variables has not been initialised and could imply that an incorrect protocol selection characters have been given in the Address parameter. See appendix C for a full list of standard error codes.

## Interlocked access

With many applications it may be acceptable for the value of a Slave Variable to be overwritten by the remote Master device before the current value has been processed by the user program, or for a new value to be locally written to be Slave Variable before the remote device has read the last value.

However, there are situations where the exchange of data from the remote device should ensure that every value is transferred and acknowledged by the user program. This can be achieved by using a flag Slave Variable as an interlock. Figure 3-9 depicts a typical situation where a remote device is sending a stream of values to the user program via Slave Variable Val1. The user program ensures that only one remote write is possible before processing the new value written to Val1 by changing the mode to Wr\_Once. Although the mode Wr\_Once will block the remote device from overwriting the value, it will also cause a communications error to be reported within the remote device if a new value is written. A better arrangement is to use a second Slave Variable as an interlock. The user program sets the Flag to signal that a new value can be written. The remote device then polls the Flag and when set, is free to send the next value.

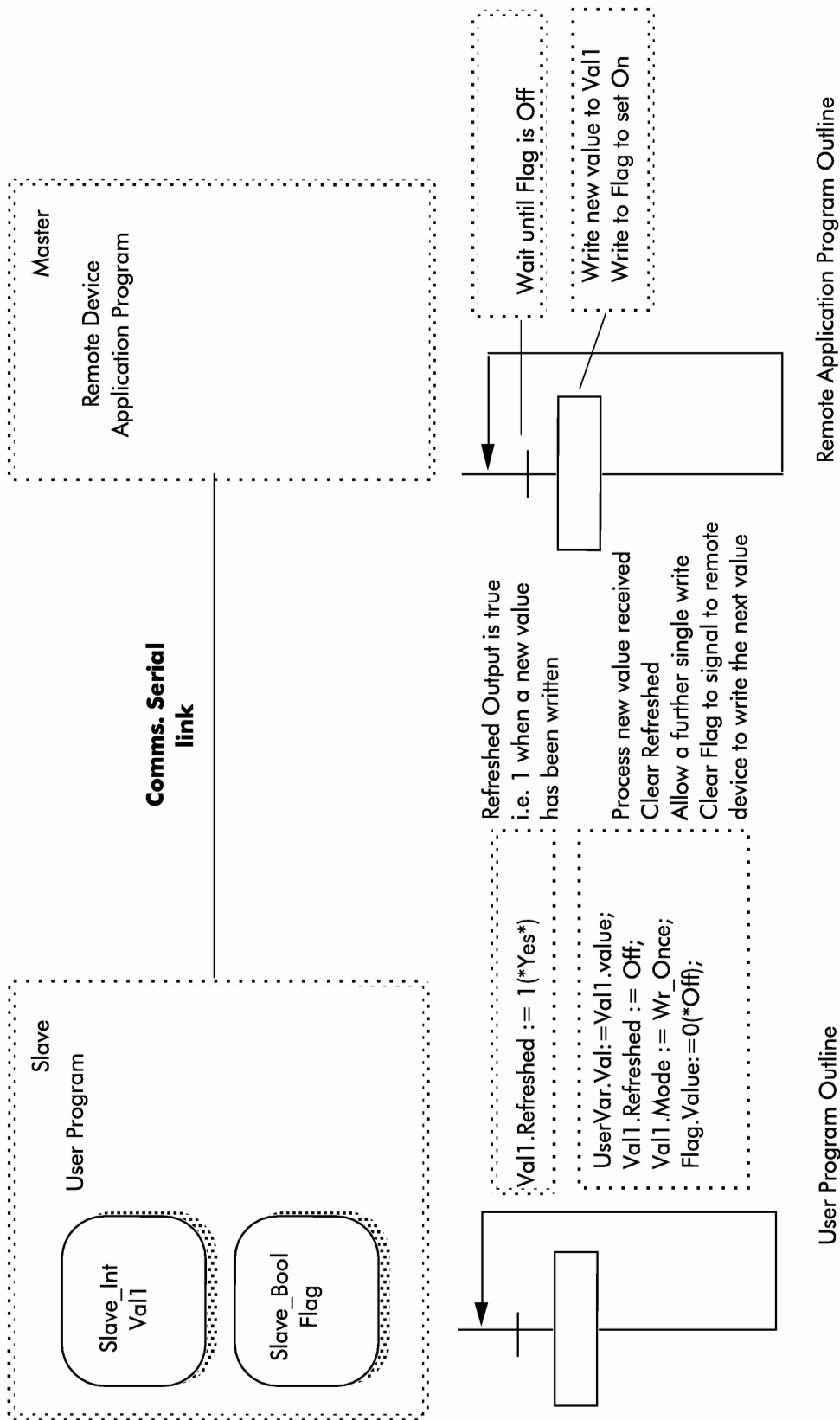


Figure 3-9 Example of interlock arrangement



## Slave variables timing constraints

If the communications driver function block is assigned to a faster PC3000 task than the Slave Variable function block, some side-effects may be observed.

The following situation may arise. When the remote device sends data to be written to a Slave Variable, the communications driver writes the data to an internal buffer within Slave Variable function block. The data is not written to the Value output parameter of the Slave Variable until the task to which the Slave Variable is associated, executes. Consequently, if a fast responding remote device writes to a Slave Variable and then attempts to immediately read back the value before the Slave Variable has updated, the remote device may unexpectedly read back the older value from the Slave Variable and not the value just written.

Further write attempts to a Slave Variable, that occur before the internal buffer has been transferred to the Slave Variable output will be blocked and will cause the communications driver to return a negative acknowledgement (NAK) for each attempt. If the Slave Variable is in the write once mode `Wr_Once`, the internal buffer will hold the first write value and cannot be overwritten by further writes. However, the new value held in the internal buffer will not be copied to the output of the Slave Variable until the Slave's task executes.

## Slave variables worse case timing

The worse case delay between receiving a serial message from a remote device to write to a Slave Variable and it appearing on the output of a Slave Variable is:

= task interval for the Communications Driver function block + task interval for the Slave Variable function block

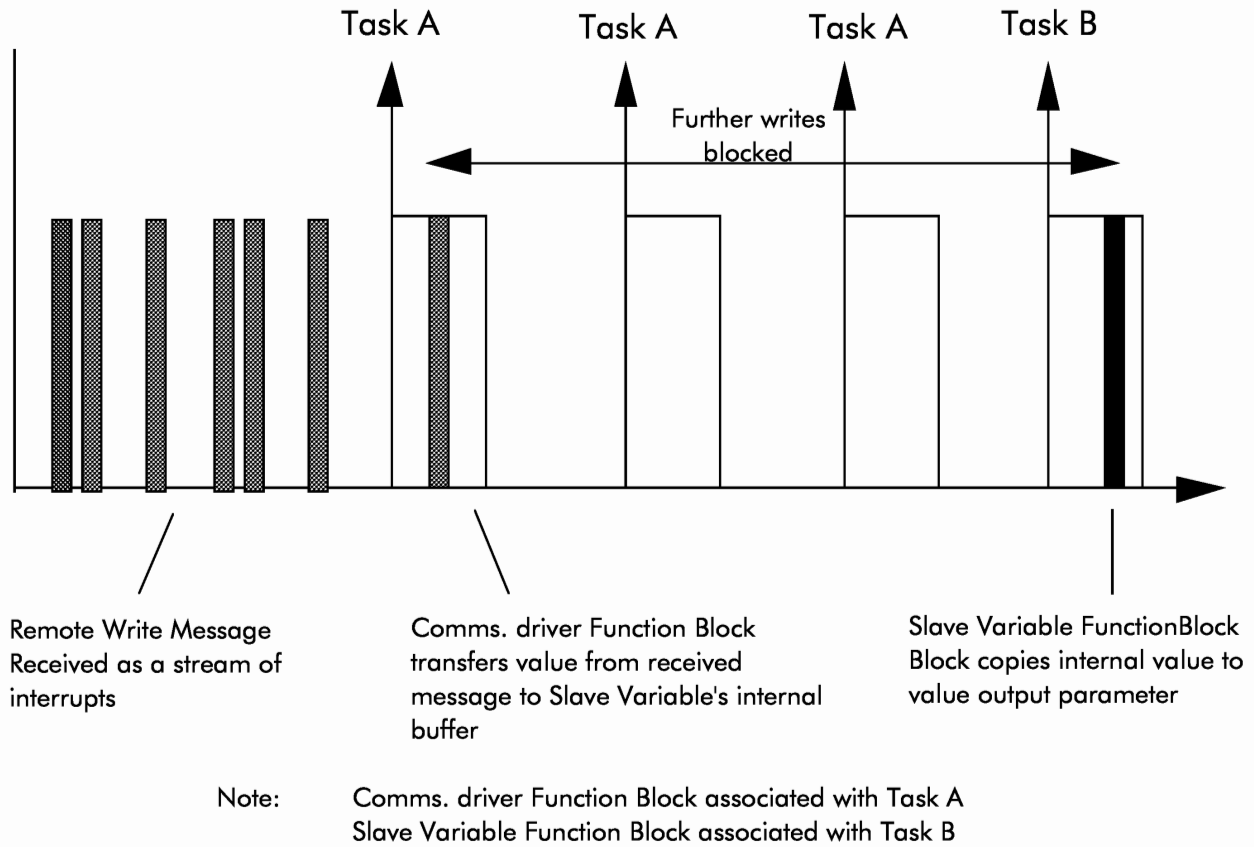


Figure 3-10 Delays associated with updating a slave variable

For example, with the Communications Driver and the Slave Variable both running in a 10ms task, the worst case time between receiving the last character of a serial message and the decoded value appearing on the Slave Variable Value output parameter will be 20ms.

The worst case delay between a user program writing a new value to a Slave Variable, and the value being available to be read over communications from a remote device is:

= task interval for the Communications Driver function block

Note that the worst case end-to-end times between the remote application program and the user program should also include the serial communications transmission times which will vary with the Baud rate, and inter-character and message latencies of the remote device.

## Remote variable function blocks

A variety of Remote Variable Function Blocks are provided to read from, and write to parameters of different data types in remote devices. Remote Variables can only be used with ports the support protocols that function in the Master mode, such as EI-Bisync-M, JBus-M and Siemens-M-S.

Typically Remote Variables are used when the PC3000 is Master of a serial link from which a number of devices such as Eurotherm 900 series instruments are multi-dropped. In such cases, Remote Variables can be used to read and write selected parameters within any of the instruments.

Each Remote Variable is configured using an address string that defines the port to be used, the Slave address of the remote device connected to the port, and the address of the parameter within the remote device.

The facilities provided by each Remote Variable Function Block are controlled using a standard set of parameters and include:

- On demand a single read of a remote parameter

- Continuous reading, i.e. polling of a remote parameter; the duration between reads can be specified.

- On demand, a single write to a remote parameter

- Measurement of the actual elapsed time between continuous reads or writes.

- A time stamp for the last successful read or write.

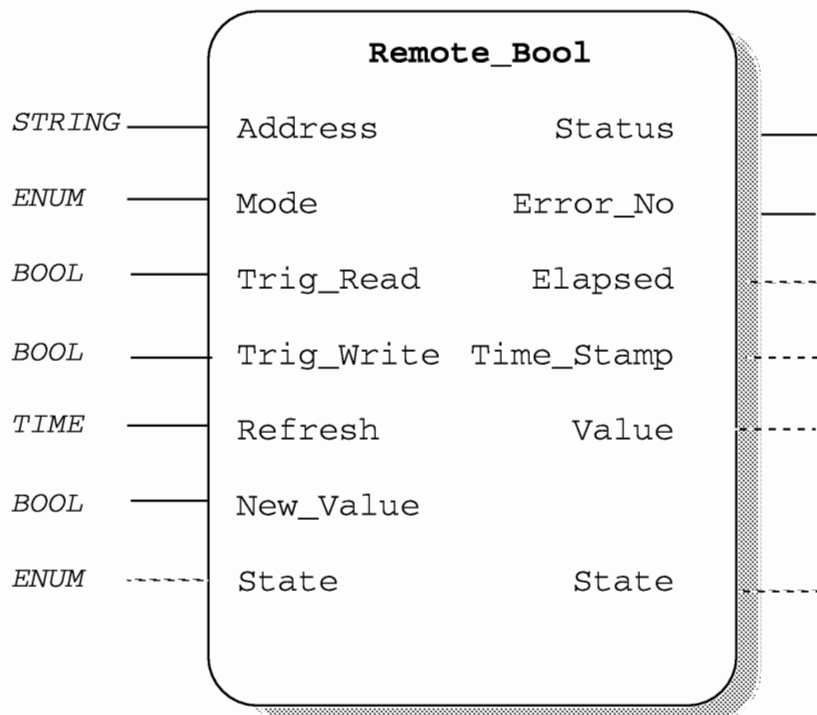


Figure 3-11 Example of a Remote Variable function block

There is a variety of different Remote Variables to interface to remote parameters of different data types. The types supported include:

Usage	Function Block Type
Single Boolean	Remote_Bool
Single Real	Remote_Real
Single Integer	Remote_nt
Single String	Remote_Str
read via coms as single 16	
bit integer (Status word)	Remote-SW

Refer to the latest 'PC3000 Technical Summary HA022230' for a full list.

### Assigning Addresses to Remote Variables

A Remote Variable can be associated with a parameter in a remote device by defining a value for the Address parameter. This should be a string where the first two characters define the PC3000 port and the rest of the string defines the protocol specific address. In some cases, extra characters are added to the end of the address string, to provide extra information such as, the format used to transmit the parameter over the serial link. The port selection uses the same convention as used with Comms. Driver Function Blocks, see section 'Assigning a Comms. Function Block to a Port' where first and second characters define the PC3000 rack slot and port respectively.

Examples are:

'OA011PVf'

Addresses Port A of the LCM (slot 0), -

assuming the port is allocated to a EI Bisync Master communications.

Function Block (EI-Bisync-M),

then the rest of the address implies:

GID = 0, UID = 1, - defines Slave device address

Channel ID = 1, Mnemonic = 'PV',

- defines Parameter address

-format f - defines the Bisync format used to transmit the value.

'1B011234IR1'

Addresses Port B of a communications module in slot 1, -

assuming the port is allocated to a JBus Master Communications.

Function Block (JBus-M),

Then the rest of the address implies:

Unit-id = 01 - defines JBus Slave device address

Register address = 1234

I - defines Input address space

format R1 - defines a JBus format

For details on the protocol specific part of the address, refer to the appropriate Communications Driver Master Function Block description. Figure 3-11 shows:

A Boolean Remote Variable addressing port '0A' which has been allocated to the EI Bisync Master protocol. The address selects a Slave device with GID=0, UID=1, and a parameter with mnemonic = 'HD' using the default format. There is no channel identity.

A floating point (REAL) Remote Variable addressing port '1B' which is allocated to the EI Bisync Master protocol. The address selects a Slave device with GID=0, UID=0, and parameter with channel identity = '2' and a mnemonic = 'PV' using the format '1'.

An integer Remote Variable addressing port '0C' which is allocated to the JBus Master protocol. The address selects a Slave Device with Unit-Id = '020', and a register at address '1000' using format 'R'.

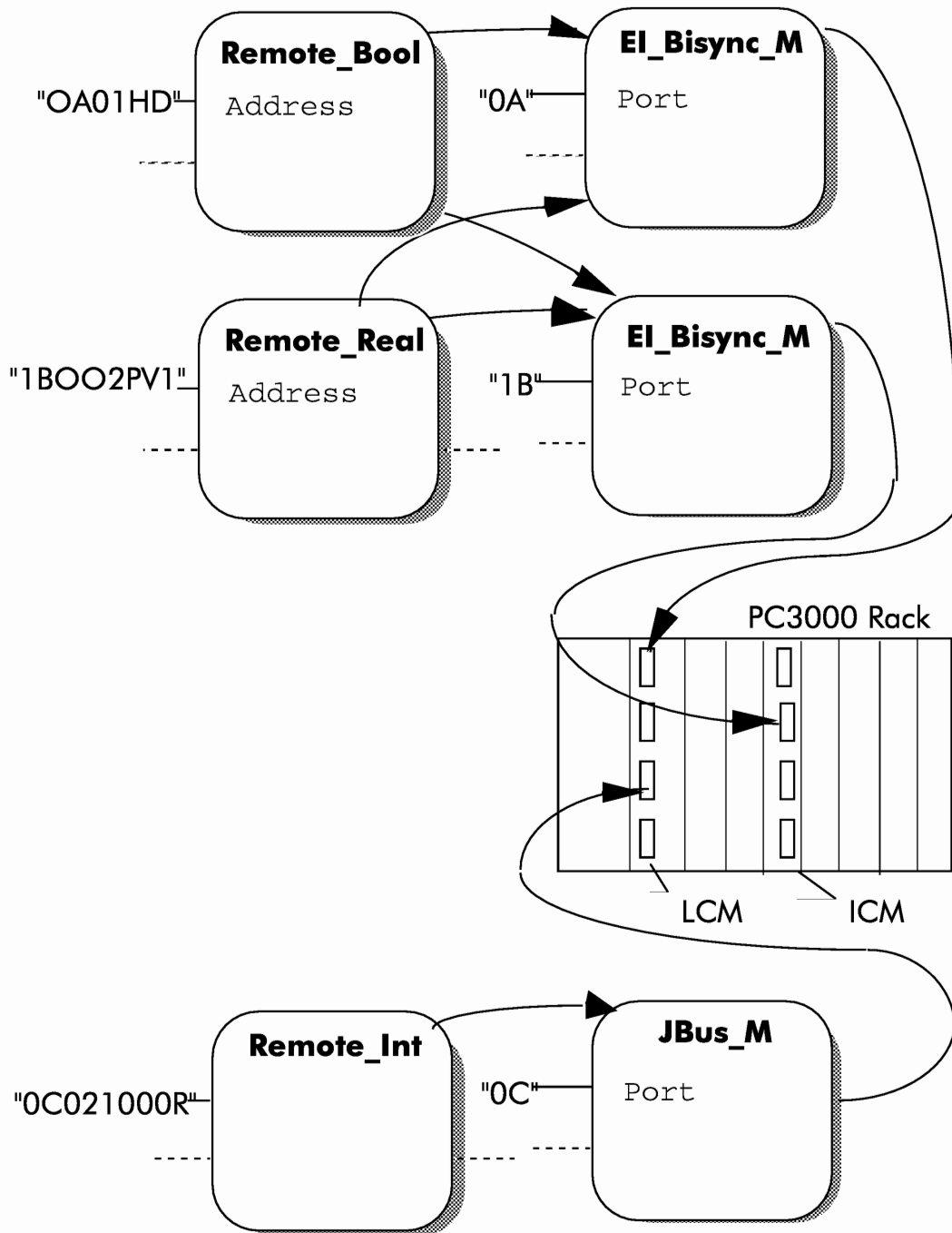


Figure 3-12 Remote variable usage example

### Changing remote variable addresses

Address string of a Remote Variable can be changed at any time by the user program and the new address comes into effect the next time the Remote Variable executes. This provides a very flexible system where it is possible for a single

Remote Variable to be used to address different parameters of the same data type on different ports and using different protocols.

For example, by changing the address, it is possible to read the Process Value PV from a number of multi-dropped instruments on a number of ports.

e.g.: addresses to read the PV for four instruments, 2 multi-dropped off each of ports 1B and 1C.

```
'1B01 PV'  
'1B02 PV'  
'1C01 PV'  
'1C02 PV'
```

## Control of remote variable function blocks

When in a 'read' mode, the **Value** parameter will contain the last value read from the parameter in the remote device.

In a 'write' mode, the value held by the **New Value** input parameter is written to the remote parameter. If the write operation is successful, the Value parameter is updated to match the **New Value** parameter.

In general, the **Value** parameter will hold the value last successfully written to or read from the remote parameter. However, if no successful read or write operation has ever occurred, the **Value** parameter should be ignored.

All Remote Variable parameters have a set of parameters to control the various modes of operation, these include:

### Mode

This specific basic operational mode which can be:

'Demand' mode - allows a remote read or write can be triggered by changing the State to Write or by the Trig\_Read or Trig\_Write parameters changing from Off to On.

'R\_Cont' mode - allows a remote parameter to be continuously read (polled) at a regular period defined by the Refresh TIME parameter.

'W\_Cont' mode - allows a remote parameter to be continuously written to, at a regular period defined by the Refresh TIME parameter.

'Change' mode - causes the remote parameter to be written to when the value of the New Value input parameter does not match the Value parameter. This mode can be used as an alternative to the write continuous mode when a reduction in the number of communications transactions is desirable, because writes that attempt to send the same value are avoided. The minimum elapsed time between changes being transmitted to a remote device is given by the Refresh time. If an error occurs, the write will be repeated at a regular period defined by the Refresh TIME parameter.

Trig\_Read, Trig\_Write. These parameters should be used when it is necessary to trigger a single read or write on demand from source derived by soft-

wiring. For example, a digital I/O input could be soft-wired to either of these parameters to trigger a read or write from an externally generated digital pulse. They should not be written to using ST in a Sequential Function Chart (SFC) (see State parameter).

### **Refresh**

This TIME parameter defines the period between communications transactions for the continuous read or write modes. See Mode for other uses of this parameter.

### **State**

This is an input/output parameter that can have the following values: OK, Pending, Error, Write, Read.

The output State parameter can take values: 'Pending' while a transaction has started, and is awaiting completion, usually while waiting for a remote Slave device to respond, 'Ok' when a successful transaction has completed or before any transaction has been made, and 'Error' when a transaction has failed for some reason (see Error-No).

The input State parameter can be written to in an SFC to make the Remote Variable to perform various functions. Setting the State parameter to 'Write' will trigger a single write of the value of the current New Value parameter to the Remote device irrespective of the current selected mode. For example, if Mode is 'R Cont' for continuous read, setting State to 'Write' will trigger a single write between the read transactions. Similarly, setting State to 'Read' will always trigger a single read transaction. The State parameter will always trigger a single read transaction. The State parameter will remain as 'Write' or 'Read' until the next execution of the Remote Variable function block when it will change to 'Pending' if the transaction can be initiated successfully, otherwise it will change to 'Error'.

Setting State to 'Ok' will cause any current transaction to be aborted. For example, if a write transaction is waiting for an acknowledgement from a remote device, forcing the State parameter to change from 'Pending' to 'Ok' will cause the acknowledgement message to be ignored when it is received. Setting State to 'Error' can be used for test purposes when no transactions are pending, to simulate an error condition; the Error No is set to 255 on the next execution of the Remote Variable Function Block in this case.

**Error-No** This is an integer that identifies an error condition, 0 implies 'Ok' other values are set if the State parameter is 'Error'. A full list of standard error codes is given in the appendix C.

### **Elapsed**

A TIME parameter that gives the elapsed time since the last read or write modes; in other modes its value has no significance. If the Elapsed time is significantly longer than the required Refresh period,



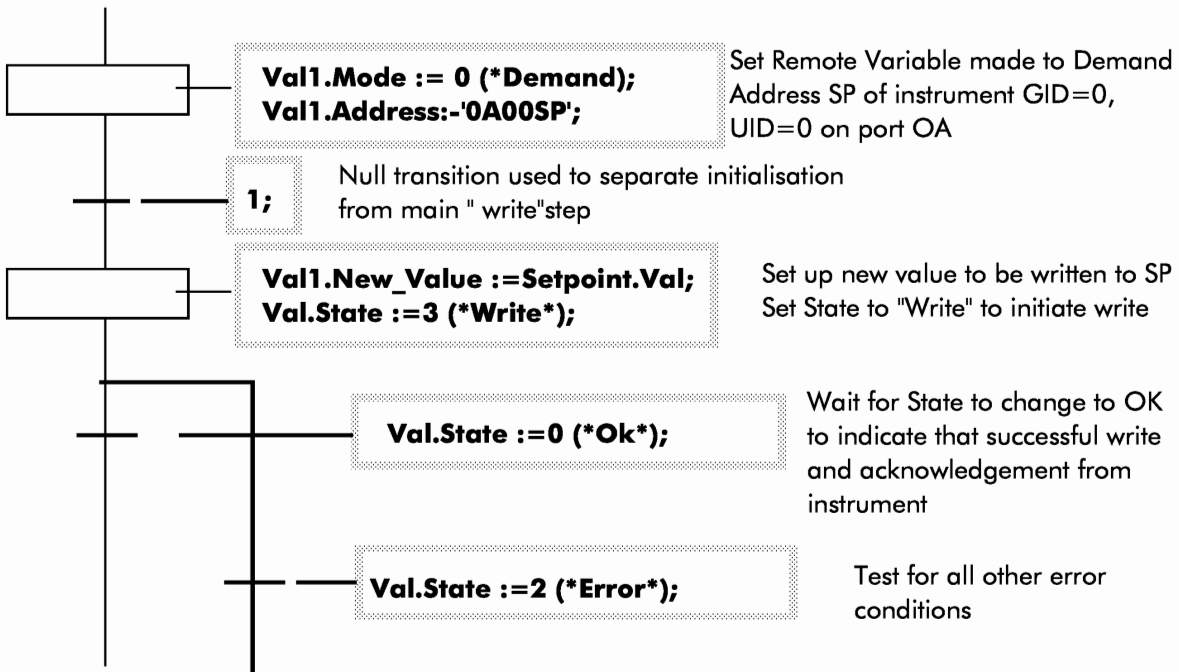
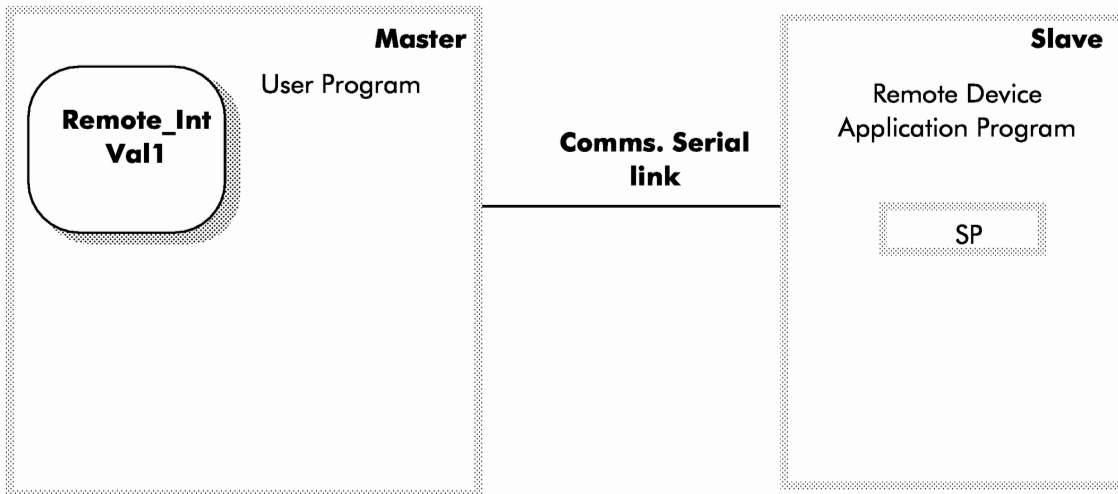
this may indicate that either the Remote device, the serial link or the PC3000 system cannot handle communications transactions at the rate required. If the transaction times-out because no response is received, the Elapsed time is set on the termination of the transaction.

**Time\_Stamp** This defines the time and date at which the last read or write transaction completed successfully. It is updated when the Remote Variable function block executes and is set coincident with the State parameter changing to 'Ok'.

**Status**

An output boolean parameter that can either be GO or NOGO to indicate that an error has occurred while running with the Remote Variable.

Example using remote variable



User Program Outline

Figure 3-13 Using a remote variable example

Figure 3-13 shows in outline part of SFC needed to write a new Setpoint value to a remote instrument. Note that only the address is specific to a particular protocol the rest of the code will be identical for different protocols.

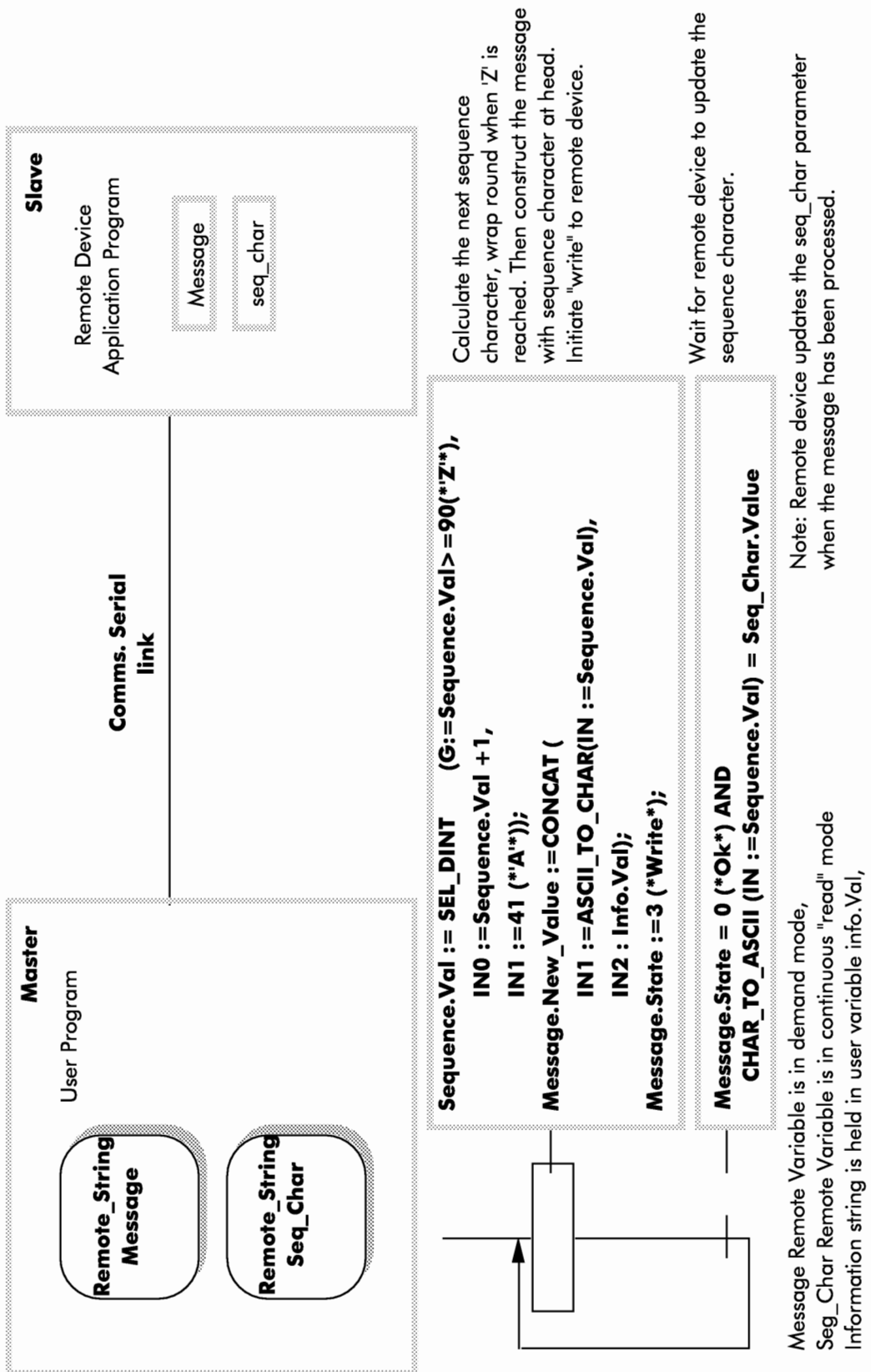


Figure 3-14 Interlock access to a remote device example

### Interlocked access to remote device

With some applications it is important that every message is received by a remote device i.e. there is no possibility that a message can be overwritten before being processed by the remote device. One arrangement for this is shown in the figure 3-14 'Interlocked Access to a Remote Device Example'. The information is assembled into a string that is written to a parameter within the remote device. A character is added to the head of the string as a sequence number. Each new string transmitted is prefixed with the next character in a defined sequence, e.g. A,B,,X,Y,Z,A,B etc. The sequence wraps round to A on reaching Z.

The PC3000 has two Remote Variables, one to write the information string and the other to read back the sequence character which is updated by the remote device when the message has been processed. The sequence character within the remote device is polled by the PC3000. When it matches the sequence character used in the last string, new information can be transmitted.

### Remote variable worse case timing

Assuming that there are not other Remote Variable transactions queued, the worse case delay between writing to Remote Variable within a user program to initiate a communications transaction and detecting the completion of that transaction is:

```
= task interval of the Remote Variable Function Block +
  task interval of the port communications Driver Function
    block +
communications serial link transmission time to Remote
  Device +
Remote Device processing time +
communications serial link transmission time back to PC3000
  +
task interval of the port Communications Driver Function
  Block +
task interval of the Remote Variable Function Block
```

For example, to read a Process Value (PV) from a 905 series instrument using a Remote Variable assigned to 20ms task, a EI Bisync master communications function block assigned to 10ms task, using a serial link running at 9600 Baud will have a worse case latency of:

```
= 20ms +
  10ms +
  10ms+
  2ms +
```

8ms +  
20ms +  
10ms+  
80ms

### Queuing remote variable transactions

It is possible to initiate a number of Remote Variables to make transactions on the same port within a short period of time. Each communications driver function block that supports Master mode operation has an internal queue to hold outstanding transactions from Remote Variables. Typically a driver can hold up to 100 outstanding transactions but refer to the specific communications driver function block description for the exact queue size. The Queue\_Space output parameter for the communications driver function block can be used to monitor the space available.

Transactions are queued on a first come, first served basis. Note that there is no specific queue order for transactions for Remote Variables to the same port that are initiated within the same SFC execution, for example, within the same SFC Step and that only one transaction can be queued at a time for a particular Remote Variable.

Serial communications drivers currently provided are unable to overlap transactions, therefore each transaction has to complete, i.e. a communications message should be received from a remote device and passed to the associated Remote Variable, before the next transaction can be started; the queue space used for the completed transaction is freed.

If there are transactions already queued, the turn-around time for a fresh Remote Variable transaction includes the time to complete all the transactions already in the queue.

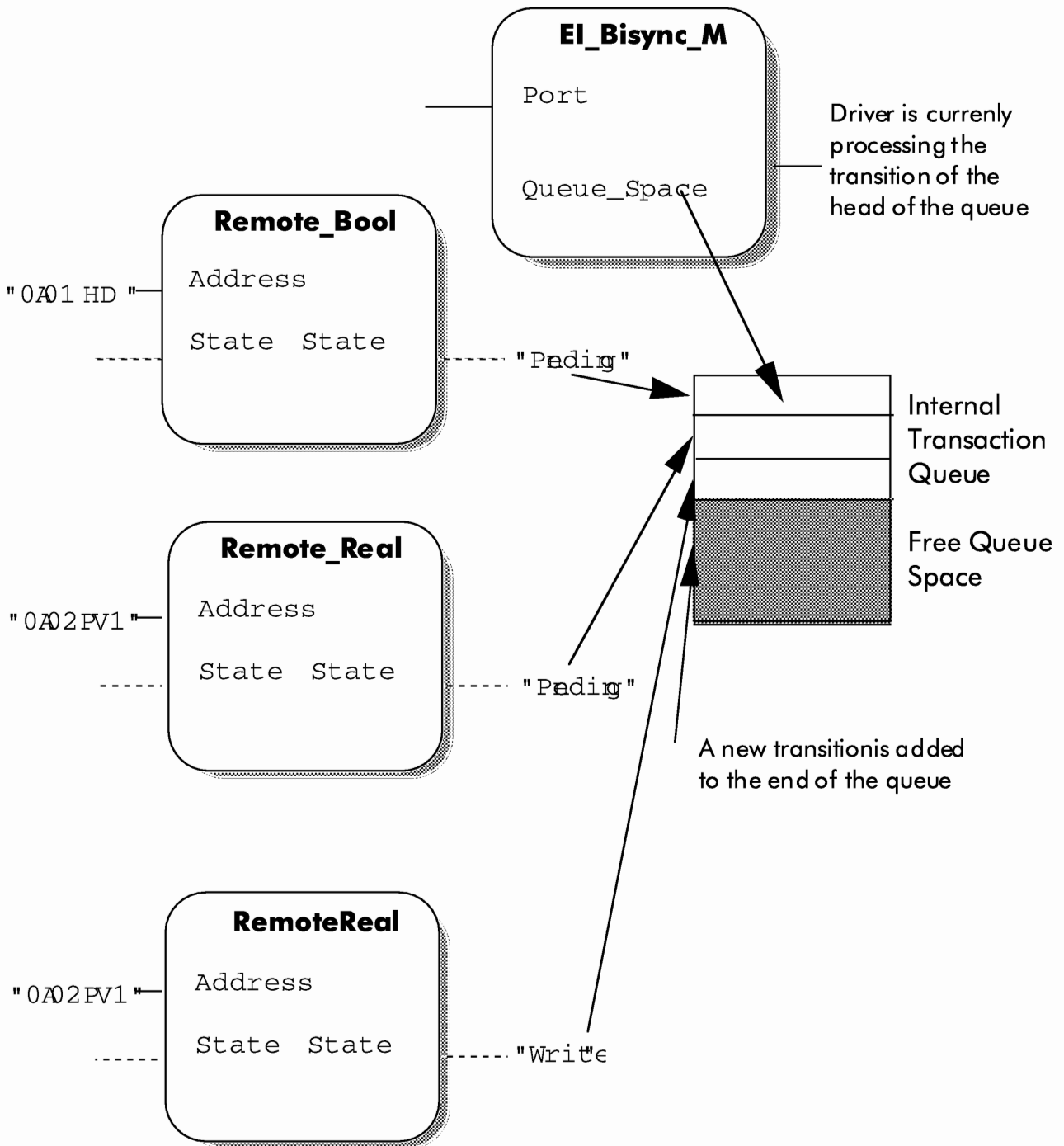


Figure 3-15 Queuing transactions from remote variables

### Queue space problems

If there is no queue space available for a new Remote Variable transaction, the Remote Variable will continually retry the transaction until space is available. This will increase the transaction turn-around time.

To detect communications delays due to this condition, the User Program should monitor the communications driver Queue Space parameter and if the queue space is low, take corrective action to reduce the communications traffic on the associated port, for example, by increasing the Remote Variable Refresh period for continuous Reads or Writes.

The Queue Space value can be used to indicate problems with a particular serial link or with a remote device. If the queue space rapidly decreases, this implies that communications transactions are taking longer to complete than usual. This may be because communications transmission errors are being detected and certain messages need to be re-transmitted, or because a particular device cannot respond, so transactions to it are being cleared by the driver time-out mechanism.

## **Raw communications block**

The Raw\_Comms driver function block is provided for applications where it is necessary to have low level control of the communications port and require the flexibility to construct or analyse messages exactly as transmitted or received over a serial link. Because there is no protocol, no extra messages or message formats are created by the driver.

Just as for other communications driver function blocks, the Raw\_Comms block can be assigned to any serial port using the Port input parameter (see section 'Assigning a Communications Function Block to a Port').

Raw-Comms provides a wide range of low level facilities including:

- Direct access to the message transmitted or received over the serial link.

- Independent control of message transmission and reception including separate baud rate selection on transmit and receive lines.

- Message flow control using Clear to Send (CTS) and Request to Send.(RTS)  
See note 1.

- Selectable echoing of received characters when required.

- User selectable delete sequence for character deletion in the receive buffer.

Developing user programs to operate with Raw\_Comms is more complex than using the protocol driver function blocks because the structure of messages and timing will all need to be handled by the program.

**Note** that it is not possible to use Slave or Remote Variables with Raw\_Comms.

For a full description, refer to the 'Raw Communications Driver' document.

## **Raws\_Comms usage**

Typical applications for the Raw\_Comms driver function block include:

- Communications with devices using simple non-standard protocols,

- Sending reports to serial line printers or to special purpose printers, for example, for label generation.

Communication with character based terminals such as DEC VT100 or with simple display devices.

**Note 1** This is not currently supported on LCM or ICM ports

## Euro-Panel driver function block

A special slave communications driver function block is provided to handle the interface with a Euro-Panel, a 2 by 40 character display with numeric keypad and function keys. The Euro-Panel function block manages all the low level communications message dialogue with the display, so that creation of user specific messages, display fields and data entry fields is easy to program. Although the Euro-Panel uses RS422, only one panel can be assigned to any RS422 port.

The format of messages displayed on the Euro-Panel is defined by a set of format parameters. These are input parameters to which strings of characters can be assigned to define the format of message, input fields, numeric display formats etc. using a simple display format language called Operator Interface Formatting Language (OIFL).

The Euro-Panel display is the master of the RS422 serial link. Slave variables can be associated with one or more Euro-Panel driver function blocks by using the protocol selection characters 'EP' in the Slave address. (see section 4 'Assigning Addresses and Protocols to Slave Variables'). The rest of the Slave address is used to assign a unique name to be used by the Euro-Panel driver within the format strings, this can be any alpha-numeric string (including underscore '\_'). The value of a Slave variable assigned to a Euro-Panel is automatically displayed on the panel, if it is referenced in a format string.

Example of Slave Variable addresses used with the Euro-Panel

```
'EPpvl'  
'EPmaxtemp'  
EPpump'
```

These can be displayed using format strings assigned to the Euro-Panel driver. Examples are:

```
''PV 1 is:', pvl'  
'Max temp: ', maxtemp:7.2'  
'@10:1, 'Pump', pump (Off,Vacuum,Purge)'
```

The display format strings can be changed dynamically during user program execution, so that menu lists and parameter lists can be changed in response to function keys on the panel.

Refer to 'the Euro-Panel Driver' function block description for details on the capabilities and programming the Euro\_Panel display.



## PROBLEMS AND SOLUTIONS

This section lists some common problems with serial communications and are given for general guidance; the list should not be regarded as comprehensive. In the first instance, identify the Error No of associated communications function blocks. A full list of error codes is given at the end of each communications driver function block description.

- a) Problem - Basic communications between a remote device and PC3000 cannot be established.

Check the following:

The cable is correctly connected, check the polarity of the Receive and Transmit lines. With RS422 check that all tee points are connected correctly. With RS232, check that the common line is connected end-to-end and if required, that the appropriate modem control signals are simulated for the remote device.

The Baud rate, Stop bits set up for the communications driver match those expected by the remote device.

The protocol of the communications function block matches the remote device,

Check that there is a single master device for protocols that require a single master with slaves, i.e. either the PC3000 or the remote device is the master.

- b) Problem - A remote device cannot read or write to a Slave Variable.

Check the following:

Slave address is assigned to a known protocol

A communications function block for the selected slave protocol has been created,

The communications function block is assigned to the same port as connected to the remote device,

The correct slave address is being used by the remote master device

If a write to the Slave is failing, check that the slave mode permits write transactions, and that the communications driver function block has any write protect parameter such as **Wr\_Protect** set to No.

- c) Problem - Remote Variable cannot communicate with a remote device:

Check the following:

The Remote Variable addresses a port associated with a communications driver function block that supports a protocol that operates in master mode,

The correct port for the remote device is addressed,

The correct address string is given for the parameter within the remote device, (With EI Bisync devices ensure that the correct channel character is

given, if the device does not use channel numbers ensure that a space character is given for the channel character).

The correct format character has been postfixed to the address for the remote device parameter,

That each remote slave device has a unique slave identity, for example, with EI Bisync each remote instrument has a different GID.

- d) **Problem** - A communications system may involve connecting numerous multi-dropped instruments to the PC3000 on a single serial link. During commissioning, some of the instruments may not be present or may not be powered-up. A common requirement is for the user program to use Remote Variables to poll parameters in each of the instruments. However, with some instruments not present on the serial link, transactions to the missing instruments will take much longer to complete since the driver will wait for a time-out and then retry each failed transaction. This will considerably slow down the communications transaction to the other instruments and may cause the driver's queue space to decrease.

**Solution** - The user program should test the **Status** and **Error\_No** on each Remote Variable to check for a time-out condition. The **Status** will be set to NOGO and the **Error\_No** will be a non-zero value according to the appropriate error code for time-out condition. The actual error code value will depend on the communications driver being used.

On detecting a Remote Variable transaction that has timed-out, the **Refresh** time for repeating read or write transactions with missing instruments should be lengthened. When the Status returns to Go i.e. indicating that the instrument is again present, the **Refresh** time can be returned to the normal value. This can be achieved using soft-wiring to the Remote Variable function block, such as:

```
remotePV.Refresh:= SEL_TIME (G:= remotePV.Status <> 1
(* GO *),
                               IN0:= normRate.Val,
                               IN1:= failRate.Val) ;
```

Where remotePV is a Remote Variable Function block, and normRate and failRate are user variables containing the normal and failure mode refresh times.

- e) **Problem** - it is not always possible to fully test a user program with communications without having fully functional external communications devices.

**Solution** - Because the PC3000 supports both Master and Slave protocols in some cases it is possible to simulate the external devices using the PC3000 itself. For example, an external JBus master device can be simulated by creating Remote Variables associated with a JBus-M function block on a spare communications port. This can then be connected to the port under test.

## COMMUNICATIONS

### EI\_BISYNC\_M FUNCTION BLOCK

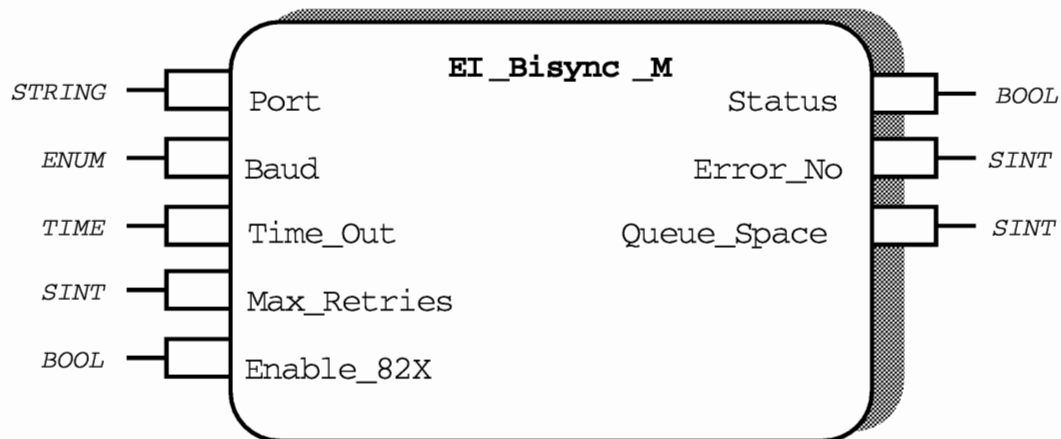


Figure 3-16 EI\_Bisync\_M Diagram

### Functional Description

The EI\_Bisync\_M Function Block supports serial communications on a designated serial communications port using the EI Bisync ASCII protocol. This function block configures the serial port to function in Master mode. Detailed knowledge of the Bisync protocol is not normally required to use this function block. However, you may refer to the EI Bisync Communications Handbook for specific details if required. Before reading this description, you are advised to gain a general understanding of the PC3000 communications system by reading the PC3000 Communications Overview.

This function block will be required when designing or programming the PC3000 to use the EI Bisync protocol to function in master mode, i.e. for a PC3000 serial communications port connected to one or more EI Bisync Slave devices, such as 900 Series discrete controllers.

### Function Block Attributes

Type:..... 8 40  
 Class: .....COMMS  
 Default Task: .....Task\_1  
 Short List: .....Port Status Queue\_Space  
 Memory Requirement:.....2054 Bytes  
 Execution Time: .....20  $\mu$  Secs

The EI Bisync Master (**EI\_Bisync\_M**) driver block deals with the protocol specific details of the EI Bisync communications and is supported by the generic Remote Variable function blocks. The Remote Variable blocks are linked to the driver by means of a protocol specific address and can request the polling or updating of parameters of a communicating instrument via a specified communications port.

**Note:** Only the ASCII version of the EI Bisync protocol is supported.

## Parameter Description

### Driver Configuration Parameters

The **EI\_Bisync\_M** block has several configuration input parameters which define various aspects of the driver and should be set prior to running the user program. Changing these parameters while the user program is executing will have no effect on the driver, except under special circumstances - see 'PC3000 Communications. Overview' section 'Temporarily changing configuration parameters'.

#### Port

The Port parameter is the two character address of the port on which the EI Bisync protocol is to run. The first character is a number from 0 to 5 representing the rack slot and the second character is a letter representing the port within that slot. e.g. '0C' would be port C on the LCM and if there were an ICM in slot 3 '3A' could refer to its top port.

#### Baud

The Baud parameter gives a choice of 11 different rates from 75 baud up to 115.2 kbaud (as shown in Table 3-1 with a default of 9600 baud. Note that not all ports will be able to support all baud rates. This will be indicated by an error when the function block is first run (see description of **Error\_No** ).

#### Time\_Out

The **Time\_Out** parameter specifies the length of time that the Bisync Master will wait for a response message to a transmitted request. After this time the driver assumes there was a transmission error and the request may be retransmitted or an error returned to the Remote Variable function block which initiated the request. **Time\_Out** defaults to 5 Seconds.

Enum Value	Baud Rate
0	75
1	300
2	600
3	1200
4	2400
5	4800
6	9600
7	19200
8	38400
9	57600
10	115200

Table 3-1 EI\_Bisync\_M Baud Rates

#### Max\_Retries

The **Max\_Retries** parameter specifies the number of times that a request will be retransmitted if a transmission error, such as a timeout or a message checksum error, is detected. If a valid response is not received after this number of retries an error is returned to the remote parameter block which initiated the request.

**Max\_Retries** defaults to 2, so a request will be sent three times before an error is reported and the request aborted.

#### Enable\_82X

The **Enable\_82x** parameter is set to allow communication with the 82x family of instruments, eg 825.

These instruments require an extra stop bit.

Note: this should only be used when there are 82x instruments connected to the port since it will reduce the communications rate by 10 .

### Driver Status Parameters

The driver status is indicated by three output parameters in the **EI\_Bisync\_M** function block.

#### Status

The Status parameter is a boolean indication of the state of the link controlled by the driver. If there are no problems with the link this parameter reads Go ,

but when an error occurs , the **Error\_No** parameter indicates the reason for the problem.

#### Error\_No

The **Error\_No** parameter indicates the reason for any errors with the link. If the link is functioning correctly, **Error\_No** will be 0 (OK). For full details of error codes see the section on Error Reporting.

#### Queue\_Space

The **Queue\_Space** parameter indicates the amount of space left in the queue for remote parameter operations. If this reaches zero then it implies that the link bandwidth is not sufficient to cope with the number of Remote Variable requests being made and data will be lost. If this situation arises the parameter polling rates should be reduced.

### Remote Variable Operation

The requests made by the PC3000 are controlled by one or more Remote Variable blocks which can initiate read and write requests via a driver that supports the Master mode of operation.

The EI\_Bisync\_M driver currently supports the **Remote\_Bool**, **Remote\_Real**, **Remote\_Int**, **Remote\_Time**, **Remote\_Str** and **Remote\_SW** block types.

### Addressing

It is necessary to set up a protocol specific Address in the Remote Variable block which is the address used to access the remote devices. An example address format is shown in Figure 2. The port field is defined as in the function block Port parameter as a rack slot number followed by a letter for the port within that slot.

The protocol specific part of the address begins with a slave identify (ID) which specifies the address of the slave device which should respond to a request. This consists of two characters, the Group Identifier ( Gid ) and the Unit Identifier ( Uid ). The range of both the Gid and Uid is '0' (30h) to 'o' (6Fh).

These are followed by the Channel Identifier ( Chid ) which is in the range SPACE (20h) to ' ' (7Eh). If set to SPACE this indicates that no Chid is to be sent to the slave instrument. Next is a two character mnemonic ( Mn0 Mn1 ), each character can be in the range '!' (21h) to ' ' (7Eh).

Finally, zero, one or more format characters indicate how the data field is to be encoded. This is required because the Ei Bisync standard provides a number of different encodings for the same data type. Multiple format characters are only used when the Remote Variable is multi-element, in which case all fields of the same type are encoded in the same way. If no format character is present, default encoding is used. The formats are defined in the following section.

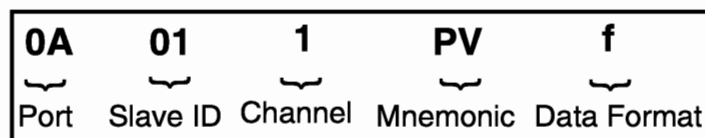


Figure 3-17 An Example Remote Variable Address

The Address parameter of a Remote Variable function block may be changed at any time so the block may be re-used to communicate with different instruments and different parameters. It is not recommended that it is changed when the function block's **State** parameter is 'Pending', i.e. a request is outstanding.

## Data Formats

This section describes the different data encodings that can be selected for each basic data type. The format character, which is part of the Remote Variable block's address string, specifies to the driver block which encoding is to be used.

### Floating Point (REAL)

The `Remote_Real` function block provides access to a single floating-point parameter on a slave instrument.

Format	Description	
P	24-bit IEEE Packed	DEFAULT
p	32-bit IEEE Packed†	
F	Free ASCII, 8 characters max.	
f	Free ASCII, 6 characters max.	
J	Free ASCII, 6 characters max.	TCS Floating Format, 1st Char. not '.' nor '-'
0	TCS Fixed Format,        XXXX. or XXXX-	
1	TCS Fixed Format,        XXX.X or XXX-X	
2	TCS Fixed Format,        XX.XX or XX-XX	
3	TCS Fixed Format,        X.XXX or X-XXX	
4	TCS Fixed Format,        .XXXX or -XXXX	
Q	SSD 64-bit Format	

Table 3-2 El\_Bisync\_M Floating Point Formats

**Note:**†The 32 bit format is not currently supported.

### Integer

The Remote\_Int function block provides access to a single Signed Integer or Status-Word parameter on a slave instrument. The Remote\_SW function block packs/unpack the value from/to sixteen Boolean parameters.

<b>Format</b>	<b>Description</b>	
Z	Free, 0 to 8 Hex Characters (32-bit maximum)	DEFAULT
B	8-bits, 2 Hex Characters	
X	16-bits, 4 Hex Characters	
Y	32-bits, 8 Hex Characters	

Table 3-3 El\_Bisync\_M Integer Formats

### Bool

The Remote\_Bool function block provides access to a single Boolean parameter on a slave instrument.

There is no user selectable format for this type. It is sent as a 'Free Format' integer with a value of 0 for False or 1 for True.

### Time

The Remote\_Time function block provides access to a single duration (TIME) parameter on a slave instrument.

There is no user selectable format for this type. It is sent as a 'Free Format' integer. The value is the number of milliseconds.

### String

The Remote\_Str function block provides access to a single String parameter on a slave instrument.

<b>Format</b>	<b>Description</b>	
S	Standard	DEFAULT
r	Raw	

Table 3-4 El\_Bisync\_M String Formats



The Standard encoding consists of the string preceded by an apostrophe (60h). All non-printable characters are replaced by the escape character (1Bh) followed by a 2 digit hexadecimal number which represents the ASCII code of the character. A character is non-printable if its ASCII code is less than 20h or greater than 7eh.

The Raw encoding implies that there is no encoding. The string is sent unchanged. This is the format normally used only for debugging communications but may be used to access parameters of a non-standard instrument or for sending composite data parameters for which there is no Remote Variable function block type.

### Composite Data

The driver is capable of handling composite data parameters but there are currently no Remote Variable blocks that support this.

### Multi-Block Messages

There is only limited support of Multi-Block messages in this driver.

A received message is accepted if it is in the Multi-Block message format but only one block is accepted. In this case the message begins with the SOH ('05h') character and terminates with the ETX ('03h') character.

No transmitted messages are Multi-Block.

## Example

This example shows the function blocks required to access the Process Variable PV and Status Word SW parameters of three instruments, two of which are connected to port A of the LCM and the other is connected to port B of the LCM.

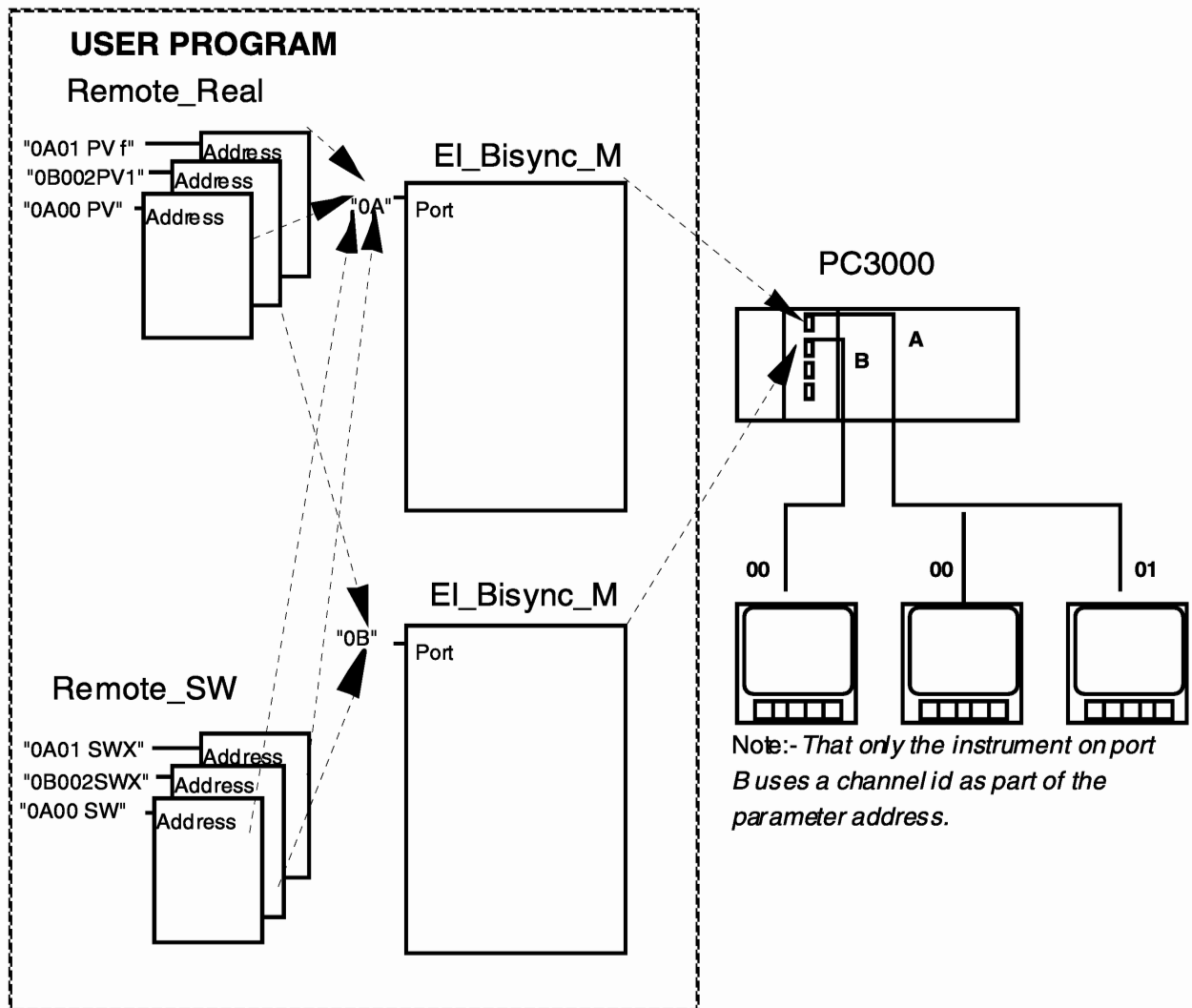


Figure 3-18 EI\_Bisync\_M Usage Example

## Errors

If there is a specific EI\_Bisync\_M Function Block or EI Bisync driver error, the error is reported via the **Error\_No** parameter of the EI\_Bisync\_M Function Block. Errors concerning access to a specific remote variable are reported via the associated Remote Variable block.

## Error Reporting

### Function Block Errors

The following errors are reported by the EI\_Bisync\_M block via the **Error\_No** parameter.

<b>Error_No</b>	<b>Error Description</b>
1	PORT ERROR NO ADDRESS There are less than two characters in the Port parameter of the function block.
5,6	PORT ERROR BAUD RATE NOT AVAILABLE The baud rate requested is not available on this serial port.
11	PORT ERROR ILLEGAL SLOT The slot number selected is not legal. The slot number is the first character of the Port parameter and should be in the range '0' to '5'.
12	PORT ERROR ILLEGAL PORT The port number selected is not legal. The second character of the Port parameter and should be in the range 'A' to 'C' for an LCM or 'A' to 'D' for an ICM.
17	PORT ERROR PORT IN USE The selected Port is already in use for another driver.

Table 3-5 EI\_Bisync\_M Error Codes

## Remote Variable Error Codes

These errors are reported via the **Error\_No** of the Remote Variable block.

<b>Error_No</b>	<b>Error Description</b>
11	ADDRESS ERROR ILLEGAL SLOT The first character in the Address parameter is not in the valid range of '0' to '5'.
12	ADDRESS ERROR ILLEGAL PORT The second character in the Address parameter is not in the valid range of 'A' to 'C' for a LCM port or 'A' to 'D' for an ICM port.
16	ADDRESS ERROR NO REMOTE PARAMETER SERVICE The port given in the Address parameter does not have a suitable master driver allocated to it.
50	DRIVER ERROR CANNOT INITIALISE The communications driver cannot initialise, for example the requested Baud Rate is not available on the selected port. The Error_No parameter of the EI_Bisync_M block will give the reason.
60	COMMS ERROR CHECKSUM FAILURE The response to a parameter read request had a checksum failure.
61	COMMS ERROR TIMEOUT No response to a request was received within the timeout period. Either the slave instrument has failed or the requests are so badly corrupted that the slave does not recognise its' own address.
62	DATA ERROR TOO MANY CHARACTERS The response to a parameter read request had too many characters in it. This may be caused by a transmission error.
63	ADDRESS ERROR STRING TOO SHORT The address string contained in the Address parameter is too short to be valid.
64	DATA ERROR HEX TO LONG The response to a parameter read request of an integer parameter had a value encoded with more than 8 hexadecimal character.
65	DATA ERROR NON HEX CHARACTER The response to a parameter read request of an integer parameter had a value encoded with a non hexadecimal character.
66	DRIVER ERROR INVALID PARAM TYPE The parameter/field type used by the Remote Variable block is not yet supported by the driver.

Table 3-6 Remote Variable Error Codes

<b>Error_No</b>	<b>Error Description</b>
67	<p>DATA ERROR NON BOOL VALUE</p> <p>The response to a parameter read request of a boolean parameter had a value other than 0 or 1.</p>
68	<p>DATA ERROR TOO MANY FIELDS</p> <p>The response to a parameter read request for a multi-element parameter contained more fields than were expected.</p>
69	<p>DATA ERROR VALUE TOO LARGE</p> <p>When encoding a value for transmission it has been found that the value is too large to be encoded using the specified format. For example a REAL value greater than 9999.0 cannot be represented using any of the TCS formats.</p>
70	<p>DATA ERROR MESSAGE TOO LONG</p> <p>When encoding a message for transmission, the driver has exceeded its' maximum buffer size. This can only occur with multi-element parameters.</p>
71	<p>DATA ERROR STRING TOO LONG</p> <p>The response to a parameter read request for a STRING parameter contained more characters than the Remote Variable block can support.</p>
72	<p>DATA ERROR INVALID CHARACTER</p> <p>The response to a parameter read request for a STRING parameter contained a non-printable character which was not converted to an Escape sequence.</p>
73	<p>DATA ERROR INVALID FIELD LENGTH</p> <p>The response to a parameter read request contained a data item of incorrect length. For example a REAL encoded using the SSD ('Q') format should have exactly 16 hexadecimal characters.</p>
100	<p>DATA ERROR NAK</p> <p>The response to a parameter write request was a Negative-Acknowledgement. This may be caused by incorrect parameter mnemonics, writing to a read-only parameter or data out of range. The value of the slave instruments' EE parameter should give the reason.</p>
101	<p>COMMS ERROR NOT ACK NOR NAK</p> <p>The response to a parameter write request was neither a Positive-Acknowledgement nor a Negative-Acknowledgement. This usually indicates that the response has been corrupted, possibly by two slaves having the same address.</p>
103	<p>DATA ERROR EOT</p> <p>The response to a parameter read request was an EOT character. It indicates that the slave has rejected the request. This may be caused by incorrect parameter mnemonics or reading a write-only parameter. The value of the slave instruments' EE parameter should give the reason.</p>

Table 3-6 Remote Variable Error Codes (continued)

<b>Error_No</b>	<b>Error Description</b>
104	<p>COMMS ERROR NOT STX</p> <p>The response to a parameter read request did not start with an STX character. This usually indicates that the response has been corrupted, possibly by two slaves having the same address.</p>
105	<p>ADDRESS ERROR INVALID GID</p> <p>The Group Identification character within the Address string is not within the valid range. (3rd character.)</p>
106	<p>ADDRESS ERROR INVALID UID</p> <p>The Unit Identification character within the Address string is not within the valid range. (4th character.)</p>
107	<p>ADDRESS ERROR INVALID CHID</p> <p>The Channel Identification character within the Address string is not within the valid range. (5th character.)</p>
108	<p>ADDRESS ERROR INVALID MNEMONIC</p> <p>The Mnemonic characters within the Address string are not within the valid range. (6th and 7th characters.)</p>
109	<p>DATA ERROR NON PACKED CHARACTER</p> <p>The response to a parameter read request for a REAL parameter encoded using the IEEE packed format ('P') contained an invalid character.</p>
110	<p>DATA ERROR NOT RS</p> <p>The response to a read request for a 2-level multi-element parameter did not have a RS character at the start of the data.</p>
111	<p>DATA ERROR STRUCTURE TOO DEEP</p> <p>The driver currently supports multi-element parameters to a maximum of 2 levels.</p>
112	<p>INTERNAL ERROR EMPTY STRUCTURE</p> <p>The Remote Variable block has requested the transmission of nothing!</p>
113	<p>COMMS ERROR MNEMONIC MISMATCH</p> <p>The mnemonics returned as part of the response to a read parameter request are different to those contained in the request.</p>
114	<p>DATA ERROR GT EXPECTED</p> <p>The response to a parameter read request for an Integer or Status Word parameter did not have a '&gt;' at the start of the data.</p>
115	<p>DATA ERROR APOSTROPHE EXPECTED</p> <p>The response to a parameter read request for a String parameter using the Standard encoding did not have an apostrophe at the start of the data.</p>

Table 3-6 Remote Variable Error Codes (continued)

<b>Error_No</b>	<b>Error Description</b>
201	COMMS ERROR RX OVERRUN An overrun error was detected on a received character.
202	COMMS ERROR RX PARITY A parity error was detected on a received character.
203	COMMS ERROR RX PARITY & OVERRUN A parity and an overrun error were detected while receiving.
204	COMMS ERROR RX FRAMING A framing error was detected on a received character.
205	COMMS ERROR RX FRAMING & OVERRUN A framing and an overrun error were detected while receiving.
206	COMMS ERROR RX FRAMING & PARITY A framing and a parity error were detected while receiving.
207	COMMS ERROR RX FRAMING, OVERRUN & PARITY A framing, parity and overrun error were detected while receiving.
208-215	BREAK CONDITION CHANGED Break condition has been detected or cleared Line has become disconnected Remote device Baud rate is too slow

Table 3-6 Remote Variable Error Codes (continued)

## Parameter Attributes

Name	Type	Cold Start	Read Access	Write Access	Type Specific InforNamemation	
Port	<b>STR ING</b>	'0A'	Oper	Config		
Baud	<b>ENUM</b>	_9600	Oper	Config	Numerated	75(0) 300(1) 600(2) 1200(3) 2400(4) 4800(5) 9600(6) 19200(7) 38400(8) 57600(9) 115200(10)
Time_Out	<b>TIME</b>	5s	Oper	Super	High Lim. Low Lim.	24day 100ms
Max_Retries	<b>SINT</b>	2	Oper	Super	High Lim Low Lim	1000 0
Enable_82X	<b>BOOL</b>	No	Config	Config	Senses	No (0) Yes(1)
Status	<b>BOOL</b>	NoGo	Oper.	Block	Senses	NOGO(0) Go(1)
Error_No	<b>SINT</b>	0	Oper	Block	High Lim Low Lim	255 0
Queue_Space	<b>SINT</b>	0	Oper	Block	High Lim Low Lim	255 0

Table 3-7 El\_Bisync\_M Parameter Attributes



## EI\_BISYNC\_S FUNCTION BLOCK

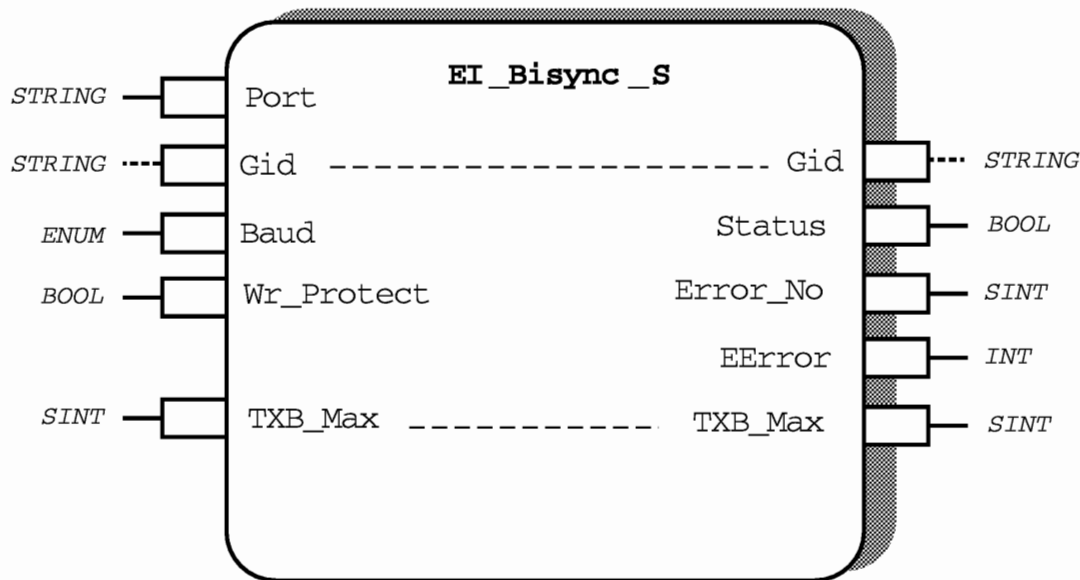


Figure 3-19 EI\_Bisync\_S Function Block Diagram

### Functional Description

The EI\_Bisync\_S Function Block supports communications on a designated serial communications port using the EI Bisync ASCII protocol and configures the serial port to function in Slave mode. Detailed knowledge of the Bisync protocol is not normally required to use this function block. However, you may refer to the EI Bisync Communications Handbook for specific details if required. Before reading this description, you are advised to gain a general understanding of the PC3000 communications system by reading the PC3000 Communications Overview.

The use of Slave Variables with the EI\_Bisync\_S function block and associated addressing is described in detail in this function block description. Because the default protocol adopted for the PC3000 communications behaves as a EI Bisync Slave, details of the default protocol are also defined.

This function block will be required when designing or programming the PC3000 to use an EI Bisync interface that functions in slave mode, i.e. when the PC3000 is connected to a EI Bisync master device via a serial link. Typical master devices are the Eurotherm Supervisory System (ESP) and Production Orchestrator (PO).

**Note:** A few features described in this document are not supported in the current release of this function block; these are listed in the appendix.

The Eurotherm International Bisynchronous Communications protocol (EI Bisync) is implemented on the PC3000, with the PC3000 acting as a slave, by means of a driver function block called EI\_Bisync\_S.

The driver block deals with the protocol specific details of the communications and is supported by generic Slave Variable function blocks. The Slave Variable blocks are linked to the driver by means of a protocol specific address and define values which can be read or written by a remote device using the protocol type specified in the address parameter, in this case `EB' for EI Bisync.

In addition to providing access to the Slave Variables, the driver also provides direct access to system parameters and function block parameters using reserved communications addresses.

The addresses of the function block parameters may change each time the PC3000 user program is re-built, so the EI Bisync master devices should be capable of importing a function block parameter address file to speed the modifications of the parameter addressing tables within the master device. ESP is an examples of EI Bisync master devices that use function block parameter addresses that are automatically generated by the Programming Station in address files. When a user program is built the user may optionally create a .GAT file which holds the function block parameters as a list of ESP Gates. To understand how these address files are used within these products, refer to the ESP manuals.

**Note:**-LCM ports A,B and C provide an EI Bisync Slave protocol as default, if no communications driver function blocks are assigned to them.

Although the default protocol can be used for EI Bisync communications with a master device, it is recommended that an EI Bisync Slave function block, EI\_Bisync\_S, is assigned to these ports when used. This includes port A which is normally used for the PC3000 Programming Station.

Using a EI\_Bisync\_S communications driver function block instead of using the default communications has the following advantages :

The EI\_Bisync\_S function block provides diagnostic information including error codes.

The user program Structure Text (ST) listing can show the full assignment of ports by listing the communications driver function blocks.

Slave Variables can be used. Default communications cannot address any EI Bisync Slave Variables.

Communications characteristics such as **Baud** rate can be changed using the communications driver function block.

**Note:**-That a EI\_Bisync\_S communications driver function block only has control over an assigned port when the user program is executing, i.e. when the PC3000 Programming Station displays the PC3000 mode as 'Running'. In any other mode, the PC3000 always reverts to default communications. which provides EI Bisync Slave protocols on ports A,B and C of the LCM. For example, an EI\_Bisync\_S communications. driver function block assigned to port C could set the baud rate to 19200 Baud. The port will operate at 19200 Baud only when the user program is executing; whenever the user program ceases execution, the port will revert to the default EI Bisync Slave protocol operating at 9600 Baud.

## Function Block Attributes

Execution Time: ..... 12  $\mu$  Secs  
 Type:..... 8 50  
 Class: .....COMMS  
 Default Task: .....Task\_1  
 Short List: .....Port Status Wr\_Protect Gid  
 Memory requirements: .....2466 Bytes

## Parameter Description

### Driver Configuration Parameters

The EI\_Bisync\_S block has several configuration input parameters which define various aspects of the driver and should be set prior to running the user program. Changing these parameters while the user program is executing will have no effect on the driver except under special circumstances - see 'PC3000 Communications. Overview' section 'Temporarily changing configuration parameters'. The only input which can affect the block whilst running is **Wr\_Protect**.

#### Port

The Port parameter is the two character address of the port on which the Bisync protocol is to run. The first character is a number from 0 to 5 representing the rack slot and the second character is a letter representing the port within that slot. e.g. '0C' would be port C on the LCM and if there were an ICM in slot 3 '3A' could refer to its top port.

## Baud

The Baud parameter gives a choice of 11 different rates from 75 baud up to 115.2 kbaud (as shown in Table 3-9 with a default of 9600 baud. Note that not all ports will be able to support all baud rates. This will be indicated by an error when the function block is first run (see description of **Error\_No** ).

<b>Enum Value</b>	<b>Baud Rate</b>
0	75
1	300
2	600
3	1200
4	2400
5	4800
6	9600
7	19200
8	38400
9	57600
10	115200

Table 3-8 El\_Bisync\_S Baud Rates

## Gid

The Gid parameter specifies the Bisync Group Identifier to which the PC3000 will reply over the communications link. The PC3000 will respond to all Unit Identifiers within this group.

The range of the Gid is '0' (30h) to 'o' (6Fh).

If the Gid is empty (null string), a hardware selected Identifier is used. This is set by either links on the LCM or a rotary switch on an ICM depending upon which module the port is on.

**Version 1 LCM**

<b>LCM Links (LK12 LK11 LK2 LK1)</b>	<b>ICM Switch</b>	<b>GID</b>
1000	0	'0'
1001	1	'1'
1010	2	'2'
1011	3	'3'
1100	4	'4'
1101	5	'5'
1110	6	'6'
1111	7	'7'
0000	8	'8'
0001	9	'9'
0010		'A'
0011		'B'
0100		'C'
0101		'D'
0110		'E'
0111		'F'

Table 3-9 Hardware Selected GID

**Note:-**That LCMs are supplied with all links fitted. This gives a factory default GID of '7'. On version 2 LCM the rotary switch corresponds to GID directly.

**Wr\_Protect**

The Wr\_Protect parameter when set inhibits all incoming writes through this port. If a write operation is attempted when write protection is on then an error response is generated by the driver.

**TXB\_Max**

The TXB\_Max parameter controls the maximum length of a transmission block.. If set to zero the multi-block feature is disabled and messages with a data field longer than 255 bytes may be transmitted. If non-zero, any message that would exceed the maximum is split into two or more blocks.

The value may be read or written via. the communications link using the BL system parameter.

## Driver Status Parameters

The driver status is indicated by three output parameters in the EI\_Bisync\_S function block.

### Status

The **Status** parameter is a boolean indication of the state of the link controlled by the driver. If there are no problems with the link this parameter reads Go, but when an error occurs it changes to NOGO. If the Status is NOGO the **Error\_No** parameter indicates the reason for the problem.

### Error\_No

The **Error\_No** parameter indicates the reason for any errors with the link. If the link Status is Go then the **Error\_No** will be 0 (OK). For full details of error codes see the section on Error Reporting.

### EError

The **EError** parameter normally indicates the reason for the last incoming transaction being rejected. The PC3000 functioning as a Slave, will indicate rejected incoming read transactions by returning an EOT character or a NAK character for a rejected write transaction. However, in some cases the PC3000 will not respond on receiving a corrupt transaction with certain types of error such as a parity error and **EError** parameter is not changed.

The value is accessible over the communications link by reading the standard EE system parameter.

The value will remain the same until another transaction is rejected.

For full details of EE error codes see the section on Error Reporting.

## Parameter Addressing

A particular PC3000 is selected on a multi-dropped serial link by the Gid ; within a PC3000 there are three different parameter domains that can be addressed using the Uid, Channel Identity and EI Bisync mnemonic. The domains are System Parameters, Slave Variables and Function Block parameters.

An EI Bisync parameter address consists of a Unit Id (Uid), a Channel Id (Chid) and a two character mnemonic (Mn0 Mn1). Table 11 shows which of these address components define the mapping to the different domains :

<b>Uid</b>	<b>Mn0</b>	<b>Parameter Domain</b>
Any value	Not '0' to '9'	System
'0'	'0' to '9'	Slave
Not '0'	'0' to '9'	Function Block

Table 3-10 Domain Mapping

System mnemonics do not require a specific Uid; any value can be used by an external master device. All system mnemonics have an alphabetic character for the first character, eg. 'EE','MN'. A full list is given in the section PC3000 Bisync System Mnemonics.

Slave Variables are always assigned a Uid of '0' and should be given mnemonics in the Slave Variable Address that start with a numeric character, eg. '1A', '4B', '90'.

Function Block parameters are assigned Uid, Channel IDs and mnemonics by the PC3000 Programming Station when a user program is created. The Uid and Channel ID are used to address a particular function block instance, the mnemonic defines the parameter within the instance. The assignments are given in the .CEL and .GAT files which can be created when a user program is built, to assist ESP integration.

#### Addressing for ESP

Whenever a PC3000 is addressed with a Uid in the range '0' (30h) to 'O' (4Fh) the parameter data is encoded/decoded as normal. However if the Uid character is offset by adding 20h to its position in the ASCII code set, so that it is in the range 'P' (50h) to 'o'(6Fh), the data is encoded/decoded for the Eurotherm Supervisory Package (ESP). For example, if the normal Uid='1', using Uid='Q', will cause ESP data encoding/decoding to be used.

The ESP gate definitions given in the .GAT file generated by the PC3000 Programming Station have all Uid values offset by 20h to ensure that the correct encoding is used.

#### Channel Identities

A channel identity is normally required for all transactions except when reading System parameters when a Channel Id is optional and is ignored; i.e. it is possible to read a system mnemonic with or without a channel identity. However, note that a Channel Id is required when writing to system parameters.

### Slave Variables

The user may associate instances of Slave Variable function blocks with this driver by starting the Address string of an instance with 'EB'. Any instance of an EI\_Bisync\_S block may then read or write the Value parameter of the Slave Variable block and so is accessible through more than one communications port.

These parameters may only be accessed when the PC3000 is in the RUNNING mode.

The Address is entered in the form 'EB<Chid> < Mn0 Mn1>'. Where Chid can be in the range '!' (21h) to '~' (7Eh), Mn0 can be in the range '0' (30h) to '9' (39h) and Mn1 is in the range '0' (30h) to '~' (7Eh). Note that Mn0 is restricted to numeric characters to ensure that the address does not clash with any pre-defined System parameter. Lists of acceptable characters for Slave Variable addresses are given in section Slave Variables Address Ranges.

Both the Channel Id and Mn1 can use numeric and alphabetic characters, However, the ranges can be extended to some special characters if required.

Examples of valid Slave Variable Addresses are :

**'EB110'**

**'EB111'**

**'EBA6B'**

**'EBZ99'**

**Note:-**That Slave Variables assigned to EI Bisync Slave protocol using the 'EB' protocol selector are accessible from any ports to which a EI\_Bisync\_S communications. driver function block has been assigned. However, although ports A,B and C of the LCM operates by default, as EI Bisync Slaves when no communications. driver function block is assigned to them, access to Slave Variables using the default protocol is not possible.

A full description of how Slave Variables can be used is given in the 'PC3000 Communications Overview'.

### Multi-Element

If the Slave Variable block is multi-element, eg. Slave\_Real\_8, the Address refers to a composite data parameter containing all of the elements. To access individual elements an offset is applied to Mn1 . For example, a Slave\_Real\_8 with an Address of 'EBx90' the PC3000 will respond to 18 addresses. These are:



<b>GID</b>	<b>Value</b>	<b>Encoding</b>
GID 0x90	composite parameter	standard encoding
GID 0x91	Value_1	standard encoding
GID 0x92	Value_2	standard encoding
GID 0x93	Value_3	standard encoding
GID 0x94	Value_4	standard encoding
GID 0x95	Value_5	standard encoding
GID 0x96	Value_6	standard encoding
GID 0x97	Value_7	standard encoding
GID 0x98	Value_8	standard encoding
GID Px90	composite parameter	ESP encoding
GID Px91	Value_1	ESP encoding
GID Px92	Value_2	ESP encoding
GID Px93	Value_3	ESP encoding
GID Px94	Value_4	ESP encoding
GID Px95	Value_5	ESP encoding
GID Px96	Value_6	ESP encoding
GID Px97	Value_7	ESP encoding
GID Px98	Value_8	ESP encoding

Table 3-11 Multi-element Addressing

Several Slave Variables may be read as a composite parameter by specifying a Mn1 of '\*' via the communications link. For example, if there were three Slave Variables 'EBy11', 'EBy12' and 'EBy15' they could be read or written as composite data using the address GID 0y1\* or GID Py1\*. If a multi-element Slave is included then a two level composite data parameter is generated.

### Function Block Parameters

Any Function Block parameter may be accessed if its' address is known. This would normally be read from the .CEL or .GAT files produced by the Programming Station.

Function Block parameters may only be accessed when the PC3000 contains a valid User Program.

A parameter within the User Program is always part of a Function Block instance. The parameters within an instance are numbered from 1 to 790 and instances within a User Program are numbered 1 to 2914.

## Standard Address Mapping

For standard data encoding the mapping of function block instance number and parameter number within instance to EI Bisync addresses is as follows :-

$$\begin{aligned}\text{UID} &= (\text{instance} / 5\text{Eh}) + 31\text{h} \\ \text{CHID} &= (\text{instance} \% 5\text{Eh}) + 21\text{h} \\ \text{Mn0} &= (\text{parameter} / 4\text{Fh}) + 30\text{h} \\ \text{Mn1} &= (\text{parameter} \% 4\text{Fh}) + 30\text{h}\end{aligned}$$

Where % implies the Modulus operator.

This gives the ranges :--

$$\begin{aligned}\text{UID} &\dots\dots\dots '1' (30\text{h}) \text{ to } 'O' (4\text{Fh}) \\ \text{CHID} &\dots\dots\dots '!' (21\text{h}) \text{ to } ' ' (7\text{Eh}) \\ \text{Mn0} &\dots\dots\dots '0' (30\text{h}) \text{ to } '9' (39\text{h}) \\ \text{Mn1} &\dots\dots\dots '0' (30\text{h}) \text{ to } ' ' (7\text{Eh})\end{aligned}$$

## ESP Address Mapping

For ESP data encoding the mapping of instance number and parameter number within instance to EI Bisync addresses is the same as for standard encoding except 20h is added to the Uid. i.e. :-

$$\text{UID} = (\text{instance} / 5\text{Eh}) + 51\text{h}$$

This gives the range:--

$$\text{UID} \text{ 'R' } (30\text{h}) \text{ to } 'o' (4\text{Fh})$$

### Multi-Parameter

To access all the parameters of one function block instance as a single composite parameter the mnemonic i.e. Mn0 Mn1 of '00' is used.

### System Parameters

All PC3000s have a built in set of parameters. These include the standard Instrument Identify (II) and Mode (MN) parameters. Others, for example, support the Downloading/Uploading of user programs and the monitoring of the state of the I/O racks. Details of using the System parameters are not included in this document although a full list of system parameters and functions accessible by EI Bisync mnemonics is given in the section PC3000 EI System mnemonics

When reading (polling) a parameter any valid Uid may be specified. Some system mnemonics also require a channel identity, Chid. A Chid can also be

given when reading those mnemonics that do not specifically require it, in which case providing it is within the EI Bisync valid set, it is ignored. In all cases, if a Chid is used, it is returned in the response message.

## Data Encoding

The data content of the communications message is briefly described in this section. Two formats are available for each data type and the format is automatically selected depending upon the Uid that is used when selecting the PC3000.

The first is a subset of formats and has been chosen to be the most efficient to transfer and process data of each basic type. It is used if the Uid is in the range 'O' (30h) to 'O' (4Fh).

The second is tailored to the requirements of the Eurotherm Supervisory System (ESP) which supports only a limited number of data types. It is used if the Uid is in the range 'P' (50h) to 'o' (6Fh).

### Slave Variable Format Specification

From version 3.00 onwards, the complete set of data formats supported in the EI Bisync Master function block are also supported in the EI Bisync Slave function block.

The EI Bisync format used when accessing Slave variables can be defined by using a format character which is appended to the Slave variable address. Full details of the valid format characters for each data type can be found in the EI Bisync Master function block documentation (EI\_Bisync\_M).

For example for a Slave\_Real :

Address = 'EB060' - implies default 24 bit precision format

Address = 'EB060p' - selects the 32 bit precision format

### Standard Encoding

The following encoding is used for normal EI\_Bisync access.

#### Floating Point (REAL)

Floating Point (Real) values are transmitted and received in Packed Single-precision IEEE format. This is a variable length format that requires only significant data to be transmitted. The format type character is '@'.

	<b>Bits</b>	<b>Max. Chars. Sent</b>	<b>Example address</b>	
P	24	4	'EBx99' or 'EBx99P'	DEFAULT
p	32	6	'EBx99p'	Not currently supported

Table 3-12 EI\_Bisync\_S Floating Point Formats

To reduce the number of characters transmitted by the PC3000 the resolution of the number may be limited. By default a maximum of 24-bits of the 32-bit value are transmitted but full resolution can be sent if requested as part of the Address string using the 'p' format character. Format character 'P' can be used to explicitly select 24 bit precision.

The following table provides examples of floating point values and the associated ASCII characters used to encode the values when transmitted via EI Bisync to 24 bit precision.

<b>REAL Value</b>	<b>ASCII Characters encoding</b>
0.0	@
2.0	@P
-2.0	@p
0.5	@Op
-0.5	@op
1000.0	@QGh
10000.0	@Qap
100000.0	@Q1Mp

Table 3-13 EI\_Bisync\_S Floating Point Values Encoded

A real number in IEEE format is broken down into number of fields as shown:

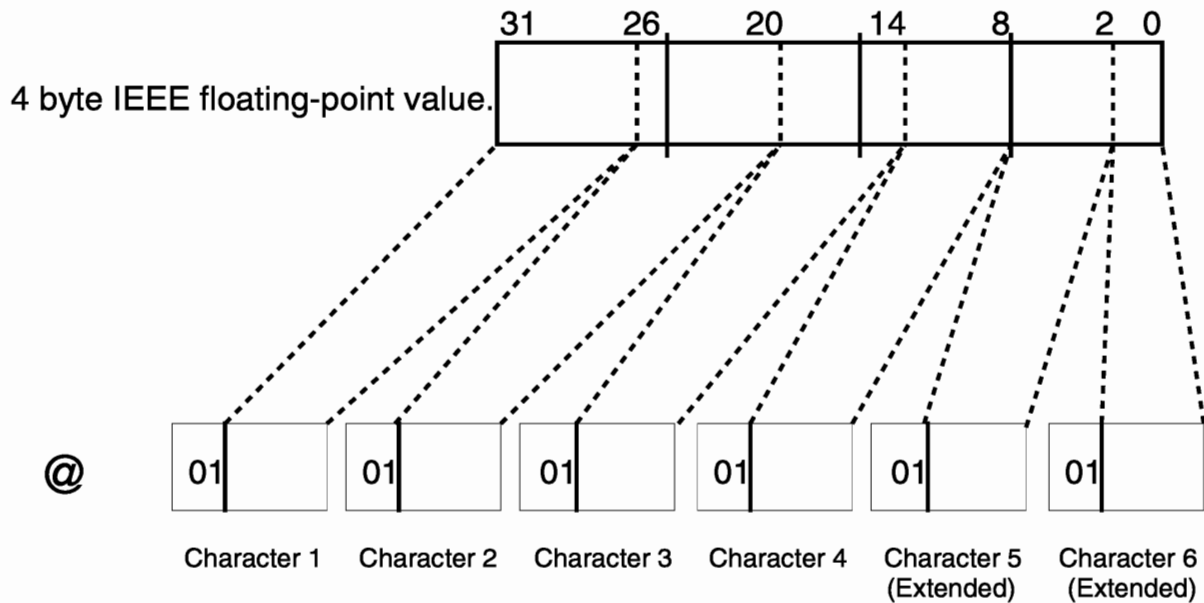


Figure 3-20 Floating Point Encoding

More details on this format are given in EI Bisync Communications Protocol Extensions (HP024113)

### Integer

Integer values are transmitted and received as ASCII hexa decimal format. This is a variable length format that requires only significant data to be transmitted. The format type code is ' and the data content may be 0 upto 8 characters in length to represent a 32-bit value. Negative values require the transmission of the full 8 characters.

The following table shows integer values and the associated ASCII encoding.

<b>Integer Value</b>	<b>ASCII encoding</b>
0	>
1	>1
16	>10
256	>100
4096	>1000
65536	>10000
1048576	>100000
16777216	>1000000
268435456	>10000000
2147483647	>7FFFFFFF
-1	>FFFFFFFF

Table 3-14 El\_Bisync\_S Integer Values

### BOOL

Boolean values are sent in the same way as Integer values.

<b>Boolean Value</b>	<b>ASCII encoding</b>
0	>
1	>1

Table 3-15 El\_Bisync\_S Boolean Encoding.

## TIME

Duration (Time ) values are sent as an unsigned integer number of milliseconds. Table 3-17 shows examples of duration values and the associated ASCII encoding.

<b>TIME value</b>	<b>ASCII encoding</b>
0ms	>
1ms	>1
16ms	>10
256ms	>100
4s96ms	>1000
1m5s536ms	>10000
17m28s576ms	>100000
4h39m37s216ms	>1000000
3d2h33m55s456ms	>10000000
49d17h2m47s295ms	>FFFFFFFF

Table 3-16 El\_Bisync\_S Time Encoding.

## DATE

Date values are sent as an unsigned integer number of days since 1970-01-01.

<b>DATE value</b>	<b>ASCII encoding</b>
1970-01-01	>
1970-01-02	>1
1991-04-04	>1E53
2020-12-31	>48C3

Table 3-17 El\_Bisync\_S Date Encoding.

### DATE\_AND\_TIME

Data And Time values are sent as an integer number of milliseconds since 1970-01-01-00:00:00. Table 3-19 shows examples of DATE\_AND\_TIME values and the associated ASCII encoding.

<b>DATE_AND_TIME value</b>	<b>ASCII encoding</b>
1970-01-01-00:00:00	>
1970-01-01-00:00:01	>1
1991-04-04-16:14:32	>27FB50E8
2038-01-19-03:14:07	>7FFFFFFF

Table 3-18 El\_Bisync\_S Date and Time Encoding.

### TIME\_OF\_DAY

Time Of Day values are sent as an integer number of seconds since 00:00:00.

Table 20 shows examples of TIME\_OF\_DAY values and the associated ASCII encoding.

<b>TIME_OF_DAY value</b>	<b>ASCII encoding</b>
00:00:00	>
00:00:01	>1
00:00:16	>10
00:04:16	>100
01:08:16	>1000
18:12:16	>10000
23:59:59	>1517F

Table 3-19 El\_Bisync\_S Time of Day Encoding.

### STRING

String values are sent by prefixing an apostrophe (27h) and expanding all non-printable characters to a three character Escape sequence. This Escape sequence consists of the ESCAPE character (1Bh) followed by two hexadecimal digits representing the ASCII code of the character.

A non-printable character is defined as one with an ASCII code of less than 20h or greater than 7Eh.



## Composite Data

Up to three levels of nested structures are supported.

Four characters are used as separators between the fields in a hierarchical manner.

Abbreviation	Hex Code		
FS	1C	3 level start	
GS	1D	2 level start	3rd level separator
RS	1E	1 level start	2nd level separator
US	1F		1st level separator

Table 3-20 EI\_Bisync\_S Composite Data Encoding.

The first data character indicates how many levels there are to the structure. Fields within the structure are then separated by lower level separator characters.

The occurrence of a higher level separator implies the simultaneous occurrence of all lower level separators.

For example :

***RS* >1234*US* @Pqr**

is a 1 level structure with 2 fields.

***GS* >1234*US* @Pqr*RS* >1**

is a 2 level structure with 2 fields at the top level with the 1st being a structure containing 2 fields.

When writing to the parameter, fields that are not to be changed may be omitted. So to write to the last field of the top level in the previous example would be :

***GSR*S >1**

Also when writing, any level may be prematurely terminated if there is no more data to send. So to write to just the 1st field of the lower structure in the previous example would be :

***GS* >1234**

### Eurotherm Supervisory System (ESP) Encoding

The following encoding is used when parameters are addressed using the normal Uid offset by 20h. This is provided for communications with ESP which requires an alternative encoding scheme.

#### Floating Point (REAL)

This is the same as standard encoding, except with fixed length data field.

Examples (24-bit resolution) :

<b>REAL value</b>	<b>ASCII encoding</b>
0.0	@@@@@
2.0	@P@@@
-2.0	@p@@@
0.5	@Op@@
-0.5	@op@@
1000.0	@QGh@
10000.0	@Qap@
100000.0	@Q1Mp

Table 3-21 El\_Bisync\_S Floating Point (REAL) Encoding.

#### INTEGER

Converted to REAL.

#### BOOL

Same as standard encoding, except with fixed length data field.

<b>Boolean Value</b>	<b>ASCII encoding</b>
0	>00
1	>01

Table 3-22 El\_Bisync\_S Boolean Encoding for ESP.

#### TIME

Sent as Floating Point (REAL) number of seconds.

**DATE**

Sent as Floating Point (REAL) number of days since 1970-01-01.

**DATE\_AND\_TIME**

Sent as Floating Point (REAL) number of seconds since 1970-01-01-00:00:00.

**TIME\_OF\_DAY**

Sent as Floating Point (REAL) number of seconds since 00:00:00.

**STRING**

There is no special ESP encoding for String values, so they are sent using the standard encoding.

**Composite Data**

The field separators and rules are the same as the standard encoding, but the data is encoded as defined in this section for ESP.

**Link Layer Protocol**

A very brief summary of the EI Bisync Link Layer Protocol, i.e. the basic message exchange formats is included here. For a full description refer to the 'EI Bisync Communications Handbook (HP022047)'.

**Instrument Addressing**

The PC3000 is considered to be addressed if it receives the EOT (04h) character followed by its' Gid repeated twice and a valid Uid repeated twice.

For example :

*EOT* **0011**

would address a PC3000 with a Gid of '0'. the Uid of '1' is used as part of the address of a parameter within the PC3000.

The PC3000 remains addressed until either an EOT is received or there is a power-up/reset.

**Parameter Read**

To read a parameter, the PC3000 should be addressed, as given, followed by the parameter address fields CHID Mn0 Mn1 and an ENQ (05h). For example :

*EOT* **0011123***ENQ*

If valid, the PC3000 will respond with an STX (02h), the CHID Mn0 Mn1 as sent, the data, an ETX (03h) and check character. This check character is the binary sum of (same as exclusive OR) of each character following the STX upto, and including the ETX. For example :

*STX* **123>456***ETX* :

where ':' is the check character.

If invalid the PC3000 will respond with an EOT (04h) and the EE system parameter will contain a code referring to the reason.

### Parameter Write

To write a parameter, the PC3000 should be addressed, as above , then sent an STX (02h) , the parameters' CHID Mn0 Mn1, the data, an ETX (03h) and a check character (as for the read). For example :

*EOT* **0011***STX* **123>456***ETX* :

If valid the PC3000 will respond with an ACK (06h).

If invalid the PC3000 will respond with an NAK (15h) and the EE system parameter will contain a code referring to the reason.

### Example

This example shows the function blocks required to provide access to three floating point (REAL) and three status word (SW) Slave Variables through both ports A and C of the LCM.

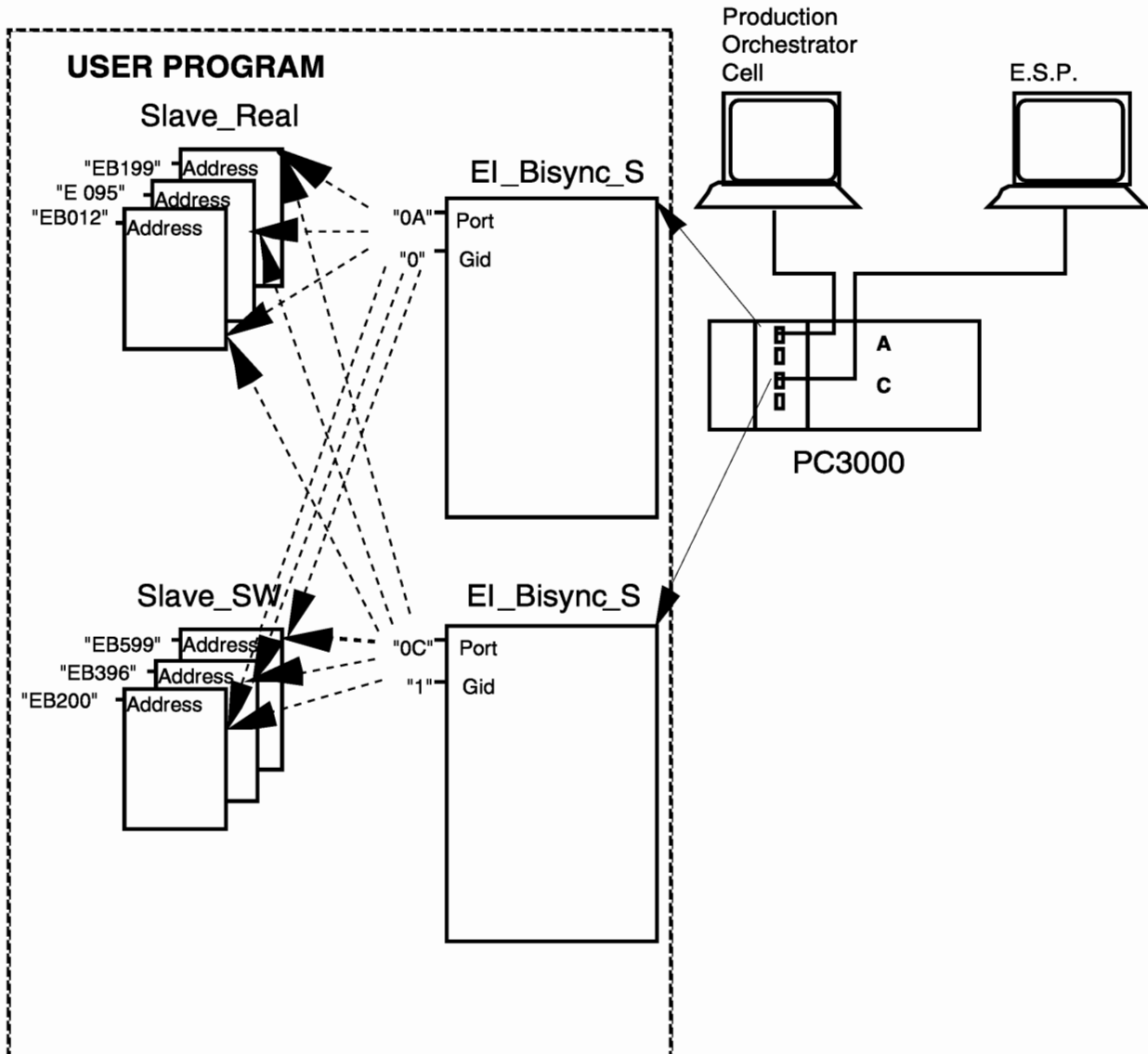


Figure 3-21 EI\_Bisync\_S Usage Example.

**Note:-**All EI\_Bisync\_S Communications Driver Function Blocks have access to Slave Variables with an Address containing the 'EB' protocol selector.

### Error Reporting

If there is a specific EI\_Bisync\_S Function Block or EI Bisync driver error, the error is reported via the **Error\_No** parameter of the EI\_Bisync\_S Function Block. Errors concerning a specific slave variable are reported via the associated Slave Variable block.

Function block Errors

<b>Error_No</b>	<b>Error description</b>
1	PORT ERROR NO ADDRESS •There are less than two characters in the Port parameter of the function block.
5/6	PORT ERROR BAUD RATE NOT AVAILABLE •The baud rate requested is not available on this serial port.
11	PORT ERROR ILLEGAL SLOT •The slot number selected is not legal. The slot number is the first character of the Port parameter.
12	PORT ERROR ILLEGAL PORT •The port number selected is not legal. The port number is the second character of the Port parameter.
17	PORT ERROR PORT IN USE •The selected Port is already in use for another driver.
18	ERROR TOO MANY SLAVE PARAMETERS •Too many Slave Variables have been declared to the system.
19	PORT ERROR TOO MANY DRIVER TYPES •Too many slave driver types have been declared to the system.

Table 3-23 El\_Bisync\_S Error Codes.

## Slave Variable Error Codes

These errors are reported via the Error\_No parameter of the Slave Variable block.

Error_No	Error description
1	ADDRESS STRING TOO SHORT •The address string contained in the Address parameter is too short o be valid.
11	ADDRESS ERROR ILLEGAL SLOT •The first character in the Address parameter is not in the valid range of '0' to '5'.
12	ADDRESS ERROR ILLEGAL PORT •The second character in the Address parameter is not in the valid range of 'A' to 'C' for a LCM port or 'A' to 'D' for an ICM port.
100	ADDRESS CHANNEL OR MNEMONIC INVALID •The Channel Identity, or Mnemonic is not within the allowed range.
255	DRIVER NOT FOUND •No port driver has been found that supports this Slave Variable. Either there is not a driver that supports the selected protocol (i.e. EB in this case) or the driver has not been able to initialise. The Error_No parameter of the EI_Bisync_S block will give the reason.

Table 3-24 Slave Variable Error Codes.

## EE Error Codes

These errors are given by the **EError** parameter of the EI\_Bisync\_S block and by the EE system parameter read via. the serial link. The decimal values for these codes will be used with the EError parameter when displayed on the PC3000 Programming Station. The hexadecimal value is used when the error is read from the PC3000 using the EI Bisync mnemonic EE.

<b>EError</b>	<b>Error description</b>
0 (0000h)	<p>EERROR NONE</p> <ul style="list-style-type: none"> <li>•No request has failed since the PC3000 system was reset.</li> </ul>
503 (01F7h)	<p>EERROR INVALID CHANNEL OR MNEMONIC</p> <ul style="list-style-type: none"> <li>•The Channel Identity. or Mnemonic within the request is not recognised.</li> </ul>
754 (02F2h)	<p>EERROR INVALID BCC</p> <ul style="list-style-type: none"> <li>•The checksum (BCC) of a write request did not correspond with the contents of the message.</li> </ul>
1528 (05F8h)	<p>EERROR READ ONLY</p> <ul style="list-style-type: none"> <li>•The parameter referred to in a write request was not writable.This may be due to the Write_Protect being set either on the El_Bisync_S block or the Remote Variable block.</li> </ul>
2040 (07F8h)	<p>EERROR INVALID DATA</p> <ul style="list-style-type: none"> <li>•The data content of a write request was invalid. For example a non-printable character in a string or a missing parameter type character.</li> </ul>
2296 (08F8h)	<p>EERROR DATA OUT OF RANGE</p> <ul style="list-style-type: none"> <li>•The data content of a write request exceeded the allowable range of the parameter.</li> </ul>
8443 (20FBh)	<p>EERROR UNSUPPORTED PARAMETER TYPE</p> <ul style="list-style-type: none"> <li>•An attempt was made to access a Slave Variable that has a value of a type that is not yet supported by the El_Bisync_S driver.</li> </ul>
8699 (21FBh)	<p>EERROR UNABLE TO ENCODE</p> <ul style="list-style-type: none"> <li>•The parameter being read could not be encoded because of an internal limitation.</li> </ul>
8955 (22FBh)	<p>EERROR COMPOSITE ACCESS NOT YET SUPPORTED</p> <ul style="list-style-type: none"> <li>•Reading or writing all the parameters of a function block as a single multi-element parameter has not yet been supported.</li> </ul>
9211 (23FBh)	<p>EERROR RS EXPECTED</p> <ul style="list-style-type: none"> <li>•The RS character (1Eh) was expected to be the first data character of the write request. The parameter is a one level structure.</li> </ul>
9467 (24FBh)	<p>EERROR TOO MANY ELEMENTS</p> <ul style="list-style-type: none"> <li>•The write request for a multi-element parameter contained too many fields.</li> </ul>
9723 (25FBh)	<p>EERROR STRUCTURE TOO DEEP</p> <ul style="list-style-type: none"> <li>•An attempt was made to access a Slave Variable that is more than a single level structure. This currently is not supported.</li> </ul>

Table 3-25 EE Error Codes.



<b>EError</b>	<b>Error description</b>
9979 (26FBh)	<p>EERROR INTERNAL FAILURE</p> <ul style="list-style-type: none"> <li>•A failure occurred within the El_Bisync_S driver whilst processing the request. This should never happen.</li> </ul>
33275 (81FBh)	<p>EERROR FILE TOO MANY OPEN</p> <ul style="list-style-type: none"> <li>•Either a file is already open through this port or the total number of files open is the maximum allowed.</li> </ul>
33276 (82F8h)	<p>EERROR FILE INVALID FILE SYSTEM</p> <ul style="list-style-type: none"> <li>•Unsupported file system selected.</li> </ul>
33278 (84F8h)	<p>EERROR FILE INVALID OPEN MODE</p> <ul style="list-style-type: none"> <li>•Invalid mode used when opening a file. Only Create, Read, Write and Append supported.</li> </ul>
33279 (85F8h)	<p>EERROR FILE INCORRECT FORMAT</p> <ul style="list-style-type: none"> <li>•Contents of format string incorrect.</li> </ul>
33280 (86F8h) and 33281 (87F8h)	<p>EERROR FILE NOT ENOUGH SPACE</p> <ul style="list-style-type: none"> <li>•Not enough memory when formatting file system.</li> </ul>
33282 (88F8h)	<p>EERROR FILE DOES NOT EXIST</p> <ul style="list-style-type: none"> <li>•File does not exist in file system and so, for example, cannot be deleted.</li> </ul>
33283 (89F8h)	<p>EERROR FILE ALREADY EXISTS</p> <ul style="list-style-type: none"> <li>•File already exists in file system and so, for example, a new file with the same name cannot be created.</li> </ul>
33284 (8AFBh)	<p>EERROR FILE MAXIMUM NUMBER</p> <ul style="list-style-type: none"> <li>•The file store already contains the maximum number of files allowed.</li> </ul>
33285 (8BFBh)	<p>EERROR FILE ALREADY OPEN FOR READ</p> <ul style="list-style-type: none"> <li>•The file is already open for reading so cannot be opened for writing.</li> </ul>
33286 (8CFBh)	<p>EERROR FILE ALREADY OPEN FOR WRITE</p> <ul style="list-style-type: none"> <li>•The file is already open for writing so cannot be opened for reading.</li> </ul>
33287 (8DFBh)	<p>EERROR FILE NOT OPEN FOR READ</p> <ul style="list-style-type: none"> <li>•The file is not open for reading so cannot be read from.</li> </ul>
33288 (8EFBh)	<p>EERROR FILE NOT OPEN FOR WRITE</p> <ul style="list-style-type: none"> <li>•The file is not open for writing so cannot be written to.</li> </ul>

Table 3-25 EE Error Codes. (continued)

<b>EError</b>	<b>Error description</b>
33289 (8FFBh)	ERROR FILE NO MORE SPACE <ul style="list-style-type: none"><li>•The file store has no more space for file storage.</li></ul>
33290 (90FBh)	ERROR FILE STORE UNFORMATTED <ul style="list-style-type: none"><li>•The file store has not been formatted.</li></ul>
33291 (91FBh)	ERROR FILE INVALID NAME <ul style="list-style-type: none"><li>•The file name contains invalid characters (eg. control characters).</li></ul>

Table 3-25 EE Error Codes. (continued)

### PC3000 EI Bisync System Mnemonics

The following mnemonics are provided either to support the EI Bisync protocol or to access system functions within the PC3000. System mnemonics are supported on LCM ports A,B or C when either operating in the default communications. mode or when configured as for EI Bisync Slave operation using the (EI\_Bisync\_S) communications. driver function block. Although system functions are provided as part of the interface with the PC3000 Programming Station, they may be used for some special applications.

In the following table, fields that are not applicable are shown as NA. Note that data returned when reading a mnemonic sometimes contains a number of fields. In some cases writing to a mnemonic performs a specific function.

<b>Mnemonic</b>	<b>Read Description</b>	<b>Write Description</b>
EE	Last Bisync error code	NA
FF	Read file system format	Format file system
FO	NA	Open an LCM file
FC	NA	Close an LCM file
FR	Read back block from LCM file	NA
Fr	Read back block from LCM file	NA
FW	NA	Write a block to an LCM file
FD	Read first directory entry	NA
Fd	Read next directory entry	NA
FK	NA	Delete an LCM file
FY	NA	Copy an LCM file
II	Instrument identifier	NA
V0	Version number	NA
CI	Configuration information	NA
MN	Read the PC3000 mode	Request a change of PC3000 mode

Table 3-26 PC3000 System Mnemonics

The following system mnemonics are unsupported and are subject to change between different releases of the PC3000. These are listed for diagnostic purposes only.

PC3000 System Mnemonics

<b>Mnemonic</b>	<b>Read Description</b>	<b>Write Description</b>
ss	Reads PC3000 startup strategy	NA
se	Read system error record	NA
	(Chid selects entry)	
sc	Reads count of system errors	Clears system error log
r1	Reads rack 1 module info.	NA
	(CHID selects slot)	
r2	Reads rack 2 module info.	NA
	(CHID selects slot)	
r3	Reads rack 3 module info.	NA
	(CHID selects slot)	
r4	Reads rack 4 module info.	NA
	(CHID selects slot)	
r5	Reads rack 5 module info.	NA
	(CHID selects slot)	
r6	Reads rack 6 module info.	NA
	(CHID selects slot)	
r7	Reads rack 7 module info.	NA
	(CHID selects slot)	
r8	Reads rack 8 module info.	NA
	(CHID selects slot)	
dd	Upload block of source code	Download block of source code
ds	Read source code status info	NA
db	Read source code block number	Select source code block
ms	Read details of user program area selected	Select user program area for read/write
mw	Read user program area selected by 'ms'	Write user program area selected by 'ms'

Table 3-27 PC3000 Unsupported System Mnemonics

<b>Mnemonic</b>	<b>Read Description</b>	<b>Write Description</b>
xe	Read use program status information	NA
mi	Read user program identity information	NA
mv	Read user program unique ID	NA
xs	Read details of user program area selected	Select user program area for read/write
xw	Read user program area selected by 'xs'	Write user program area selected by 'xs'
xe	Read use program status information	NA
xi	Read user program identity information	NA
xv	Read user program unique ID	NA

Table 3-27 PC3000 Unsupported System Mnemonics (continued)

## Slave Variable Address Ranges

The following tables show which characters are acceptable for Slave Variable Addresses, i.e. can be used for a Slave Channel Identity CHID

<b>ASCII Table Hexadecimal - Character</b>							
	21 !	22 '	23 #	24 \$	25 %	26 &	27 '
28 (	29 )	2A *	2B +	2C ,	2D -	2E .	2F /
30 0	31 1	32 2	33 3	34 4	35 5	36 6	37 7
38 8	39 9	3A :	3B ;	3C <	3D =	3E >	3F ?
40 @	41 A	42 B	43 C	44 D	45 E	46 F	47 G
48 H	49 I	4A J	4B K	4C L	4D M	4E N	4F O
50 P	51 Q	52 R	53 S	54 T	55 U	56 V	57 W
58 X	59 Y	5A Z	5B [	5C \	5D ]	5E ^	5F _
60 `	61 a	62 b	63 c	64 d	65 e	66 f	67 g
68 h	69 i	6A j	6B k	6C l	6D m	6E n	6F o
70 p	71 q	72 r	73 s	74 t	75 u	76 v	77 w
78 x	79 y	7A z	7B {	7C	7D }	7E ~	

Table 3-28 Acceptable Slave Address Characters

Valid Characters which can be used for the first mnemonic character Mn0 of a Slave Variable Address are given in Table 3-29.

<b>ASCII Table Hexadecimal - Character</b>							
30 0	31 1	32 2	33 3	34 4	35 5	36 6	37 7
38 8	39 9						

Table 3-29 Acceptable Characters for Mnemonics Mn0

Valid characters which can be used for the second mnemonic character Mn1 of a Slave variable Address are given in table 3-30.

<b>ASCII Table Hexadecimal - Character</b>							
30 0	31 1	32 2	33 3	34 4	35 5	36 6	37 7
38 8	39 9	3A :	3B ;	3C <	3D =	3E >	3F ?
40 @	41 A	42 B	43 C	44 D	45 E	46 F	47 G
48 H	49 I	4A J	4B K	4C L	4D M	4E N	4F O
50 P	51 Q	52 R	53 S	54 T	55 U	56 V	57 W
58 X	59 Y	5A Z	5B [	5C \	5D ]	5E ^	5F _
60 `	61 a	62 b	63 c	64 d	65 e	66 f	67 g
68 h	69 i	6A j	6B k	6C l	6D m	6E n	6F o
70 p	71 q	72 r	73 s	74 t	75 u	76 v	77 w
78 x	79 y	7A z	7B {	7C	7D }	7E ~	

Table 3-30 Acceptable Characters for Mnemonics Mn1

The ASCII code table is given for general reference.

<b>ASCII Table Hexadecimal - Character</b>							
00 NUL	01 SOH	02 STX	03 ETX	04 EOT	05 ENQ	06 ACK	07 BEL
08 BS	09 HT	0A NL	0B VT	0C NP	0D CR	0E SO	0F SI
10 DLE	11 DC1	12 DC2	13 DC3	14 DC4	15 NAK	16 SYN	17 ETB
18 CAN	19 EM	1A SUB	1B ESC	1C FS	1D GS	1E RS	1F US
20 SP	21 !	22 '	23 #	24 \$	25 %	26 &	27 '
28 (	29 )	2A *	2B +	2C ,	2D -	2E .	2F /
30 0	31 1	32 2	33 3	34 4	35 5	36 6	37 7
38 8	39 9	3A :	3B ;	3C <	3D =	3E >	3F ?
40 @	41 A	42 B	43 C	44 D	45 E	46 F	47 G
48 H	49 I	4A J	4B K	4C L	4D M	4E N	4F O
50 P	51 Q	52 R	53 S	54 T	55 U	56 V	57 W
58 X	59 Y	5A Z	5B [	5C \	5D ]	5E ^	5F _
60 `	61 a	62 b	63 c	64 d	65 e	66 f	67 g
68 h	69 i	6A j	6B k	6C l	6D m	6E n	6F o
70 p	71 q	72 r	73 s	74 t	75 u	76 v	77 w
78 x	79 y	7A z	7B {	7C	7D }	7E ~	7F DEL

Table 3-31 ASCII Codes

## Features Not Supported

This describes the operation of the EI\_Bisync\_S function block included in Version 2.00 firmware onwards with the exception of the features below which have not yet been implemented.

- Multi-block message transfer.
- Access to all the parameters of a function block instance as a single composite parameter.
- 32-bit IEEE packed floating point values ( only 24-bit format for IEEE floating point values are supported)



## Parameter Attributes

Name	Type	Cold Start	Read Access	Write Access	Type Specific Information	
Port	<b>STRING</b>	'0A'	Oper	Config		
Baud	<b>ENUM</b>	_9600	Oper	Config	Enumerated Values	_75(0) _300(1) _600(2) _1200(3) _2400(4) _4800(5) _9600(6) _19200(7) _38400(8) _57600(9) _115200(10)
Wr_Protect	<b>BOOL</b>	No	Oper	Super	Senses	No(0) Yes(1)
TXB_Max	<b>SINT</b>	0	Oper	Config	See Note 1	
Gid	<b>STRING</b>	' '	Oper	Config		
Status	<b>BOOL</b>	NOGO	Oper		Block Senses	NOGO(0) Go(1)
Error_No	<b>SINT</b>	0	Oper	Block	Hi Limit Lo Limit	255 0

Table 3-32 EI\_Bisync\_S Parameter Attributes

## RAW\_COMMUNICATIONS FUNCTION BLOCK

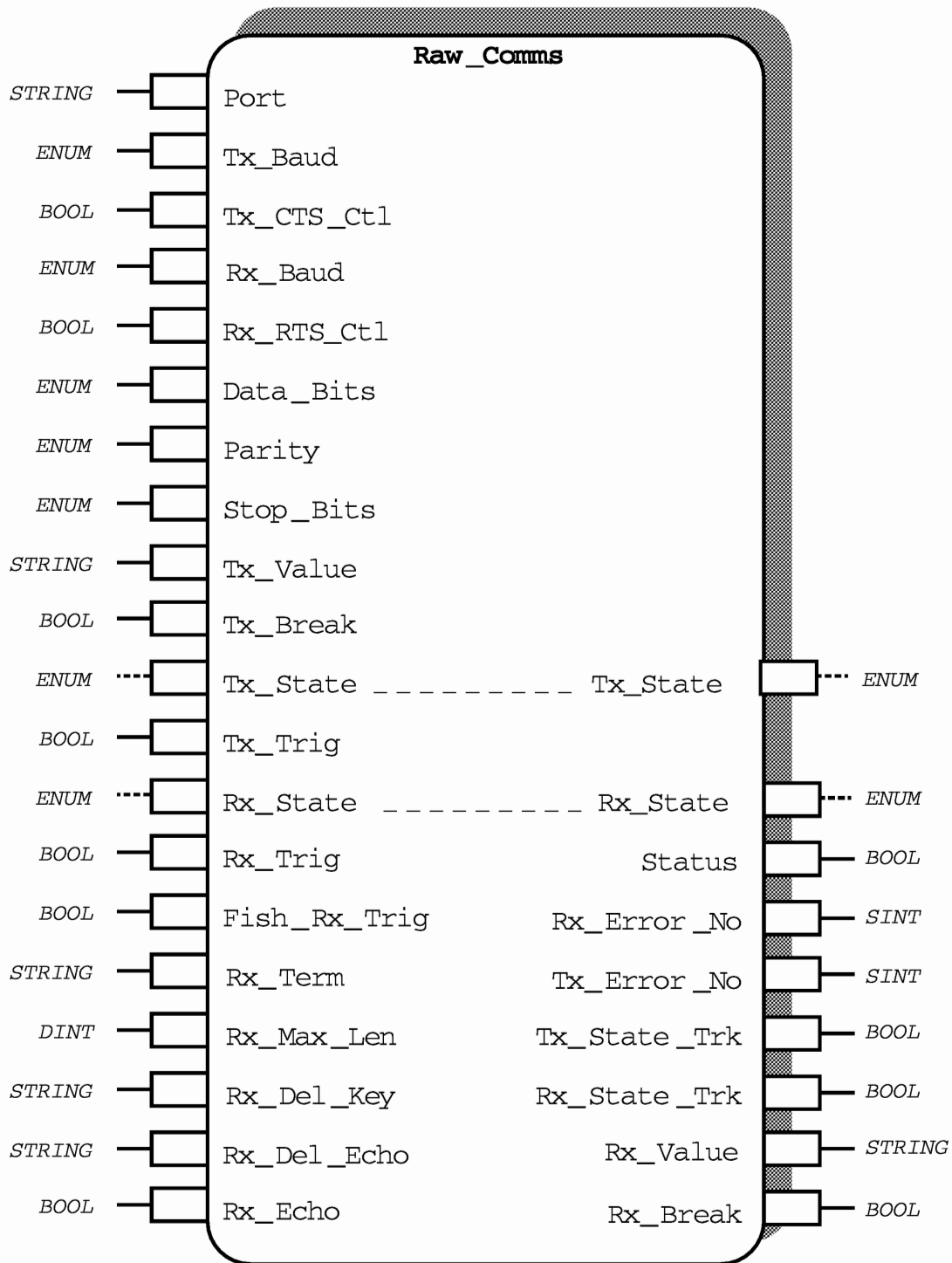


Figure 3-22 Raw\_Comms Diagram

## Functional Description

The Raw\_Comms Function Block provides facilities to directly control the transmission and reception of messages over a serial link. Before reading this description, you are advised to gain a general understanding of the PC3000 communications system by reading the PC3000 Communications Overview.

The Raw\_Comms function block is available for applications where it is necessary to have low level control of the serial communications port and have the flexibility to construct or analyse messages exactly as transmitted or received over a serial link. Because there is no protocol, no extra messages or message formats are created by the driver.

The Raw\_Comms block can be assigned to any serial port using the Port input parameter ( see 'PC3000 Communications Overview' section 'Assigning a Communications Function Block to a Port').

Raw\_Comms provides a wide range of low level facilities including :

- Direct access to messages as transmitted or received over the serial link.
- Independent control of message transmission and reception including separate baud rate selection on transmit and receive lines.
- Message flow control using Clear To Send (CTS) and Request To Send. This is not currently supported on LCM or ICM ports
- Selectable echoing of received characters when required
- User selectable delete sequence for character deletion in the receive buffer.

Developing user programs to operate with Raw\_Comms is more complex than using the protocol driver function blocks because the structure of messages and timing will all need to be handled by the program.

**Note:** Unlike protocol specific communications driver function blocks, it is not possible to use Slave or Remote Variables with Raw\_Comms

## Raw\_Comms Usage

Typical applications for the Raw\_Comms driver function block include :

- Communication with devices using simple non-standard protocols,
- Sending reports to serial line printers or to special purpose printers, for example, for label generation.
- Communication with character based terminals such as DEC VT100 or with simple display devices.

The Raw\_Comms function block gives the user the lowest possible level of interface to the serial ports on the PC3000. All serial data encoding parameters such as baud rate and parity are user programmable. The block does the minimum of processing on the received and sent characters in order to maintain

maximum flexibility. Some simple facilities for echo, deletion and receive string termination have been included in the block to aid programming but all these can be disabled if not required.

## Function Block Attributes

Type:..... 8 60  
Class: ..... COMMS  
Default Task: ..... Task\_1  
Short List: ..... Port Status  
Memory Requirement: ..... 864 Bytes

## Parameter Description

The Raw\_Comms block has a total of 25 parameters. These conveniently break down into seven groups which are configuration, transmit, transmit control, receive, receive control, echo/deletion and status.

### Driver Configuration Parameters

The Raw\_Comms block has a number of inputs which configure various aspects of the driver. Changing these parameters while the user program is executing will have no effect on the driver, except under special circumstances - see 'PC3000 Communications. Overview' section 'Temporarily changing configuration parameters'.

### Port

The Port parameter is the two character address of the port to which the block refers. The first character is a number from 0 to 5 representing the rack slot and the second character is a letter representing the port within that slot. e.g. '0C' would be port C on the LCM and if there were an ICM in slot 3 '3A' would refer to its top port.

### Tx\_Baud

The Tx\_Baud parameter gives a choice of 11 different transmit baud rates from 75 baud up to 115.2 Kbaud.

Enum Value	Baud Rate
0	75
1	300
2	600
3	1200
4	2400
5	4800
6	9600
7	19200
8	38400
9	57600
10	115200

Table 3-33 Raw\_Comms Transmit/Receive Baud Rate

**Note:-** That not all ports will be able to support all baud rates. If the port chosen will not support the given rate an error will be indicated when the function block is first run (see description of Rx/Tx\_Error\_No). For details of which baud rates are supported by given ports/modules please refer to the module documentation.

#### Tx\_CTS\_Ctl

If the Tx\_CTS\_Ctl parameter is On then the transmitter will only send when the CTS line is active. If the Tx\_CTS\_Ctl parameter is Off the transmitter will not respond to the CTS line but will always transmit when requested. If CTS transmit flow control is not available on the given port this parameter should be set to Off.

If the facility is not available but the parameter is On then an error will be indicated when the function block is first run (see description of Rx/Tx\_Error\_No). For details of which ports/modules support CTS flow control please refer to the module documentation.

#### Rx\_Baud

The Rx\_Baud parameter is the same as Tx\_Baud except that it refers to the receive baud rate.

### Tx\_RTS\_Ctl

The Tx\_RTS\_Ctl parameter is used to request flow control for the receiver. If Tx\_RTS\_Ctl is On the RTS output will be active if and only if the receiver is ready to receive characters. If the parameter is set to Off RTS will remain active always. If RTS flow control is not supported by the port then Tx\_RTS\_Ctl should be Off or an error will be indicated when the function block is first run (see description of Rx/Tx\_Error\_No ). For details of which ports/modules support RTS flow control please refer to the module documentation.

### Data\_Bits

The Data\_Bits parameter sets the number of bits per character for both receive and transmit. If this is set less than 8 then the most significant (8 -Data\_Bits ) bits will be ignored when sending and assumed zero when receiving. The number of bits per character available are 5, 6, 7 or 8 and they are represented by enumerated values of \_5, \_6, \_7 and \_8 .

### Parity

The Parity parameter can be set to EVEN, ODD, SPACE, MARK or NONE. If the Parity is NONE then only the start, data and stop bits are sent. If parity is set to EVEN, ODD, SPACE or MARK the extra bit will be sent following the data bits. On receiving this bit will be checked and an error indicated if it does not comply to the Parity parameter setting.

### Stop\_Bits

The Stop\_Bits parameter sets the number of stop bits expected by the receiver and sent by the transmitter. There may be 1, 1.5 or 2 stop bits, represented by the enumerated values \_1, \_1\_5 and \_2 .

## Transmit Parameters

### Tx\_Value

The Tx\_Value parameter is used to hold the string to be transmitted. The string may be up to 128 characters long. Where the **Data\_Bits** parameter has been set less than \_8, only the specified number of least significant bits will be sent for each character. One, two or three of the most significant bits of the character will be ignored depending on the number of Data\_Bits.

### Tx\_Break

When the Tx\_Break boolean parameter is set to On any transmission currently in progress will be aborted and a break condition will be generated on the serial link. If characters are lost the Tx\_Error\_No will reflect this. Note that transmitting of characters is not possible while Tx\_Break is On.

### Transmit Control Parameters

In order to initiate transmission of the Tx\_Value there are two transmit control parameters. Both these parameters can be used to initiate transmission but one is more readily suited to control from wiring and the other to control by sequence program. The Tx\_State parameter also indicates status of the transmitter.

#### Tx\_State

The Tx\_State parameter can have values of OK, PENDING, ERROR and WRITE which indicate the current state of the transmitter. Assigning Write to this parameter from sequence program can be used to start the transmission of the Tx\_Value. For full details of transmitter operation see Transmitting Strings page 3-108.

#### Tx\_Trig

The Tx\_Trig parameter is provided to ease control of the Raw\_Comms function block by wiring. When the Tx\_Trig goes from Off to On transmission is initiated as described in Transmitting Strings page 3-108.

### Receive Parameters

#### Rx\_Value

The Rx\_Value string contains characters received via the serial port. For details of how the receiving is controlled/buffered see Receiving Strings page 3-109

#### Rx\_Break

If the Rx\_Break parameter is On it indicates that a break condition has been detected on the receive line. When the break condition clears the Rx\_Break parameter will return to Off .

### Receive Control Parameters

The Raw\_Comms function block includes five parameters which control the input of strings of characters. The parameters are outlined below and the full details of the receiving process is described in Receiving Strings page 3-109

#### Rx\_State

The Rx\_State parameter is similar to the Tx\_State parameter, having possible values of OK, PENDING, ERROR, READ and FLUSH. In a similar way to the Tx\_State parameter the Rx\_State parameter can be read to discover the current state of the receiver and can also be written to control it. For details of how the receiver is controlled see Receiving Strings page 3-109.

### Rx\_Trig

Rx\_Trig, like Tx\_Trig is designed to make it easy to control the receiving using PC3000 wiring. When Rx\_Trig goes from Off to On receiving is started, as described in Receiving Strings page 3-109.

### Flsh\_Rx\_Trig

Flush\_Rx\_Trig is like Rx\_Trig except that it flushes the receive buffer instead of starting receiving. Again there is a detailed description of the operation of Flush\_Rx\_Trig in Receiving Strings page 3-109

### Rx\_Term

The Rx\_Term parameter is a one character long string which is used to identify the end of a line of input (see Section **\*\*REF rx method**). If the Rx\_Term is left blank the input will only be read from the receive buffer into the function block Rx\_Value parameter when Rx\_Max\_Len characters have been received.

### Rx\_Max\_Len

Rx\_Max\_Len specifies the maximum number of characters which will be held in the receive buffer before they are passed on to the Rx\_Value. The Rx\_Max\_Len should be in the range 1..128, 128 being the maximum length of the Rx\_Value string.

## Echo And Deletion Parameters

A simple echo and delete facility is provided by the Raw\_Comms block in order to make it easier to implement data entry on a device which has no local echo. The operation of the echo facility is explained in full in section Echo on page 3-1.

### Rx\_Echo

This boolean turns On or Off the echo facility.

### Rx\_Del\_Key

This one character string parameter specifies the character which is to be used for deletion, for example '\$7F'. If left blank deletion will not be performed.

### Rx\_Del\_Echo

In order to delete a character on some screens it is necessary to echo a number of characters, for example '<BS><space><BS>'. The Rx\_Del\_Echo parameter is used to specify this string of characters.



## Status Parameters

The final group of parameters are the status parameters, which indicate any errors which occur in relation to receiving and transmitting.

### Status

The Status parameter is a boolean parameter which takes the value Go or NoGo to indicate if the communications is functioning without problems. If an error occurs the Status parameter changes to NoGo and a non zero error number will appear in the Rx\_Error\_No or Tx\_Error\_No accordingly.

### Rx\_Error\_No

The Rx\_Error\_No indicates if an error has occurred in receiving of data. This could be anything from requesting a receive baud rate which the serial port could not support to framing errors while receiving. An Rx\_Error\_No of 0 indicates that no error has occurred. For full details of the error codes see section Error Codes.

### Tx\_Error\_No

The Tx\_Error\_No acts in a similar way to the Rx\_Error\_No except that it is exclusively for errors which are relevant to receiving. For full details of the error codes see section Error Codes.

## Configuration

Before transmitting or receiving begins it is necessary set up the configuration of the port. The configuration parameters, as described in Section Driver Configuration Parameters should be chosen before the user program is run as the configuration is fixed when the user program first begins executing.

### Transmitting Strings

After configuring the Raw\_Comms Function Block, a string can be transmitted by placing the string value into the **Tx\_Value** parameter and then triggering the sending of that string using either the **Tx\_State** parameter or the **Tx\_Trig** parameter. If the control of transmission is to be done from a sequence program it is recommended that the **Tx\_State** parameter be used as shown here :-

```
Port.Tx_Value := `The quick brown fox jumps over the lazy dog.`;  
Port.Tx_State := 3 (*Write*) ;
```

Setting the **Tx\_State** parameter to Write starts the transmission of the string. While the string is being transmitted the **Tx\_State** parameter changes to Pending and then, on completion, reverts to Ok. If an error occurs at any stage then the **Tx\_State** will go to Error, the **Tx\_Status** will go to NoGo and a non-zero error code will appear in **Tx\_Error\_No**. If an attempt is made to transmit again before the last string has been completely sent the old string will be

aborted, the new one begun and an error logged in the **Tx\_Error** parameter. For this reason, normally, the transition following such a pair of assignments contains a check for the transmission being completed, as shown here :-

```
( Port.Tx_State = 0 (*Ok*) )
```

If it is required to trigger the transmission of the string by wiring then the **Tx\_Trig** parameter should be used. A transition from Off to On of this parameter will initiate the sending of the **Tx\_Value**. Initiating transmission using **Tx\_Trig** acts in exactly the same way as assigning Write to the State parameter, with the state going to Pending until the **Tx\_Value** has been sent.

**Note:-** That control characters can be embedded in transmitted strings. These can be input from the PC3000 Programming Station using a special \$ 'dollar' format which is described in the appendix.

### Flushing The Transmit Buffer

There are two methods to flush the transmit buffer. When using ST wiring, the **Tx\_Value** can be set to the empty string i.e. ' ' and then the **Tx\_Trig** changed from Off to On, as if sending the string. The Raw\_Comms driver will treat this as a special case and will abort the current transmission.

Alternatively, when controlled by sequence program, changing the **Tx\_State** from Pending to Ok will cause the driver to abort the current transaction, clear the transmit buffer and reset the **Tx\_Error** to 0 (Ok).

### Receiving Strings

Receiving strings is controlled in a similar way to sending strings. The receiving of characters by the Raw\_Comms block is buffered to allow for echo, automatic deletion and to allow processing by the user program to be line at a time, rather than character at a time. Initially, for simplicity, it will be assumed that echo and deletion are not required. For receiving it is always necessary to consider the **Rx\_Term** and **Rx\_Max\_Len** parameters. These two parameters govern the point at which a received string is transferred from the receive buffer within the function block into the **Rx\_Value**. If an **Rx\_Term** character has been specified (i.e. the string is not empty) then as soon as that character is received the input (including the terminating character) will appear in the **Rx\_Value**. For example if it was necessary to process input which was terminated by <CR> then **Rx\_Term** would be set to '\$R' and an example value received in **Rx\_Value** might be 'This is some input\$R'.

**Note:-** That in ST strings

\$R represents the Carriage Return character, ASCII 13.

The **Rx\_Max\_Len** parameter is used to specify the maximum number of characters which are to be received before they are transferred into the **Rx\_Value**. If the **Rx\_Max\_Len** were set to 1 then characters would be input one at a time. The **Rx\_Term** and **Rx\_Max\_Len** when used together will cause the **Rx\_Value** to be produced when either of the conditions are met.

If the characters received were :-

```
'This_is_the_first_sentence._This_is_a_longer_second_sentence.'
```

and **Rx\_Term** and **Rx\_Max\_Len** were (.) and 30 respectively then the strings received in **Rx\_Value** would be :-

```
'This_is_the_first_sentence.'  
'_This_is_a_longer_sente'  
'nce.'
```

**Note:-** That if the received message contains unprintable control characters these are shown on the PC3000 Programming Station by using the \$ 'dollar' format which is described in later paragraphs.

### Flushing The Receive Buffer

The receive buffer can be cleared at any time by setting **Rx\_State** to Flush or by using changing the **Rx\_Flsh\_Trig** parameter from Off to On. This does not clear characters already the **Rx\_Value** but clears any characters which have been received and are queued to be copied into **Rx\_Value**.

### Echo And Deletion

The receive buffer within the Raw\_Comms function block allows two more facilities to be provided, namely echo and deletion.

#### Echo

If the Echo parameter is set to On then as characters are received they will be echoed out of the transmitter. These echo characters will not interfere with any transmission which is currently in progress but will be buffered up until the transmitter is free. While characters are being echoed the **Tx\_State** parameter will be set to Pending in just the same way as if a string had been written by the user program. For this reason it is necessary to check that the **Tx\_State** is Ok or Error before attempting to write strings when the Echo is On.

## Deletion

To support deletion of characters within the buffer before they reach the **Rx\_Value** there are two further parameters to be set up which are **Rx\_Del\_Key** and **Rx\_Del\_Echo**. If deletion is required then the **Rx\_Del\_Key** parameter should contain the character to be used for deletion. This will most often be either '\$7F' or '\$08'. If the deletion facility is not required then the **Rx\_Del\_Key** should be left as an empty string. Because it is often necessary to echo a number of characters in order to implement a deletion on a terminal the characters echoed for the deletion character come from the **Rx\_Del\_Echo** parameter. For example, on a VT100 terminal the delete key returns the code 7Fh and to delete a character from the display it is necessary to send '<BS><space><BS>'.

To configure the deletion for a VT100 it is therefore necessary to set **Rx\_Del\_Key** to '\$7F' and **Rx\_Del\_Echo** to '\$08 \$08'. If Echo is not On it is not necessary to set up the **Rx\_Del\_Echo** parameter but if deletion from the receive buffer is still required then **Rx\_Del\_Key** should be defined.

## Example

The example outlined here shows how a VT100 terminal could be used as a simple operator control panel for a process. Details of panel driving will be given but controlling of the process will be excluded to minimise complexity.

For this example the application will be assumed to be a simple batch process which requires the batch number and a setpoint to be input before the process runs. The panel is required to allow for entry of these values and, once entered, to display batch start time, date and setpoint. Please enter batch number and setpoint to start next run.

```
Batch number = 123456
Setpoint = 12.5
Batch number 123456, Setpoint 12.5 - Run started at 12:15:30 on 29-10-90.
PHASE 1..... Completed.
PHASE 2..... Completed.
PHASE 3..... Completed.
Batch number 123456 completed at 12:20:47 on 29-10-90
Please enter batch number and setpoint to start next run.
Batch number = _
```

Figure 3-23 Example Display Output

While the process is running it should show when each phase of the process is complete and, when the run is finished, show the finish time and date. It will then return to requesting another batch number and set point for the next run. An example of the display output after one run is shown in Figure 3-23.

#### Setting Up The Raw\_Comms Block

The VT100 terminal will be assumed to have been set for 9600 baud, 7 data bits, even parity, 1 stop bit and no local echo. It will also be assumed that the DEL key is to be used for deletion and it produces the ASCII code 7Fh. No flow control will be implemented in either direction as it is assumed that the terminal and PC3000 have sufficiently large receive buffers. Table 3-34 shows the function block parameter values which are needed for this example.

Parameter	Value	Comments
Port	'OB'	Port B on the LCM.
Tx_Baud	_9600	Transmit baud rate 9600 baud.
Tx_CTS_Ctl	Off	CTS flow control is not needed.
Rx_Baud	_9600	Receive baud rate 9600 baud.
Rx_RTS_Ctl	Off	RTS flow control is not needed.
Data_Bits	_7	7 data bits (receive and transmit).
Parity	Even	Even parity (receive and transmit).
Stop_Bits	_1	1 stop bit (receive and transmit).
Tx_Value	''	†
Tx_Break	Off	It is not necessary to send BREAK in this application.
Tx_State	Ok	†
Tx_Trig	Off	Tx_Trig is not needed here because transmitting is controlled by the sequence program using Tx_State .
Rx_State	Ok	†
Rx_Trig	Off	Rx_Trig is not needed here because receiving is controlled by the sequence program using Rx_State .
Flsh_Rx_Trig	Off	Flsh_Rx_Trig is not required here because it is not necessary to flush the receive buffer in this application.
Rx_Term	'\$R'	Input is terminated by carriage return.
Rx_Max_Len	128	The maximum number of characters which can input is 128.
Rx_Del_Key	'\$7F'	The VT100 delete key returns ASCII 7Fh.
Rx_Del_Echo	'\$08 \$08'	To delete a character on the VT100 it is necessary to send '<BS><space><BS>'.
Rx_Echo	On	Echo by the PC3000 will be required.

† These are the initial values; the parameter will be controlled from the sequence program.

Table 3-34 Example Parameter settings

### Sequencing Control Of The Machine

The application example is assumed to be controlled entirely from sequence program. The SFCs (Sequential Function Charts) for the application are shown in Figure 3-9.

**Note:-** That the three phases of the process, are exactly the same in the example and their SFC/ST are only shown once as Phase\_n. The three phases of the process would obviously not, be the same in practice but in the example the process control part of the application is not shown for clarity.

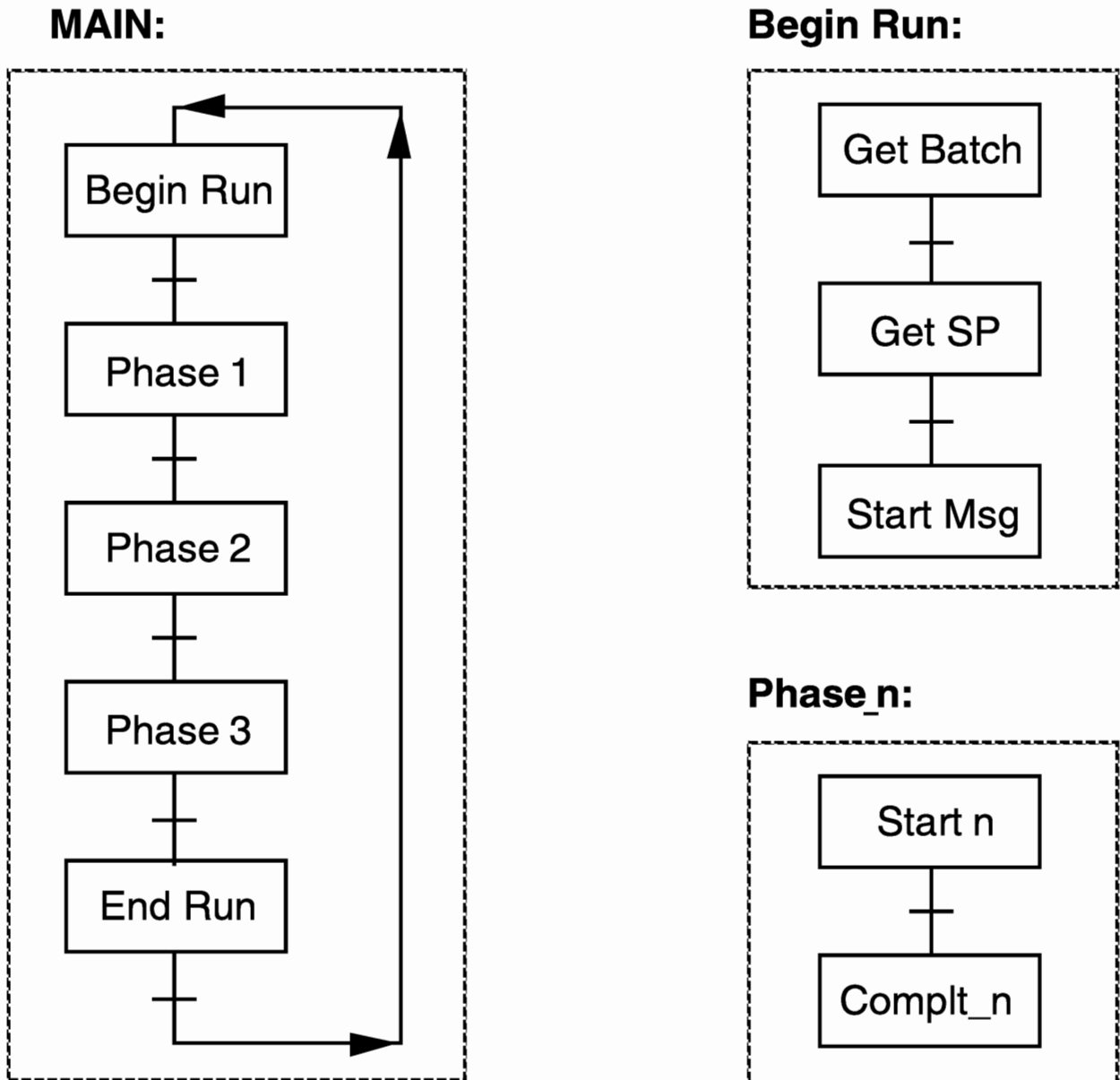


Figure 3-24 Example Sequential Function Charts

**BeginRun:** Get Batch , the first step within BeginRun , outputs a message/prompt by assigning a string to the **Tx\_Value** and then setting **Tx\_State** to **Write** .

**Note:-** The use of '\$R\$L' (carriage return, line feed) to move onto the next line allowing two lines to be displayed by sending one string.

After the prompt has been sent **Rx\_State** is set to **Read** which initiates input of the batch number. The transition from GetBatch to Get\_SP waits for the input to be completed. The data entry is terminated either by the operator typing carriage return or entering 128 characters. The 128 character limit is not expected to be used but is necessary as this is the maximum number of characters which can be held in the **Rx\_Value** string. A test for **Tx\_State** being **Ok** is also included in the transition from GetBatch to Get\_SP in order to ensure that the batch number prompt has been displayed before the Get\_SP step puts up the setpoint prompt. Once the batch number string has been entered Get\_SP strips the carriage return from the end of the **Rx\_Value** and stores it away in a user variable.

**Note:-** That for simplicity in the example no checking has been done on the string although in a real application it may be necessary to do some form of validation of the batch number. Also that the batch number is kept in string format. This allows an alpha-numeric batch number to be used.

The Get\_SP step, having dealt with the batch number, prompts for the setpoint and initiates the read in a similar way as the GetBatch step. Once the setpoint has been read in it is decoded (using the **STRING\_TO\_REAL** function) and stored in a user variable by the StartMsg step. The rest of the StartMsg step composes the '... Run started ...' message into **Tx\_Value** using a number of concatenations and then sends it by setting **Tx\_State** to **Write** . The CONCAT/ function is used to join strings together.

**Note:-** That no validation of the setpoint is shown here for clarity although it may be necessary in a real application.

**Phase\_n:** The three phases of the process are each implemented as macros consisting of two steps and are all the same except in name. The Start\_n step outputs the message '**PHASE n .**' in the same way as was shown earlier. It is within this step that any sequencing for the process would be included. Following this the Cmpltd\_n step outputs the 'Completed.' message to show that the phase is over.

**End\_Run:**The End\_Run step is very similar to the second half of the StartMsg step except that it composes and starts transmission of the 'Batch ... completed ..' message. After the message has been sent the flow of control returns to BeginRun which prompts for the next batch run.



## ST For The Example Application MAIN Macro

```
INITIAL_STEP MAIN : (* MACRO *)
```

```
TRANSITION
```

```
    FROM BeginRun (* MACRO *)
    TO   Phase_1 (* MACRO *)
:= VT100.Tx_State = 0(*Ok*)
END_TRANSITION
```

```
TRANSITION
```

```
    FROM Phase_1 (* MACRO *)
    TO   Phase_2 (* MACRO *)
:= VT100.Tx_State = 0(*Ok*)
END_TRANSITION
```

```
TRANSITION
```

```
    FROM Phase_2 (* MACRO *)
    TO   Phase_3 (* MACRO *)
:= VT100.Tx_State = 0(*Ok*)
END_TRANSITION
```

```
TRANSITION
```

```
    FROM Phase_3 (* MACRO *)
    TO   End_Run
:= VT100.Tx_State = 0(*Ok*)
END_TRANSITION
```

```
STEP End_Run :
```

```
    VT100_Tx_Value      := CONCAT( IN1 := 'Batch number ' , IN2
:=                               BtchName.Val ) ;
    VT100_Tx_Value      := CONCAT( IN1 := VT100.Tx_Value , IN2
:= '                               completed at ' ) ;
    VT100_Tx_Value      := CONCAT( IN1 := VT100.Tx_Value , IN2
:=                               TIME_OF_DAY_TO_STR( IN :=
RT_Clock.Time_Of_Day ) )
;
    VT100_Tx_Value      := CONCAT( IN1 := VT10.Tx_Value , IN2
:= '                               on ' ) ; VT100_Tx_Value :=
CONCAT( IN1 := VT100.Tx_Value , IN2
:=                               DATE_TO_EURO_STRING( IN:=
RT_Clock.Date ) ) ;
    VT100_Tx_Value      := CONCAT( IN1 := VT100.Tx_Value , IN2
:=                               '.RLRL' ) ;
```

```

        VT100_Tx_State      := 3(*Write*) ;
        END_STEP
TRANSITION
        FROM End_Run
        TO   BeginRun (* MACRO *)
:= VT100.Tx_State = 0(*Ok*)
END_TRANSITION

END_STEP (* MAIN *)

```

## ST For The Example Application BeginRun Macro

```

STEP BeginRun : (* MACRO *)
    STEP GetBatch :
        VT100_Tx_Value      :=
            start           Please enter batch number and setpoint to
                           next run.$R$LBatch number = ' ;
        VT100_Tx_State      := 3(*Write*) ;
        VT100_Rx_State      := 3(*Read*) ;
    END_STEP

TRANSITION
    FROM GetBatch
    TO   Get_SP
:= ( VT100.Rx_State = 0(*Ok*) ) AND ( VT100.Tx_State = 0(*Ok*) )
END_TRANSITION

STEP Get_SP :
    BtchName_Val := LEFT( IN := VT100.Rx_Value , L :=
                        LEN( IN := VT100.Rx_Value ) - 1 ) ;
    VT100_Tx_Value := 'LSetpoint = ' ;
    VT100_Tx_State := 3(*Write*) ;
    VT100_Rx_State := 3(*Read*) ;
END_STEP

TRANSITION
    FROM Get_SP
    TO   StartMsg
:= ( VT100.Rx_State = 0(*Ok*) ) AND (
    VT100.Tx_State = 0(*Ok*) )
END_TRANSITION

STEP StartMsg :
    Setpoint_Val := STRING_TO_REAL( IN := VT100.Rx_Value ) ;

```

```

VT100_Tx_Value      := CONCAT( IN1 := 'LBatch number ' , IN2 :=
                               BtchName.Val ) ;
VT100_Tx_Value      := CONCAT( IN1:= VT100.Tx_Value , IN2 := ',
                               Setpoint ' );
VT100_Tx_Value      := CONCAT( IN1 := VT100.Tx_Value , IN2:=
                               REAL_TO_STRING( IN :=
                               Setpoint.Val, DPS:=          1 ) ) ;

VT100_Tx_Value      := CONCAT( IN1 := VT100.Tx_Value , IN2 := '
                               - Run started at ' ) ;
VT100_Tx_Value      := CONCAT( IN1 := VT100.Tx_Value , IN2 :=
                               TIME_OF_DAY_TO_STR( IN:=
                               RT_Clock.Time_Of_Day ) )
;
VT100_Tx_Value      := CONCAT( IN1 := VT100.Tx_Value , IN2 := '
                               on ' ) ;
VT100_Tx_Value      := CONCAT( IN1 := VT100.Tx_Value , IN2 :=
                               DATE_TO_EURO_STRING( IN :=
                               RT_Clock.Date ) ) ;
VT100_Tx_Value      := CONCAT( IN1 := VT100.Tx_Value , IN2 :=
                               '.RL' ) ;
VT100_Tx_State      := 3(*Write*) ;
END_STEP
END_STEP (* BeginRun *)

```

## ST For The Example Application Phase\_n Macros

```

STEP Phase_n : (* MACRO *)

STEP Start_n :
  VT100_Tx_Value := 'Phase n. ' ;
  VT100_Tx_State := 3(*Write*) ;
END_STEP

TRANSITION
  FROM Start_n
  TO  Cmpltd_n
  :=  VT100.Tx_State = 0(*Ok*)
END_TRANSITION

STEP Cmpltd_n :
  VT100_Tx_Value := 'Completed.RL' ;
  VT100_Tx_State := 3(*Write*) ;
END_STEP
END_STEP (* Phase_n *)

```

## Display of Control Characters in Strings

On the PC3000 Programming Station, unprintable control characters can be inserted into a string using the \$ 'dollar' format. The same format is used when displaying the contents of strings. The \$ sign is used to indicate that the character(s) that follow define a control character. Formats used are :

\$nn	- the character with ASCII code 'nn' in hexadecimal - eg. 0A
\$\$	- the dollar sign
\$'	- the single quote
\$L	- line feed ( ASCII 0A in hexadecimal )
\$N	- newline (converted to a RL pair)
\$P	- form feed (ASCII 0C in hexadecimal)
\$R	- carriage return (ASCII 0D hexadecimal)
\$T	- tab (ASCII 09 hexadecimal)

Table 3-35 Control Characters for Strings

## Error Reporting

The error codes which appear in **Rx\_Error\_No** and **Tx\_Error\_No** are shown in Table 3-36. Some of the error codes may appear when the block is first run indicating a port initialisation failure. These errors will be logged in both **Rx\_Error\_No** and **Tx\_Error\_No** and are marked with a † in the table. The error code 0 represents OK and indicates that the block is operating normally. The table shows under error code 0 (OK) the situations in which the **Tx/Rx\_Error\_No** s will return to zero following an error.

**Note:-** That it is possible to force the error codes back to zero by writing Ok to the appropriate Rx/Tx\_State parameter.

<b>Rx_Error_No</b>	<b>Tx_Error_No</b>	<b>Error description</b>
0	0	<p>OK</p> <ul style="list-style-type: none"> <li>•The port has been initialised ok.</li> <li>•A receive has been aborted by changing Rx_State from Pending to Ok .</li> <li>•A receive error has been cleared by changing Rx_State from Error to Ok .</li> <li>•A transmission has been aborted by changing Tx_State from Pending to Ok .</li> <li>•A transmit error has been cleared by changing Rx_State from Error to Ok .</li> <li>•A transmit has been completed successfully.</li> <li>•A string has been received successfully.</li> <li>•A receive buffer flush has been completed successfully.</li> </ul>
1	1	<p>NO PORT ADDRESS †</p> <ul style="list-style-type: none"> <li>•The Port string has less than two characters.</li> </ul>
2	2	<p>REQUESTED DATA BITS NOT AVAILABLE †</p> <ul style="list-style-type: none"> <li>•The specified port is not able to support the number of data bits selected.</li> </ul>
3	3	<p>REQUESTED PARITY SETTING NOT AVAILABLE †</p> <ul style="list-style-type: none"> <li>•The selected Parity is not supported by this port.</li> </ul>
4	4	<p>REQUESTED STOP BITS NOT AVAILABLE †</p> <ul style="list-style-type: none"> <li>•The selected number of stop bits is not supported by this port.</li> </ul>
5	5	<p>RX BAUD RATE NOT AVAILABLE †</p> <ul style="list-style-type: none"> <li>•The chosen receive baud rate is not available on this port.</li> </ul>
6	6	<p>TX BAUD RATE NOT AVAILABLE †</p> <ul style="list-style-type: none"> <li>•The chosen transmit baud rate is not available on this port.</li> </ul>
7	7	<p>RTS NOT AVAILABLE †</p> <ul style="list-style-type: none"> <li>•RTS flow control is not available on this port.</li> </ul>
8	8	<p>CTS NOT AVAILABLE †</p> <ul style="list-style-type: none"> <li>•CTS flow control is not available on this port.</li> </ul>

Table 3-36 Raw\_Comms Error Codes

Rx_Error_No	Tx_Error_No	Error description
11	11	ILLEGAL SLOT † •The slot number specified in the Port parameter does not have any serial ports in it.
12	12	ILLEGAL PORT † •The slot specified does not have a port with the number specified in the Port parameter.
14	14	RX & TX BAUD RATES INCOMPATIBLE † •The combination of receive and transmit baud rates chosen are not possible on this port.
17	17	PORT IN USE † •Another driver block has already attached to this port.
-	95	FORCED TX ERROR •The Tx_State parameter has been set to Error to force an error.
96	-	FORCED RX ERROR •The Rx_State parameter has been set to Error to force an error.
97	-	ECHO LOST •Tx_Break was On when receiving a character which should be echoed causing it to be lost. •The echo buffer overflowed causing echo characters to be lost.
98	-	RX CHARS LOST •The receive buffer has overflowed causing characters to be lost. •Rx_State has been changed from Pending to Read causing a new read to begin after flushing the receive buffer.
-	99	TX CHARS LOST •Tx_Break has been turned on while characters were being transmitted so some characters were lost. •An attempt was made to transmit characters while Tx_Break was on so the transmit was aborted.
100	-	RX OVERRUN ERROR •An overrun error was detected on a received character.
101	-	RX PARITY ERROR •A parity error was detected on a received character.

Table 3-36 Raw\_Comms Error Codes (continued)

---

<b>Rx_Error_No</b>	<b>Tx_Error_No</b>	<b>Error description</b>
102	-	RX PARITY & OVERRUN ERROR •A parity and an overrun error were detected while receiving.
103	-	RX FRAMING ERROR •A framing error was detected on a received character.
104	-	RX FRAMING & OVERRUN ERROR •A framing and an overrun error were detected while receiving.
105	-	RX FRAMING & PARITY ERROR •A framing and a parity error were detected while receiving.
106	-	RX FRAMING, OVERRUN & PARITY ERROR •A framing, parity and overrun error were detected while receiving.

Table 3-36 Raw\_Comms Error Codes (continued)

## Parameter Attribute

Name	Type	Cold Start	Read Access	Write Access	Type Specific Information	
Port	<b>STRING</b>	'0A'	Oper	Config		
Tx_Baud	<b>ENUM</b>	_600	Config	Config	Enumerated Values	_75(0) _300(1) _600(2) _1200(3) _2400(4) _4800(5) _9600(6) _19200(7) _38400(8) _57600(9) _115200(10)
Tx_CTS_Ctl	<b>BOOL</b>	ON	Config	Config	Senses	OFF(0) ON(1)
Rx_Baud	<b>ENUM</b>	_600	Config	Config	Enumerated Values	_75(0) _300(1) _600(2) _1200(3) _1200(3) _2400(4) _4800(5) _9600(6) _19200(7) _38400(8) _57600(9) _115200(10)
Rx_RTS_Ctl	<b>BOOL</b>	ON	Config	Config	Senses	OFF(0) ON(1)
Data_Bits	<b>ENUM</b>	_7	Config	Config	Enumerated Values	_5(0) _6(1) _7(2) _8(3)
Parity	<b>ENUM</b>	SPACE	Config	Config	Enumerated Values	EVEN(0) ODD(1) SPACE(2) MARK(3) NONE(4)
Stop_Bits	<b>ENUM</b>	_2	Config	Config	Enumerated Values	_1(0) _1_5(1) _2(2)

Table 3-37 Raw\_Comms Parameter Attributes (continued)



<b>Name</b>	<b>Type</b>	<b>Cold Start</b>	<b>Read Access</b>	<b>Write Access</b>	<b>Type Specific Information</b>	
Tx_Value	<b>STRING</b>	''	Oper	Oper		
Tx_Break	<b>BOOL</b>	ON	Oper	Oper	Senses	OFF(0) ON(1)
Rx_Trig	<b>BOOL</b>	YES	Oper	Oper	Senses	NO(0) YES(1)
Flsh_Rx_Trig	<b>BOOL</b>	YES	Oper	Oper	Senses	NO(0) YES(1)
Rx_Term	<b>STRING</b>	''	Oper	Oper		
Tx_State	<b>ENUM</b>	ERROR	Oper	Oper	Enumerated Values	OK(0) PENDING(1) ERROR(2) WRITE(3)
Rx_Max_Len	<b>SINT</b>	1	Oper	Oper	High Limit Low Limit	128 1
Rx_Del_Key	<b>STRING</b>	''	Oper	Oper		
Rx_Del_Echo	<b>STRING</b>	''	Oper	Oper		
Rx_Echo	<b>BOOL</b>	ON	Oper	Oper		
Status	<b>BOOL</b>	GO	Oper	Block	Enumerated Values	NOGO(0) Go(1)
Rx_Error_no	<b>SINT</b>	0	Oper	Block	High Limit Low Limit	255 0
Tx_Error_no	<b>SINT</b>	0	Oper	Block	High Limit Low Limit	255 0
Tx_State_Trk	<b>BOOL</b>	YES	Super	Block	Senses	NO(0) YES(1)
Rx_State_Trk	<b>BOOL</b>	YES	Super	Block	Senses	NO(0) YES(1)
Rx_Value	<b>STRING</b>	''	Oper	Block		
Rx_Break	<b>BOOL</b>	YES	Oper	Block	Senses	NO(0) YES(1)

Table 3-37 Raw\_Comms Parameter Attributes (continued)

## SIEMENS\_M\_S FUNCTION BLOCK

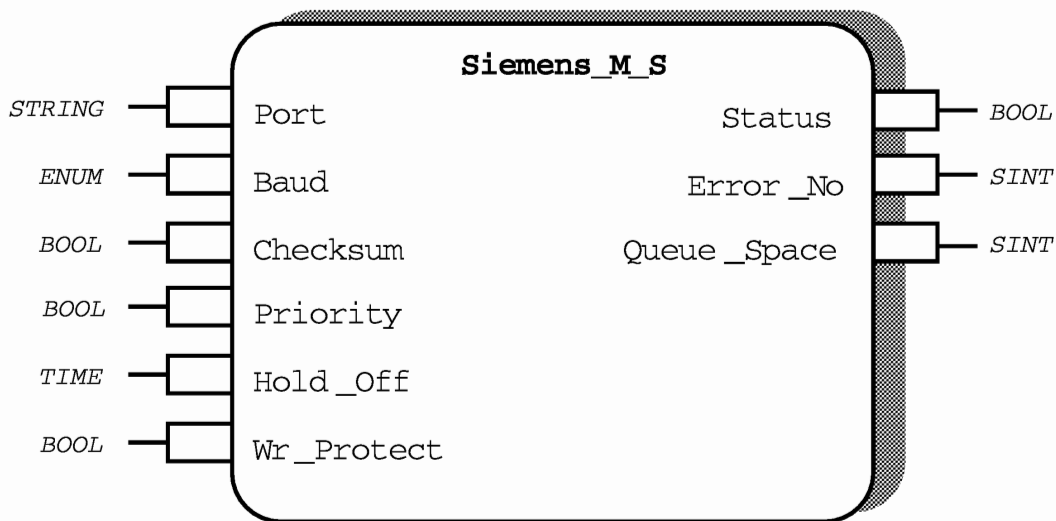


Figure 3-25 Function Block Diagram

### Functional Description

The Siemens\_M\_S function block supports serial communications on a designated serial communications port using the Siemens 3964(R) protocol. The function block implements the 3964(R) procedure and the RK 512 interpreter as specified in the Siemens protocol specification with modifications to conform with the PC3000 function block methodology.

There are two features of the RK 512 interpreter which are not supported by the PC3000 implementation :

- Follow on telegrams are not supported.
- The only 'command type' supported is data block (D).

Before reading this description, you are advised to gain a general understanding of the PC3000 communications system by reading the PC3000 Communications Overview.

This function block is required when designing or programming the PC3000 to communicate with a single Siemens PLC using a serial link for a point-to-point connection. Details in this manual will be useful when developing applications that use the Siemens communications driver function block. As the Siemens driver can function in both master and slave modes, this document also includes information on using both Slave and Remote Variables.

The following documents should be referenced if a detailed understanding of the Siemens 3964(R) protocol is required :

- Siemens Aktiengesellschaft. Siemens COM525 Programming Package for the Communications Processors CP524 and CP525 (S5-DOS) User Guide Volume 1 (chapter 7, section 5).

- Siemens Aktiengesellschaft. Siemens COM525 Programming Package for the Communications Processors CP524 and CP525 (S5-DOS) User Guide Volume 1 (chapter 7, section 4).
- Siemens Aktiengesellschaft. Siemens COM525 Programming Package for the Communications Processors CP524 and CP525 (S5-DOS) User Guide Volume 1 (chapter 7, section 4.1).
- Siemens Aktiengesellschaft. Siemens COM525 Programming Package for the Communications Processors CP524 and CP525 (S5-DOS) User Guide Volume 1 (chapter 7, section 8.2).

## Special Terminology

To comply with Siemens terminology the following term has been adopted :

**Partner** an alternative term for the remote device which is communicating with the PC3000; this will generally be a Siemens PLC.

This function block deals with the protocol specific details of the communications with a Siemens PLC and is supported by generic remote variable and slave variable function blocks. The remote variable and slave variable blocks are linked to the driver by means of a protocol specific address. The remote variable blocks correspond to the master side of the driver and initiate reads from and writes to remote devices. On the slave side, the slave variable blocks define values which can be read or written by a remote device using the protocol type specified in the address parameter, in this case 'SI' for Siemens.

**Note:** This driver supports the Remote/Slave String and Remote/Slave Integer function blocks.

## Function Block Attributes

Type:..... 8 70  
 Class: .....COMMS  
 Default Task: .....Task\_1  
 Short List: .....Port Status Queue\_Space  
 Memory Requirement:.....2580 Bytes

## Parameter Description

### Driver Configuration Parameters

The Siemens\_M\_S block has a number of inputs which configure various aspects of the driver and must be set before the user program is run. Changing these parameters while the user program is executing will have no affect on the driver except under special circumstances - see 'PC3000 Comms. Overview' section 'Temporarily changing configuration parameters'. The only inputs to the

Siemens\_M\_S function block which may be changed while it is running are **Hold\_Off** and **Wr\_Protect**.

#### Port

The **Port** parameter is the two character address of the port on which the 3964(R) procedure is to run. The first character is a number from 0 to 5 representing the rack slot and the second character is a letter representing the port within that slot. e.g. '0C' would be port C on the LCM and if there were an ICM in slot 3 '3A' could refer to its top port.

#### Baud

The **Baud** parameter gives a choice of 11 different rates from 75 baud up to 115.2 kbaud (see Table 4-2) with a default of 9600 baud.

**Note:-** That not all ports will be able to support all baud rates. This will be indicated by an error when the function block is first run (see description of **Error\_No**).

<b>Enum Value</b>	<b>Baud Rate</b>
0	75
1	300
2	600
3	1200
4	2400
5	4800
6	9600
7	19200
8	38400
9	57600
10	115200

Table 3-38 Siemens\_M\_S\_Baud Rates

#### Checksum

When the **Checksum** parameter is set to **Yes** the 3964R procedure, which includes checksumming, is used. When **Checksum** is set to **No** the 3964 procedure is used which does not use checksumming. In all other respects the two procedures are the same.

### Priority

The **Priority** parameter must be set so that one end of the link is **High** and the other **Low**. This will be used to decide which end of the link must back down in the case of contention on the link.

### Hold\_Off

The **Hold\_Off** parameter is the length of time the driver will wait while a write is protected before returning an error telegram to the partner. For an explanation of how hold off works and suggested values for this parameter see page 3-134.

## Driver Status Parameters

The driver status is indicated by three output parameters in the Siemens\_M\_S function block.

### Status

The **Status** parameter is a boolean indication of the state of the link controlled by the driver. If there are no problems with the link this parameter reads **Go**, but when an error occurs it changes to **NoGo**. If the **Status** is **NoGo** the **Error\_No** parameter indicates the reason for the problem.

### Error\_No

The **Error\_No** parameter indicates the reason for any errors with the link. If the link **Status** is **Go** then the **Error\_No** will be 0 (OK). For full details of error handling and error codes see Section Error Reporting, page 3- and Error Codes page 3- .

### Queue\_Space

The **Queue\_Space** parameter indicates the amount of space left in the queue for remote parameter operations. If this reaches zero then it implies that the link bandwidth is not sufficient to cope with the number of remote variable requests being made and data will be lost. If this situation arises the parameter polling rate should be reduced.

## Driver Write Protection

### Wr\_Protect

The **Wr\_Protect** input parameter on the Siemens\_M\_S driver block is a global write protect which can be used to disable writing to all addresses through this driver's port.

## Remote Variable Operation

The master requests made by the PC3000 are controlled by having one or more remote variable blocks. If only slave operation is required it is not necessary to have any Remote Variable blocks. The Siemens\_M\_S driver supports Remote\_Str, Remote\_Int and Remote\_SW parameters.

## Addressing

It is necessary to set up a protocol specific **Address** in the remote variable block which is the address used to access the partner. An example address is shown in Figure 4-2 The port is defined as in the Siemens\_M\_S function block **Port** parameter as a rack slot number followed by a letter for the port within that slot. The protocol specific part of the address begins with the CPU number which would normally be from 1 to 7. This is followed by the command type letter which should be D for data block. The data block number and data word number then follow separated by a period. The length of block to be fetched/sent can be specified in three ways.

- The first option is to end the address here, missing off the range type and length/end address. This indicates an implied block length of one word.e.g. 0A1D7.5
- The second option is to specify the length in words in which case a plus sign precedes the length. e.g. 0A2D20.16+8
- The final alternative is to specify a range. In this case a dash follows the data word number followed by the end data block number. e.g. 0A5D78.1-50

In practice either of the last two methods can be used to specify any address and the alternative methods are only given for ease of use. For example '0A1D78.10+1', '0A1D78.10-10' and '0A1D78.10' all represent the same address. Note that the range is inclusive.

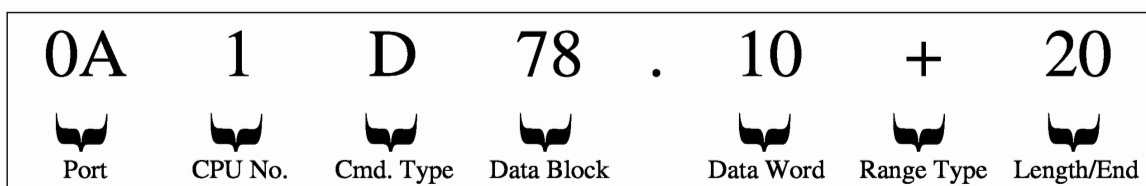


Figure 3-26 Remote Variable Address Example

## Usage Remote\_Str

Refer to the 'PC3000 Communications. Overview' for further information on Remote Variable Function Blocks.

The address which is specified in the **Address** parameter of the Remote\_Str block should have a block length which is less than half the length of the

**New\_Value** parameter when writing or an error will be flagged indicating insufficient data to be sent. The words of data are represented by pairs of characters in the string, most significant byte first.

### Using of Remote\_Int

Addressing of remote integers is exactly the same as addressing of remote strings except for one respect. In the case of remote integers the address should specify a block length of one or two words only. If the address specifies a block length of one it can be used to read or write a one word integer. For writes of a one word integer the least significant 16 bits of the value will be written. If the address is two words long the whole integer will be written as two words, most significant word first.

### Using of Remote\_S\_W

Remote status words are a special form of integer which is separated into its constituent binary digits. Addresses for status words should be one word long only. In fact as far as the Siemens driver is concerned the Status Word (SW) is handled in the same way as an integer and so may be two words long. If a two word long address is used the driver will not object but the most significant word will be written as 0 and ignored when read. .

### Slave Variable Operation

The slave half of the Siemens driver is used to simulate a Siemens PLC within the PC3000. The addressing space of this pseudo PLC is defined by having a number of slave variable function blocks. The Siemens driver supports Slave\_Str and Slave\_Int. **ONLY**

#### Addressing

The address for the slave variable uses the same format as the remote variable address except that it has a two letter driver mnemonic, which in this case is 'SI' for Siemens, instead of the port address (see Figure 3-27)

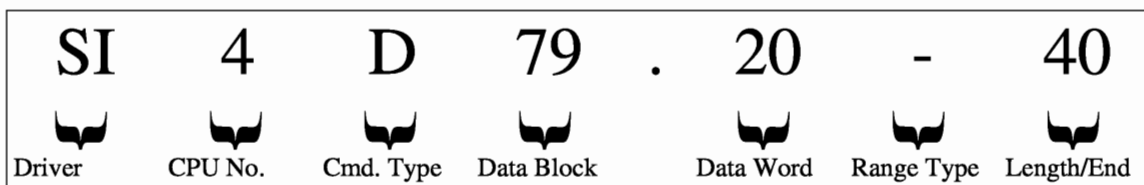


Figure 3-27 Slave Variable Address Example

### Using Slave Variables.

Refer to the 'PC3000 Comms. Overview' and the 'PC3000 Function Block Manual' for further information on the use of Slave Variable Function Blocks.

The slave will respond to any reads/writes by the partner ( remote device ) if they fall within the range specified by the **Address** parameter. If a number of slave variable function blocks are made to have consecutive addresses e.g. 1D79.10-19, 1D79.20+10, 1D79.30... then the partner can write into more than one slave variables with one block write (e.g. of words 10-30). For multi-parameter writes the write protect is ORed together. e.g. if any parameter within a block is write protected then the whole write will be blocked. Note that coordination flags are not supported by the PC3000 Siemens driver. The address is specified in terms of driver type rather than port address because a slave variable can respond to requests from many ports which have the same driver.

**Note:-** That like the remote integers the slave integers should have an address which is one or two words long to represent 16 or 32 bit integers. Again the 32 bit integers are represented with the most significant word first.

### Write Protection.

There are two ways in which a slave variable can be protected against being written via the Siemens communications protocol. In the driver block the **Wr\_Protect** parameter can be used to globally protect against all writes through the port specified in that block's **Port** parameter. If, for example, a monitoring device were to be attached to the PC3000 using Siemens 3964(R) then the **Wr\_Protect** parameter would be set to **Yes** to protect the PC3000 slave variables against writes.

The second form of protection is on a per slave variable basis. Each slave variable has a **Mode** parameter which can be either **Rd\_Wr**, **Rd\_Only** or **Wr\_Once**. In the **Rd\_Wr** mode no restrictions are put on reading or writing of the parameter by a remote device. When in the **Rd\_Only** mode reads are freely permitted but writes by a remote device will return an error reply telegram informing the remote device that the write was rejected

**Note:-** That it is possible to hold off this error reply for a short time to allow received values to be processed and the write protection to be removed.

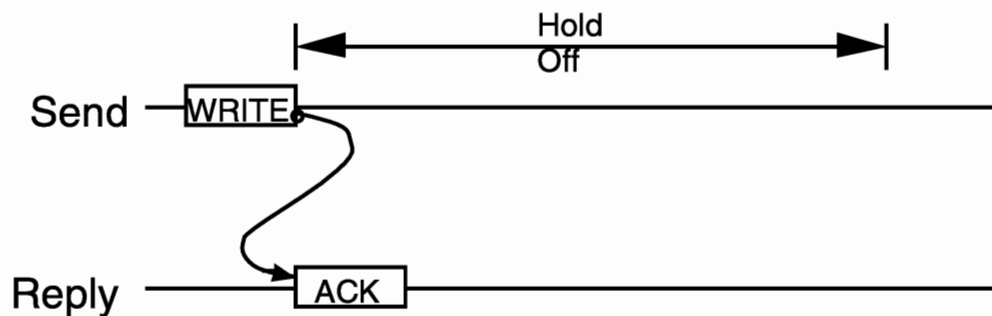
This will result in the error DB/DX DISABLED BY CF AT PARTNER being logged in the remote device. **Wr\_Once** mode is used in the situation where it is necessary to process each value as it is received in the slave variable block. The **Wr\_Once** mode works in the same way as **Rd\_Wr** mode initially, allowing read and write by the remote device. The difference comes once the value is written. When a write occurs while in **Wr\_Once** mode it is permitted but following this the **Mode** immediately changes to **Rd\_Only**, protecting the newly received value against being overwritten. Once the value has been processed by the user program the **Mode** can then be restored to **Wr\_Once** ready for the next value to be received. This is done from the sequence program simply by assigning **Wr\_Once** to the **Mode** parameter, or in the



wiring by making the **Trig\_Wr1** parameter go from **Off** to **On** which in turn makes the **Mode** change to **Wr\_Once**.

Hold Off.

If the write protection of a parameter is not on (i.e the slave variable **Mode** is **Rd\_Wr** or **Wr\_Once** and the driver block **Wr\_Protect** is **Off**) then a write to that parameter will immediately be acknowledged back to the sender as shown in Figure 3-28 and the value written into the parameter.



Write Protected

Figure 3-28 Writing With Write Protection Off

In order to stop the slave variable value being overwritten before the user program has had a chance to process it, the Siemens driver can be made to delay the write and its acknowledgement using the write protection facilities described previously. If write protection is on when a write is received, the write and its acknowledgement will be delayed until protection is turned off as shown in Figure 3-29

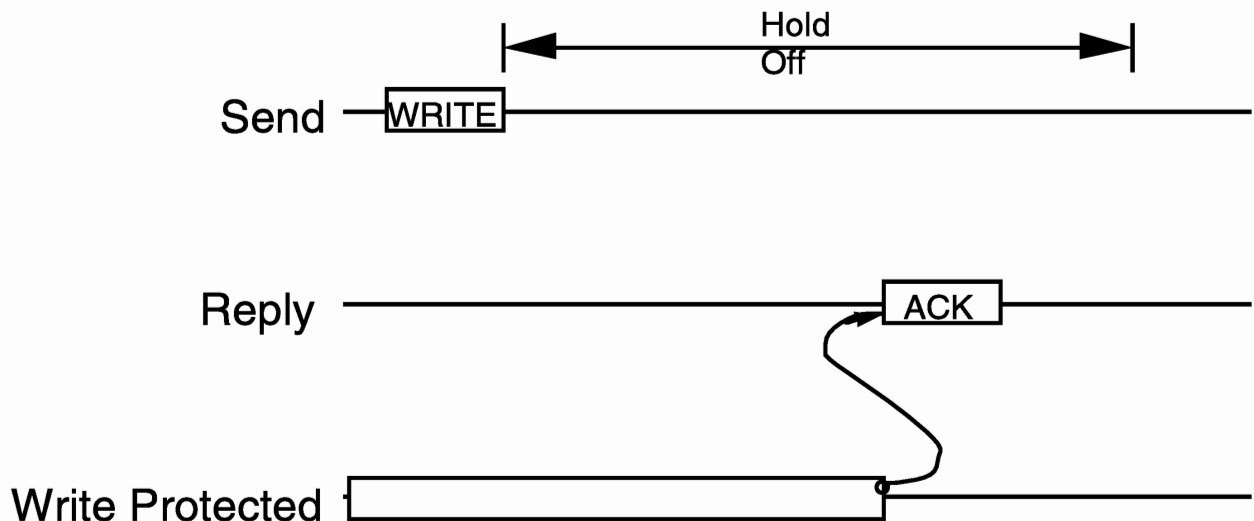


Figure 3-29 Writing Delayed With Write Protection Is On

If the write protection is not removed before the **Hold Off** time is exceeded an error telegram will be returned and the write aborted. See Figure 3-30.

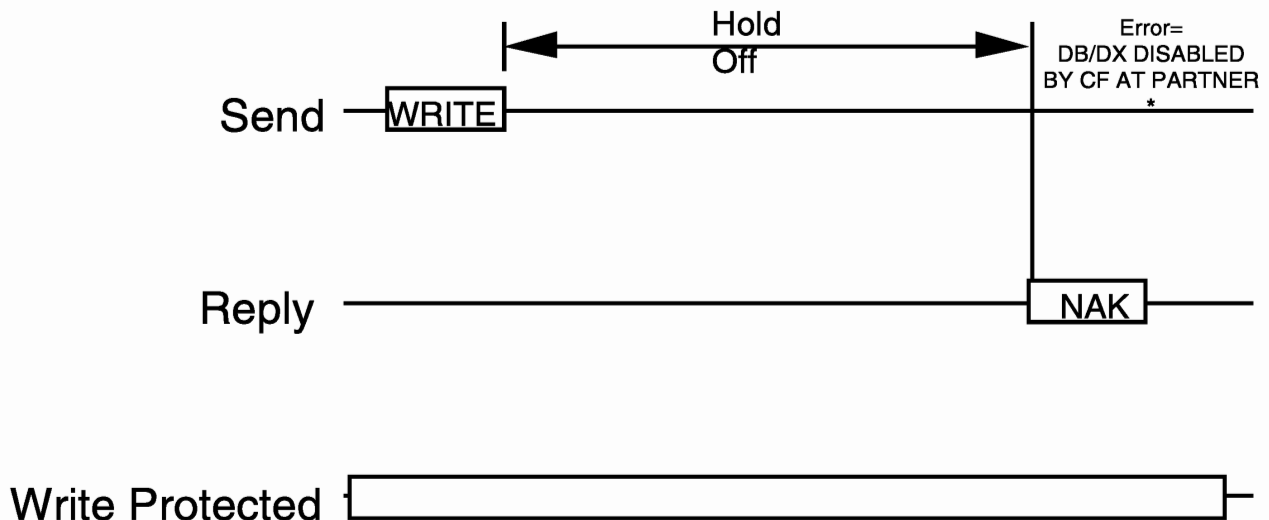
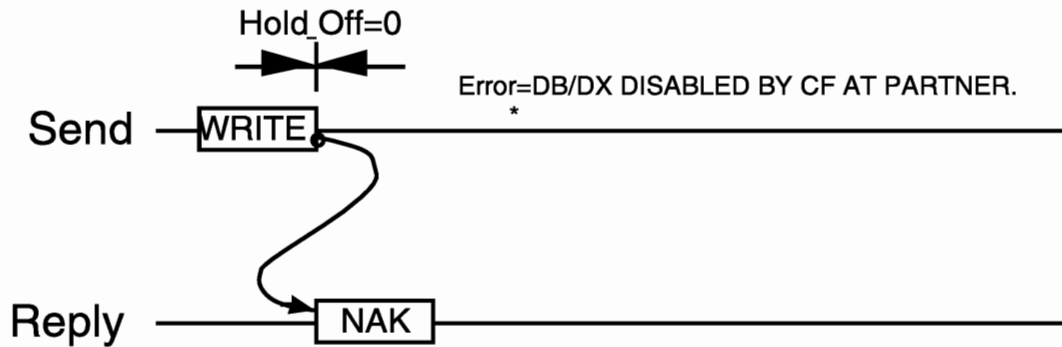


Figure 3-30 Writing Fails Due To Write Protection Being On

The **Hold Off** parameter in the driver function block should be set to the maximum time needed to process the received data. This will ensure that, in the case where write protection remains on, the sender will be informed as soon as possible that the parameter is protected. If it is not necessary to hold off writes for a given application the **Hold Off** can be set to zero in which case writes to protected parameters will be immediately reported back to the sender as an error. This situation is shown in Figure 3-31.



## Write Protected

Figure 3-31 Writing With Hold\_Off Of 0 And Write Protection On

It is important to note that there is an upper limit imposed on the **Hold\_Off** by the senders time out. The sending half of the 3964(R) procedure specifies a 'Monitoring Time' after which the telegram is assumed to have been lost. This monitoring time varies depending on baud rate and is shown in Table 3-41

Baud Rate	Monitoring Time	Recommended Maximum Hold_Off
75	40s	38s 400ms
300	10s	9s 500ms
600	7s	6s 700ms
1200	5s	4s 800ms
2400	5s	4s 800ms
4800	5s	4s 800ms
9600	5s	4s 800ms
19200	5s	4s 900ms
38400	5s	4s 900ms
57600	5s	4s 900ms
115200	5s	4s 900ms

Table 3-39 Monitoring Times

Figure 3-32 shows the effect of setting the **Hold\_Off** greater than the monitoring time .

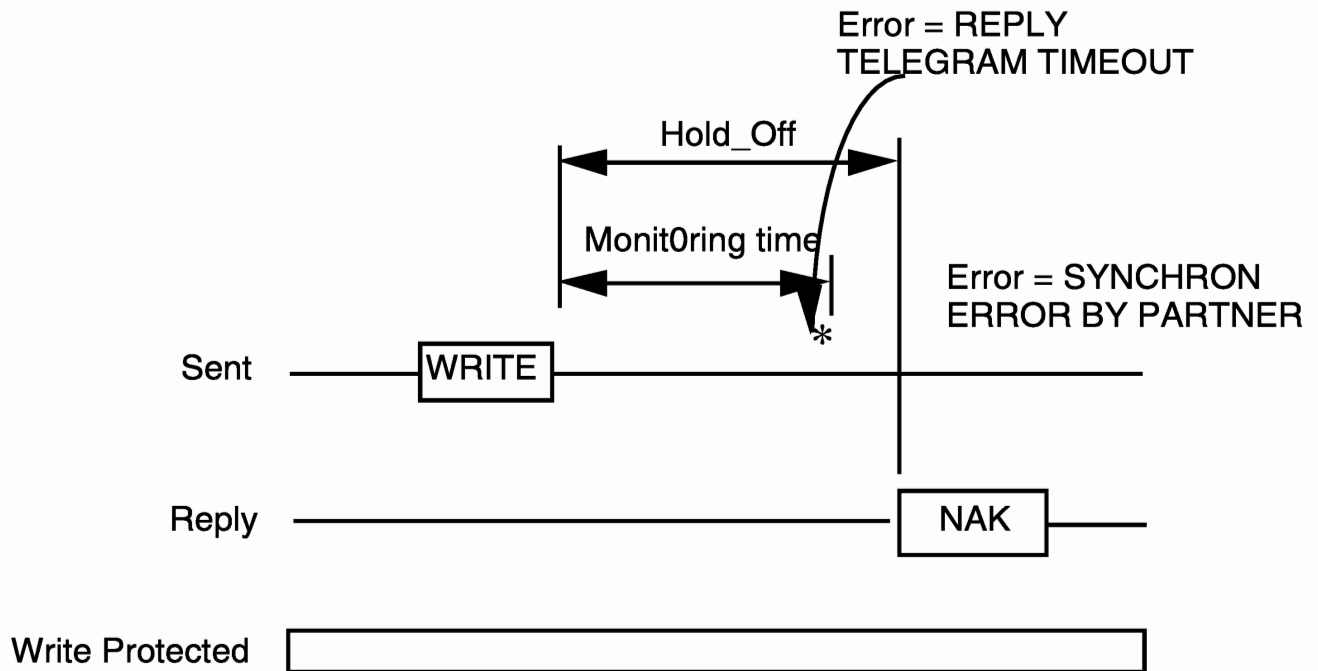


Figure 3-32 Writing With Hold\_Off Greater Than The Monitoring Time

**Note 1:-** That two errors will be logged by the sender in this situation if write protect remains on. The first error will indicate that the send transaction has timed out. The second error will occur because when the reply telegram finally arrives there will be no send telegram to match it up with (because the send will be assumed to have aborted). It is recommended that the **Hold\_Off** be set as low as possible and always less than the recommended maximum shown in Table 3-41 This will leave some leaway for latency and to allow time for the reply telegram to reach the sender.

**Note 2:-** That latency time is not shown on Figures 3-42 to 3-45. For example in Figure 3-42 the acknowledgment telegram will, in fact, not be returned immediately after the send telegram is received; there will be a small processing delay.

## Example

This section considers an example application. The full sequence programming of the application will not be shown for clarity but the communications function blocks will be discussed in detail.

In this example application a Siemens PLC is being used to control a machine. This machine controller accepts commands written into a data word (say DW 70.1) on the PLC and updates a status word (say DW 71.9) which can be read by the PC3000. It also sends unsolicited alarm codes to a data word (say DW 50.0) within the PC3000. When first run the controller also reads a 6 character batch name from the PC3000 (say DW 51.0-2). The exact workings of the machine are not relevant to the example. Tables 3-40 -3-41 and -3-42 show

the machine command codes, status bits and alarm codes assumed for this example.

<b>Command</b>	<b>Code</b>
run	1
hold	2
continue	3
stop	4
reset	5

Table 3-40 Example Machine Application Command Codes

<b>Status</b>	<b>Bit Number</b>
running	0
holding	1
manual/auto	2
alarm active	3
battery warning	4
tolerance exceeded	5
m/c failed	6
emergency stop	7
fire !	8

Table 3-41 Example Machine Application Status Bits

<b>Alarm Type</b>	<b>Code</b>
tolerance exceeded	1
m/c failure	2
emergency stop	3
manual override	4
fire!	5

Table 3-42 Example Machine Application Alarm Codes

It is assumed that port B on the LCM is linked to the Siemens PLC. The Siemens will be assumed to have been programmed to expect the 3964R procedure with a baud rate of 9600 baud and to have its priority set to high. The priority of the Siemens would be set to high in order that alarm messages

from the machine take priority over commands from the PC3000. The Siemens\_M\_S function block would be set up as shown in Table 3-43 for this application.

Parameter	Value	Comments
Port	'0B'	'0' means slot 0 in the rack (the LCM) and 'B' means port B.
Baud	_9600 (6)	
Checksum	Yes (1)	This parameter is used to choose between procedure 3964 and 3964R. 3964R, which is used in this example, is 3964 with the addition of a block check character (checksum).
Priority	Low (0)	One end of the link should be Low priority and the other High . The a low priority end of the link will back down in the case of a contention situation arising.
Hold_Off	1s	When an alarm code is received in the user program it should be dealt with well within a second. If it has not been dealt with within 1s and another alarm has occurred it will be signaled back to the PLC as write protected.
Wr_Protect	No (0)	In this application it is never necessary to globally write protect all accesses.

Table 3-43 Example Siemens\_M\_S Function Block Settings

This example application requires four communications parameter function blocks; two remote variables and two slave variables. The remote variable blocks are a Remote\_Int, to send commands, and a Remote\_SW, to fetch the machine status information. The slave variables are a Slave\_Int, to receive the alarm codes, and a Slave\_Str, for the batch name to be read from. Notice the difference between slave and remote variable blocks. The remote variables are associated with the master half of the driver and *fetch* or *send* data. The slave variable blocks are associated with the slave half of the driver and can be *read* or *written* by the partner, i.e. a Siemens PLC.

In order to send the commands to the PLC it is necessary to create a remote variable function block which in this example is called MC\_Cmd. The configuration of this block is shown in Table 3-44.

<b>Parameter</b>	<b>Cold Start Value</b>	<b>Comments</b>
Address	`0B1D70.1'	This address indicates that the parameter should be sent out of port B on the LCM to CPU number 1, DW 70.1.
Mode	Demand	Demand mode is chosen as the commands will only need to be sent when the PC3000 sequence program requires.
Trig_Read	Off (0)	This parameter is not used for this application as it is not necessary to read this address. The parameter can stay at its default value of Off .
Trig_Write	Off (0)	This parameter is not used for this application and can be left at its default value of Off . In this application the sequence program will be controlling the writing of the commands which means the State parameter is a more convenient way to initiate the writing of commands.
Refresh	10s	As Demand mode is used for this application this parameter is not used and can be left at its default value of 10s.
New_Value	-	This is the value to be sent to the PLC to represent a machine command.  Note- That only the least significant 16 bits will be sent or read from the Siemens as one word block is specified.
State	Ok (0)	The State parameter is written to by the sequence program to initiate the sending of the command.

Table 3-44 MC\_Cmd Remote\_Int Function Block Cold Start Values

In order to continuously poll the status of the machine from the Siemens PLC a Remote\_SW function block instance called MC\_Stat is set up within the PC3000 as in Table 3-45 It is assumed that the status only needs to be updated at a rate of once every second on the PC3000.

<b>Parameter</b>	<b>Cold Start Value</b>	<b>Comments</b>
Address	'0B1D71.9'	This address indicates reading of CPU 1, DW 71.9 from the device connected to port B on the LCM. This is the address of the status word in the Siemens PLC.
Mode	R_Cont (1)	R_Cont mode is chosen to read the status information from the PLC at regular intervals.
Trig_Read	Off (0)	This parameter is not used for this application as the R_Cont mode is used to trigger the reading. The parameter can be left at its default value of Off .
Trig_Write	Off (0)	This parameter is not used for this application and can be left at its default value of Off .
Refresh	1s	A polling rate of 1 second will be assumed sufficient for this application.
New_Value	Off	As the machine status is not going to be written the New_Value s will not be used and can remain at their default values.
State	Ok (0)	The State parameter need not be initialised when a parameter using the read continuous mode to control the polling. The State can be left at the default value and will be controlled by the function block to initiate the reads.

Table 3-45 MC\_Stat Remote\_SW Function Block Cold Start Values

The alarms from the Siemens PLC will be received by the slave half of the driver. To control this a Slave\_Int function block is generated which has been called MC\_Alarm. The cold start values for this are shown in Table 3-46.



<b>Parameter</b>	<b>Cold Start Value</b>	<b>Comments</b>
Address	'SI1D50.0'	This address indicates that the block simulates a Siemens address CPU 1, DW 50.0.
Trig_Wr1	Off (0)	The Trig_Wr1 parameter is designed for convenience when using wiring to control the write protection of the parameter. In the case of this application alarm handling will be done by the sequence program so this parameter will be left at its default value of Off .
Mode	Wr_Once (2)	The Mode will initially be Wr_Once. This enables one write but following that it changes the mode to Rd_Only. For a full explanation of the operation see the text.
Changed	No (0)	The Changed flag is initially defaulted to No so that it is possible to see when an alarm is received by the slave.

Table 3-46 MC\_Alarm Slave\_Int Function Block Cold Start Values

The batch name will be read from a slave string block in the PC3000 called Batch\_Nm. The cold start values for this block are shown in Table 3-47.

<b>Parameter</b>	<b>Cold Start Value</b>	<b>Comments</b>
Address	'SI1D51.0+3'	This address indicates that the block simulates Siemens addresses CPU 1, DW 51.0-2. Note:- That three words can hold six ASCII characters.
Trig_Wr1	Off (0)	The Trig_Wr1 parameter is designed for convenience when using wiring to control the write protection of the parameter. In the case of this application the batch number will be handled by the sequence program so this parameter will be left at its default value of Off.
Mode	Rd_Only (1)	The batch name is only read by the machine, not written so the Mode should be set to Rd_Only for maximum security.
Changed	No (0)	The Changed flag is of no interest as the parameter is read only so this flag can be left at its default of No .
Value	"	The Value will be filled in with the batch name by the sequence program before the machine is set to run.

Table 3-47 Batch\_Nm Slave\_Str Function Block Cold Start Values

The **Value** parameter of the Batch\_Nm will be filled in with the appropriate string by the sequence program prior to sending the run machine command.

### Controlling The Communications

The communications function blocks need to be controlled to a certain extent by sequence programming. This is outlined here although the full application side of the sequence program is not.

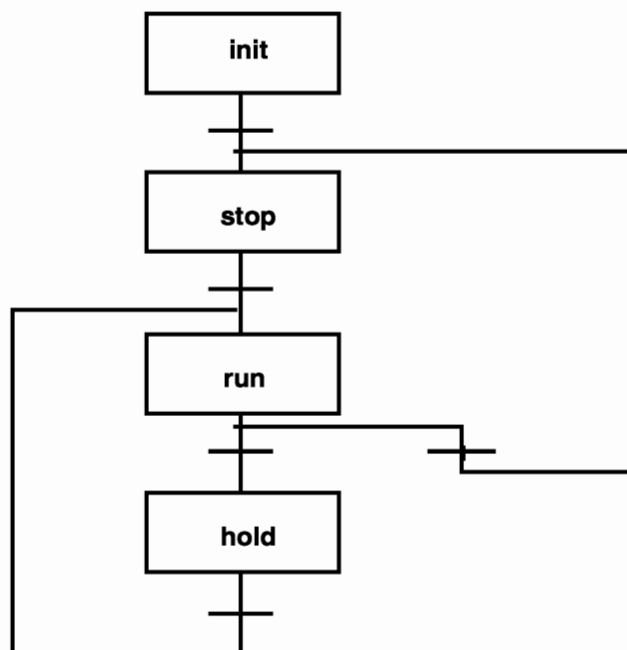


Figure 3-33 Main SFC For The Example Application

The Sequential Function Chart (SFC) for the main control sequencing is shown in Figure 3-46 with the Structured Text (ST) as follows: .

```

STEP  init :
    MC_Cmd.New_Value := 5 ; (*reset*)
    MC_Cmd.State := 3 (*Write*) ;
END_STEP

```

```

TRANSITION
    FROM  init
    TO    stop
:= MC_Cmd.State = 0 (*Ok*)
END_TRANSITION

```

```

STEP  stop :
    MC_Cmd.New_Value := 4 ; (* stop *)
    MC_Cmd.State := 3 (*Write*) ;
END_STEP

```

```

TRANSITION
    FROM  stop
    TO    run
:= ( MC_Cmd.State = 0 (*Ok*) ) AND
   ( run_proc.Val = 1 (*On*) ) ;
   (* run -> On to start process *)

```

```
END_TRANSITION

STEP run :
    Batch_Nm.Value := Next_Batch.Val;
    MC_Cmd.New_Value := 1 ; (* run *)
    MC_Cmd.State := 3 (*Write*) ;
END_STEP

TRANSITION
    FROM run
    TO hold
:= ( MC_Cmd.State = 0 (*Ok*) ) AND
    ( hold.Val = 1 (*On*) ) ;
END_TRANSITION

TRANSITION
FROM run
TO stop
:= ( MC_Cmd.State = 0 (*Ok*) ) AND
    ( run_proc.Val = 0 (*Off*) );
END_TRANSITION

STEP hold :
    MC_Cmd.New_Value := 2; (* hold *)
    MC_Cmd.State := 3 (*Write*) ;
END_STEP

TRANSITION
    FROM hold
    TO run
:= ( MC_Cmd.State = 0 (*Ok*) ) AND
    ( hold.Val = 0 (*Off*) ) ;
END_TRANSITION
```

The four states names reflect the commands which are sent to the machine by the ST within the steps. The init step sends a reset command to the machine to initialise it. It starts by setting the **New\_Value** to the command code reset and then the **State** is set to **Write**. The exit transition from this step then waits for the **State** to return to **Ok**, indicating that the value has been written successfully. The stop step is similar except that it does not move on until the stop command has been sent and the run\_proc boolean user variable is **On**. All the other states and transitions work in a similar fashion to one or other of these two. Notice the use of another boolean user variable to control the holding of the process in a similar way to the run\_proc user variable. Error handling has been omitted for clarity. However it is vital that the exit transitions after initiating a remote write should test for **Error** as well as **Ok** in the **State**. The

example shown assumes that the link is perfect and would lock up if an error did occur. In a real application it would be essential to consider the error handling in detail.

## Alarm Handling

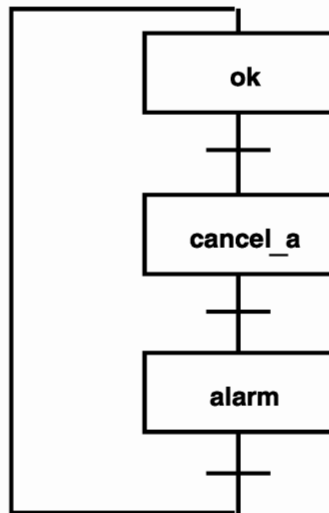


Figure 3-34 Alarm Handling SFC For The Example Application

The Sequential Function Chart (SFC) for the machine alarm handling is shown in Figure 3-34 with the Structured Text (ST) as follows:

```

STEP ok :
END_STEP
TRANSITION
  FROM ok
  TO alarm
:= MC_Alarm.Value <> 0 ;
END_TRANSITION

STEP alarm :
  run.Val := 0 (*Off*) ;(* stop the machine *)
  hold.Val := 0 (*Off*) ;
END_STEP

TRANSITION
  FROM alarm
  TO cancel_a
:= Al_Clr.Val = 1 (*On*) ;
  (* setting Al_Clr.Val On (1) clears the alarm
  and acknowledges it *)
END_TRANSITION

```

```
STEP cancel_a :
    Al_Clr.Val := 0 (*Off*) ;
    MC_Alarm.Value = 0 ;
        (* clear the MC_Alarm slave variable to acknowledge the
           alarm *)
MC_Alarm.Mode = 2 (* Wr_Once *) ;
        (* Re-enable writes ready for next alarm *)
END_STEP

TRANSITION
    FROM alarm
    TO ok
:= 1
END_TRANSITION
```

Because the alarm code is received by the slave it is not necessary to initiate any transactions with the partner. This simplifies the coding considerably. The **ok** step does not contain any ST. It has a simple exit condition which causes the sequence to move on to the **alarm** step if Alarm.Val is non zero.

**Note:-** That the slave variable function block will change from **Wr\_Once** mode to **Rd\_Only** mode to protect against any alarms being missed when this happens.

In the **alarm** step the run and hold booleans are both cleared to halt the process. The alarm is cleared by the boolean **Al\_Clr.Val** being set to **On**. **Al\_Clr.Val** going **On** moves the sequence program on to step cancel\_a which clears **Al\_Clr.Val**, ready for the next alarm to be cleared. This step also sets **MC\_Alarm.Val** to 0 to indicate to the Siemens PLC that the alarm has been noted/dealt with. **MC\_Alarm.Val** will be polled by the Siemens PLC to detect that the alarm has been acknowledged/cleared. Finally **MC\_Alarm.Mode** is reset to **Wr\_Once** in order to re-enable writing of the alarm code by the PLC.

## Status Reporting

The polled status bits may be used in many ways depending on the application but for this example some of them will be displayed using the messages function block. The following wiring will display the run/hold status and alarm status on the messages line of the PC3000 programming station software. The other status bits could be displayed in a similar manner if required.

```
Messages.P_Message := CONCAT( IN1 := SEL_STRING( G :=
    MC_Status.Value_1 IN0 := 'HOLDING ', IN1 :=
    'RUNNING ' ) , IN2 := SEL_STRING
    ( G := MC_Status.Value_2 , IN0 := 'IN AUTO', IN1 :=
    'IN MANUAL' ) );
```

```
Messages.S_Message := SEL_STRING( G := MC_Status.Value_3 , IN0 :=  
    '*** ALARM  
ACTIVE          ***', IN1 := 'Ok' );
```

## Error Reporting

The majority of the error codes have been made to match the SYSTAT errors used by Siemens so that they can be referenced in the COM 525 manual. There is one error which is in the COM 525 manual which does not have a matching error code due to implementation restrictions. The error BREAK (255 (FFh)) what has had to be moved to 127 (7Fh) for the PC3000. There are also some extra errors which are specific to the PC3000 implementation which have codes from 128 to 164 and 255.

Errors can be reported by the Siemens\_M\_S function block in three different places. The errors are reported in the **Error\_No** parameter of the Siemens\_M\_S, slave variable and remote variable function blocks according to the type of error involved. For example if, while sending a Remote\_Str, there is no response from the partner then error 226 (LINK ESTABLISHMENT TIMEOUT) would appear in the **Error\_No** parameters of the Siemens\_M\_S and Remote\_Str function blocks. In general

- If an error can be directly attributed to a specific remote variable it is reported to that parameter.
- If an error is detected which implies a problem with the link it is reported to the driver block.
- Only errors in the **Address** of a slave variable are reported in the slave parameter block.

Table 3-48 shows the location where various types of errors are reported.

Type Of Error And When Detected	Block(s) Reported To		
	Driver	Remote	Slave
An error detected in the link, when it is idle, which cannot be attributed to any parameter. e.g. SYNCHRON ERROR BY PARTNER	•		
When processing a remote variable read/write the reply was garbled. <sup>1</sup> e.g. ERROR IN REPLY TELEGRAM FORMAT.	•		
An error with the Siemens_M_S initialisation when the Siemens_M_S function block first runs. e.g. PORT ERROR ILLEGAL PORT.	•		
BREAK received at any time.	•	.2	
When processing a remote variable read/write an error in the link was detected while sending the request. e.g. ERROR DURING LINK ESTABLISHMENT.	•	•	
When processing a remote variable read/write an error was reported back by the partner in a 'REPTTEL'. e.g. DB/DX ACCESS ERROR AT PARTNER.		•	
An error in the Address parameter of a remote variable was detected when a read/write was initiated by the remote variable block. e.g. CPU NUMBER OUT OF RANGE.		•	
When processing a remote variable read/write no driver was found on the port specified in the remote variable Address. e.g. PORT ERR NO REMOTE PARAM SERV.		•	
An error in the Address parameter of a slave variable was detected when the slave parameter was first initialised. <sup>3</sup> e.g. ADDRESS OUT OF RANGE.			•
When the slave variable was first initialised an overflow of the systems slave variable or driver type tables occurred. e.g. PORT ERROR TOO MANY SLAVE PARAMS.			•

Table 3-48 Error Reporting Locations

<sup>1</sup>The garbled telegram cannot be attributed to the remote variable request in progress because it could have been an incoming request for the slave\_variable.



<sup>2</sup>When a BREAK is detected any remote variable read/writes which are in progress will be terminated with an error. If any further read/writes are initiated before the BREAK is cleared they will be immediately terminated with an error.

<sup>3</sup>slave variables are declared to the driver when they are first run. The driver then encodes and checks their addresses on its first execution after all blocks have been executed once. This means that the SLAVE VARIABLE NOT INITIALISED error code (255) may be seen in the slave variable **Error\_No** for a short time after running the user program. If this error persists it implies that there is no driver of the name declared to deal with the slave variable.

## Error Codes

The error codes listed in Table 3-49 correspond to the SYSTAT codes in chapter 7 of the CP525 manual except for errors 128 to 164 and 255 which are specific to the PC3000. The exception to this is the BREAK error which is 255 in the Siemens SYSTAT and 127 on the PC3000. Note that not all the errors are seen in the driver block. Errors can also be flagged by the driver in the slave and remote variable blocks (see footnotes at the end of the error codes ).

Error_No	Error description
0	OK <ul style="list-style-type: none"> <li>•No error has been detected.<sup>1 3</sup></li> <li>•The last operation was completed without error.<sup>2 3</sup></li> </ul>
33	ERROR IN DATA TYPE <sup>1 2</sup> <ul style="list-style-type: none"> <li>•The data type of the slave/remote variable was set to something other than `D' (data block). At present only the `D' type is supported.</li> </ul>
34	ADDRESS OUT OF RANGE <sup>1 2</sup> <ul style="list-style-type: none"> <li>•The data block number specified in the slave/remote variable Address is greater than 255. The data block number should be 0 to 255.</li> <li>•The data word number specified in the slave/remote variable Address is greater than 255. The data word number should be 0 to 255.</li> <li>•There is an error in the format of the data block/word in the Address parameter of the slave/remote variable.</li> <li>•The block length specified in the slave/remote variable Address is greater than 65535. The block length should be 0 to 65535.</li> <li>•The block end address specified in the slave/remote variable Address is greater than 65535.The block end address should be 0 to 65535.</li> <li>•The block end address specified in the slave/remote variable Address is less than the start address.</li> <li>•There is an illegal character following the data word number in the Address .</li> <li>•The size of block specified is too large for the type of lave/remote block being used.</li> <li>•There are excess characters in the Address.</li> </ul>
36	CPU NUMBER OUT OF RANGE <sup>1</sup> <ul style="list-style-type: none"> <li>•The CPU number specified for the slave/remote variable is not a legal one. It should be 0 to 7.</li> </ul>

Table 3-49 Siemens\_M\_S Error Codes

Error_No	Error description
38	LENGTH TOO GREAT •The block length specified in the remote variable Address is too large for the length of the string New_Value .
41	SYNCHRON ERROR BY PARTNER <sup>3</sup> •A reply telegram arrived, though no request was made or it was not completed.
42	ERROR IN REPLY TELEGRAM FORMAT <sup>3</sup> •1st byte is not 0. Note:- That the driver does not support follow on telegrams so \$FF is illegal.
43	REPLY TO FETCH TOO LONG <sup>3</sup> •The reply to a fetch contains too much data.
44	REPLY TO FETCH TOO SHORT <sup>3</sup> •The reply to a fetch contains too little data.
45	REPLY TO SEND HAS DATA <sup>3</sup> •A reply was received following a send which contained data.
47	REPLY TELEGRAM TIMEOUT <sup>2 3</sup> •No reply telegram came from the partner within the monitoring time after sending a FETCH/SEND telegram.
48	DB/DX DISABLED BY CF (OR WR_PROTECT) AT PARTNER <sup>2</sup> •This error will be reported if writing to another PC3000 Siemens_M_S slave variable which is write protected. •Coordination flags are not supported by the driver at present so this error should never occur when connected to Siemens PLC's.
49	HARDWARE ERROR AT PARTNER <sup>2</sup> <span style="float: right;">e.g. With CP525 as partner:</span> •Partner reports an illegal source/destination type detected. •Partner reports a memory error in partner PC. •Partner reports an error in handshake between CP/CPU at partner. This is only applicable to SIEMENS PCs. •Partner reports PC is in STOP state.

Table 3-49 Siemens\_M\_S Error Codes (continued)

Error_No	Error description
50	MEMORY ACCESS ERROR AT PARTNER <sup>2</sup> e.g. With CP525 as partner: <ul style="list-style-type: none"> <li>• Partner reports wrong area for condition code word.</li> <li>• Partner reports date area does not exist.</li> <li>• Partner reports data area too small (except DB/DX).</li> </ul>
52	COMMAND ERROR AT PARTNER <sup>2</sup> e.g. With CP525 as partner: <ul style="list-style-type: none"> <li>• Partner reports wrong first command letter in telegram header.</li> </ul>
53	COMMAND TYPE ERROR AT PARTNER <sup>2</sup> e.g. With CP525 as partner: <ul style="list-style-type: none"> <li>• Partner reports wrong second command letter in telegram header.</li> </ul>
54	PARTNER DETECTS WRONG TELEGRAM LENGTH <sup>2</sup> e.g. With CP525 as partner: <ul style="list-style-type: none"> <li>• Partner reports that the length encoded in the header does not agree with the actual telegram length read.</li> </ul>
55	PARTNER DETECTS SYNCHRON ERROR <sup>2</sup> e.g. With CP525 as partner: <ul style="list-style-type: none"> <li>• Partner reports that the order of telegrams is wrong.</li> </ul>
56	NO COLD RESTART AT PARTNER <sup>2</sup> e.g. With CP525 as partner: <ul style="list-style-type: none"> <li>• Partner reports that since power up no 'SYNCHRON' HDB has run.</li> <li>• Partner reports that the mode selector is switched to STOP/PGR.</li> </ul>
57	PARTNER SIGNALS SYS COMMAND ILLEGAL <sup>2</sup> e.g. With CP525 as partner: <ul style="list-style-type: none"> <li>• This is an incorrect reaction by the partner. The CP525 never outputs a system command!</li> </ul>
58	UNKNOWN ERROR NUMBER FROM PARTNER <sup>2</sup> <ul style="list-style-type: none"> <li>• An unrecognized error number was received in the reply telegram from the partner.</li> </ul>

Table 3-49 Siemens\_M\_S Error Codes (continued)

<b>Error_No</b>	<b>Error description</b>
64	ERROR IN 1ST COMMAND BYTE <sup>3</sup> <ul style="list-style-type: none"> <li>• 1st command byte is not 0.</li> </ul> Note: - That the driver does not support follow on telegrams so \$FF is illegal.
127	BREAK <sup>2 3</sup> <ul style="list-style-type: none"> <li>• A break (continuous space condition) has been detected on the serial line.</li> </ul>
128	ILLEGAL EVENT <sup>3</sup> <ul style="list-style-type: none"> <li>• An internal error within the block which should never be seen.</li> </ul>
129	ILLEGAL STATE <sup>3</sup> <ul style="list-style-type: none"> <li>• An internal error within the block which should never be seen.</li> </ul>
130	SLAVE ADDRESS OVERLAP <sup>1</sup> <ul style="list-style-type: none"> <li>• Two slave addresses overlap.</li> </ul>
145	PORT ERR NO ADDRESS <sup>4</sup> <ul style="list-style-type: none"> <li>• There are less than two characters in the Port parameter of the function block.</li> </ul>
149	PORT ERR RX BAUD RATE NOT AVAILABLE <sup>4</sup> <ul style="list-style-type: none"> <li>• The receive baud rate requested is not available on this serial port.</li> </ul>
150	PORT ERR TX BAUD RATE NOT AVAILABLE <sup>4</sup> <ul style="list-style-type: none"> <li>• The transmit baud rate requested is not available on this serial port.</li> </ul>
155	PORT ERR ILLEGAL SLOT <sup>4</sup> <ul style="list-style-type: none"> <li>• The slot number selected is not legal. The slot number is the first character of the Port parameter.</li> </ul>
156	PORT ERR ILLEGAL PORT <sup>4</sup> <ul style="list-style-type: none"> <li>• The port number selected is not legal. The port number is the second character of the Port parameter.</li> </ul>
160	PORT ERR NO REMOTE PARAM SERV <sup>2</sup> <ul style="list-style-type: none"> <li>• The port given in the Address parameter of the remote variable function block does not have a driver allocated to it.</li> </ul>

Table 3-49 Siemens\_M\_S Error Codes (continued)

Error_No	Error description
161	PORT ERR PORT IN USE <sup>4</sup> •The selected Port is already in use for another driver.
162	PORT ERR TOO MANY SLAVE PARAMS <sup>1</sup> •Too many slave variables have been declared to the system.
163	PORT ERR TOO MANY DRIVER TYPES <sup>1</sup> •Too many slave driver types have been declared to the system.
164	ILLEGAL WRITE HOLD OFF TIME <sup>4</sup> •The Hold_Off time selected is too long for this baud rate.
225	ERROR DURING LINK ESTABLISHMENT <sup>2 3</sup> •After an <STX> was sent a non <DLE> or <STX> was received.
226	LINK ESTABLISHMENT TIMEOUT <sup>2 3</sup> •After sending <STX> the partner did not respond within the reply timeout.
227	ABORTED BY PARTNER <sup>2 3</sup> •A <NAK> or other character was received while sending causing the send to be aborted.
228	ERRORS AT END OF LINK <sup>2 3</sup> •After sending a telegram it was rejected by the partner with a <NAK> or other non <DLE> character.
229	TIMEOUT AT END OF LINK <sup>2 3</sup> •After sending link termination <DLE> <ETX> **BCC** no acknowledgement was received from the partner within the timeout.
241	LINK TERMINATION ERROR <sup>3</sup> •A character (other than <NAK> or <STX> was received after link termination i.e while the line was idle.
242	LOGICAL ERROR WHILE RECEIVING <sup>3</sup> •An illegal character was received following a <DLE> Only a <DLE> or <ETX> are allowed to follow a <DLE>
243	CHARACTER TIME OUT •Rx buffer not freed (i.e. no EOT)

Table 3-49 Siemens\_M\_S Error Codes (continued)

Error_No	Error description
244	BCC ERROR <sup>3</sup> <ul style="list-style-type: none"> <li>•The block check character (BCC) does not agree with the internally calculated value. This is only applicable when Checksuming is on.</li> </ul>
246	NO RX BUFFER FREE <sup>3</sup> <ul style="list-style-type: none"> <li>• The receive buffer was not freed up within the timeout. This should never occur and should be reported as a bug if it does.</li> <li>•A character was received after an &lt;STX&gt; but before the link establishment was acknowledged by a &lt;DLE&gt;</li> <li>•The receive buffer has overflowed while building up a received message.</li> </ul>
254	TRANSMISSION ERROR <sup>2 3</sup> <ul style="list-style-type: none"> <li>•An error has been detected while receiving a character. e.g. Framing, parity or overrun errors have been detected.</li> </ul>
255	SLAVE VARIABLE NOT INITIALISED <sup>1</sup> <ul style="list-style-type: none"> <li>•The slave variable has not yet been initialised.</li> <li>•There is no driver block in the user program which has the name given in the slave variable Address.</li> </ul>

Table 3-49 Siemens\_M\_S Error Codes

<sup>1</sup>This error may be reported in a slave variable block when the user program is first run.

<sup>2</sup>This error may be reported in a remote variable block when a read or write is executed.

<sup>3</sup>This error may be reported in the Siemens\_M\_S function block at any time.

<sup>4</sup>This error may be reported in the Siemens\_M\_S function block when it is first run.

## Parameter Attributes

Name	Type	Cold Start	Read Access	Write Access	Type Specific Information	
Port	STRING	'0A'	Oper	Config		
Baud	ENUM	_9600	Oper	Config	Enumerated Values	_75(0) _ _300(1)* _600(2)* _1200(3)* _2400(4)* _4800(5)* _9600(6)* _19200(7)* _38400(8) _57600(9) _115200(10) * Rates supported by Siemens
Checksum	BOOL	No	Oper	Super	Senses	No(0) Yes(1)
Priority	BOOL	Low	Oper	Super	Senses	Low(0) High(1)
Hold_Off	TIME	0s	Oper	Config	High Limit Low Limit	40s 0s
Wr_protect	BOOL	No	Oper	Super	Senses	No(0) Yes(1)
Status	BOOL	NOGO	Oper	Block	Senses	NOGO(0) Go(1)
Error_No	SINT	0	Oper	Block	High Limit Low Limit	255 0
Queue_Space	SINT	0	Oper	Block	High Limit Low Limit	255 0

Table 3-50 Siemens\_M\_S Parameter Attributes



## JBus\_M FUNCTION BLOCK

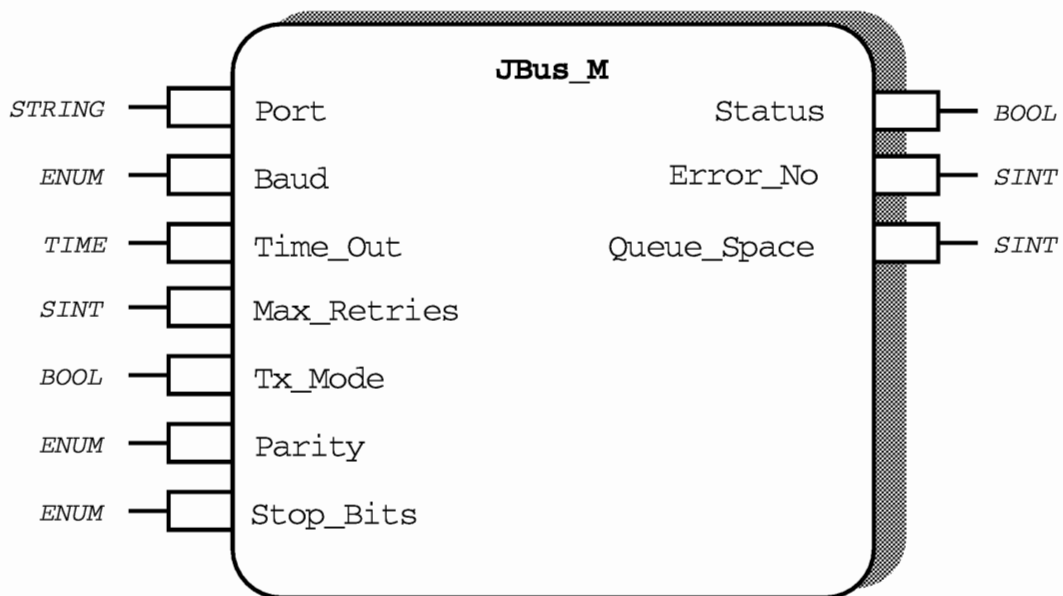


Figure 3-35 JBus\_M Function Block Diagram

## Functional Description

JBus\_M Function Block supports serial communications on a designated serial communications port using the JBus protocol in master mode. The Function Block also supports the Modbus protocol which is identical to JBus except slave locations are referenced with an offset of one. Detailed knowledge of the JBus protocol is not normally required to use this function block. However, you may refer to the Gould Modbus Protocol Reference Guide produced by Gould Inc. Programmable Control Division for specific details if required. An outline of the JBus protocol is included in the JBus\_S function block section. See 'JBus Protocol Reference on page 3-180. Before reading this description, you are advised to gain a general understanding of the PC3000 communications system by reading the PC3000 Communications Overview.

This function block will be required when designing or programming the PC3000 to use a JBus or Modbus interface that functions in master mode, i.e. the PC3000 is connected to one or more JBus slave devices via a serial link.

The Jbus\_M function block provides an JBus master driver and deals with the protocol specific details of the serial communication. It is supported by generic Remote Variable function blocks which are linked to the driver by means of a protocol specific address. The Remote Variable blocks use the master driver Function Block to initiate reads from and writes to remote devices. For more details of the use of Remote Variable function blocks refer to the 'PC3000 Communications Overview'. Further information on the Jbus protocol is given in chapter 3 in the JBus\_S function block description.

## Function Block Attributes

Type:..... 8 75  
Class: ..... COMMS  
Default Task: ..... Task\_1  
Short List: ..... Port Status Queue\_Space  
Memory Requirements:..... 1992 Bytes  
Execution Time: .....23 $\mu$  Secs

## Parameter Description

### Driver Configuration Parameters

The JBus\_M block has a number of configuration input parameters which define various aspects of the driver and should be set before the user program is run. Changing these parameters while the user program is executing will have no effect on the driver, except under special circumstances - see 'PC3000 Communications. Overview' section 'Temporarily changing configuration parameters'.

#### Port

The **Port** parameter is the two character address of the port on which the JBus protocol is to run. The first character is a number from 0 to 5 representing the rack slot and the second character is a letter representing the port within that slot. e.g. '0C' would be port C on the LCM and if there were an ICM in slot 3 '3A' could refer to its top port.

#### Baud

The **Baud** parameter gives a choice of 11 different rates from 75 baud up to 115.2 kbaud (as shown in Table 3-51) with a default of 9600 baud.

**Note:** Not all ports will be able to support all baud rates. This will be indicated by an error when the function block is first run (see description of **Error\_No**).

Enum Value	Baud Rate
0	75
1	300
2	600
3	1200
4	2400
5	4800
6	9600
7	19200
8	38400
9	57600
10	115200

Table 3-51 JBus\_M Baud Rates

#### Parity

The **Parity** parameter defines the parity used in transmission of characters. It can take the values **None**, **Even**, **Odd**, **Space** or **Mark**.

#### Stop\_Bits

The **Stop\_Bits** parameter defines the number of stop bits used in transmission of characters. Allowable values are **1**, **1.5** and **2**.

#### Time\_Out

The **Time\_Out** parameter specifies the length of time that the JBus\_Master will wait for a response message to a transmitted request. After this time the driver assumes there is a transmission error and the request may be retransmitted or an error returned to the remote parameter block which initiated the request.

#### Max\_Retries

The **Max\_Retries** parameter specifies the number of times that a request will be retransmitted if a transmission error, such as a timeout or a message checksum error, is detected. If a valid response is not received after this number of retries an error is returned to the Remote Variable which initiated the request.

## Driver Status Parameters

The driver status is indicated by three output parameters in the JBus\_M function block.

### Status

The **Status** parameter is a boolean indication of the state of the link controlled by the driver. If there are no problems with the link this parameter reads **Go**, but when an error occurs it changes to **NoGo**. If the **Status** is **NoGo** the **Error\_No** parameter indicates the reason for the problem.

### Error\_No

The **Error\_No** parameter indicates the reason for any errors with the link. If the link **Status** is **Go** then the **Error\_No** will be 0 (OK). For full details of error handling and error codes see Table 3-52 on page 3-166.

### Queue\_Space

The **Queue\_Space** parameter indicates the amount of space left in the queue for Remote Variable transactions. If this reaches zero then it implies that the link bandwidth is not sufficient to cope with the number of Remote Variable requests being made and data will be lost. If this situation arises the parameter polling rates should be reduced.

## Remote Variable Operation

The master requests made by the PC3000 are controlled by one or more Remote Variables. The JBus\_M driver supports Remote\_Bool, Remote\_Real, Remote\_Int, Remote\_Str and Remote\_SW parameters.

## Addressing

It is necessary to set up a protocol specific **Address** in the Remote Variable block which is the address used to access the remote devices. An example address format is shown in Figure 3-21. The port is defined as in the function block **Port** parameter as a rack slot number followed by a letter for the port within that slot.

The protocol specific part of the address begins with a slave id which specifies the address of the slave device which should respond to a request. This is a two digit hexadecimal number in the range 0 to 247. A slave id of zero specifies broadcast mode where all slaves are addressed. This is followed by a four digit address which identifies a starting address in the selected slave device. The next character (upper or lowercase 'I') is optional and specifies whether the location addressed is accessed as an input only. The specification of an input only address is important as it can select a different address space in the slave device. The last characters are data format characters and specify for each type of Remote Variable how the data should be interpreted.



Figure 3-36 An Example Remote Variable Address

### Remote Bool Parameter

The address string of the block has the following format characters.

**B** - Bit addressing mode (1 bit).

**R** - Register addressing mode (1 word).

Examples

**1000** - Default of bit addressing mode.

**1000B** - Bit addressing mode.

**1000R** - Register addressing mode.

Boolean values are encoded as follows.

**Bit Mode** - A boolean value of zero is converted to a bit result of zero. A boolean value of one is converted to a bit result of one.

**Word Mode** - A boolean value of zero is converted to a word result of zero. A boolean value of one is converted to a non-zero word result.

### Remote Real Parameter

The address string of the block has the following format characters.

**En** - Register addressing, length = 1, exponent mode. Optional n = -9..9.

**Llow,high** - Register addressing, length = 1, limits mode, low < high.

Examples

**3000** - Default of exponent mode, index = 0.

**3000E** - Exponent mode, index = 0.

**3000E-4** - Exponent mode, index = -4.

**3001E+2** - Exponent mode, index = 2.

**3001L1.5,3.5** - Limits mode, low = 1.5, high = 3.5.

**3002L-4.0,+4.0** - Limits mode, low = -4.0, high = 4.0.

Real values are encoded as follows.

**Exponent Mode** - Word = integer value of Real \* 10<sup>index</sup>

Limits Mode - Word = integer value of  $\frac{\text{Real-Low}}{\text{High-Low}} * \text{FFFFh}$

#### Remote Int Parameter

The address string of the block has the following format characters.

**Bn** - Bit addressing mode. Optional n = 1..32.

**Rn** - Register addressing mode. Optional n = 1 or 2.

#### Examples

**2000** - Default of register addressing mode, length 1 word.

**2000B** - Bit addressing mode, length 1 bit.

**2000R** - Register addressing mode, length 1 word.

**2001B12** - Bit addressing mode, length 12 bits.

**2001R2** - Register addressing mode, length 2 words.

Integer values are encoded as follows.

**Bit Mode** - A byte is formed by extracting the required number of bits from the double integer. The resultant byte is padded with zero bits if necessary.

**Word Mode** - No conversion necessary. If double integer defined as one word long then least significant word is used.

#### Remote String Parameter

The address string of the block has the following format characters.

**Bn** - Bit addressing mode. Optional n ≤ maximum length of string in bits.

**Rn** - Register addressing mode. Optional n ≤ maximum length of string in words.

#### Examples

**4000** - Default of register addressing mode, length 1 word, 2 characters.

**4000B** - Bit addressing mode, length 1 bit.

**4000R** - Register addressing mode, length 1 word, 2 characters.

**4001B12** - Bit addressing mode, length 12 bits.

**4001R6** Register addressing mode, length 6 words, 12 characters.

String values are encoded as follows.

**Bit Mode** - A byte is formed by extracting the required number of bits from the string. The resultant byte is padded with zero bits if necessary.

**Word Mode** - Words are encoded in integer type format with most significant byte first and least significant byte second.

i.e. Word = Byte\_0<sub>hi</sub> Byte\_1<sub>lo</sub> .

Block reads and writes are only possible using remote string parameters where the data will be stored as the 16 bit words or bits packed into bytes. It is therefore only suitable for character or integer storage unless conversion to another data type is performed first.

## Error Reporting

The following errors are reported by the JBus\_M function block

Error_No	Error description
145	PORT ERR NO ADDRESS •There are less than two characters in the Port parameter of the function block.
149	PORT ERR RX BAUD RATE NOT AVAILABLE •The receive baud rate requested is not available on this serial port.
150	PORT ERR TX BAUD RATE NOT AVAILABLE •The transmit baud rate requested is not available on this serial port.
155	PORT ERR ILLEGAL SLOT •The slot number selected is not legal. The slot number is the first character of the Port parameter.
156	PORT ERR ILLEGAL PORT •The port number selected is not legal. The port number is the second character of the Port parameter.
160	PORT ERR NO REMOTE PARAM SERV •The port given in the Address parameter of the remote parameter function block does not have a suitable master driver allocated to it.
161	PORT ERR PORT IN USE The selected Port is already in use for another driver.

Table 3-52 JBus\_M Error Codes

## Remote Variable Error Codes

The following errors are reported by Remote Variable function blocks operating with a serial communications port assigned to JBus\_M.

<b>Error_No</b>	<b>Error description</b>
128	ADDRESS STRING TOO SHORT •The address string contained in the Address parameter is too short to be valid.
129	ADDRESS OUT OF RANGE •The address specified in the Address parameter is not in the valid range of 0000 to 9999.
130	INVALID CHARACTER IN ADDRESS STRING •An invalid character has been detected in the data format field of the Address parameter.
131	NUMERIC FORMAT VALUE OUT OF RANGE •A parameter size value specified in the Address parameter is out of range for that data type.
135	INVALID SLAVE ID •A slave id was specified that was not in the valid range of 0 to 247.
136	BROADCAST NOT ALLOWED •A read was attempted using broadcast mode when only writes are allowed.
137	CANNOT WRITE TO INPUT PARAMETER •A write was attempted to a slave location specified as input only in the Address parameter.

Table 3-53 JBus\_M Remote Variable Error Codes



## Parameter Attributes

Name	Type	Cold Start	Read Access	Write Access	Type Specific Information	
Port	STRING	'0A'	Oper	Config		
Baud	ENUM	_9600	Oper	Config	Enumerated Values	_75(0) _300(1) _600(2) _1200(3) _2400(4) _4800(5) _9600(6) _19200(7) _38400(8) _57600(9) _115200(10)
Time_Out	TIME	1s	Oper	Super	High Limit Low Limit	24day 100ms
Max_Retries	SINT	2	Oper	Super	High Limit Low Limit	1000 0
Tx_Mode	BOOL	RTU	Oper	Config	Senses	RTU(0) Ascii(1)
Parity	ENUM	NONE	Config	Config	Enumerated Values	EVEN(0) ODD(1) SPACE(2) MARK(3) NONE(4)
Stop_Bits	ENUM	_1	Config	Config	Enumerated Values	_1(0) _1_5(1) _2(2)
Status	BOOL	NoGo	Oper	Block	Senses	NOGO(0) Go(1)
Error_No	SINT	0	Oper	Block	High Limit Low Limit	255 0
Queue_Space	SINT	0	Oper	Block	High Limit Low Limit	255 0

Table 3-54 JBus\_M Parameter Attributes

## JBUS\_S FUNCTION BLOCK

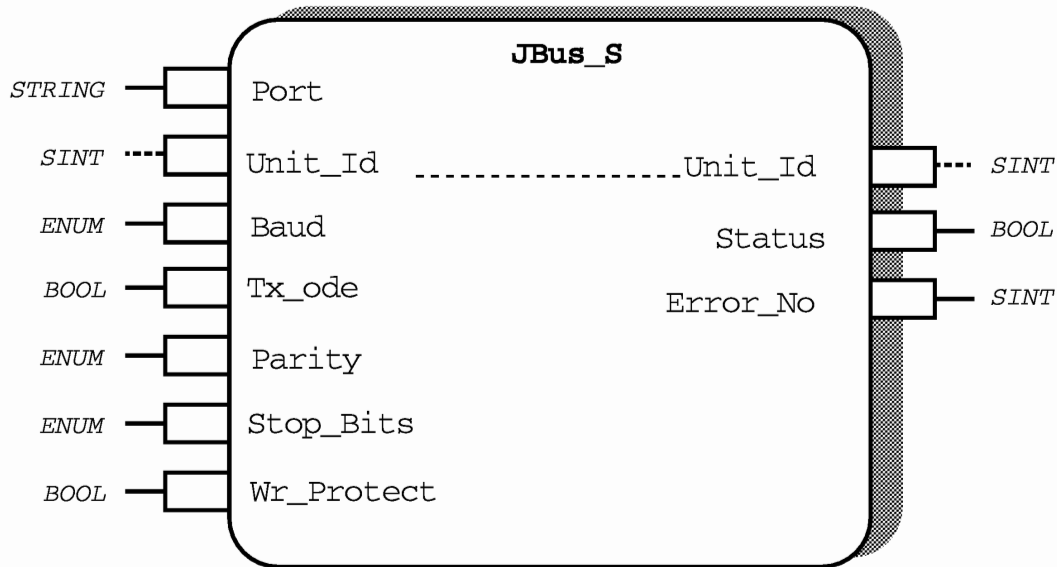


Figure 3-37 JBus\_S Comms Diagram

### Functional Description

This document details the JBus protocol slave driver for the PC3000 and the associated function block.

**Note:-** That the Modbus protocol is also supported by this driver; Modbus is identical to JBus except that slave locations are referenced with an offset of one.

You are advised to gain a general understanding of the PC3000 communications system as given in the 'PC3000 Communications Overview' prior to reading this document.

This function block will be required when designing or programming the PC3000 to use a JBus or Modbus interface that functions in slave mode, i.e. the PC3000 is connected to a JBus master device via a serial link.

The JBus\_S function block provides a driver for JBus protocol that enables a PC3000 serial port to function as a JBus slave. The driver block deals with the protocol specific details of the communications and is supported by generic Slave Variable function blocks. The Slave Variable blocks are linked to the driver by means of a protocol specific address and define values which can be read or written by a remote device using the protocol type specified in the address parameter, in this case 'JB' for JBus. Parameter values can be placed in either a register address space or a separate bit address space if appropriate. For more details of the Slave Variable blocks see the 'PC3000 Communications Overview'

## Function Block Attributes

Type:..... 8 80  
 Class: .....COMMS  
 Default Task: .....Task\_1  
 Short List: .....Port Status Wr\_Protect Unit\_Id  
 Memory Requirements: .....1706 Bytes  
 Execution Time: .....17  $\mu$  Secs

## Parameter Descriptions

### Driver Configuration Parameters

The JBus\_S block has several configuration input parameters which define various aspects of the driver and should be set prior to running the user program. Changing these parameters while the user program is executing will have no effect on the driver except under special circumstances - see 'PC3000 Communications, Overview' section 'Temporarily changing configuration parameters'. The only input parameter which can affect the block whilst running is **Wr\_Protect** .

### Port

The **Port** parameter is the two character address of the port on which the JBus protocol is to run. The first character is a number from 0 to 5 representing the rack slot and the second character is a letter representing the port within that slot. e.g. '0C' would be port C on the LCM and if there were an ICM in slot 3 '3A' could refer to its top port.

### Baud

The **Baud** parameter gives a choice of 11 different rates from 75 baud up to 115.2 kbaud (as shown in Table 3-55) with a default of 9600 baud.

**Note:-** That not all ports will be able to support all baud rates. This will be indicated by an error when the function block is first run (see description of **Error\_No** ).

Enum Value	Baud Rate
0	75
1	300
2	600
3	1200
4	2400
5	4800
6	9600
7	19200
8	38400
9	57600
10	115200

Table 3-55 JBus\_S Baud Rates

#### Parity

The Parity parameter defines the parity used in transmission of characters. It can take the values Even, Odd, Space or Mark

#### Stop\_Bits

The **Stop\_Bits** parameter defines the number of stop bits used in transmission of characters. Allowable values are 1, 1.5 and 2.

#### Unit\_Id

The **Unit\_Id** parameter specifies the JBus address to which the PC3000 will respond. The range is 0 to 247.

If the **Unit\_Id** is zero, a hardware selected Identifier is used. This is set by either links /rotary switch on the LCM or a rotary switch on an ICM depending upon which module the port is on.

LCM Links	ICM Switch	Unit_Id
0000	0	1
0001	1	16
0010	2	31
0011	3	46
0100	4	61
0101	5	76
0110	6	91
0111	7	106
1000	8	121
1001	9	136
1010		151
1011		166
1100		181
1101		196
1110		211
1111		1111

Table 3-56 JBus Hardware Selected Unit\_Id

### Tx\_Mode

The **Tx\_Mode** parameter specifies the type of encoding used to send and receive JBus messages and can take the values `RTU` or `Ascii`. The difference between the two modes is that `Ascii` format messages use printable characters and are approximately twice the size of `RTU` format messages which use binary coding. The message content in each mode is identical.

### Wr\_Protect

The **Wr\_Protect** parameter when set inhibits all JBus writes. If a write operation is attempted when write protection is on then an error response message is generated by the driver.

### Driver Status Parameters

The driver status is indicated by two output parameters in the `JBus_S` function block.

### Status

The **Status** parameter is a boolean indication of the state of the link controlled by the driver. If there are no problems with the link this parameter reads `Go`, but when an error occurs it changes to `NoGo`. If the **Status** is `NoGo` the **Error\_No** parameter indicates the reason for the problem.

### Error\_No

The **Error\_No** parameter indicates the reason for any errors with the link. If the link Status is `Go` then the Error\_No will be 0 (OK). For full details of error codes see section Error Reporting.

Instances of Slave Variable function blocks are created within the PC3000 to define parameters that may be read and written by a remote JBus device. Currently most base data types and a few array data types are supported.

All of the slave variable block types have the following parameters :

**Address** protocol and address to access parameter (see later for details).

**Value** an Input/Output parameter that contains the parameter value.

**Mode** an Input/Output parameter to select read only, read and write, and write once modes.

**Trig\_Wr1** on a rising edge causes the Mode to change to the write once mode.

**Status** output boolean to indicate block is operational.

**Error\_No** output to indicate the reason for the block not being operational.

Slave Variable Operation Data is written to and read from the PC3000 via Slave Variables.

### Addresses

The address parameter consists of a two character protocol identifier followed by protocol specific location and format information. For example,

Address String	Protocol Identifier	Location	Format
'JB1234R1'	JB	1234	R1

A Slave Variable may be accessed via any communications port configured to support the protocol selected by the Protocol Identifier part of the address, in this case 'JB' is used to select the Jbus\_S driver function block.

The location part of the address is assigned a decimal value in the range 0000-3999, (4000-9999 are reserved for future access to system parameters). Since JBus has two address spaces, Bit and Register, the location only needs to be unique in each of these spaces. Depending upon the variable type and format selected, a variable may extend across several locations. For example a Slave String function block (Slave\_Str) with a location of 0000 and a format of B64

would occupy locations 0000-0063. Note that overlapping addresses will cause one or more Slave Variables to enter an error state.

Consecutive variables may be accessed using a block read or write. Attempts to read or write across 'gaps' between variables will cause an error to be returned on JBus. A block read or write may also access a part of a variable if the variable is more than one location long.

Whenever the Slave Variable is put into a write once mode, the next write to it via JBus will cause it to change to read only. This protects the value until the PC3000 user program can acknowledge that it has seen it by changing the mode back to 'write once'. This is described in more detail in the 'PC3000 Communications Overview' document.

The value of a Slave Variable is encoded into a 16 bit word for register addressing and encoded into a packed byte for bit addressing.

### Slave Bool Parameter

The address string of the block has the following format characters.

**B** - Bit addressing mode (1 bit).

**R** - Register addressing mode (1 word).

Examples

**1000** - Default of bit addressing mode.

**1000B** - Bit addressing mode.

**1000R** - Register addressing mode.

Boolean values are encoded as follows.

**Bit Mode** - A boolean value of zero is converted to a bit result of zero. A boolean value of one is converted to a bit result of one.

**Word Mode** - A boolean value of zero is converted to a word result of zero. A boolean value of one is converted to a non-zero word result.

### Slave Real Parameter

The address string of the block has the following format characters.

**E<n>** - Register addressing, length = 1, exponent mode. Optional n = -9..9.

**L<low>,<high>** - Register addressing, length = 1, limits mode, low < high.

**S IEEE** - Single Precision Floating Point (2 registers)

Examples

**3000** - Default of exponent mode, index = 0.

**3000E** - Exponent mode, index = 0.

**3000E-4** - Exponent mode, index = -4. **3001E+2** - Exponent mode, index = 2.

**3001L1.5,3.5** - Limits mode, low = 1.5, high = 3.5.

**3002L-4.0,+4.0** - Limits mode, low = -4.0, high = 4.0.

Real values are encoded as follows.

**Exponent Mode** - Word = integer value of Real \* 10<sup>index</sup>

**Limits Mode** - Word = integer value of  $\frac{\text{Real-Low}}{\text{High-Low}} * \text{FFFFh}$

**IEEE** - Single Precision

1st register: sign, exponent and top 7 bits of mantissa.

2nd register: bottom 16 bits of mantissa

e.g. float = 1.001

1st reg = 3F80 (hex)

2nd reg = 20C5 (hex)

### Slave Int Parameter

The address string of the block has the following format characters.

**Bn** - Bit addressing mode. Optional n = 1..32.

**Rn** - Register addressing mode. Optional n = 1 or 2.

Examples

**2000** - Default of register addressing mode, length 1 word.

**2000B** - Bit addressing mode, length 1 bit.

**2000R** - Register addressing mode, length 1 word.

**2001B12** - Bit addressing mode, length 12 bits.

**2001R2** - Register addressing mode, length 2 words.

Integer values are encoded as follows.

**Bit Mode** - A byte is formed by extracting the required number of bits from the double integer. The resultant byte is padded with zero bits if necessary.

**Word Mode** - No conversion necessary. If double integer defined as one word long then least significant word is used.

### Slave String Parameter

The address string of the block has the following format characters.

**Bn** - Bit addressing mode. Optional n ≤ maximum length of string in bits.

**Rn** - Register addressing mode. Optional n ≤ maximum length of string in words.

Examples

**4000** - Default of register addressing mode, length 1 word, 2 characters.

**4000B** - Bit addressing mode, length 1 bit.



**4000R** - Register addressing mode, length 1 word, 2 characters.

**4001B12** - Bit addressing mode, length 12 bits.

**4001R6** - Register addressing mode, length 6 words, 12 characters.

String values are encoded as follows.

**Bit Mode** - A byte is formed by extracting the required number of bits from the string. The resultant byte is padded with zero bits if necessary.

**Word Mode** - Words are encoded in integer type format with most significant byte first and least significant byte second. i.e. Word = Byte\_0<sub>hi</sub> Byte\_1<sub>lo</sub>.

### Slave Array Parameter

The array types supported currently are boolean, integer and real with each containing eight values. They are equivalent to eight single parameters of the same type occurring at consecutive addresses and all sharing the same format specification.

### Address Examples

If the following Slave Variables are created :

Param	Block Type	Address
A	Slave_Bool	JB0000
B	Slave_Str	JB0001B7
C	Slave_Int	JB0008B16
D	Slave_Real	JB0000
E	Slave_Int	JB0001
F	Slave_Str	JB0002R4

The address mapping as seen by JBus would be :

Domain	Location	Param
Bit	0000	A
Bit	0001-0007	B
Bit	0008-0023	C
Register	0000	D
Register	0001	E
Register	0002-0005	F

### Xycom Format Parameters

The PC3000 implementation of the JBus protocol supports 3 additional non-standard parameter types. This is implemented specifically for the formatting of datatypes for communication with a Xycom Terminal but may be used in any application where full compliance with the JBus protocol is not required. The main feature is the use of an extra JBus word to transmit the parameter information.

#### Double Integer (DINT)

A xycom DINT parameter is specified using the character X for the protocol specific field in the parameter address. eg JB1000X.

The parameter value is transmitted using two JBus registers as follows :

$$\text{LS register} = (\text{abs}(i) \text{ MOD } 10000) * 2 + \text{sign\_bit}$$

$$\text{MS register} = \text{abs}(i) \text{ DIV } 10000$$

where sign\_bit is 1 for negative i, otherwise 0.

This format allows a range of -99999999 to +99999999.

#### Floating Point (REAL)

The address field for a xycom REAL parameter is specified as for a xycom DINT. eg JB2000X.

The parameter value is transmitted using two JBus registers as follows :

$$\text{LS register} = (\text{abs}(R_m) \text{ mod } 1000) * 16 + R_e * 2 + \text{sign\_bit}$$

$$\text{MS register} = \text{abs}(R_m) \text{ div } 1000$$

where sign\_bit is 1 for negative, otherwise 0 and  $R_m$  is an integer value and  $R_e$  is an exponent of 10.

This format allows a range of -9999999 to 9999999 with seven significant digits.

#### Duration (TIME)

The address field for a xycom TIME parameter has no special format characters. eg JB3000.

The parameter value is transmitted using two JBus registers as follows :

$$\text{LS register} = T_{sec} * 1000 + T_{millisec}$$

$$\text{MS register} = T_{day} * 10000 + T_{hour} * 100 + T_{min}$$

This allows a range of 0ms to 6d\_23h\_59m\_59s\_999ms.

### Timing Information

The following table gives information on the slave execution times for several different types of parameter operations. In the current PC3000 implementation the JBus\_S slave is normally associated with the high priority 10ms task. The total execution time for all 10ms function blocks should not exceed 10ms and thus the following table can be used to estimate the maximum block size that can be used.

<b>Operation</b>	<b>Block Size</b>	<b>Execution Time (ms)</b>	<b>Parameter Time (<math>\mu</math>s)</b>
BOOL word read	3	1.2	60
BOOL word write	3	1.6	43
DINT word read	1	0.8	66
DINT word write	1	0.8	48
STRING word read	20	3.1	69
STRING word read	40	5.3	69
STRING word write	20	3.7	70
STRING word write	40	6.0	71
REAL E mode read	1	1.3	326
REAL E mode read	8	4.5	326
REAL E mode write	1	0.8	187
REAL E mode write	8	3.7	181
REAL L mode read	1	2.1	385
REAL L mode read	8	5.1	385
REAL L mode write	1	0.9	254
REAL L mode write	8	4.4	249
REAL X mode read	1	2.2	600
REAL X mode read	6	7.8	646
REAL X mode write	1	2.7	212
REAL X mode write	6	5.0	206
STRING bit read	128	1.5	390
STRING bit write	128	1.9	361

Table 3-57 JBus\_S Timing Information

Each JBus transaction consists of a request message and a response message. The size of each message is related to the block size and the transmission mode

(RTU or ASCII.) Table 3-39 gives the sizes of the messages generated for each type of parameter operation for RTU mode. ASCII mode messages are related to RTU mode messages by the simple formula : Ascii = RTU \* 2 + 1.

Function	Address Space	Block Size	Request Message	Response Message
1/2	Bit	n	8	5 + 2n
3/4	Word	n	8	5 + 2n
5	Bit	1	8	8
6	Word	1	8	8
15	Bit	n	$10 + \frac{(n-1)}{8}$	8
16	Word	n	9 + 2n	8

Table 3-58 JBus\_S Message Sizing

The message size and the transmission baud rate has a direct effect on the turnaround time for each transaction. Using :

$$T_{rx} = \text{Request\_Msg\_Size} * \text{Bits\_Per\_Character} / \text{Baud\_Rate}$$

and

$$T_{tx} = \text{Response\_Msg\_Size} * \text{Bits\_Per\_Character} / \text{Baud\_Rate}$$

then

$$T_{total} = T_{rx} + \text{Slave\_Variable\_Execution\_Time} + T_{tx}$$

## JBus Protocol Reference

A definitive definition of Modbus protocol can be found in : ' Gould Modbus Protocol Reference Guide ' produced by Gould Inc. Programmable Control Division. Modbus is almost identical to Jbus except that slave device locations are referenced with an offset of one.

The JBus protocol provides for one master and up to 247 slaves on a single network. Only the master initiates a transaction. Transactions are either a query/response type (only one slave is addressed) or a broadcast/no response type (all slaves are addressed). A transaction consists of a single query and single response frame or a single broadcast frame.

Certain characteristics of the JBus protocol are fixed such as the frame format, handling of errors and functions performed. Others are user selectable such as baud rate, parity, number of stop bits and transmission modes (ASCII or RTU). All characteristics are selected on startup and cannot be changed while the system is running.

Messages transmitted by the master are in the form of a slave address, a function code, data and an error checking code. The addressed slave, if there are no message errors, performs the action defined by the function code. The slave then sends a response message consisting of the slave address, action performed, data acquired and an error checking code. No response is sent if the message is a broadcast as indicated by an address of 0.

In general, the master can send another message to any slave as soon as it receives a valid response, or after a user-selected time interval if no response is received. All messages may be sent as queries generating a slave response. However, only messages that do not need a response, such as a write function, can be sent as broadcast messages.

### Modes of Transmission

Two modes of transmission are available for use in a JBus system. The modes are ASCII and RTU as described below. Only one mode at a time can be used in a system.

<b>Characteristic</b>	<b>ASCII</b>	<b>RTU</b>
Coding System	Hexadecimal (0-9,A-F)	8-bit binary
Start bits	1	1
Data bits	7	8
Parity bits (optional)	1	1
Stop bits	1 or 2	1 or 2
Error Checking	Longitudinal Redundancy Check	Cyclical Redundancy Check

Table 3-59 JBus Modes of Transmission

### Message Format

The protocol defines two types of message format which are ASCII and RTU. The interpretation of fields within the two types of message are completely identical. The major differences are the type of error check performed and that approximately twice as many characters are used in ASCII. Instead of sending a single 8-bit binary character, the equivalent pair of 7-bit ASCII (0-9,A-F) characters are sent.

### ASCII Framing

Framing in ASCII transmission mode is accomplished by the use of the unique colon (:) character to indicate the beginning of a frame and carriage return (CR) line feed (LF) to indicate the end.

<b>Beginning of Frame</b>	<b>Address</b>	<b>Function</b>	<b>Data</b>	<b>Error Check</b>	<b>EOF</b>
:	2-char	2-char	N x 4-char	2-char	CR LF
Colon	16-bits	16-bits	N x 16-bits	16-bits	

Table 3-60 JBus Message Framing

### Remote Terminal Unit (RTU) Framing

The end of an RTU mode message is defined as a 3 character silence at the working baud rate. This timing may not be exact in the current implementation. The start of the next message is assumed to have occurred when this break is detected.

<b>Beginning of Frame</b>	<b>Address</b>	<b>Function</b>	<b>Data</b>	<b>Error Check</b>	<b>EOF</b>
T1 T2 T3	8-bits	8-bits	8-bits	16-bits	T1 T2 T3

Table 3-61 JBus Message Structure

### Address Field

The address field immediately follows the beginning of the frame and consists of 8-bits (RTU) or 2 characters (ASCII). This field indicates the user defined address of the slave that is to receive the message sent by the master. Each slave should be assigned a unique address. In a broadcast message an address of 0 is used. All slaves will respond to a broadcast message but no reply will be sent.

### Function Field

The function code field tells the addressed slave what action to perform. The list of supported functions is as follows :

<b>Function Code</b>	<b>Action</b>
1 or 2	Read N bits
3 or 4	Read N words
5	Write 1 bit
6	Write 1 word
15	Write N bits
16	Write N words

Table 3-62 JBus Function Codes

where N = 1..256.

### Data Field

The data field contains information needed by the slave to perform the specific function or it contains data collected by the slave in response to a query.

### Error Check Field

This field allows the master and slave devices to check a message for errors in transmission. The error check field uses a longitudinal redundancy check (LRC) in ASCII mode and a CRC-16 check in RTU mode.

### Exception Responses

The supported exception response codes are listed below. When a slave detects one of these errors it sends a response message to the master consisting of slave address, function code, error code and error check fields. To indicate that the response is a notification of an error bit 7 of the function code is set to 1.

<b>Code</b>	<b>Name</b>	<b>Meaning</b>
01	ILLEGAL FUNCTION	The message received is not an allowable action for address slave
02	ILLEGAL DATA ADDRESS	The address referenced in the data field is invalid
04	FAILURE IN ASSOCIATED DEVICE	An abortive error occurred. eg. Trying to write to write a protected address.

Table 3-63 JBus Exception Response Codes

Example Exception Response :

Slave Address	Function	Exception Code	Error Check
0A	81	02	73

Table 3-64 JBus Exception Response Codes

**Note:** Exception Response 03 - ILLEGAL DATA VALUE not implemented as PC3000 user program should ensure that values cannot exceed defined ranges.

### Function Details

The purpose of this section is to define the general format for the specific functions available in the JBus protocol. The format of the request message from the master is shown followed by the response message sent by the slave. The format is shown in RTU format which can easily be converted to ASCII format. Words are transmitted most significant byte first. Bit functions allow a maximum transfer of 1024 bits and word functions a maximum of 125 words in one operation.

### Read N bits (Function Code 01 or 02)

Request:

Byte 1	Slave Address (1..247).
Byte 2	Function Code (1 or 2).
Byte 3 & 4	Address of first bit to read.
Byte 5 & 6	Number of bits to read.
Byte 7 & 8	CRC-16 checksum.

Response :

Byte 1	Slave Address (1..247).
Byte 2	Function Code (1 or 2).
Byte 3	Number of bytes read, N.
Byte 4 to 4+N-1	bytes containing bits read.
Byte 4+N & 4+N+1	CRC-16 checksum

The bits read are packed into bytes starting at bit 0 for the first address. The last byte is padded with zero bits where necessary.



Example : Read bits from 0020 to 0056 from slave number 17.

Request : 11 01 00 14 00 25 CRC-16

Response : 11 01 05 CD 6B B2 0E 1B CRC-16

Read N words (Function Code 03 or 04)

Request :

Byte 1	Slave Address (1..247)
Byte 2	Function Code (3 or 4).
Byte 3 & 4	Address of first word to read.
Byte 5 & 6	Number of words to read.
Byte 7 & 8	CRC-16 checksum.

Response :

Byte 1	Slave Address (1..247).
Byte 2	Function Code (3 or 4).
Byte 3	Number of bytes read, N.
Byte 4 to 4+N-1	Words stored MSB byte, LSB byte
Byte 4+N & 4+N+1	CRC-16 checksum.

Example : Read bits from 0107 to 0110 from slave number 17.

Request : 11 03 00 6B 00 03 CRC-16

Response : 11 03 06 02 2B 00 00 00 64 CRC-16

**Write 1 bit (Function Code 05)**

Request :

Byte 1	Slave Address (1..247).
Byte 2	Function Code (5).
Byte 3 & 4	Address of bit to write.
Byte 5	Bit value. FF to set, 00 to clear.
Byte 6	Equals 0.
Byte 7 & 8	CRC-16 checksum.

Response :

Same as Request.

Example : Activate bit at address 0173 in slave number 17.

Request : 11 05 00 AD FF 00 CRC-16

Response : 11 05 00 AD FF 00 CRC-16

**Write 1 word (Function Code 06)**

Request :

Byte 1	Slave Address (1..247).
Byte 2	Function Code (6).
Byte 3 & 4	Address of word to write.
Byte 5 & 6	Word value.
Byte 7 & 8	CRC-16 checksum.

Response :

Same as Request.

Example : Write 128 to word at address 0135 in slave number 17.

Request : 11 06 00 87 00 80 CRC-16

Response : 11 06 00 87 00 80 CRC-16

## Write N bits (Function Code 15)

Request :

Byte 1	Slave Address (1..247).
Byte 2	Function Code (15).
Byte 3 & 4	Address of first bit to write.
Byte 5 & 6	Number of bits to write, n.
Byte 7	Number of bytes to write, $N = \text{int}(n/8)+1$ .
Byte 8 to 8+N-1	Bytes containing bits to write.
Byte 8+N & 8+N+1	CRC-16 checksum.

Response :

Byte 1	Slave Address (1..247).
Byte 2	Function Code (15).
Byte 3 & 4	Address of first bit written.
Byte 5 & 6	Number of bits written.
Byte 7 & 8	CRC-16 checksum.

Example : Write 10 bits starting at address 0020 in slave number 17.

Request : 11 0F 00 14 00 0A 02 CD 00 CRC-16

Response : 11 0F 00 14 00 0A CRC-16

## Write N words (Function Code 16)

Request :

Byte 1	Slave Address (1..247).
Byte 2	Function Code (16).
Byte 3 & 4	Address of first word to write.
Byte 5 & 6	Number of words to write, n.
Byte 7	Number of bytes to write, $N = n * 2$ .
Byte 8 to 8+N-1	Words stored MSB byte, LSB byte
Byte 8+N & 8+N+1	CRC-16 checksum.

Response :

Byte 1	Slave Address (1..247).
Byte 2	Function Code (16).
Byte 3 & 4	Address of first word written.
Byte 5 & 6	Number of words written.
Byte 7 & 8	CRC-16 checksum

Example : Write 2 words starting at address 0135 in slave number 17.

Request : 11 10 00 87 00 02 04 00 0A 01 02 CRC-16

Response : 11 10 00 87 00 02 CRC-16

## Error Reporting

If there is a specific JBus\_S Function Block or JBus protocol error, the error is reported via the **Error\_No** parameter of the JBus Function Block. Errors concerning a specific variable are reported via the associated Slave Variable block.

<b>Error_No</b>	<b>Error description</b>
145	PORT ERR NO ADDRESS <ul style="list-style-type: none"> <li>•There are less than two characters in the Port parameter of the function block.</li> </ul>
149	PORT ERR RX BAUD RATE NOT AVAILABLE <ul style="list-style-type: none"> <li>•The receive baud rate requested is not available on this serial port</li> </ul>
150	PORT ERR TX BAUD RATE NOT AVAILABLE <ul style="list-style-type: none"> <li>•The transmit baud rate requested is not available on this serial port.</li> </ul>
155	PORT ERR ILLEGAL SLOT <ul style="list-style-type: none"> <li>•The slot number selected is not legal. The slot number is the first character of the Port parameter.</li> </ul>
156	PORT ERR ILLEGAL PORT <ul style="list-style-type: none"> <li>•The port number selected is not legal. The port number, the second character of the Port parameter, should be valid for the module, for example 'A' to 'C' for an LCM port or 'A' to 'D' for an ICM port.</li> </ul>
161	PORT ERR PORT IN USE <ul style="list-style-type: none"> <li>•The selected Port is already in use for another driver.</li> </ul>
162	PORT ERR TOO MANY SLAVE PARAMS <ul style="list-style-type: none"> <li>•Too many slave parameters have been declared to the system.</li> </ul>
163	PORT ERR TOO MANY DRIVER TYPES <ul style="list-style-type: none"> <li>•Too many slave driver types have been declared to the system.</li> </ul>

Table 3-65 JBus\_S Error Codes

## Slave Variable Error Codes

<b>Error_No</b>	<b>Error description</b>
128	ADDRESS STRING TOO SHORT <ul style="list-style-type: none"> <li>•The address string contained in the Address parameter is too short to be valid.</li> </ul>
129	ADDRESS OUT OF RANGE <ul style="list-style-type: none"> <li>•The address specified in the Address parameter is not in the valid range of 0000 to 3999.</li> </ul>
130	INVALID CHARACTER IN ADDRESS STRING <ul style="list-style-type: none"> <li>•An invalid character has been detected in the data format field of the Address parameter.</li> </ul>
131	NUMERIC FORMAT VALUE OUT OF RANGE <ul style="list-style-type: none"> <li>•A parameter size value specified in the Address parameter is out of range for that data type.</li> </ul>
132	INVALID PARAMETER TYPE <ul style="list-style-type: none"> <li>•A slave variable block type has been created that cannot be handled by the current implementation of the driver.</li> </ul>
134	ADDRESS OVERLAP <ul style="list-style-type: none"> <li>•The address range specified in the Address parameter wholly or partly overlaps the address range of another slave parameter.</li> </ul>
135	MIXED MULTIELEMENT TYPES <ul style="list-style-type: none"> <li>•A slave variable block has array elements of differing types. The current implementation of the driver cannot support mixed types.</li> </ul>

Table 3-66 J Bus\_S Slave Variable Error Codes

## Parameter Attributes

Name	Type	Cold Start	Read Access	Write Access	Type Specific Information	
Port	<b>STRING</b>	'0A'	Oper	Config		
Unit_Id	<b>SINT</b>	0	Oper	Oper	High Limit Low Limit	247 0
Baud	<b>ENUM</b>	_9600	Oper	Config	Enumerated Values	_75(0) _300(1) _600(2) _1200(3) _2400(4) _4800(5) _9600(6) _19200(7) _38400(8) _57600(9) _115200(10)
Tx_Mode	<b>BOOL</b>	RTU	Oper	Config	Senses	RTU(0) Ascii(1)
Parity	<b>ENUM</b>	NONE	Config	Config	Enumerated Values	EVEN(0) ODD(1) SPACE(2) MARK(3) NONE(4)
Stop_Bits	<b>ENUM</b>	_1	Config	Config	Enumerated Values	_1(0) _1_5(1)
Wr_Protect	<b>BOOL</b>	No	Oper	Config	Senses	No(0) Yes(1)
Status	<b>BOOL</b>	NoGo	Oper	Block	Senses	NOGO(0) Go(1)
Error_No	<b>SINT</b>	0	Oper	Block	High Limit Low Limit	255 0

Table 3-67 JBus\_S Parameter Attributes

## TOSHIBA\_M FUNCTION BLOCK

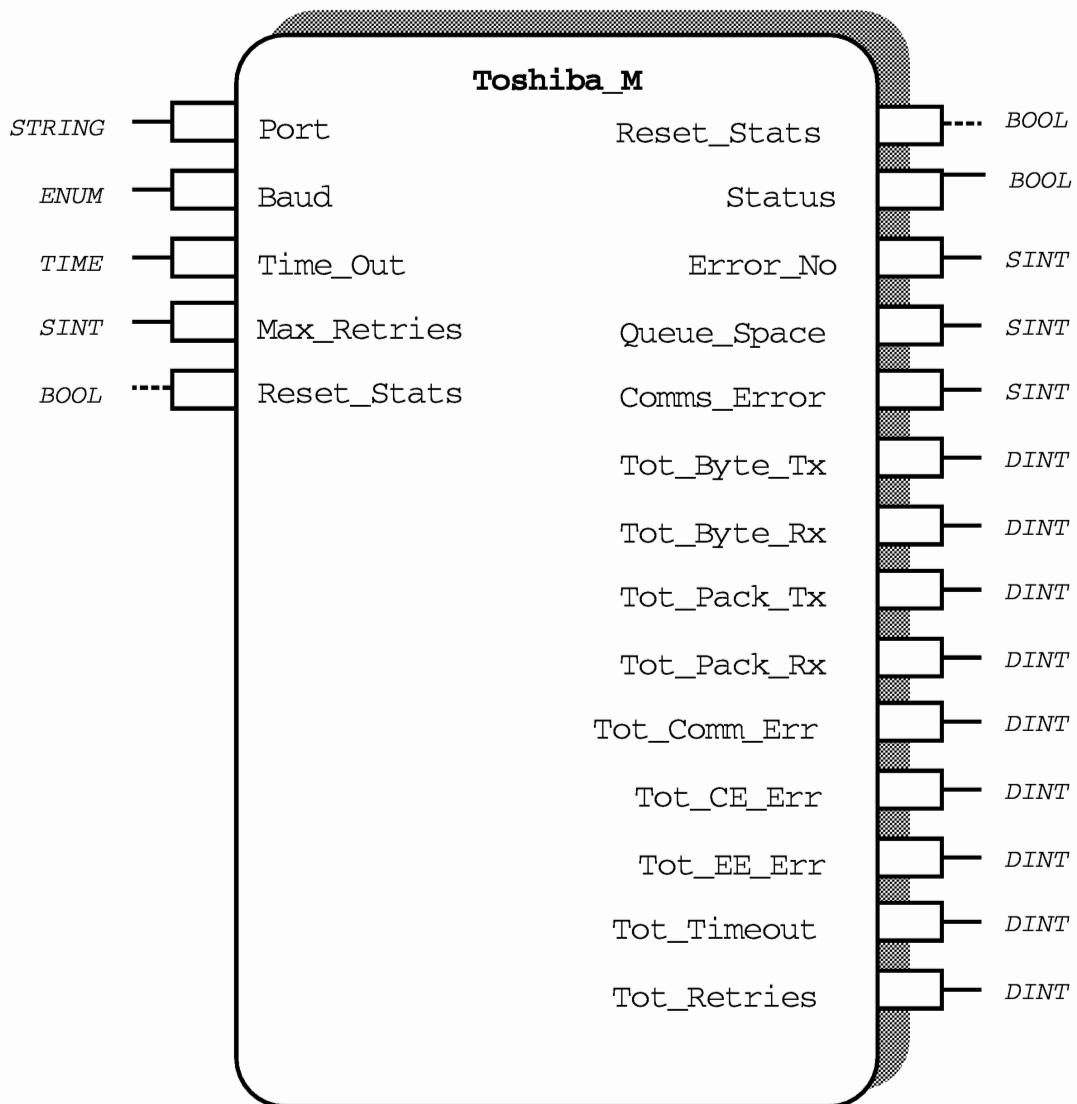


Figure 3-38 Toshiba\_M Function Block Diagram



## Functional Description

The Toshiba\_M Function Block supports serial communications on a designated serial communications. port using the Toshiba PLC protocol. This function block description includes the Toshiba Commands and Register Information for quick reference. Before reading this description, you are advised to gain a general understanding of the PC3000 communications system by reading the PC3000 Communication Overview.

This function block will be required when designing or programming the PC3000 to use a Toshiba PLC interface that functions in the master mode, i.e. the PC3000 is connected via a serial link to a Toshiba PLC configured as a slave.

You are advised to reference the following documents, if a detailed understanding of the Toshiba PLC protocol is required:

- EX250500 Toshiba Corporation. *Toshiba Programmable Controller Computer Link Module. CMP-6236 EX250/500 User's Manual. (Reference Number: UM-EX25U-E104(Mar. '87(2)))*

- EX2000 Toshiba Corporation. *Toshiba Programmable Controller Computer Link EX2000 User's Manual. (Reference Number: UM-EX2K\*\*\*-E004(Mar. '88(1)))*

The Toshiba\_M function block provides a driver for the Toshiba Programmable Logic controller (PLC) communication protocol for Toshiba devices EX250, EX500 and EX2000; the function block allows the PC3000 to act as the serial link master.

The function block deals with the protocol specific details of the communications and is supported by generic remote variable function blocks. The remote variable blocks are linked to the driver by means of a protocol specific address and can request the polling or updating of parameters of a communicating instrument via. a specified communication port.

For more details of the Remote Variable function blocks see the 'PC3000 Comms. Overview'.

Only the ASCII version of the protocol is supported. The driver always operates in **Even Parity**.

## Function Block Attributes

Type:..... 8 88

Class: ..... COMMS

Default Task: ..... Task\_1

Short List: ..... Port status Queue\_Space Comms\_Error

Memory Requirement:..... 2026 Bytes

## Parameter Descriptions

### Driver Configuration Parameters

The **Toshiba\_M** block has two configuration input parameters Port and Baud which configure the driver and should be set prior to running the user program. Changing these parameters while the user program is executing will have no affect on the driver except under special circumstances - see 'PC3000 Comms. Overview' section 'Temporarily changing configuration parameters'. The other input parameters can be changed while the user program is running.

#### Port

The **Port** parameter is the two character address of the port on which the Toshiba protocol is to run. The first character is a number from 0 to 5 representing the rack slot and the second character is a letter representing the port within that slot. e.g. '0C' would be port C on the LCM and if there were an ICM in slot 3 '3A' could refer to its top port.

#### Baud

The **Baud** parameter gives a choice of 6 different rates from 300 baud up to 9600 baud (as shown in Table 4-20) with a default of 9600 baud.

**Note:-** That not all ports will be able to support all baud rates. This will be indicated by an error when the function block is first run (see description of **Error\_No** ).

Enum Value	Baud Rate
0	300
1	600
2	1200
3	2400
4	4800
5	9600

Table 3-68 Toshiba\_M Baud Rates

#### Time\_Out

The **Time\_Out** parameter specifies the length of time that the Toshiba Master will wait for a response message to a transmitted request. After this time the driver assumes there was a transmission error and the request may be retransmitted or an error returned to the remote variable block which initiated the request.

**Time\_Out** defaults to 1 Second.

## Max\_Retries

The **Max\_Retries** parameter specifies the number of times that a request will be retransmitted if a transmission error, such as a timeout or a message checksum error, is detected. If a valid response is not received after this number of retries an error is returned to the remote variable block which initiated the request.

**Max\_Retries** defaults to 2, so a request will be sent three times before an error is reported and the request aborted.

## Driver Status Parameters

The driver status is indicated by three output parameters in the **Toshiba\_M** function block.

### Status

The **Status** parameter is a boolean indication of the state of the link controlled by the driver. If there are no problems with the link this parameter reads **Go**, but when an error occurs it changes to **NOGO**. If the **Status** is **NOGO** the **Error\_No** parameter indicates the reason for the problem.

### Error\_No

The **Error\_No** parameter indicates the reason for any errors with the link. If the link **Status** is **Go** then the **Error\_No** will be 0 (OK). For full details of error codes see Error Reporting section on page 3-

### Queue\_Space

The **Queue\_Space** parameter indicates the amount of space left in the queue for remote parameter operations. If this reaches zero then it implies that the link bandwidth is not sufficient to cope with the number of remote variable requests being made and data will be lost. If this situation arises the parameter polling rates should be reduced.

### Comms\_Error

The **Comms\_Error** parameter indicates the status of the last transaction executed by the driver. If there was some error detected then this will be reflected here. It should be noted that if many transactions are occurring then any error detected and reported will only remain here until the next transaction has completed.

## Driver Statistical Parameters

The driver statistical data is indicated by nine output parameters in the **Toshiba\_M** function block.

### Reset\_Stats

The **Reset\_Stats** parameter is used to reset all the statistical information outputs. This is done by setting the boolean value to **Yes**. The parameter automatically changes to **No** once the statistical outputs have been reset.

### Tot\_Byte\_Tx

The **Tot\_Byte\_Tx** parameter indicates the total number of bytes that have been transmitted by the driver since either the start of execution of the driver or, the last reset of the statistical outputs.

### Tot\_Byte\_Rx

The **Tot\_Byte\_Rx** parameter indicates the total number of bytes that have been received by the driver since either the start of execution of the driver or, the last reset of the statistical outputs.

### Tot\_Packet\_Tx

The **Tot\_Packet\_Tx** parameter indicates the total number of packets that have been transmitted by the driver since either the start of execution of the driver or, the last reset of the statistical outputs.

### Tot\_Packet\_Rx

The **Tot\_Packet\_Rx** parameter indicates the total number of packets that have been received by the driver since either the start of execution of the driver or, the last reset of the statistical outputs.

### Tot\_Comm\_Err

The **Tot\_Comm\_Err** parameter indicates the total number of communications errors that have been detected by the driver since either the start of execution or, the last reset of the statistical outputs. It includes both the **Tot\_CE\_Err** and **Tot\_EE\_Err** values as well as the total number of timeouts, checksum errors and any other errors that may have been detected during the transmission/reception process.

### Tot\_CE\_Err

The **Tot\_CE\_Err** parameter indicates the total number of **Computer Link Errors** that have been received from the Toshiba EX device and detected by the driver since either the start of execution or, the last reset of the statistical outputs.

### Tot\_EE\_Err

The **Tot\_EE\_Err** parameter indicates the total number of **Erroneous Errors** that have been received from the Toshiba EX device and detected by the driver since either the start of execution or, the last reset of the statistical outputs.

## Tot\_Timeouts

The **Tot\_Timeouts** parameter indicates the total number of timeouts that have been detected by the driver since either the start of execution or, the last reset of the statistical outputs. A timeout occurs when the time between transmitting a request and receiving the response from the Toshiba device exceeds the limit set by the input parameter **Time\_Out**.

## Tot\_Retries

The **Tot\_Retries** parameter indicates the total number of retries that have occurred since either the start of execution or, the last reset of the statistical outputs. A retry occurs when either a timeout is detected or some error is detected in the communications process.

## Remote Variable Operation

The requests made by the PC3000 are controlled by one or more remote variable blocks.

The **Toshiba\_M** driver currently supports the **Remote\_Int**, **Remote\_Str** and **Remote\_SW** block types.

## Addressing

It is necessary to set up a protocol specific **Address** in the remote variable block which is the address used to access the remote devices. This address is made up as follows.

The port is defined as in the function block **Port** parameter as a rack slot number followed by a letter for the port within that slot.

The protocol specific part of the address begins with a Station Number which specifies the Toshiba device which should respond to the request. This consists of either one or two characters depending upon the Toshiba device to which the request is being sent. For an EX250 or EX500 Toshiba device the address is one character in the range '0' to '7', as only eight devices can be connected together on one serial link from the host computer.

For an EX2000 Toshiba device the Station Number is two characters in the range '01' to '32' if the EX2000 device is on a RS485 serial link or, it is in the range '01' to '08' if it is on a RS422 serial link, again these are limited by the number of devices that can be placed on one serial link from the host computer.

This is then followed by the Command that is to be executed on the Toshiba device. The Command is always a two character mnemonic as described in Toshiba documentation for an EX250 or EX500 device or for an EX2000 device.

The end of the address is dependent upon the command specified. In the case of the Device / Register read or write commands the address must end with the Device / Register type, the start Device / Register and the number of Device's / Register's to be read or written. The format of this information is shown in the

examples below and can also be found in the related Toshiba documents. With any other command there is no additional information required.

Device / Register read and write commands, **DR** or **DW**, may optionally be specified as **D?**; the appropriate read or write mnemonic will be substituted internally depending on whether a read or write operation is triggered.

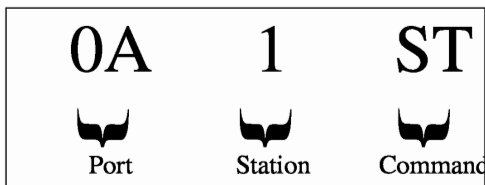


Figure 3-39 Example 1 of Remote Variable Address

**Note 1:** Example in figure 3-39 shows a 'Status Read' command issued to an EX250/500 device.

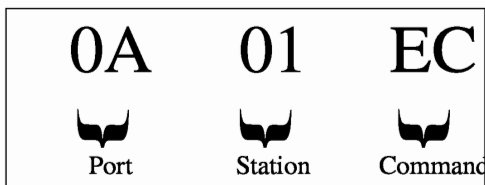


Figure 3-40 Example 2 of Remote Variable Address

**Note 2:** Example in figure 3-40 shows an 'EX Control' command issued to an EX2000 device.

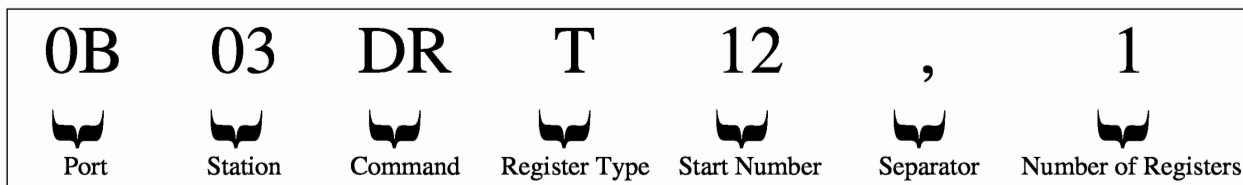


Figure 3-41 Example 3 of Remote Variable Address

**Note 3:** Example in figure 3-41 shows a 'Device/Register Read' command issued to an EX2000 device to read Timer Register 12.

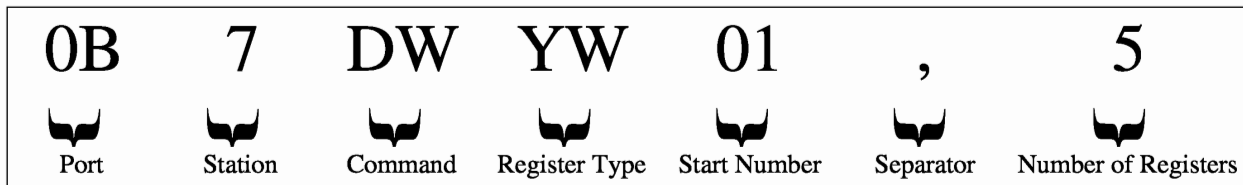


Figure 3-42 Example 4 of Remote Variable Address

**Note 4:** Example in figure 3-42 shows a 'Device/Register

**Write'** command issued to an EX250/500 device to write to 5 External Output registers starting at register 1..

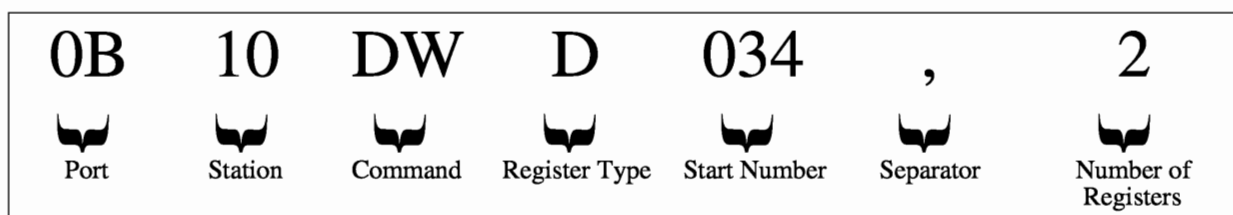


Figure 3-43 Example 5 of Remote Variable Address

**Note 5:** Example in figure 3-43 shows a '**Device/Register Write'** command issued to an EX2000 device to write to 2 Data Registers starting at register 34.

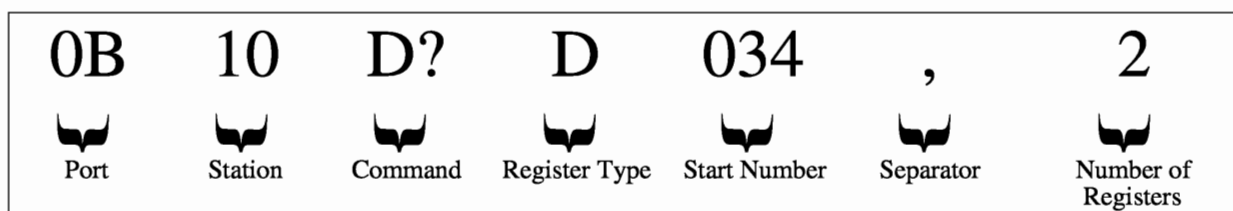


Figure 3-44 Example 6 of Remote Variable Address

**Note 6:** Example in figure 3-44 shows a '**Device/Register Read or Write'** command (depending on trigger condition) issued to an EX2000 device to write to 2 Data Registers starting at register 34.

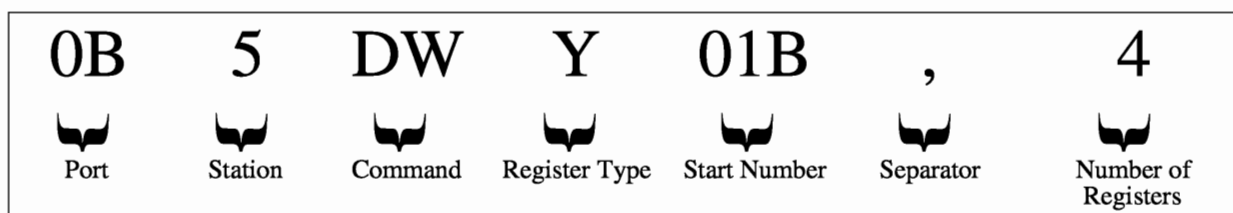


Figure 3-45 Example 7 of Remote Variable Address

**Note 7:** Example in figure 3-45 shows a '**Device/Register Write'** command issued to an EX250/500 device to write to 4 External Output Devices starting at device 1B ( Module 1, Channel B ). This will write to Devices Y1B, Y1C, Y1D, and Y1E ( that is Module 1, Channel B, C, D and E ).

The Address parameter of a Remote Variable function block may be changed at any time so the block may be re-used to communicate with different instruments and different parameters. It is not recommended that it is changed when the function blocks' State parameter is 'Pending', i.e. a request is outstanding.

## Data Formats

This section describes the different data encodings that can be selected for each basic data type.

### Integer

The Remote\_Int function block provides access to a single 'Register' location or, in the case of Input, Output and Relay Devices up to 16 'Device' locations. In the case of other 'Devices' it provides access to a single location. The Remote\_SW function block packs/unpacked the value from/to sixteen Digital parameters.

### String

The Remote\_Str function block provides access to one or more 'Device/Register' locations or access to 'System or Diagnostic' information on the slave device. When used with the 'Device/Register' commands or, commands wanting to access single integer values, the data is entered as Hex data by proceeding each character with the '\$' symbol. When used with other commands the data is entered as straight ASCII text ( this is only used for the 'TS' command ).

### Errors

If there is an error in the Address string or if there is a communications error, the Status parameter of the Remote Parameter block is set to NOGO and the Error\_No parameter will indicate the reason.

**Note:-** That the contents of the Address string is only validated when initiating a remote read or write.



## Toshiba Commands Reference

The commands listed below include EX250/500 and EX2000 commands. Where commands apply to one of the devices specifically these are identified clearly.

<b>Command</b>	<b>Description</b>
ER	EX Error Status Read • Reads the EX device error status.
TS	Test • Returns transmitted text as it was received at EX device.
ST	EX Status Read • Reads the EX Device status ( RUN/HALT/ERROR ).
DR	Device/Register Read • Reads contents of Device/Register.
DW	Device/Register Write • Writes contents of Device/Register.
BR†§	Program Block Read ( EX250/500 Only ) • Reads Programs Block by Block.
BR¥§§	Program Information Block Read ( EX2000 Only ) • Reads Program Information Block by Block.

Table 3-69 Toshiba EX250/500 and EX2000 Commands

† EX250 and EX500 devices only.

¥ EX2000 device only.

§ EX250/500 Use of This Command.

§§ EX2000 Use of This Command.

<b>Command</b>	<b>Description</b>
RB†	Program Block Read •Reads programs block by block.
BW†§	Program Block Write •Writes programs block by block.
BW†§§	Program InformationBlock Write. •Writes program information block by block.
WB†	Program Block Write •Writes programs block by block.
EC	EX Control •Controls operation mode of EX device.
IR	Keep Area Top Read •Reads the starting register for the top of the keep area.

Table 3-70 Toshiba EX250/500 and EX2000 Commands

† EX250/500 Use of This Command.

†† EX2000 Use of This Command.

§ EX250/500 Use of This Command.

§§ EX2000 Use of This Command.

<b>Command</b>	<b>Description</b>
TR	Diagnostic Table Read • Reads user defined error information.
SR†	System Parameter Read • Reads the EX250/500 system information.
S1¥	System Parameter 1 Read • Reads system information 1 from the EX2000 device.
S2¥	System Parameter 2 Read • Reads system information 2 from the EX2000 device.
RT¥	Calendar/Clock Read • Reads Calendar/Clock information.
WT¥	Calendar/Clock Write • Writes Calendar/Clock information.

Table 3-71 Toshiba EX250/500 and EX2000 Commands

† EX250 and EX500 devices only.

¥ EX2000 device only.

## Toshiba Device/Register Reference

Register	EX250	EX250 (RAM Extn)	EX500	EX2000 (16K RAM)	EX2000 (32K RAM)
XW Input Registers	XW00 - XW31	XW00 - XW31	XW00 - XW63	XW0000 - XW0499	XW0000 - XW0499
YW Output Registers	YW00 - YW31	YW00 - YW31	YW00 - YW63	YW0000 - YW0499	YW0000 - YW0499
RW Input Registers	RW00 - RW63	RW00 - RW63	RW00 - RW63	RW0000 - RW0999	RW0000 - RW0999
ZW Input Registers	ZW00 - ZW31	ZW00 - ZW31	ZW00 - ZW31	ZW0000 - ZW1999	ZW0000 - ZW1999
D Data Registers	D000 - D511 or 0000 - 0511	D000 - D999 or 0000 - 1535†	D000 - D999 or 0000 - 1535†	D00000 - D08191	D00000 - D16383
T Timer Registers	T000 - T127	T000 - T127	T000 - T127	T00000 - T00499	T00000 - T00499
C Counter Registers	C000 - C095	C000 - C095	C000 - C095	C00000 - C00499	C00000 - C00499
X External Input Devices	X000 - X15F	X000 - X15F	X000 - X31F	X00000 - X0499F	X00000 - X0499F
Y External Output Devices	Y000 - Y15F	Y000 - Y15F	Y000 - Y31F	Y00000 - Y0499F	Y00000 - Y0499F
R Auxiliary Relay Devices	R000 - R63F (Special coil: R600 - R63F)	R000 - R63F (Special coil: R600 - R63F)	R000 - R63F (Special coil: R600 - R63F)	R00000 - R0999F (Special coil: R09000 - R0999F)	R00000 - R0999F (Special coil: R09000 - R0999F)
Z Timer Registers	Z000 - Z31F	Z000 - Z31F	Z000 - Z31F	Z00000 - Z0999F	Z00000 - Z0999F
T. Timer Registers ¥				T.0000 - T.0499	T.0000 - T.0499
C. Timer Registers ¥				C.0000 - C.0499	C.0000 - C.0499

Table 3-72 Toshiba EX250/500 and EX2000 Device/Register Names and Ranges

† EX250 and EX500 devices only.

¥ EX2000 device only.

## Error Reporting

### Driver Error Codes

These errors appear as the **Error\_No** on the **Toshiba\_M** block.

<b>Error_No</b>	<b>Error description</b>
1	PORT ERROR NO ADDRESS <ul style="list-style-type: none"> <li>•There are less than two characters in the Port parameter of the function block.</li> </ul>
5/6	PORT ERROR BAUD RATE NOT AVAILABLE <ul style="list-style-type: none"> <li>•The baud rate requested is not available on this serial port.</li> </ul>
11	PORT ERROR ILLEGAL SLOT <ul style="list-style-type: none"> <li>•The slot number selected is not legal. The slot number is the first character of the Port parameter and must be in the range '0' to '5'.</li> </ul>
12	PORT ERROR ILLEGAL PORT <ul style="list-style-type: none"> <li>•The port number selected is not legal. The port number is the second character of the Port parameter and must be in the range 'A' to 'C' for an LCM or 'A' to 'D' for an ICM.</li> </ul>
17	PORT ERROR PORT IN USE <ul style="list-style-type: none"> <li>•The selected Port is already in use for another driver.</li> </ul>

Table 3-73 Toshiba\_M Error Codes

## Remote Variable Error Codes

These errors appear as the **Error\_No** on the remote variable block and the **Comms\_Error** on the **Toshiba\_M** block.

<b>Error_No</b>	<b>Error description</b>
11	ADDRESS ERROR ILLEGAL SLOT •The first character in the Address parameter is not in the valid range of '0' to '5'.
12	ADDRESS ERROR ILLEGAL PORT •The second character in the Address parameter is not in the valid range of 'A' to 'C' for a LCM port or 'A' to 'D' for an ICM port.
16	ADDRESS ERROR NO REMOTE PARAMETER SERVICE •The port given in the Address parameter does not have a suitable master driver allocated to it.
50	COMMS ERROR RX OVERRUN •An overrun error was detected on a received character.
51	COMMS ERROR RX PARITY •A parity error was detected on a received character.
52	COMMS ERROR RX PARITY & OVERRUN •A parity and an overrun error were detected while receiving.
53	COMMS ERROR RX FRAMING •A framing error was detected on a received character.
54	COMMS ERROR RX FRAMING & OVERRUN •A framing and an overrun error were detected while receiving.
55	COMMS ERROR RX FRAMING & PARITY •A framing and a parity error were detected while receiving.
56	COMMS ERROR RX FRAMING, OVERRUN & PARITY •A framing, parity and overrun error were detected while receiving.
57-64	BREAK CONDITION CHANGED •Break condition has been detected or cleared. •Line may have been disconnected. •Remote device Baud rate could be too slow.

Table 3-74 Remote Variable Error Codes

<b>Error_No</b>	<b>Error description</b>
80	STATION ADDRESS INVALID •The station address specified in the remote variable address string is invalid.
81	NUMBER OF DEVICES INVALID •An invalid number of Devices/Registers has been specified.
82	TOO MANY DEVICES SPECIFIED •The number of devices specified is greater than the maximum limit of 32.
83	NUMBER OF DEVICES AND DATA MISMATCH •There is not enough data in the remote string parameter value for the number of Devices/Registers specified.
84	COMMS CHECKSUM ERROR •A Checksum error has been detected in the response data received from the slave device.
85	COMMS TIMEOUT ERROR •A Timeout error has occurred after attempting the required number of retries with no response from the slave device within the timeout period.
101	COMMAND ERROR •The appropriate command is missing.
102	FORMAT ERROR •Transmission format mismatch detected.
103	CHECKSUM ERROR •The request received by the slave device had a checksum error.
104†	ENDING CODE ERROR •The packet ending code ')' and CR were not received by the slave device.
105†	EXCESSIVE TEXT LENGTH •Text sent exceeds 255 bytes.
106†	PROTECTION ERROR •Write command received by slave whilst 'Write Protect' still on.

Table 3-74 Remote Variable Error Codes (continued)

† EX250 and EX500 devices only.

<b>Error_No</b>	<b>Error description</b>
107†	CONVERSION ERROR •Data from EX250/500 cannot be converted.
108†	TIME-OUT 1 •A time gap of 1 second or longer occurred during reception of data at the slave device.
109†	TIME-OUT 2 •Request issued to the EX250/500 slave device was not accepted within 30 seconds.
110†	PARITY ERROR •Parity error detected at the slave device.
111†	OVERRUN ERROR •Overrun error detected at the slave device.
112†	FRAMING ERROR •Framing error detected at the slave device.
131†	NO END INSTRUCTION •No END instruction in slave program.
132†	ILLEGAL PAIR INSTRUCTION •Error in slave program on initial RUN check.
133†	PROGRAM FAILURE •Program destroyed or program checksum error in slave.
134†	MEMORY FULL •Unable to perform program write command, no more memory available.
135†	ILLEGAL PAGE/CIRCUIT NUMBER •Requested Page or Circuit does not exist.
136	MODE MISMATCH •Invalid mode command received.
137†	PROM ERROR •Attempt to write program after PROM was installed in slave.
138†	OPERAND ERROR •Operand in program in slave does not match available I/O device assignment.

Table 3-74 Remote Variable Error Codes (continued)

† EX250 and EX500 devices only.



<b>Error_No</b>	<b>Error description</b>
139	ERRONEOUS REGISTER NUMBER/SIZE •Undefined register specified or, register number out of range.
140†	I/O MISMATCH •I/O programmed did not match available slave I/O.
141†	I/O NO SYNC •No response obtained from a specified I/O module.
142†	ERRONEOUS TRANSMISSION •Text reception processing error or, unidentified command.
143	TYPE MISMATCH •Memory and I/O type recorded on storage device do not match those of the slave device.
144†	PAGE FULL •Page in program contains more than 255 instructions.
146¥	MEMORY PROTECT •Slave device has its memory protected.
147¥	TRANSMISSION IS BUSY •Command currently being executed from a local GP programmer device.
203¥	I/O BUS ERROR •An I/O bus error was encountered when the RUN command was issued to the slave.
234¥	I/O NO SYNC ERROR •No response from the I/O device when the RUN command was issued.
235¥	I/O MISMATCH ERROR •The installed I/O module does not correspond to the I/O allocation table.

Table 3-74 Remote Variable Error Codes (continued)

† EX250 and EX500 devices only.

¥ EX2000 device only.

<b>Error_No</b>	<b>Error description</b>
237¥	I/O OVERLAP ERROR •Duplication of an I/O allocated register number is detected when the RUN command is issued.
238¥	I/O NUMBER ERROR •I/O allocated register number exceeds limits.
239¥	PROM WRITE ERROR •An error is detected in the PROM write.
241¥	NO END ERROR •No END instruction is found within the main, sub or interrupt program.
242¥	PAIR INSTRUCTION ERROR •An error is detected in a pair instruction for MCS/MCR, JCS,JCR when a RUN command is issued.
243¥	OPERAND ERROR •An error is detected in an operand instruction when a RUN command is issued.
244¥	PROGRAM INVALID •An error is detected in the program management table when a RUN command is issued.
245¥	JUMP ERROR •An error is detected in the use of a JUMP instruction when a RUN command is issued.
246¥	NO LABEL ERROR •No LABEL is registered for the JUMP instruction.
247¥	NO SUB ERROR •Subroutine program is not registered.
248¥	NO RETURN ERROR •RETURN instruction in a subroutine program is not registered.

Table 3-74 Remote Variable Error Codes (continued)

¥ EX2000 device only.

---

<b>Error_No</b>	<b>Error description</b>
249¥	SUBROUTINE NESTING •An error is detected in nesting of a subroutine when a RUN command is issued.
250¥	STEP NUMBER ERROR •An error in the Toshiba SFC step number is detected when a RUN command is issued.
251¥	CONNECTOR STEP •An error of sequence connection in Toshiba SFC page is detected when a RUN command is issued.
252¥	TOSHIBA SFC SUBROUTINE ERROR •No Toshiba SFC subroutine CALL instruction registered.
253¥	ILLEGAL USER INSTRUCTION •An illegal command is detected in a user's program.

Table 3-74 Remote Variable Error Codes

¥ EX2000 device only.

## Standard Communications Error Codes

The following codes apply to other communication protocol drivers. Additional error codes will be used for all drivers when protocol specific error codes are required.

Code	Error Condition	Slave Driver	Slave Variable	Master Driver†	Remote Variable
0	OK	*	*	*	*
1	ADDRESS STRING TOO SHORT •Port address too short •Remote address too short •Slave address too short	*	*	*	*
2	ILLEGAL DATABITS	*	*	*	*
3	ILLEGAL PARITY	*	*	*	*
4	ILLEGAL STOP BITS	*	*	*	*
5	RX BAUD RATE NOT AVAILABLE	*	*	*	*
6	TX BAUD RATE NOT AVAILABLE	*	*	*	*
7	RTS NOT AVAILABLE	*	*	*	*
8	CTS NOT AVAILABLE	*	*	*	*
11	ILLEGAL SLOT •the first character in the Address parameter is not in them valid range of '0' to '5'	*	*	*	*
12	ILLEGAL PORT •the second character in the Address parameter is not in the valid range for the module , for example 'A' to 'C' for an LCM port or 'A' to 'D' for an ICM port.	*	*	*	*
14	RX TX INCOMPATIBLE	*	*	*	*
16	NO REMOTE VARIABLE SERVICE •Comms driver does not support Remote Vars.				*
17	PORT IN USE	*	*	*	*
18	TOO MANY SLAVE VARIABLES		*		
19	TOO MANY DRIVER TYPES		*		

Table 3-75 Standard Communications Error Codes

**Note:-** †These errors is also applicable to Raw communications.driver function block.

These error codes may be **offset** to a higher set of values for some drivers in order not to clash with standard error codes associated with the particular protocol.

## Parameter Attributes

Name	Type	Cold Start	Read Access	Write Access	Type Specific Information	
Port	STRING	'OA'	Oper	Config		
Baud	ENUM	_9600	Oper	Config	Enumerated Values	_300(0) _600(1) _200(2) _400(3) _4800(4) _9600(5)
Time_Out	TIME	1s	Oper	Super	High Limit Low Limit	24day 1sec
Max_Retries	SINT	2	Oper	Super	High Limit Low Limit	1000. 0
Enable_82X	BOOL	No	Config	Config	Senses	No(0) Yes(1)
Reset_Stats	BOOL	No	Oper	Oper	Senses	No(0) Yes(1)
Status	BOOL	NoGo	Oper	Block	Senses	NOGO(0) Go(1)
Error_No	SINT	0	Oper	Block	High Limit Low Limit	255 0
Queue_Space	SINT	0	Oper	Block	High Limit Low Limit	255 0
Comms_Error	SINT	0	Oper	Block	High Limit Low Limit	255 0

Table 3-76 Parameter Attributes

<b>Name</b>	<b>Type</b>	<b>Cold Start</b>	<b>Read Access</b>	<b>Write Access</b>	<b>Type Specific Information</b>	
Tot_Byte_Tx	DINT	0	Oper	Block	high Limit Low Limit	999999999 0
Tot_Byte_Rx	DINT	0	Oper	Block	High Limit Low Limit	999999999 0
Tot_Pack_Tx	DINT	0	Oper	Block	High Limit Low Limit	999999999 0
Tot_pack_Rx	DINT	0	Oper	Block	High Limit Low Limit	999999999 0
Tot_Comms_Err	DINT	0	Oper	Block	High limit Low limit	999999999 0
Tot_CE_Err	DINT	0	Oper	Block	High Limit Low limit	999999999 0
Tot_EE_Err	DINT	0	Oper	Block	High Limit Low limit	999999999 0
Tot_Timeouts	DINT	0	Oper	Block	High Limit Low Limit	999999999 0
Tot_Retries	DINT	0	Oper	Block	High Limit Low Limit	999999999 0

Table 3-76 Parameter Attributes (continued)

## EURO\_PANEL FUNCTION BLOCK

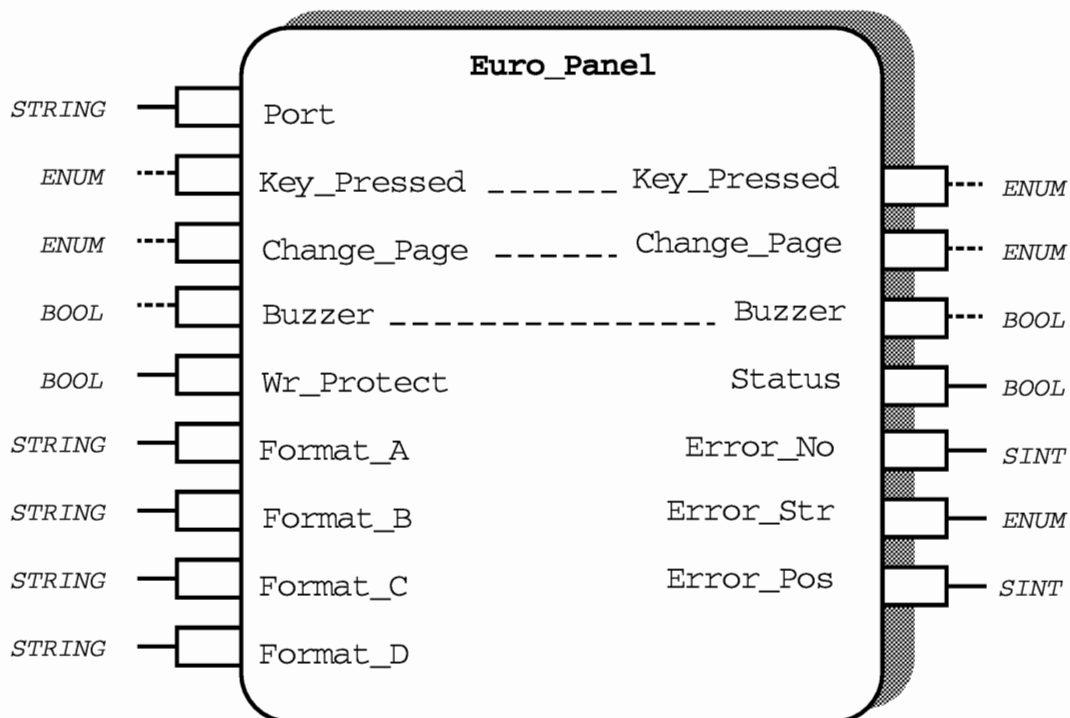


Figure: 3-46 Euro\_Panel Function Block Diagram

This block has now been replaced with the Europanel 2 function block. The pin descriptions on that block apply equally to this one.

## Functional Description

The Euro\_Panel function block is designed to simplify producing displays on the Eurotherm 2 line by 40 character panel. This function block is required when programming the PC3000 to operate with the Euro\_Panel display panel and provides facilities to control the display of messages and to support user interaction and data entry from the panel.

Before reading this description, you are advised to gain a general understanding of the PC3000 communications system by reading the PC3000 Communications Overview.

The follow terminology has been used in this description:

**OIFL** Operator Interface Format Language used to define display message and value formats for the Euro\_Panel.

**Page** Term used in this document to refer to a particular format for the complete 2 line by 40 character panel display.

The Euro\_Panel is a display panel with 2 lines by 40 characters, a numeric keypad, up/down and scroll keys and can communicate with the PC3000 via any RS422 communications port. Only one panel can be connected to a single RS422 port.

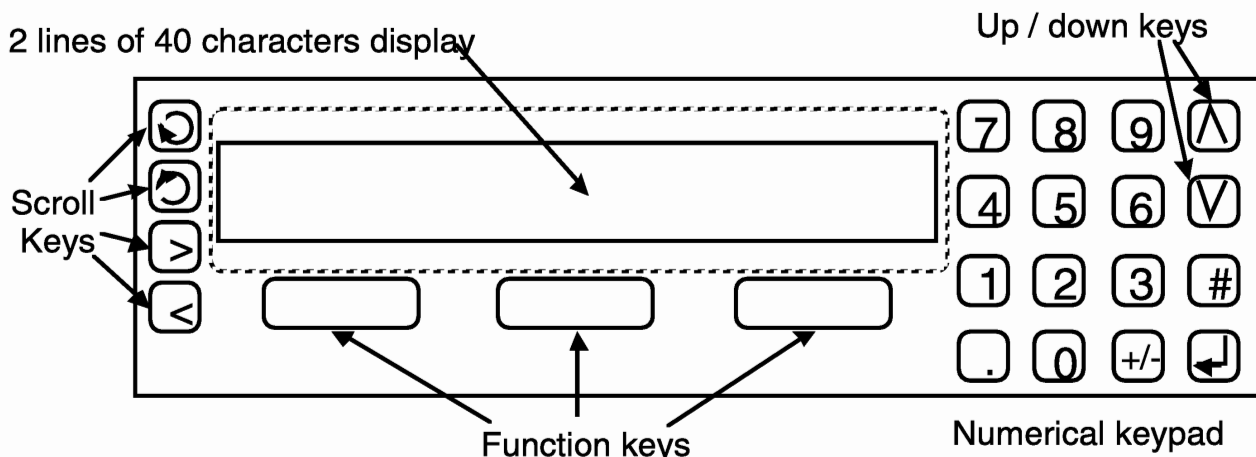


Figure: 3-47 Panel Layout

The Euro\_Panel communications driver function block provides PC3000 program support and handles all the low level functions such as data entry, cursor movement, screen updates etc and sets up the serial link to use the appropriate Baud rate, stop-bits etc. The parameters displayed on the panel are Slave Variables on the PC3000, with the Euro\_Panel function block, functioning as the serial link master. A Slave Variable can be associated with one or more Euro\_Panel communications function blocks by using the 'EP' protocol selector in the Slave Variable address.

Panel display messages can be formatted by setting up four string parameters to the Euro\_Panel function block. The formats are described using the Operator Interface Formatting Language (OIFL). OIFL can reference specific Slave Variable function blocks by a name which is supplied as part of the Slave Variable address, for example, OIFL can be used to place the value of a Slave Variable at a particular position in the panel display window. For more information on Slave Variable function blocks refer to the 'PC3000 Communications Overview'.

### Programming

When designing a system to use the Euro\_Panel, the internal function block parameters that are to be displayed or updated by the Euro\_Panel should be 'wired' to a set of Slave Variable function blocks. For example, to display the Output of a PID control loop, it is necessary to create a Slave\_Real function block which is 'wired' to the PID Output parameter. The Euro\_Panel function block refers to these slave variables via the names given the associated slave addresses.



In figure 3-13 , three Slave Variables are associated with two Euro\_Panel communications driver function blocks which are assigned to ports '0C' and '1B'. The port '0C' assigned function block has a format string that references the Slave\_Real Variable 'real1'. This results in the message 'Output is 100.0' being displayed on the panel, where 100.0 is the current value of the Slave\_Real Variable. If the Slave\_Real is wired to the Output of a PID function block, the displayed value will be automatically updated by the Euro\_Panel function block every time it executes.

A second integer slave variable identified to the Euro\_Panel block by the address 'EPval\_1', provides the value for a batch number which is displayed on the panel attached to port B of the ICM.

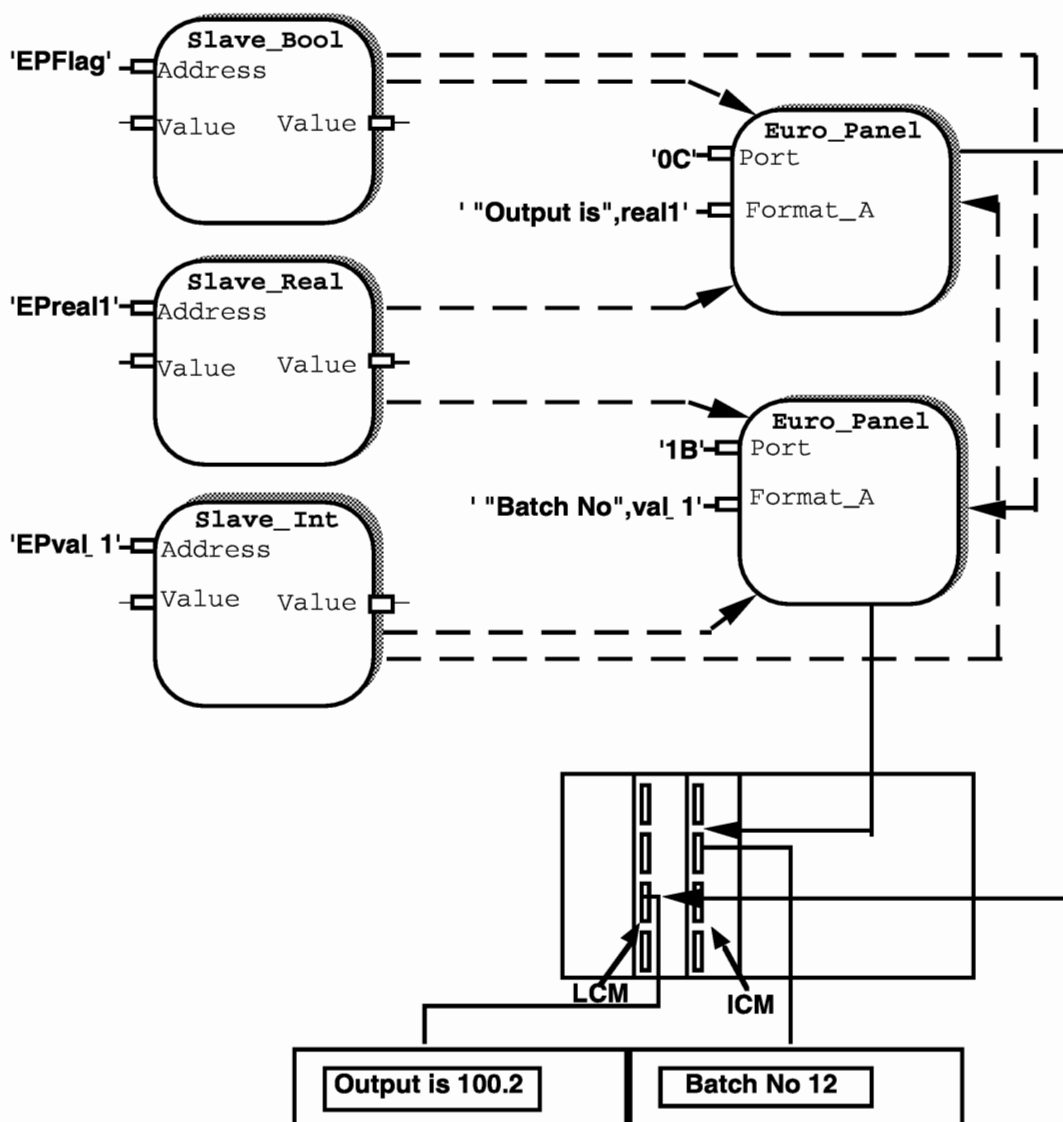


Figure: 3-48 Euro\_Panel Usage Example

## Slave Variables

The first two characters of the Slave Variable Address define the protocol with which the slave variable is associated. In this case the first two characters must be 'EP' to select the Euro\_Panel; the rest of the address should be used to assign a name to the Slave Variable which can be referenced within the OIFL format language. This is called the OIFL parameter name and can be any set of alphanumeric characters including underscore '\_' up to 12 characters in length.

Slave Variable Address examples :

```
'EPpv'  
'EPset_point'  
'EPs34_6_tt'
```

The slave variables also have a write protect parameter, which can be used to enable and disable write permission on a particular variable such as on the state of another parameter within the user program. This can be used for example, to inhibit changes to certain parameters during certain phases of a process.

## Euro\_Panel Function Block

Each Euro\_Panel function block is associated with a single communications port, and hence a single Eurotherm Panel. It performs the conversion to different formats for display, according to a set of format strings. It deals with the protocol required to drive the panel, and performs the key handling for data entry. It does not provide any facilities to change Baud rate or similar serial link characteristics as these cannot be changed within the panel.

## Use with Sequential Function Charts

In a simple application where the requirement is to display a fixed set of values on the panel, it is only necessary to set-up the Euro\_Panel format strings once, for example, as cold-start values. However, with complex applications, it may be necessary to change the displayed values using scroll lists and menus. This can be achieved by changing the format strings within a Sequential Function Chart (SFC). Four format strings are provided so that parts of a display region on the panel may be changed independently. For example, it is possible for the lower line to be part of the scroll list, while the top line always displays the same information. In this case, the SFC only has to change the format string for the lower line.

**Note:-** That the four format strings are always processed by the Euro\_Panel as if they form a single string. A User Program can modify any of the four strings at any time.

Refer to the section on 'User Program Examples' for further information.

---

## Function Block Attributes

Type:..... 8 90  
Class: .....COMMS  
Default Task: .....Task\_1  
Short List: .....Port Status Error\_No  
Memory Requirements: .....2356 Bytes  
Execution Time: .....1150  $\mu$  Secs

## EURO\_PANEL2 FUNCTION BLOCK

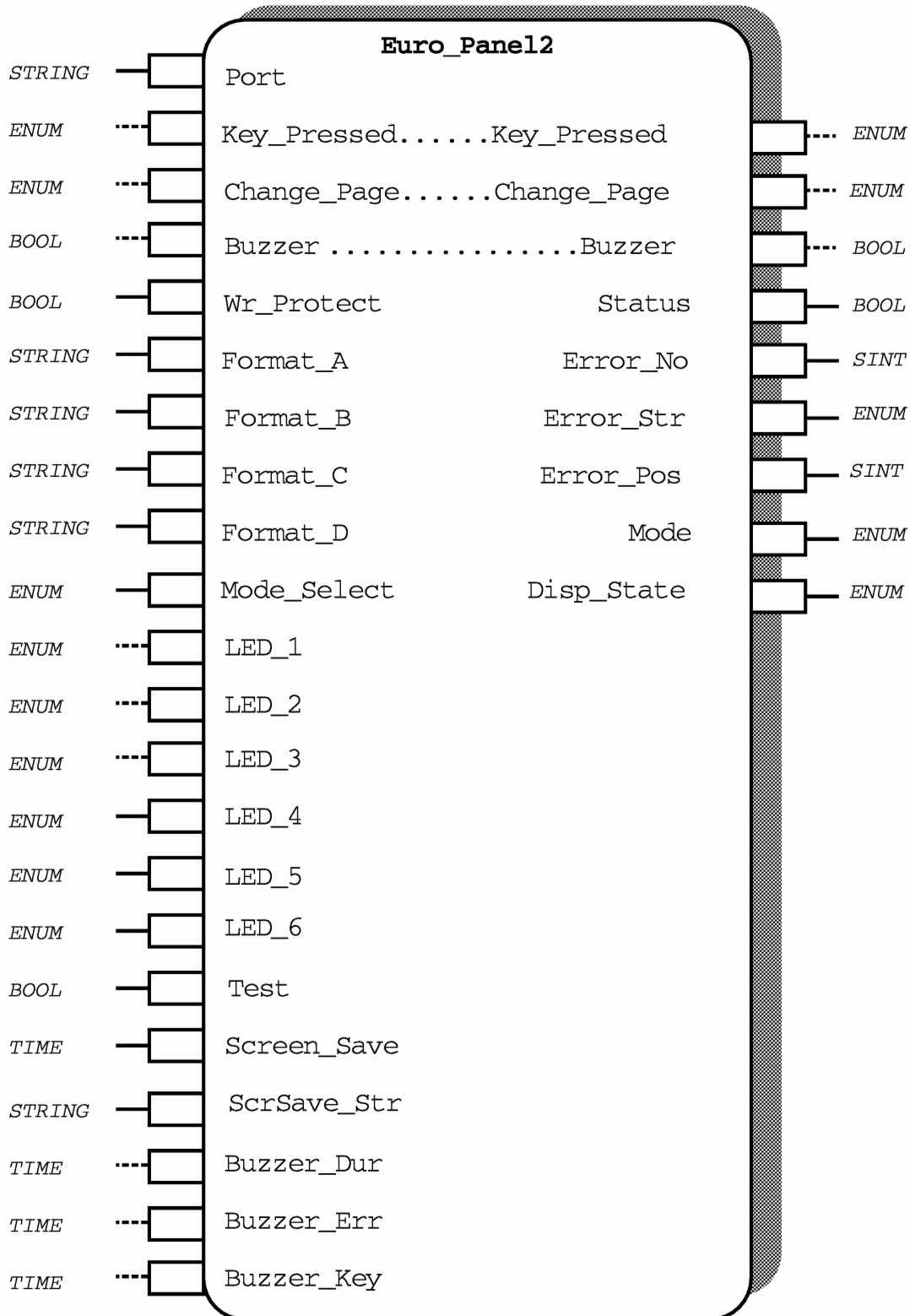


Figure: 3-49 Euro\_Panel2 Function Block Diagram

## Function Description

The Euro Panel 2 Function block is an enhanced version of the Euro Panel function block which has been created to support the new features and functionality available in the Version 2 Euro Panel.

This function block has been designed as a universal function block which can drive both the version 1 and version 2 Euro Panels. It should be used in preference to the Euro Panel function block for **all** new applications. It should be noted that in all cases, the default input values have been chosen to mimic version 1 Euro Panel behaviour.

Since this function block is an enhancement of the original Euro Panel function block, only the new features will be described. Reference should be made to the Euro Panel function block documentation for a full description of the various modes of operation and for the definition of the Operator Interface Formatting Language (OIFL).

## Function Block Attributes

Type:.....892  
Class .....COMMS  
Default Task: .....Task\_2  
Short List: .....Port, Status, Error No  
Memory Requirements: .....3084 Bytes

## Parameter Descriptions

### Mode\_Select (MS)

The **Mode\_Select** input controls the selection of the general mode of operation, The possible values are:

Auto (0)	This mode allows the semi- automatic selection of the panel type currently in use. (See later for further information).
EP (1)	This mode configures the function block to be compatible with the Version 1 Euro Panel.
EP2_EP1 (2)	This mode configures the function block to drive the Version 2 Euro Panel in a Version 1 compatible mode.
EP2 (3)	This mode configures the function block to be fully compatible with the Version 2 Euro Panel, and allows the enhanced features to be fully utilised. This mode should be used for all new applications.

Table 3-77 Mode Configurations

### LED 1 (L1) to LED 6 (L6)

These inputs activate the corresponding **LEDs** on the Version 2 Euro Panel. If **Mode Select** is set to EP1 (1) these inputs have no effect. The possible values are:

Off(0)	The LED is off
On (1)	The LED is on
Flash (2)	The LED will flash. The rate of Flash is determined by the panel hardware and cannot be changed by the user.

Table 3-78 LED Values on Version 2 Euro Panel

The LED colours and default symbols (printed on the membrane), are shown in Table 3-85

Input	Colour	Default Symbol
LED 1	Red (Bright)	Alarm
LED 2	Red	Battery Condition
LED 3	Green	Stand-by
LED 4	Green	Run
LED 5	Green	Hold
LED 6	Red	System Error

Table 3-79 LED Colours and Default Symbols

## Test (TST)

The **Test** input provides a simple way of checking that the panel is fully operational. When **Test** is set to On (1), all the display characters, underlines and LEDs will flash and the buzzer will sound continuously. When **Test** is reset to Off(O), normal operation will resume. It should be noted that **Test** does not work when the panel is in data entry mode.

## Screen\_Save (SCR)

A Screen Saver Function has been provided to allow the lifetime of the vacuum fluorescent display to be prolonged.

If the **Screen\_Save** time is set to T#Os (the default), the panel will operate continuously as with the Euro Panel block. If it is set to a non-zero duration, then the display will blank if there have been no page changes, key presses, buzzer or test activity for this duration. The six status LEDs will continue to operate.

A moving Eurotherm Logo appears on the otherwise blank display, to indicate that the panel is still working. This can be changed or disabled using the **ScrSave\_Str** input (see below).

Once asleep, the display will be woken up by a page change, a key press or any test or buzzer activity. In the case of a key press, the key press has no effect other than to restore the display.

Screen\_Save is temporarily disabled when the panel is in the data entry mode.

## ScrSave\_Str (SSS)

This string input allows the configurer to define the moving text which appears during screen save. The text is limited in length to four characters. An empty string (default) gives the standard Eurotherm Logo. For a completely blank screen, the string should be set to one or more spaces. The input can be changed at run-time, but the new string will only be picked up when the screen saver is next started.

**Buzzer\_Dur (BD)**

This controls the duration of the 'beep' activated by the **Buzzer** input. The default duration of 100ms matches the version 1 Euro Panel. **Buzzer\_Dur** has no effect when the function block is in EP1 mode.

**Buzzer\_Err (BE)**

This controls the duration of the 'beep' activated by an error e.g. an invalid data entry. The default duration of 100ms matches the version 1 Euro Panel. **Buzzer\_Err** has no effect when the function block is in EP1 mode.

**Buzzer\_Key (BK)**

This controls the duration of the 'beep' activated by any key press (except key presses during page change which are ignored). The default duration of 0ms (i.e. no beep) matches the Version 1 Euro Panel.

If the function block is in EP1 mode, any non-zero value will result in a 100 ms beep for every key press.

**Mode (M)**

The **Mode** output indicates the actual active mode of operation.

Unknown (0)	The mode has not yet been determined
EP1 (1)	The function block has been configured to be compatible with the Version 1 Euro Panel
EP2_EP1 (2)	The function block has been configured to drive the Version 2 Euro Panel in a Version 1 compatible mode
EP2 (3)	The function block has been configured for use with the Version 2 Euro Panel

Table 3-80 Possible Values of Modes of Operation



## Disp-State (DS)

This output indicates the current state of the display.

Blank(0)	The display is currently blanked e.g. during a page change
Display(1)	The normal display mode (i.e. <u>not</u> data entry)
Entry(2)	The operator has activated data entry mode
Sleep(3)	The screen saver is in operation
Test (4)	The panel is in test mode

Table 3-81 Possible Values of Display States

Some possible uses of this output include the time out of data entry to suspend unnecessary processing when the panel is asleep, etc.

## Basic Modes of Operation

The three basic modes of operation (EP1, EP2 and EP2-EP1) affect certain aspects of functionality. These are now described in more detail.

### Character Sets

Standard printable ASCII characters are handled normally in all three modes. The special characters generated by characters outside the printable range are different between Version 1 and Version 2 Euro Panel; there is a large degree of overlap in the characters available but the codes are different. In EP1 and EP2 modes the Version 1 and Version 2 character sets are used respectively. In EP2\_EP1 mode, the codes are mapped from the Version 1 set to the Version 2 set so as to give maximum compatibility. However, a few Version 1 characters have become obsolete in Version 2. In these cases a near match has been used where available, otherwise a filled block is used to highlight the problem.

### Data Entry

In EP2\_EP1 mode this works exactly as in Euro\_Panel, but using red underlines (i.e. flashing underline cursor, with 'live' data flashing and data being entered non-flashing).

In EP2 mode exactly the same system is used, except that the cursor is green for 'live' data, red for entered data.

## Mode Selection

### Basic Mode Selection

The selection of the basic modes EP1, EP2\_EP1 or EP2 is straightforwardly achieved using Mode\_Select. This would normally be set off-line in the userware, then possibly altered online while the PC3000 is in the Reset or Halted state.

### Auto Mode

If Mode\_Select is set to Auto then the basic mode is selected semi-automatically.

Note that this feature is only operational for LCM ports. On an ICM, it simply results in the mode being set to EP2\_EP1.

The remainder of this section refers to operation on an LCM port.

Because it involves characters temporarily being sent and/or received at incorrect baud rate and parity, this feature cannot be regarded as fully deterministic. Some specific limitations and recommendations are mentioned below. Testing has shown that it works very well in practice.

In the first instance, Mode will be set to Unknown, since the hardware cannot be determined until keys are pressed; in this mode nothing can be displayed. Pressing some keys will eventually result in the mode being set to either EP1 or EP2\_EP1 as appropriate. Several key presses may be required, and one or two particular keys may prove ineffective, e.g. '+/-' on Version 1 Euro Panel. (None of these key presses has any effect beyond determining the mode). A Change Page = Next Pge (3) will be automatically generated at this point, to activate the display.

If the Mode\_Select remains at Auto then if the panel hardware is changed over to the other type, further key presses will eventually cause the mode to flip over to match the new panel. Again, page change is automatically invoked, so that display may commence immediately on the new panel. Note that this mode change could possibly happen in error if the comms environment is very noisy; to avoid this problem, the user program could include a Step which recognises

the mode change from Unknown to EP1 or EP2\_EP1 and then changes Mode\_Select to match Mode, in effect freezing the mode permanently from then on. Note also that the Version 1 panel, if plugged into a port currently displaying in a Version 2 mode, can very occasionally enter a lock-up state. In this case the panel must be switched off and on again.

### Mode change at runtime

A further feature is that the Mode Select may be used to force a mode change 'on the fly', i.e. at runtime, by setting it to a non-auto value which differs from the current mode. The mode change then takes immediate effect, including an automatic page change. (Note that this feature, like auto mode, is not available on ICM).

## OIFL enhancements

Several OIFL enhancements have been made in order to make use of the enhanced functionality of the new panel. Some of these may be used in EP1 mode.

### Positioning for Function Key legends

This feature is a new form of the '@' command, introduced to simplify the positioning of legends for the function keys. The new commands are as follows:

@f1      @f2      @f3

The position, default width and justification are automatically set up for the next field, which should be the function key legend - usually a constant string or a string variable. Subsequent fields will be positioned at subsequent function key locations, until another '@' command is reached.

Example:

@f1, "Barrel", zone, "MORE " (zone is a STRING variable)

The actual string length should not exceed 13 characters for f1 or f3, or 12 characters for f2.

## Underline Formats

In EP1 mode 'U' works exactly as before, giving blue underlines.

In the other modes we have:

U	Generates red (upper) underlines
u	Generates green (lower) underlines

In EP1 mode, 'u' is equivalent to 'U'.

## Bar Chart Formats

The underscore characters may be used to implement bar charts. The new format characters, which may be thought of as alternatives to the 'E' and 'S' formats are:

B	Bar chart using red (upper) underlines
b	Bar charts using green (lower) underlines

This is for EP2 and EP2\_EP1 modes. For EP1 mode, 'b' and 'B' are equivalent, and use the blue underlines; the resolution is halved in this case, because the underlines are not split into two segments as on the version 2 Euro Panel

The justification character 'R' may be used to justify the bar to the right (i.e. it grows towards the left). By default (or if 'L' or 'C' is used) it is justified to the left.

The slave value may be DINT or REAL. If min and max limits are specified then they are used; if not, then they default to 0 and 100 respectively.

If the maximum is exceeded by the live value, then the entire bar will flash. Similarly below minimum, except that in EP2\_EP1 and EP2 modes only half-underlines are flashed. These features may be (independently) disabled by using the 'U' and/or 'u' qualifiers respectively.

The 'F' ('flip-sign') qualifier may be used to negate the parameter so that the bar grows as the value decreases. This can sometimes be useful - see the 'error chart' example below. Note that the limits are not negated.

Text may occupy the same position as bar charts, but 'U', 'u' or 'W' qualifiers should not be used with this text. Such an overlap will generally give error 131, except when a bar chart overlaps with underlines of the opposite colour; this case is not detected as an error, but should be avoided, as the chart will flicker where it overlaps.

A bar chart should not overlap with underlines of either colour it is not detected as an error, but the chart will flicker.

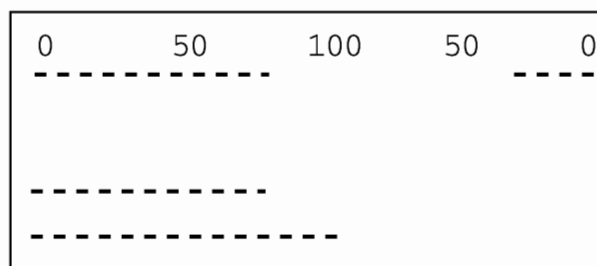
A general example:

```
Format_A := '0      50      100    50      0'
```

```
Format_B := '@0,xxx:20LB, @20, yyy:20RB'
```

```
Format_C := '@0: 1 ,yyy<=50:40B,@0,10<=yyy<=30:40bUu'
```

(and suppose xxx = 70, yyy = 20)



(In the case of the last bar, neither underflow or overflow will cause the flashing behaviour described above.)

As a second example, note that a useful 'error chart' may be achieved as follows:

```
'xxx:20bFRu,xxx:20Bu'
```

This grows to the right in red when xxx is positive (flashing if xxx is over 100) and to the left in green when xxx is negative (flashing if xxx is below -100).

## CR justification

By default, and as in Euro\_Panel, centre justified fields tend towards the left if the specified and actual widths are neither both even nor both odd. If the 'C' specifier is followed by 'R' then the tendency will be towards the right. Previously this would have given error 127 ("More than one justification specified"); 'R' followed by 'C' will still give this error.

Example:

"xx":5UC will give \_xx\_ \_

"xx":5UCR will give \_ \_xx \_

This feature is available in all modes.

## Non-alphanumeric characters in enumerations

Enumeration strings are no longer restricted to alphanumeric. Any character above \$17 is permitted, except for the delimiters ',' and '}'.

## Modified OIFL error codes

The following two tables list the differences between OIFL error codes in Euro\_Panel2 versus Euro\_Panel. The first table give the Euro\_Panel meanings, the second the Euro\_Panel2 meanings. In most cases the meaning has simply been generalised. However, codes 129 and 131 have been merged into 129, and 131 is now a new code, which is generated in EP2 and EP2\_EP1 modes only.

<b>OIFL Error Codes (Euro_Panel)</b>	
122	Non alphanumeric in enum list
129	Scientific notation specified for non real param
130	Can't have scientific and engineering notation
131	Engineering notation specified for non real param
132	No more attribute space
138	No room for const string on display

Table 3-82 Euro\_Panel Error Codes

<b>OIFL Error Codes (Euro_Panel2)</b>	
122	Invalid character in enum list
129	Format is invalid for this parameter type
130	Conflicting formats specified
131	Overlapping display items
132	Too many display items
138	Item does not fit within display area

Table 3-83 Euro\_Panel 2 Error Codes

## Parameter Attributes

Name	Type	Cold Start	Read Access	Write Access	Type Specific Information	
Port	<b>STRING</b>	'0A'	Oper	Config		
Key_Pressed	<b>ENUM</b>	No_Key	Oper	Oper	Enumerated Values	Zero(0) One(1) Two(2) Three(3) Four(4) Five(5) Six(6) Seven(7) Eight(8) Nine(9) Clkwise(10) AClkwise(11) Right(12) Left(13) F1(14) F2(15) F3(16) Point (17) Chg_Sgn(18) Enter(19) Down(20) Up(21) No_Key(22)
Change_Page	<b>ENUM</b>	Ok	Oper	Oper	Enumerated Values	Ok(0) Pending(1) Error(2) Nxt_Pge(3)
Buzzer	<b>BOOL</b>	Off	Oper	Config	Senses	Off(0) Beep(1)
Wr_Protect	<b>BOOL</b>	No	Oper	Super	Senses	No(0) Yes(1)
Format_A	<b>STRING</b>	"	Oper	Config		
Format_B	<b>STRING</b>	"	Oper	Config		
Format_C	<b>STRING</b>	"	Oper	Config		
Format_D	<b>STRING</b>	"	Oper	Config		

Table 3-84 Euro\_Panel2 Parameter Attributes



Name	Type	Cold Start	Read Access	Write Access	Type Specific Information	
Status	<b>BOOL</b>	NoGo	Oper	Block	Senses	NoGo(0) Go(1)
Error_No	<b>SINT</b>	0	Oper	Block	High Limit Low Limit	255 0
Error_str	<b>ENUM</b>	No_Err	Oper	Block	Enumerated Values	No_Err(0) Fmt_A(1) Fmt_B(2) Fmt_C(3) Fmt_D(4)
Error_Pos	<b>SINT</b>	0	Oper	Block	High Limit Low Limit	255 0
Mode_Select	<b>ENUM</b>	Auto(0)	Oper	Oper	Enumerated Values	Auto(0) EP1(1) EP2_EP1(2) EP2(3)
LED_1 to LED_6	<b>ENUM</b>	Off (0)	Oper	Oper	Enumerated Values	Off(0) On(1) Flash(2)
Test	<b>BOOL</b>	Off (0)	Oper	Oper	Senses	Off(0) On(1)
Screen_Save	<b>TIME</b>	0ms	Oper	Oper	High Limit Low Limit	20days 0ms
ScrSave_Str	<b>STRING</b>	"	Oper	Oper		
BuzzerDur	<b>TIME</b>	100ms	Oper	Oper	High Limit Low Limit	2s540ms 0ms
Buzzer_Err	<b>TIME</b>	100ms	Oper	Oper	High Limit Low Limit	2s540ms 0ms
Buzzer_Key	<b>TIME</b>	0ms	Oper	Oper	High Limit Low Limit	2s540ms 0ms
Mode	<b>ENUM</b>	Unknown (0)	Oper	Oper	Enumerated Values	Unknown(0) EP1(1) EP2_EP1(2) EP2(3)
Disp_State	<b>ENUM</b>	Blank(0)	Oper	Oper	Enumerated Values	Blank(0) Display(1) Entry(2) Sleep(3) Test(4)

Table 3-84 Euro\_Panel2 Parameter Attributes (continued)

## Character Codes

Hex	Dec	Description	Char	Hex	Dec	Description	Char
18	24	inv c	<b>C</b>	33	51	three	3
19	25	inv h	<b>H</b>	34	52	four	4
1A	26	inv i	<b>I</b>	35	53	five	5
1B	27	inv p	<b>P</b>	36	54	six	6
1C	28	inv s	<b>S</b>	37	55	seven	7
1D	29	inv a	<b>A</b>	38	56	eight	8
1E	30	inv e	<b>E</b>	39	57	nine	9
1F	31	not used		3A	58	colon	:
20	32	space	u	3B	59	semi colon	;
21	33	exclamation mark	!	3C	60	less than	<
22	34	duoble quote	"	3D	61	equals sign	=
23	35	hash	#	3E	62	greater than	>
24	36	dollar	\$	3F	63	query	?
25	37	percent	%	40	64	commercial at	@
26	38	ampersand	&	41	65	capital a	A
27	39	right quote	'	42	66	capital b	B
28	40	left bracket	(	43	67	capital c	C
29	41	right bracket	)	44	68	capital d	D
2A	42	star	*	45	69	capital e	E
2B	43	plus sign	+	46	70	capital f	F
2C	44	comma	,	47	71	capital g	G
2D	45	minus sign	-	48	72	capital h	H
2E	46	full stop	.	49	73	capital i	I
2F	47	slash	/	4A	74	capital j	J
30	48	zero	0	4B	75	capital k	K
31	49	one	1	4C	76	capital l	L
32	50	two	2	4D	77	capital m	M

Table 3-85 Character Code for Euro\_Panel2


Hex	Dec	Description	Char	Hex	Dec	Description	Char
4E	78	capital n	N	6B	107	lower case k	k
4F	79	capital o	O	6C	108	lower case l	l
50	80	capital p	P	6D	109	lower case m	m
51	81	capital q	Q	6E	110	lower case n	n
52	82	capital r	R	6F	111	lower case o	o
53	83	capital s	S	70	112	lower case p	p
54	84	capital t	T	71	113	lower case q	q
55	85	capital u	U	72	114	lower case r	r
56	86	capital v	V	73	115	lower case s	s
57	87	capital w	W	74	116	lower case t	t
58	88	capital x	X	75	117	lower case u	u
59	89	capital y	Y	76	118	lower case v	v
5A	90	capital z	Z	77	119	lower case w	w
58	91	l sq bracket	[	78	120	lower case x	x
5C	92	back slash	\	79	121	lower case y	y
5D	93	r sq bracket	]	7A	122	lower case z	z
5E	94	up chevron	^	7B	123	l curly bracket	{
5F	95	underscore	_	7C	124	pipe	
60	96	left quote	`	7D	125	r curly bracket	}
61	97	lower case a	a	7E	126	tilde	~
62	98	lower case b	b	7F	127	page symbol	
63	99	lower case c	c	80	128	uper case c cidilla	Ç
64	100	lower case d	d	81	129	l c u umlaut	ü
65	101	lower case e	e	82	130	l c e acute	é
66	102	lower case f	f	83	131	l c a circumflex	â
67	103	lower case g	g	84	132	l c a umblaut	ä
68	104	lower case h	h	85	133	l c a grave	à
69	105	lower case i	i	86	134	l c a bol	â
6A	106	lower case j	j	87	135	l c c cidilla	ç

Table 3-85 Character Code for Euro\_Panel2 (Continued)

Hex	Dec	Description	Char	Hex	Dec	Description	Char
88	136	l c e circumflex	ê	A6	166	l c <u>a</u>	<u>a</u>
89	137	l c e umblaut	ë	A7	167	l c <u>o</u>	<u>o</u>
8A	138	l c e grave	è	A8	168	inverted ?	¿
8B	139	l c i umblaut	ï	A9	169	superscript +	<sup>+</sup>
8C	140	l c i circumflex	î	AA	170	superscript -	<sup>-</sup>
8D	141	l c i grave	ì	AB	171	double down chevron	⇩
8E	142	u c a umblaut	Ä	AC	172	double -up chevron	⇧
8F	143	u c a bol	Å	AD	173	inverted l	¡
90	144	u c e acute	É	AE	174	much less than	≪
91	145	l c ae	œ	AF	175	much greater than	≫
92	146	u c ae	Æ	B0	176	superscript 0	<sup>0</sup>
93	147	l c o circumflex	ô	B1	177	superscript 1	<sup>1</sup>
94	148	l c o umblaut	ö	B2	178	superscript 2	<sup>2</sup>
95	149	l c o grave	ò	B3	179	superscript 3	<sup>3</sup>
96	150	l c u circumflex	û	B4	180	superscript 4	<sup>4</sup>
97	151	l c u grave	ù	B5	181	superscript 5	<sup>5</sup>
98	152	l c y umblaut	ÿ	B6	182	superscript 6	<sup>6</sup>
99	153	u c o umblaut	Ö	B7	183	superscript 7	<sup>7</sup>
9A	154	u c u umblaut	Ü	B8	184	superscript 8	<sup>8</sup>
9B	155	cent	¢	B9	185	superscript 9	<sup>9</sup>
9C	156	pound sign	£	BA	186	subscript 0	<sub>0</sub>
9D	157	yen sign	¥	BB	187	subscript 1	<sub>1</sub>
9E	158	l h zone end	↵	BC	188	subscript 2	<sub>2</sub>
9F	159	mid zone sign	┌	BD	189	subscript 3	<sub>3</sub>
A0	160	l c a acute	á	BE	190	subscript 4	<sub>4</sub>
A1	161	l c i acute	í	BF	191	subscript 5	<sub>5</sub>
A2	162	l c o acute	ó	C0	192	subscript 6	<sub>6</sub>
A3	163	l c u acute	ú	C1	193	subscript 7	<sub>7</sub>
A4	164	l c n - tilde	ñ	C2	194	subscript 8	<sub>8</sub>

Table 3-85 Character Code for Euro\_Panel2 (Continued)

Hex	Dec	Description	Char	Hex	Dec	Description	Char
A5	165	u c n - tilde	Ñ	C3	195	subscript 9	₉
C4	196	inv 0	◻	DE	222	up arrow	↑
C5	197	inv 1	◻	DF	223	down arrow	↓
C6	198	inv 2	◻	EO	224	alpha	α
C7	199	inv 3	◻	E1	225	beta	β
C8	200	inv 4	◻	E2	226	cap gamma	Γ
C9	201	inv 5	◻	E3	227	pi	π
CA	202	inv 6	◻	E4	228	uc sigma	Σ
CB	203	inv 7	◻	E5	229	lc sigma	σ
CC	204	inv 8	◻	E6	230	mu	μ
CD	205	inv 9	◻	E7	231	tau	Τ
CE	206	row 2	—	E8	232	phi	φ
CF	207	row 3	—	E9	233	theta	θ
D0	208	row 4	—	EA	234	omega	Ω
D1	209	row 5	—	EB	235	lc delta	δ
D2	210	row 6	—	EC	236	infinity	∞
D3	211	row 7	—	ED	237	psi	Ψ
D4	212	row 8	—	EE	238	uc epsilon	Ε
D5	213	col 2		EF	239	uc eta	Η
D6	214	col 3		F0	240	equivalence sign	≡
D7	215	col 4		F1	241	plus/minus	±
D8	216	col 5		F2	242	greater than or equals	≥
D9	217	intgral	∫	F3	243	less than or equals	≤
DA	218	bell symbol	🔔	F4	244	inv -	◻
DB	219	filled block	■	F5	245	inv +	◻
DC	220	left arrow	←	F6	246	divide sign	÷
DD	221	right arrow	→	F7	247	approx equals	≈

Table 3-85 Character Code for Euro\_Panel2 (Continued)

<b>Hex</b>	<b>Dec</b>	<b>Description</b>	<b>Char</b>	<b>Hex</b>	<b>Dec</b>	<b>Description</b>	<b>Char</b>
F8	248	degree sign	°	FC	252	lc eta	η
F9	249	large bullet	1	FD	253	superscript 2	2
FA	250	small bullet	1	FE	254	small block	■
FB	251	sq root sign	√	FF	255	not used	

Table 3-85 Character Code for Euro\_Panel2

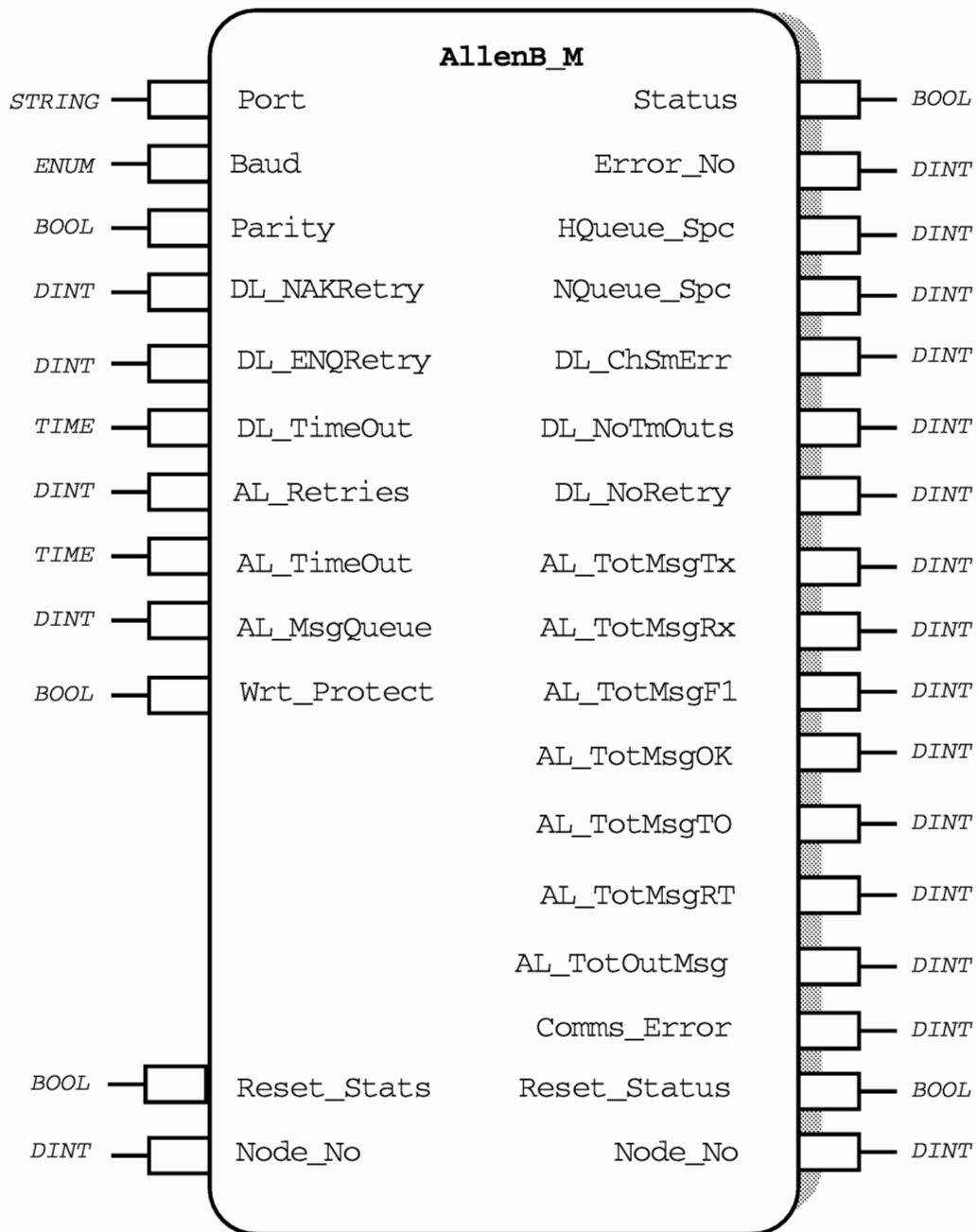
**ALLENB\_M FUNCTION BLOCK**

Figure 3-50 AllenB\_M Function Block Diagram

## Functional Description

Note: This block runs on the LCM only - it cannot be assigned to an ICM port

### Allen-Bradley Communications Protocol & Command Set Compliance

The PC3000 communications function block implementation of the Allen-Bradley Data Highway/Data Highway Plus<sup>TM</sup>/DH-485 Communication Protocol has been designed as a Master only driver with support for the full-duplex data link layer protocols.

Application layer message protocol support is a subset of three (3) command sets. The command sets supported and the functions of that command set are listed in the next sections.

#### Basic Command Set

The basic command set allows the PC3000 to communicate with PLC nodes that have been enabled to support the Allen-Bradley non-privileged commands listed below.

Non-Privileged	CMD	FNC
Unprotected read	01	N/A
Unprotected write	08	N/A

No Privileged commands are supported.

Unprotected reads and writes in a PLC allow access to the output registers of a PLC. For a PLC-5, these are the 'O' (or Output) word registers.

These commands are designed to support communications to the Output registers of any PLC-2, PLC-3, PLC-4, PLC-5, or SLC-500 which is equipped the proper Allen-Bradley communication hardware.

#### Reference:

ALLEN-BRADLEY Data Highway/Data Highway Plus <sup>TM</sup> /DH-485 Communication Protocol and Command Set, Reference Manual, Publication 1770-6.5.16 November 1991.
---



## PLC-5 Family Command Set

The PLC-5 family command set functions supported enable integers and floats to be read from and written to a PLC-5.

Message	CMD	FNC
Typed read (floats)	0F	68
Typed write (floats)	0F	67
Word Range read (integers)	0F	01
Word Range write (integers)	0F	00

The Typed and Word Range commands allow read/write data of the float (F) and integer (N) data files respectively.

The PC3000 communications function block supports logical ASCII addressing in the PLC-5 System Address message packet field.

It does **not** support logical binary addressing.

It does **not** support embedded responses. Embedded responses are response symbols transmitted within the message packet.

## PC3000 to Allen-Bradley Comms Hardware Setup & Wiring

Communications between the PC3000 and an Allen-Bradley device is accomplished through asynchronous RS-422 serial data transfer. On the PC3000, this is a serial port in an PC3000 ICM (Intelligent Communications Module) or LCM (Local Control Module).

On the Allen-Bradley network, an asynchronous serial interface port is required. Information on the required Allen-Bradley communications hardware is given for PLC-5s (using Data Highway Plus™ network) and SLC-500s (using the DH-485™ network). For more in-depth information on Allen-Bradley PLC asynchronous interface modules, contact an Allen-Bradley sales office or distributor.

Data Highway Plus™ network: Communications to the Allen-Bradley Data Highway Plus™ can be made through either of the following devices:

1. Allen-Bradley 1785-KE interface module (mounts in PLC-5 rack for RS-232 communications)
2. Allen-Bradley 1780-KF2 interface module (stand alone converter for RS-232 or RS-422 communications)

Allen-Bradley 1770-KF2 or 1785-KE settings:

- \* parity as setup by PC3000 software config tools
- \* no embedded responses
- \* no duplicate message detection
- \* no handshaking
- \* full-duplex
- \* BCC error checking
- \* baud as setup by PC3000 software config tools

RS422 Allen Bradley communications:

PC3000 LCM/ICM		Allen-Bradley 1770-KF2
pin		pin
3 Red	Tx- ----- Rx-	16
4 Black	Tx+ ----- Rx+	18
5 Green	Rx- ----- Tx-	14
6 Blue	Rx+ ----- Tx+	25

RS232 Allen Bradley communications:

PC3000 LCM/ICM	Cable	EUROTHERM 261 (RS-422 to RS-232 converter)	Cable	AB 1780-KF2 or 1785-KE	
pin		pin	pin	pin	
3 Red	Tx- ---- Rx-	16	2	Tx ----- Rx	3
4 Black	Tx+ ---- Rx+	3	3	Rx ----- Tx	2
5 Green	Rx- ---- Tx-	13	7	Com -----Com	7
6 Blue	Rx+ ---- Tx+	12			

RS-422

RS-232

Note on accessing SLC-500 information from the Data Highway Plus:

Use of an Allen-Bradley 1785-KA5 module allows routing of information between the PLC-5's Data Highway Plus network and the SLC-500's DH-485 network. However, only a PLC-5 can access information through this 'router'.

A 1785-KE or 1780-KF2 module (the port to which the PC3000 is connected) cannot use this 'router'. Thus, the PC3000 cannot directly access information from a SLC-500 PLC. A way for a PC3000 to access information in the SLC-500 is to allow a PLC-5 to act as an data transfer agent.

The PC3000 can read/write data from/to designated PLC-5 registers. Then, the SLC-500 can write/read the same data to/from the same PLC-5.

*Acceptance testing has only been performed on a PLC-5 Data Highway Plus<sup>TM</sup> network node.*

## PC3000 Software Configuration Overview

Configuration of the PC3000 to Allen-Bradley device driver requires the configuration of one 'COMMS' class function block and one or more 'REMOTE\_VARS' class function blocks with the PC3000 configuration tools.

Allen-Bradley Device Comms Setup:

To setup the Allen-Bradley device driver:

- \* Choose a COMMS class function blocks.
- \* Select an AllenB\_M type function block.
- \* Create (or edit) an **AllenB\_M** type function block. The parameters 'Port', 'Baud', 'Parity' are typically the only parameters which may require modification from the default settings.

Data Values Setup:

To setup access to a PLC value:

- \* Select a REMOTE\_VAR class function block.
- \* Select either **Rmt\_Real** (floating point) or **Rmt\_Dint**(integer) or **Rmt\_SW** (integer).  
A **Rmt\_Real** would be selected for accessing an Allen-Bradley floating point value.

A **Rmt\_Dint** would be selected for accessing an Allen-Bradley integer. **Rmt\_Real8**, **Rmt\_Real64**, **Rmt\_Dint8**, or **Rmt\_Dint64** are used if block reads are required.

A **Rmt\_SW** would be selected for accessing an Allen Bradley integer as bits.

**Rmt\_Real8 & Rmt\_dint8** support blocks up to 8 elements.

**Rmt\_Dint64** supports blocks up to 64 elements.

**Rmt\_Real64** supports blocks up to 50 elements.

- \* Enter the parameters such as 'Address', data triggering 'Mode', and data 'Refresh' rate.
- \* Accessing this function block will display the PLC value.

## Remote Parameter Function Block Detail

The **AllenB\_M** driver supports the remote parameter **Rmt\_Dint** (integer) and **Rmt\_Real** (float) function block types for single parameter reads and writes. **Rmt\_Dint8**, **Rmt\_Real8**, **Rmt\_Dint64**, and **Rmt\_Real64** remote function block types are supported for block reads.

Note: **Rmt\_SW** may be used as it is a dint displayed in Bits.

The following are the setup parameters required to configure a read or write from/to an Allen-Bradley device:

### Address

Contains a field to enter a series of characters which designate access to a remote device (in this case the Allen-Bradley PLC). The format of the field is as follows, all fields must be filled (except for size which defaults to one):

Port	Destination	Priority	Size	Identifier	Address
------	-------------	----------	------	------------	---------

If errors are made in configuration of the address, an error code number will be presented in the **Error\_No** parameter of the remote parameter function block. The meaning of the error code number is shown in the Configuration and Communication Errors page 3-354.

Note: The Address is only validated while actually communicating a read or write request to the Allen-Bradley device.

### Port

Is two characters which designate to which PC3000 physical port to attach.  
This Function Block will only run on the LCM. Ports 0A, 0B or 0C.

### Destination

Is the Allen-Bradley network Node Number which will be accessed by this function block. It is a **Octal** numerical value between 0 and 376 (254 decimal). Leading zeros are permitted, but not required.

Examples:            '5'            specifies PLC at octal address 5  
                      '034'        specifies PLC at octal address 34

### Priority

Can be set to normal ('N' or 'n') or high ('H' or 'h'). Assignment as high priority has two effects. First, within the PC3000, these parameters are moved to the front of the queue. Second, the Allen-Bradley network designates these as high priority. note: Excessive use of high priority will nullify its effectiveness, and inhibit the transmission of normal priority requests.

Examples:            'N' specifies normal priority  
                      'h' specifies high priority

### Size (optional):

Is an optional parameter to be supplied on read commands. This specifies the size of a block read. If no size is specified, a single parameter read is performed. Block reads can only be performed with remote function block types **Rmt\_Dint8** (8 value integer), **Rmt\_Real8** (8 value floating point) blocks, **Rmt\_Dint64** (64 value integer), and **Rmt\_Dint64** (50 element integer). The form of the size parameter a number (1 to 64) which specifies the size. Leading zeros are allowed, but not required.

Examples:            '7'  
                      '48'

## Identifier

Is a single character that identifies the type of command being issued. It may be one of the following:

- U (u)     Unprotected Word (read/write of PLC output value)  
          MUST be a **Rmt\_Dint** type remote function block.
- T (t)     Typed (read/write of float(F) files)  
          MUST be a **Rmt\_Real** type remote function.
- W (w)     Word Range (read/write of integer(N) files)  
          MUST be a **Rmt\_Dint** type remote function block.

The remote parameter function block determines whether a block is designated as read or write.

Examples:            'U'  
                      'w'

## Address:

Contains the data address in the PLC which is to be accessed. For a 'U' Identifier, the address field always contains a four character address. For a 'T' or 'W' Identifier, the address contains the file number, ':', element number.

Examples:	U Identifier:	'0024'	Output word 24
		'0002'	Output word 2
	W Identifier:	'N10:23'	Int file 10, element 23
		'N14:2'	Int file 14, element 2
	T Identifier:	'F13:3'	Flt file 13, element 3
		'F16:32'	Flt file 16, element 32

## Function Block Attributes

Type:..... 8E7  
 Class: ..... COMMS  
 DefaultTask: ..... Task\_2  
 Short List: ..... Port, Baud, Parity

## Parameter Descriptions

### Driver Configuration Parameters

#### Port

Is the two character address of the PC3000 LCM port on which the Allen-Bradley protocol is to be run.

example: '0C' would be port C on the LCM.

This Function Block cannot be assigned to an ICM.

#### Baud

Gives a choice of seven rates from 300 baud to 19200 as shown:

300	4800
600	9600
1200	19200
2400	

Default: 9600

#### Parity

Allows selection of communication parity as shown:

None
Even

Default: None

### DL\_NAKRetry

Sets the number of retries that will be made in transmitting a message at the Data Link Layer. Default: 3

### DL\_ENQRetry

Sets the number of Enquiries that will be made after a message has been transmitted at the Data Link Layer, but when no response has been received before the specified timeout period of parameter **DL\_TimeOut**. The enquiry will request the last response from the slave device. If no response is received within the timeout period; then, enquiries will be made up to the retry limit. Default: 3

### DL\_TimeOut

Sets the Data Link Layer Timeout between transmitting a message and receiving the response. After this timeout has elapsed, an enquiry (ENQ) will be transmitted to determine the last response from the slave. The driver then will respond according to the last response. Default: 1 second

### AL\_Retries

Sets the number of retries that will be made in transmitting a message packet at the Application Layer. A retry will be made up to the specified limit when a NAK is received from the Data Link Layer. If the number of retries is exceeded, then an error will be reported on the **Comms\_Error** output. Default: 3

### AL\_TimeOut

Sets the Application Layer timeout between transmitting a message packet and receiving the corresponding response packet. This timeout **MUST** be longer than the Maximum length of time that could be spent in the Data Link Layer. After the timeout has elapsed, a retry will be made (up to the specified limit) to transmit to the message packet again. Default: 10 seconds

### AL\_MsgQueue

Sets the number of outstanding command message packets allowed at the Application Layer. The recommended number of outstanding message is between 1 and 3. Values above this recommended limit may result in message transmission and reception errors. The maximum limit for the number of outstanding messages is 10. Default: 3

### Wrt\_Protect

Is not currently used, but has been provided for later Slave function block enhancements.



### Node\_No

Specifies the Allen-Bradley Data Highway **Octal** node address number of the network interface module (such as the 1770-KF2 or 1785-KE for a Data Highway/Data Highway Plus™ network). This parameter will automatically configure itself to the address of the connected interface module. Default: 0

### Driver Status Parameters

The driver status is indicated by six status output parameters in the **AllenB\_M** function block.

#### Status

Is a boolean (OK or NoGo) indication of the state of the communications to the Allen-Bradley device. If communications are good, 'OK' will be displayed. If error, 'NoGo' will be displayed. If the Status is NoGo, the **Error\_No** parameter will indicate the reason for the problem.

#### Error\_No

Indicates the reason for error in communications to the Allen-Bradley device. If Status is 'OK', **Error\_No** will be 0. If error exists (Status is NoGo), a number will be displayed indicating the cause. These cause numbers are listed in section 9.

#### Hqueue\_Spc

Indicates the amount of space left in the queue for High Priority remote parameter operations. If **Hqueue\_Spc** reaches zero, the serial communications bandwidth is not sufficient to cope with the number of High Priority remote parameters requests being made. In this case, High Priority remote parameter request will be added to the Normal Priority queue. If this situation arises, the number of High Priority remote parameter requests should be reduced through a PC3000 configuration change.

#### Nqueue\_Spc

Indicates the amount of space left in the queue for Normal Priority remote parameter operations. If **Nqueue\_Spc** reaches zero, the serial communications bandwidth is not sufficient to cope with the number of remote parameter request being made, and data will be lost. If this situation arises, the parameter polling rates should be reduced through a PC3000 configuration change.

#### AL\_TotOutMsg

Indicates the total number of outstanding messages that are currently being actioned by the Allen-Bradley driver block. When a message is actioned by the Allen-Bradley driver, it is removed from the corresponding Remote Parameter queue and placed on the Outstanding Message queue.

### Comms\_Error

Indicates the status of the last transaction executed by the Allen-Bradley driver. If there was some error detected, this will be reflected here. If many transactions are occurring, then any error detected and reported will remain here only until the next transaction is completed.

### Driver Statistical Parameters

Driver statistical parameters keep running totals errors, timeouts, transmissions, and receptions associated with the PC3000 to Allen-Bradley device communications. These statistics are indicated in ten output parameters.

### Reset\_Stats

Resets all statistical information outputs. This is done by setting this to 'YES'. The parameter automatically changes to 'NO' once the statistical outputs have been reset.

### DL\_ChksmErr

Indicates the total number of checksum errors that have been encountered during message transmission at the Data Link Layer.

### DL\_NoTmOuts

Indicates the total number of timeouts that have been encountered during message transmission at the Data Link Layer.

### DL\_Retry

Indicates the total number of retries that have been made due to errors or timeouts in message transmission at the Data Link Layer.

### AL\_TotMsgTx

Indicates the total number of message packets that have been transmitted at the Application Layer.

### AL\_TotMsgRx

Indicates the total number of message packets that have been received at the Application Layer.

### AL\_TotMsgFl

Indicates the total number of message packet transmission failures that have occurred at the Application Layer.

**AL\_TotMsgOK**

Indicates the total number of message packet transmissions that have been sent successfully by the Application Layer.

**AL\_TotMsgTO**

Indicates the total number of timeouts encountered in transmitting message packets from the Application Layer.

**AL\_TotMsgRT**

Indicates the total number of retries that have been made due to errors or timeouts in message packet transmission at the Application Layer.

## Configuration and Communication Errors

Errors in the PC3000 configuration or communications to the Allen-Bradley PLC are indicated in the Status parameter of the remote function block. The **Error\_No** parameter in the remote function block gives a number which indicates the cause. The following is a list of all possible errors:

1	LCM/ICM ERROR: NO ADDRESS
2	LCM/ICM ERROR: ILLEGAL DATA BITS
3	LCM/ICM ERROR: ILLEGAL PARITY
4	LCM/ICM ERROR: ILLEGAL STOP BITS
5	LCM/ICM ERROR: RX BAUD RATE NOT AVAILABLE
6	LCM/ICM ERROR: TX BAUD RATE NOT AVAILABLE
7	LCM/ICM ERROR: RTS NOT AVAILABLE
8	LCM/ICM ERROR: CTS NOT AVAILABLE
9	LCM/ICM ERROR: DUPLEX MODE NOT AVAILABLE
10	LCM/ICM ERROR: TX DISABLE NOT AVAILABLE
11	LCM/ICM ERROR: ADDRESS ERROR ILLEGAL SLOT The first parameter in the Address parameter is not in the valid range of '0' to '5'.
12	LCM/ICM ERROR: ADDRESS ERROR ILLEGAL PORT The second character in the Address parameter is not in the valid range of 'A' to 'C' for an LCM port or 'A' to 'D' for an ICM port.
13	LCM/ICM ERROR: BREAK ON
14	LCM/ICM ERROR: RX TX INCOMPATIBLE
15	LCM/ICM ERROR: ILLEGAL INITIALIZATION TYPE
16	LCM/ICM ERROR: ADDRESS ERROR NO REMOTE PARAMETER SERVICE The port given in the Address parameter does not have a suitable master driver allocated to it.
17	LCM/ICM ERROR: PORT IN USE
18	LCM/ICM ERROR: TOO MANY SLAVE PARAMETERS
19	LCM/ICM ERROR: TOO MANY DRIVER TYPES
20	LCM/ICM ERROR: TOO MANY REMOTE PARAMETERS
21	LCM/ICM ERROR: INTER BOARD
22	LCM/ICM ERROR: UNSUPPORTED FUNCTION BLOCK TYPE

23	LCM/ICM ERROR: ALREADY TRANSFERRED
24	LCM/ICM ERROR: WRITE PROTECTED
25	LCM/ICM ERROR: BOARD FAILURE
26	LCM/ICM ERROR: MEMORY
27	LCM/ICM ERROR: ABORT DUE TO HALT
70	DLL CHECKSUM ERROR There has been a checksum error detected at the Data Link Layer reception.
75	DESTINATION NODE INVALID The Destination Node entered in the remote parameter address is incorrect.
76	COMMAND PRIORITY INVALID The command priority character entered in the remote parameter address is incorrect.
77	READ BLOCK SPECIFIED IS TOO LARGE The multi-element remote block specified was larger than allowed.
78	BLOCK READ RESPONSE FOR SINGLE READ REQUEST The Allen Bradley device returned a multi-valued response for a single-valued request.
79	INVALID IDENTIFIER SPECIFIED An invalid identifier has been specified in the remote parameter address.
81	INVALID 'U' ADDRESS An invalid address has been specified for an Unprotected Read/Write command in the remote parameter address.
84	DATA VALUE TOO BIG OR SMALL FOR FLOAT The value of data is greater than 3.4E38 or less than -3.4E38.
87	APL TIMEOUT WHILE SENDING MESSAGE A timeout at the application layer occurred while sending a command message.
91	APL FAILED TO GET RESPONSE No response was received at the application layer to a command message sent.
92	INVALID WRITE ADDRESS The PLC-5 Logical ASCII address specified was invalid.
93	DATA VALUE TO LARGE FOR INTEGER The integer data value specified was larger than allowed.
94	INVALID REMOTE FUNCTION BLOCK TYPE The remote function block type is invalid for this type of command.

95	FLOAT VALUE TOO BIG OR SMALL The floating point value is greater than 3.4E38 or -3.4E38.
102	DLL SUSPECT DATA DELIVERY Cannot guarantee delivery at the DLL.
103	DUPLICATE TOKEN Duplicate token holder detected.
104	LOCAL PORT DISCONNECTED No communications is available
105	APL TIMEOUT WAITING FOR RESPONSE Application layer timed out waiting for a response.
106	DUPLICATE NODE DETECTED
107	STATION IS OFF-LINE
108	LOCAL HARDWARE FAULT
116	ILLEGAL COMMAND OR FORMAT
117	HOST CANNOT COMMUNICATE Host has a problem and will not communicate.
118	REMOTE NODE HOST NOT OPERATING Remote node host is missing, disconnected, or shutdown.
119	REMOTE HARDWARE FAULT Host could not complete function due to hardware fault.
120	ADDRESSING PROBLEM Addressing problem or memory protect rungs.
121	INVALID FUNCTION Function disallowed due to command protection selection.
122	PROCESSOR IN PROGRAM MODE
123	MISSING FILE OR ZONE PROBLEM Compatibility mode file is missing or communication zone problem.
124	REMOTE NODE BUFFER FULL Remote node cannot buffer command.
126	REMOTE NODE PROBLEM Remote node problem due to download.
127	CANNOT EXECUTE COMMAND Cannot execute command due to active IPBs.
128	LCM/ICM ERROR: SYSTEM ERROR

131	ILLEGAL VALUE IN FIELD A field contains an illegal value.
132	ADDRESS TOO SMALL Less levels specified in address than the minimum for any address.
133	ADDRESS TOO LARGE More levels specified in address than system supports.
134	SYMBOL DOES NOT EXIST Symbol not found.
135	INVALID SYMBOL Symbol is of improper format.
136	INVALID ADDRESS Address does not point to something useable.
137	INVALID FILE SIZE File is the wrong size.
138	COMMAND CANNOT BE COMPLETED Cannot complete request. Situation has changes since the start of the command.
139	DATA OR FILE TOO LARGE
140	TRANSACTION SIZE AND ADDRESS TOO LARGE Transaction size plus word address is too long.
141	ACCESS DENIED Access denied, improper privilege.
142	CONDITION CANNOT BE GENERATED Condition cannot be generated - resource not available.
143	CONDITION EXISTS Condition already exists - resource is already available.
144	COMMAND CANNOT BE EXECUTED
145	HISTOGRAM OVERFLOW
146	NO ACCESS
147	ILLEGAL DATA TYPE
148	INVALID PARAMETER OR DATA
149	ADDRESS REFERENCE EXISTS TO DELETED AREA
150	COMMAND EXECUTION FAILURE Command execution failure for unknown reason. Possible PLC-3 histogram overflow.

151	DATA CONVERSION ERROR
152	SCANNER CANNOT COMMUNICATE Scanner is not able to communicate with 1771 rack adapter.
153	ADAPTER CANNOT COMMUNICATE Adapter cannot communicate with module.
154	INVALID RESPONSE FROM 1771 MODULE
155	DUPLICATE LABEL
156	FILE ALREADY OPEN File is open. Another node owns it.
157	PROGRAM OWNED BY ANOTHER NODE Another node is the program owner.
254	LCM/ICM ERROR: QUEUE FULL
255	LCM/ICM ERROR: REMOTE PARAMETER DESCRIPTOR NOT READY TO BE RETURNED



---

**APPENDIX A      GLOSSARY OF TERMS**

<b>ASCII</b>	American Standard Code for Information Interchange - the character set used within the PC3000 system. (see Appendix D for full ASCII character set)
<b>AWG</b>	American Wire Gauge
<b>Baud</b>	The number of line signal state changes per second for sending serial information.
<b>Block Check Character(s), BCC</b>	One or more characters that contain a value that is created by using an algorithm such as summation or longitudinal parity, which is applied to bytes within a communications message. The BCC is usually added to the end of a transmission message. It can then be used by the receiving device to cross-check integrity of the message by re-running the algorithm applied to the same bytes within the message.
<b>Channel Identity</b>	Within some protocols it is possible to address a sub-set of parameters using a channel identity. With EI Bisync a channel identity (CHID) as a single printable character is sometimes required.
<b>Control characters</b>	Certain characters within the ASCII character set, that are used to frame messages or to signal control information between devices. The particular control characters used will depend on the communications protocol used. Examples are :<escape> 1BH.
<b>EI Bisync</b>	A serial protocol adopted by the Eurotherm International group of companies for use with a wide variety of instruments. For a detailed definition of the standard refer to 'Eurotherm International Bisync Handbook EI HP022047C' and 'EI Bisync Communications Protocol Extensions EI HP024113'.
<b>ESP</b>	Eurotherm Supervisory System which runs on a PC.
<b>Full Duplex</b>	When operating in this mode, a device can be transmitting and receiving data simultaneously. This normally implies that there is a separate media for the transmission and reception of data.
<b>GID</b>	Group Identity, a byte used within the EI Bisync protocol to address a group of Slave devices which used together with the Unit Identity (UID) forms a Slave device communications address.
<b>Half Duplex</b>	This mode of operation implies that at any time a device can either transmit or receive data; simultaneous transmission and reception is not possible.

<b>ICM</b>	Intelligent Communications Module - a general purpose PC3000 communications module that provides 4 user configurable special ports.
<b>LCM</b>	Local Control Module - the main processing module of a PC3000 system that provides 3 user configurable serial ports.
<b>Master</b>	A master device is able to initiate a request to read or write information from a remote device, and poll for information from a number of remote devices. (see Slave).
<b>Mnemonic</b>	A short text abbreviation which can be used within a protocol to address a particular item, function or parameter within a device. With the EI Bisync protocol, a mnemonic can be any pair of letters and numbers, e.g., 'SP', 'PV', P1, '23'.
<b>Multi-Dropped</b>	A serial communications network in which there are many Slave devices connected to a single Master device.
<b>Parity</b>	A check bit added to a byte for serial transmission to enhance error detection. Odd parity implies that there is always an odd number of bits transmitted for each byte; even parity implies an even number of bits are always transmitted.
<b>Peer-to-peer</b>	This type of protocol allows any device connected to a communications network to send information to, and request information from any other device on the network.
<b>Polling</b>	A method of gathering information from a series of remote devices connected via a serial link, by requesting information from each one in turn. This implies that the polling device is 'Master' of the link.
<b>Point-to-point</b>	A serial link used to only connect two devices
<b>Port</b>	A physical connection point on the PC3000 to which remote devices can be connected using a serial link.
<b>PLC</b>	Programmable Logic Controller
<b>Protocol</b>	Defines the rules by which two communicating devices exchange information. This includes the methods used to package and encode information being sent along a serial link.
<b>PS</b>	The PC3000 Programming Station
<b>Remote Device</b>	This is a general term for any device which is communicating to the PC3000.
<b>RS232, RS422, RS485</b>	These refer to three different standards used for electrically signalling information on a serial communications link .

<b>Serial Comms</b>	A communications system where a stream of characters is sent serially, i.e. the data is broken down into a series of bits which are sent one after the other down a serial link.
<b>Serial Link</b>	This is a general term that refers to the physical media used to connect two communicating devices and can be a single wire, a twisted pair of wires, fibre optic cables etc.
<b>SFC</b>	Sequential Function Chart
<b>Slave</b>	A device configured so that it can respond to requests to read or write information from a 'Master' device but can never initiate communications requests.
<b>Stop Bits</b>	A delay measured in terms of the time to transmit a bit, which is inserted after transmitting a character. It is used to ensure a minimum time is left between the transmission of consecutive characters and therefore delimits each character.
<b>Time-out</b>	This is a technique used in most protocol drivers to detect an error condition by measuring the time for an event, such as the arrival of a message, to occur. If the time exceeds a set value, the event is said to have timed-out.
<b>Transaction</b>	A general term for an exchange of information between devices and can imply a read or write operation.
<b>UID</b>	Unit Identity, part of the EI Bisync Slave address which is used with the GID.

## APPENDIX B STANDARD COMMUNICATIONS ERROR CODES

The following codes apply to all drivers. Additionally, error codes will be used for all drivers when protocol specific error codes are required.

Code	Error Condition	Slave Driver	Slave Variable	Master Driver †	Remote Variable
0	OK	*	*	*	*
1	ADDRESS STRING TOO SHORT	*	*	*	*
	Port address too short				
	Remote address too short				
	Slave address too short				
2	ILLEGAL DATABITS	*	*	*	*
3	ILLEGAL PARITY	*	*	*	*
4	ILLEGAL STOP BITS	*	*	*	*
5	RX BAUD RATE NOT AVAILABLE	*	*	*	*
6	TX BAUD RATE NOT AVAILABLE	*	*	*	*
7	RTS NOT AVAILABLE	*	*	*	*
8	CTS NOT AVAILABLE	*	*	*	*
11	ILLEGAL SLOT the first character in the Address parameter is not in the valid range of '0' to '5'.	*	*	*	*
12	ILLEGAL PORT the second character in the Address parameter is not in the valid range for the module, for example 'A' to 'C' for an LCM port or 'A' to 'D' for an ICM port.	*	*	*	*

Code	Error Condition	Slave Driver*	Slave Variable	Master Driver †	Remote Variable
14	RX TX INCOMPATIBLE	*	*	*	*
16	NO REMOTE VARIABLE SERVICE communications driver does not support remote vars.	*	*	*	*
17	PORT IN USE	*	*	*	*
18	TOO MANY SLAVE VARIABLES		*		
19	TOO MANY DRIVER TYPES		*		

Note: †These errors are also applicable to Raw\_Comms. driver function block.

These error codes may be offset to a higher set of values for some drivers in order not to clash with standard error codes associated with the particular protocol. For example, error 1, ADDRESS STRING TOO SHORT is offset to 145 for the JBus\_Slave driver.

## APPENDIX C ASCII TABLE

The ASCII code table is given general reference.

### ASCII Table Hexadecimal - Character

00	NUL	01	SOH	02	STX	03	ETX	04	EOT	05	ENQ	06	ACK	07	BEL
08	BS	09	HT	0A	NL	0B	VT	0C	NP	0D	CR	0E	SO	0F	SI
10	DLE	11	DC1	12	DC2	13	DC3	14	DC4	15	NAK	16	SYN	17	ETB
18	CAN	19	EM	1A	SUB	1B	ESC	1C	FS	1D	GS	1E	RS	1F	US
20	SP	21	!	22	*	23	#	24	\$	25	%	26		27	'
28	(	29	)	2A	*	2B	+	2C	,	2D	-	2E	.	2F	/
30	0	31	2	32	2	33	3	34	4	35	5	36	6	37	7
38	8	39	9	3A	:	3B	;	3C	<	3D	=	3E	.	3F	
40	@	41	A	42	B	43	C	44	D	45	E	46	F	47	G
48	H	49	I	4A	J	4B	K	4C	L	4D	M	4E	N	4F	O
50	P	51	Q	52	R	53	S	54	T	55	U	56	V	57	W
58	X	59	Y	5A	Z	5B	[	5C	\	5D	]	5E	-	5F	_
60	'	61	a	62	b	63	c	64	d	65	e	66	f	67	g
68	h	69	i	6A	j	6B	k	6C	l	6D	m	6E	n	6F	o
70	p	71	q	72	r	73	s	74	t	75	u	76	v	77	w
78	x	79	y	7A	z	7B	{	7C		7D	}	7E	-	7F	DEL