EUROTHERM

PC3000 REAL TIME OPERATING SYSTEM

Handbook

Printed in England 0199 HA022918 Issue 3.1

CONTENTS

PREFACE

This reference manual provides detailed information on the PC3000 Real Time Operating System software that manages the loading, initialisation and execution of a User Program. User Programs are created and downloaded from a PC3000 Programming Station, i.e. PS or Microcell into the PC3000 control system.

You are advised to gain a basic understanding of the PC3000 programming techniques including the use of function blocks as detailed in the PC3000 PS User Guide or PC3000 Microcell User Guide before reading this manual.

The information provided will allow you to understand:

- How to develop User Programs that make full use of the PC3000 performance.
- How to design multi-tasking programs and efficiently exchange information between tasks.
- How to diagnose faults and take recovery action.
- How to use system facilities, such as the Real Time Clock, that are provided in the PC3000.
- How PC3000 memory is organised.

For further information refer to the PC3000 Reference documents:

PC3000 Hardware Reference - provides detailed information on all the PC3000 hardware modules including calibration, wiring and physical configuration details. Part No. HA022919

PC3000 Functions Reference - describes all the functions that can be called within the Structured Text (ST) language. Part No. HA022916

PC3000 Function Block Reference - describes the numerous function blocks available to be incorporated into your control program for PID control, Ramps, Counters, Filters, Timers etc. Part No.HA022917

Chapter 1

INTRODUCTION

Contents

Overview

This overview provides a general description of the main structure and functions of the PC3000 system software. After creating a User Program on a PC3000 Programming Station, the compiled User Program is downloaded into the PC3000 Local Controller Module (LCM) and stored in non-volatile (battery backed) memory. The LCM is always the first module in the main PC3000 rack and is responsible for starting and running the User Program.

The LCM has an interface into the rest of the system provided by backplane buses, i.e. Local Bus (LBus), Parallel Interface Bus (PiBus) and a Serial Interface Bus (SiBus).

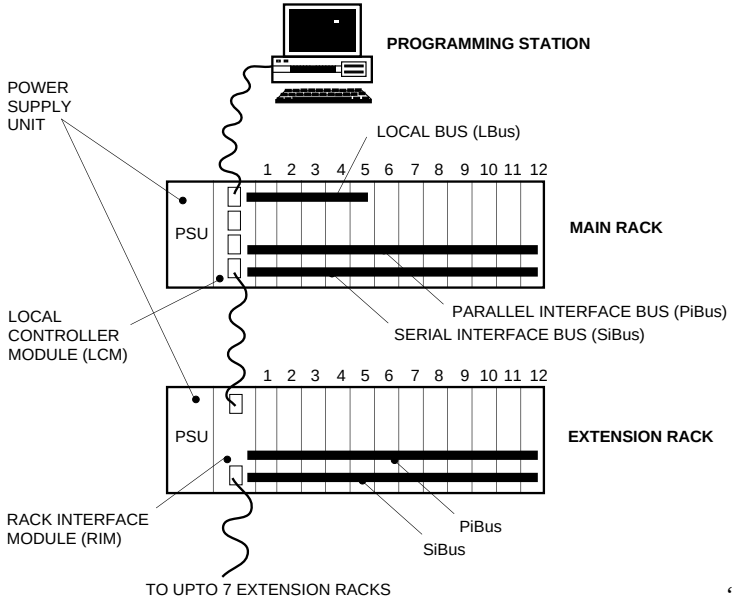

Figure 1-1 PC3000 Hardware overview

LBus

LBus is provided to interface with modules that have a requirement to exchange large quantities of data with the LCM. Examples of such modules are; the Intelligent Communications Module (ICM) that supports 4 high speed serial communications ports and the Eurotherm Network Module (ENM) that provides a high-speed peer-to-peer network for PC3000 and Production Orchestrator communications.

PiBus

PiBus is provided to rapidly read from and write to digital I/O modules such as the 14 channel Digital Input or 12 channel Relay Output modules.

SiBus

SiBus is used to exchange status, and parameter values with modules such as Analogue Input and Analogue Output modules AOM4.

The LCM also has a serial interface to a Rack Interface Module (RIM). RIMs can be daisy-chained to add up to 7 extension racks. For further details on hardware structure you should refer to the PC3000 Hardware Reference.

The LCM also has three serial communications ports A, B and C. Port B is normally reserved for connection with the Programming Station. The two remaining ports can be used for application specific communications; for further details on these ports refer to the PC3000 Communications Overview document.

The functions of the LCM are shared between two processors: a 68000 based main processor and an I/O Concentrator (IOC) which supports the I/O sub-system. The I/O System is concerned with reading input sensor values, for example, from thermocouples and writing values out to actuators, such as valve positioners. The 68000 is the main processing engine which executes the User Program. It has a high-speed interface with the IOC for the rapid exchange of I/O data with the I/O modules. The 68000 also has a 68881 co-processor that supports high speed floating point (decimal) calculations. Most of the system software described in this manual runs on the 68000 which is at the core of the PC3000 real time system.

I/O Concentrator

The IOC gathers I/O sensor input values from the hardware I/O modules for access by the User Program and writes out new output values to the I/O modules as generated by the running program to drive actuators, heaters etc. It communicates with the I/O modules directly via the SiBus and PiBus interfaces.

When a User Programs first initialised to be run, software within the LCM generates a list of I/O inputs to be read and written. This “I/O shopping list” depends on the I/O

requirements of a particular User Program. The list of I/O inputs and outputs are organised to be read in two different scans. Nominally, fast inputs and outputs are scanned every 10ms while slower analogue inputs are scanned every 100ms. The transactions with the slower I/O modules are spread out through the 100ms period so that a different small set of the slower transactions coincide with each 10ms scan. These scan times can be modified; this is discussed in the Chapter Real Time Task Scheduler.

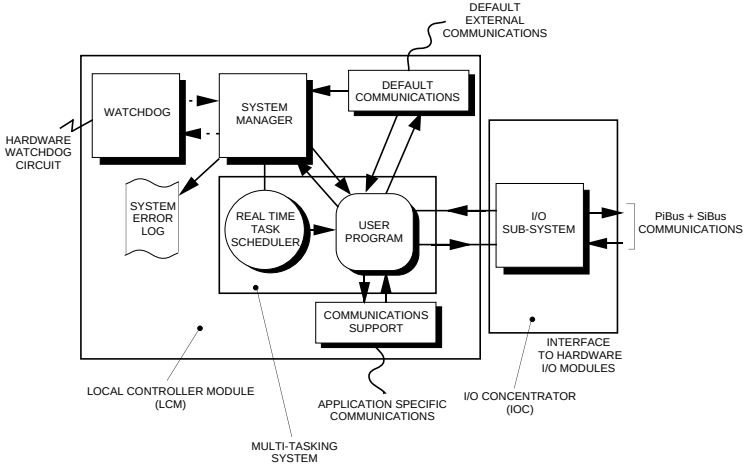

Figure 1-2 System architecture overview

When a User Program is loaded into the LCM, it requires services and support from the system to manage its start-up and execution. The program also requires an interface with the PC3000 I/O system and with the communications support software. All the components of this support infra-structure run on the LCM and are depicted in the System Architecture Overview figure 1-2.

Real time task scheduler (scheduler)

Real Time Task Scheduler is responsible for the timely execution of the User Program. It manages the synchronisation between the execution of tasks within the User Program and the gathering of input values from the I/O system and the dispatch of new values to outputs via the I/O system. The relationship between the User Program and the Scheduler is depicted in the Multi-tasking System figure 1-3.

System manager

System Manager provides a wide range of services and management functions to support the interface between the Program and the PC3000 system including:

1. Loading of the User Program into memory
2. Control of the PC3000 operational modes
3. Management of program start-up and shut-down
4. Detection and reporting of system errors in a system error log
5. Maintenance of regular trigger signals to an external hardware watchdog
6. Interface with the Real Time Clock

Default communications

Default Communication is provided to access the system manager and User Program for system administration and diagnostic purposes and is used by the PC3000 Programming Station. It is referred to as “default” because it is always available on LCM ports A,B and C, for ports that are not dedicated to a specific protocol in a User Program. For example, if a communications driver function block, such as JBus_S (J-Bus® slave), is assigned to port A, Default Communications will not be available on port A while the User Program is running. However, whenever the User Program is not ”RUNNING”, Default Communications is always provided on ports A, B and C.

The Default Communications. provides the Eurotherm EI Bisync protocol in slave mode operating at 9600 Baud. For further details refer to the PC3000 Communications Overview document.

Communications support

Communications Support is provided for a wide range of different protocols such as J-Bus®, Siemens 3969®. Most of the low level protocol details are dealt with by the communications support software, so that a detailed knowledge of communications protocols is not required. A specific protocol for a port on the LCM or on an Intelligent Communications Module (ICM) can be provided by creating an instance of the required communications driver function block in the User Program and assigning the port via the function block’s **Port** parameter. For further details refer to the PC3000 Communications Overview or the specific communications function blocks in the PC3000 Function Block Reference.

Multi-tasking system

The Real Time Task Scheduler shown in the Multi-tasking system figure 1-3, provides the PC3000 with the ability to execute a User Program that has a number of tasks running in parallel.

Note: PC3000 supports two forms of parallel execution.

A part of the User Program which may include a number of function blocks and the Sequential Function Charts (SFCs) can be controlled by a single task. For example, a collection of function blocks can be configured to execute every 20ms.

It is also possible within an SFC to have a number of sequences that are executed in parallel.

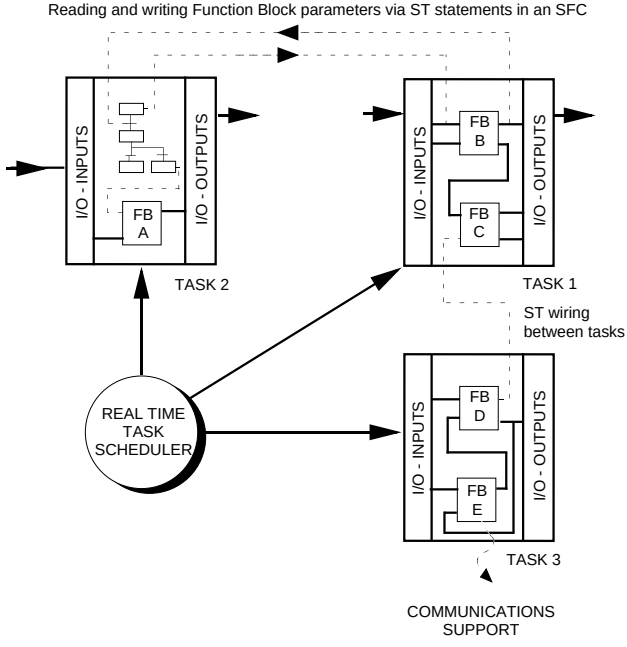

Figure 1-3 Multi-tasking system

The function blocks within the User Program are assigned to be under the control of a designated task. By default, the PC3000 Programming Station arranges function blocks to be in either a 10ms or 100ms task. Function blocks dealing with purely digital parameters such as digital I/O function blocks, are nominally run in the 10ms task, while function blocks associated with analogue I/O or floating point computation and the Sequential Function Chart (SFC) are run in the 100ms task. However, these are only default task assignments. It is possible to assign function blocks to other tasks or to create more tasks, if there are special timing or performance requirements. Additional tasks can be specified by creating Task function blocks when the User Program is being developed on the PC3000 Programming Station. Refer to Chapter 7 Real Time Task Scheduler for further details.

Figure 1-3 shows an example User Program consisting of three tasks. Each task is associated with a number of I/O input and output values defined by the I/O parameters accessed by the function blocks assigned to the task.

Each task has an interval, defined by the corresponding task function block's **Interval** parameter which defines how often the task should be executed. The Scheduler is designed to re-run each task at the required interval so that User Program execution is **deterministic**. For example, the period between sampling an analogue input and updating an associated output for a PID loop is always the same.

However, note that the actual task interval may be affected by the loading on the system and the execution time and priority of other tasks. If a task is not executed within the required deadline, the task function block will report a task overrun; this is regarded as an overload condition.

Inter-task parameter transfer

ST wiring can be used to transfer parameters between function blocks assigned to different tasks. ST statements in Step bodies and in Transitions can be used to read and write parameters from function blocks in different tasks.

Because there is no inter-task buffering of parameters the following principles should be considered when transferring parameters between function blocks in different tasks.

1. The value of input parameters of function blocks in a lower priority task may change while the task is executing, if the parameter values come from function blocks within higher priority tasks
2. The value of input parameters of function blocks in a higher priority task cannot be changed by function blocks in a lower priority task while the higher priority task is executing.

Further principles and restrictions regarding the interaction of tasks and parameter passing are described in detail in Chapter 7, Real Time Task Scheduler.

Chapter 2

MODES OF OPERATION

Contents

Operating modes

The PC3000 real time system can be in one of a number of modes. The principal operating modes are :

IDLE - This mode exists when the PC3000 real time system does not have a valid User Program to execute. This may be because no program has been down-loaded or the last loaded program has become invalid or cleared.

RESET - This mode exists when a User Program is loaded and has been initialised, ready to be executed. All function block input parameters are reset to their “Cold Start” values.

HALTED - This mode exists when a User Program is not executing and function block input parameters are not set to their “Cold Start” values.

RUNNING - This is the normal mode when the PC3000 is running a user program. The function blocks and sequences defined by the Sequential Function Charts (SFCs) are executed.

SEQ_HELD - This mode exists when the Sequences (i.e. SFCs) have been held, i.e. their execution has been suspended. However, the function blocks and associated wiring continue to execute.

There are also a number of transient modes which exist while the PC3000 is starting-up, powering down, being loaded etc. These modes are not normally visible on the Programming Station but are defined here for diagnostic and background information purposes.

INITIALISING - This mode exists while the User Program is being prepared for initial execution. During this period, the program is verified and function block input parameters are reset to their ”Cold Start” values.

LOADING - This mode exists while the Programming Station is downloading a User Program into the LCM memory.

STOPPING - This mode exists while the User Program is stopping execution. All tasks are closed down in an orderly fashion. This mode persists until the longest duration task has stopped. For example, if the longest duration task is 10 secs., this mode may exist for up to 10 secs.

PRE-RUNNING - Prepares the system to begin User Program execution. During this period, communications function blocks are associated with Slave Variable function blocks.

SHUTDOWN - A mode that exist between receiving the power failure signal from the power supply, and the power supply ceasing. External communications are disabled during this period.

Changing modes

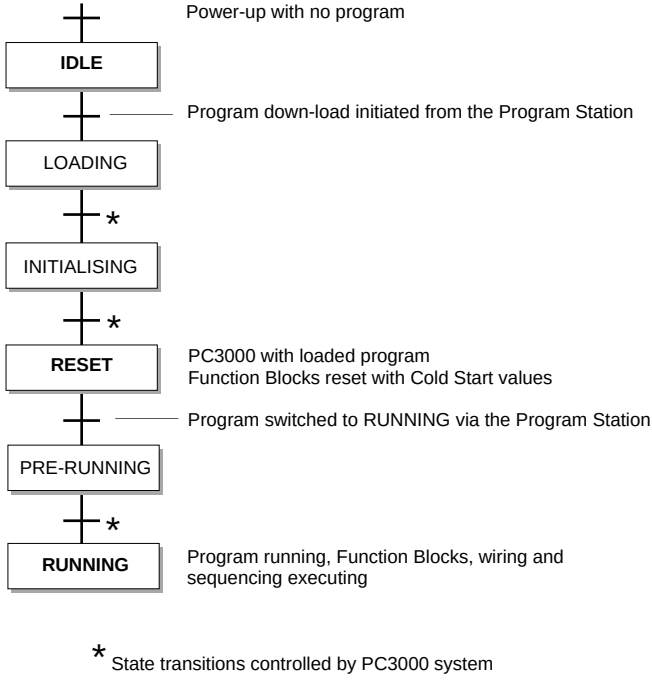

Figure 2-1 Running a program

The normal mode changes to load and run a program are shown in figure 2-1. Program down-load and switching a program into RUNNING mode require commands to be issued from the Programming Station.

> **Note:** Mode changes are made by the Programming Station via the Default Communications by writing to the PC3000 system mode parameter (EI Bisync mnemonic MN).

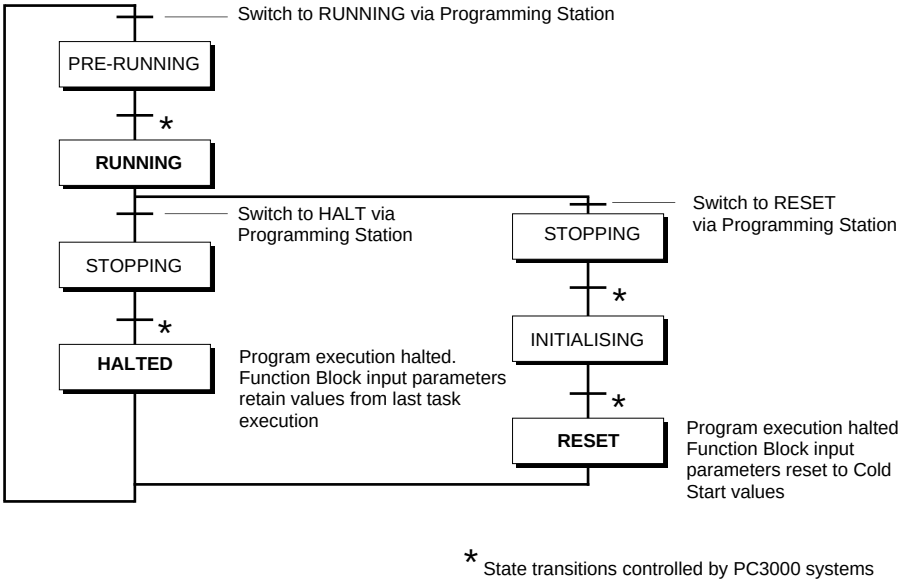

Figure 2-2 Changing the mode of a running program.

Figure 2-2 shows the two main mode changes from a RUNNING a program. By issuing a command to switch to HALT mode, the program execution ceases and enters the transient STOPPING mode before entering the HALTED mode; function block input parameters retain values from the last task execution. By issuing the command to RESET the program, the PC3000 enters transient modes STOPPING and INITIALISING before entering the RESET mode. The function block input parameters are set to their Cold Start values.

If any function block input parameter is modified via the external communications while the PC3000 is in the RESET mode, the mode is automatically changed to HALTED. The mode can be switched back to RESET by issuing a RESET command from the Programming Station.

For example, if the program has been reset and the PC3000 is in the RESET mode, changing the value such as a PID proportional band **Prop_Band** parameter via the Programming Station, will cause the mode to change to HALTED. Changing parameter values from their normal Cold Start values may be useful when it is necessary to have alternative start-up values.

Halting sequence execution

By switching the PC3000 into SEQ_HELD mode, it is possible to allow the Function Blocks and associated wiring to continue execution, but to halt sequence execution. This mode is provided for commissioning purposes; for example, when there is a requirement to evaluate PID loop performance with the sequencing part of the program disabled. SEQ_HELD mode can also be useful if there is a need to temporarily halt the sequencing to examine the program state to check that the program is behaving correctly.

It is possible to switch into SEQ_HELD by issuing a command from the Programming Station. The effect of issuing this command while PC3000 is in one of the many modes is shown in figure 2-3.

Sequencing always halts after completely executing all currently active steps.

> **Note:** The step duration parameters (.T) are not updated while the sequence is halted. Therefore placing the PC3000 in SEQ_HELD mode, if only temporarily, will cause timing derived from step durations to be inaccurate.

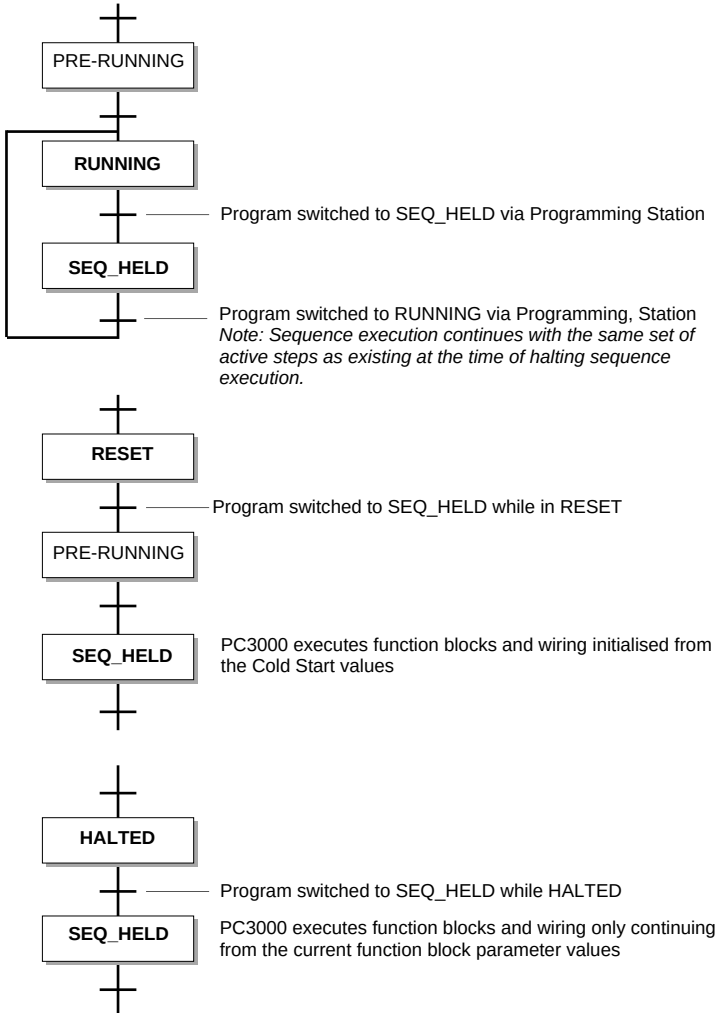

Figure 2-3 Halting sequence execution

Clearing the user program

While the PC3000 is in RESET, HALTED or LOADING modes, it is possible to clear the User Program by issuing a clear User Program command from the Programming Station. The PC3000 mode will then change to IDLE.

Initialisation and power-up tests

When PC3000 powers-up a wide range of self-diagnostic system confidence checks are made to ensure that the system is sound.

These checks include :

a) Memory

b) Hardware handling serial communications

c) System watchdog

d) System timing

e) Floating point co-processor

f) User Program verified via a checksum

If the LCM and IOC have initialised successfully, the two green indicators on the LCM module should be lit within 5 seconds after power-up, otherwise, consult the PC3000 Hardware Reference.

If a fatal hardware error prevents the system from running, an encoded error number will be signalled on the LCM system indicators. In which case, PC3000 will not enter an operating mode.

If a non fatal hardware error is detected, the LCM system indicators will signal an error number once, and then the PC3000 will enter an operating mode.

The following hardware errors are non-fatal :

a) Real Time Clock crystal or processor timing inaccurate.

b) Communications ports B or C faulty.

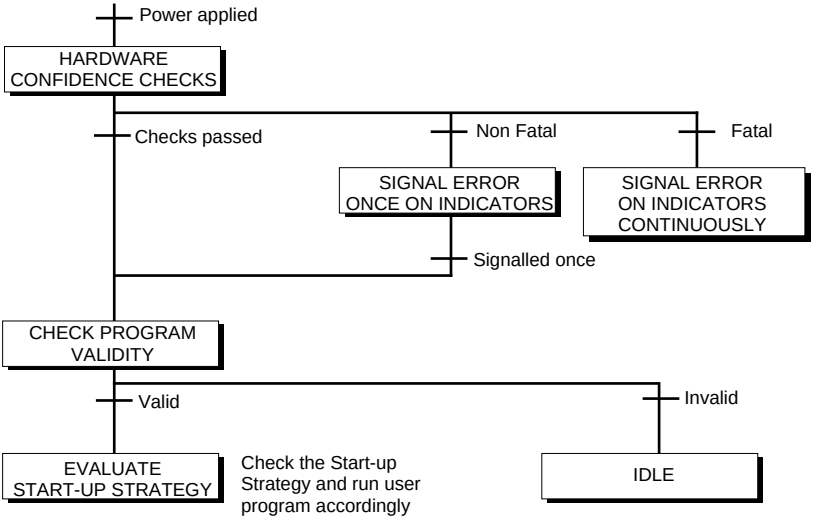

Figure 2-4 Initialisation and power-up

After completing the hardware checks, the validity of the loaded User Program is checked. If valid, the Start-up Strategy is evaluated and if appropriate, the User Program is executed. Normally this should occur within 5 seconds after power-up. If the program is not valid, has been cleared, or a program has never been loaded, the PC3000 will enter the IDLE mode.

Start-up strategy

The PC3000 provides a range of Start-up Strategies which define the behaviour of the User Program when power is restored after a power failure.

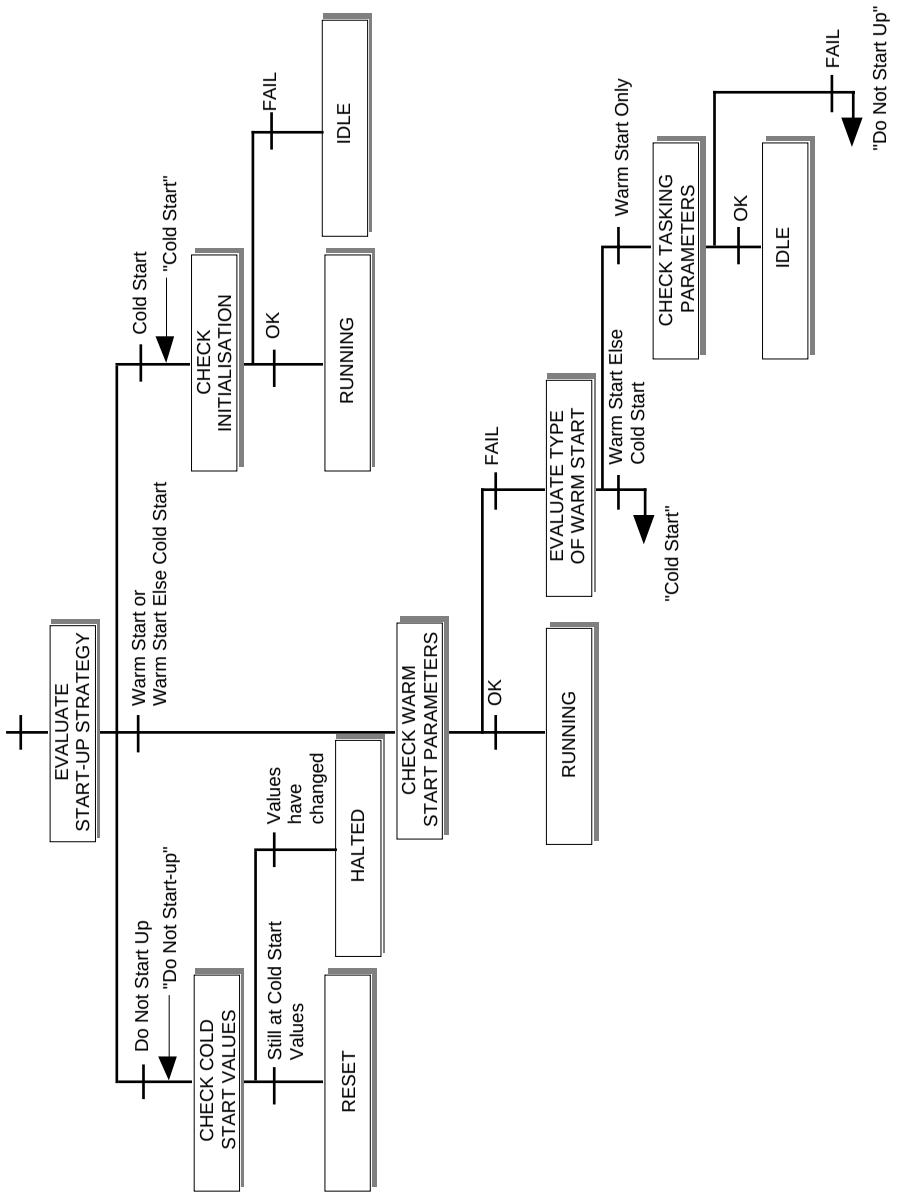

Figure 2-5 Start-up strategies

The strategies are selected when a program is created on the Programming Station by selecting a value for the **Start_Strat** parameter of the PcsSTATE function block.

Strategies on power-up are :

Do Not Start-Up -

The User Program is not run. If a program is loaded and function block parameters have been changed since the program was first initialised, PC3000 will enter the HALTED mode, otherwise it will enter the RESET mode.

Cold Start -

The User Program is initialised, function block input parameters are reset to their cold start values and the program is run. If the program is not valid, PC3000 will enter the IDLE mode.

Warm Start -

The User Program continues execution with function block parameters having the values retained from power-fail; the sequencing continues from the last set of active steps. If warm start fails, PC3000 will enter either RESET, HALTED or IDLE modes.

Warm Start Else Cold Start -

This is the same as Warm Start strategy except if checks within the warm start fail, instead of entering RESET, HALTED or IDLE, a Cold Start (as defined) is attempted.

Warm Start within Downtime else Cold Start -

The downtime, i.e. time for which power has been off is checked against the maximum acceptable downtime as defined in the **OK_Down_Time** param eter of the PcsSTATE function block. If the power has been off for less than this period, a warm start is attempted. If it is not successful, or if the downtime is greater than the acceptable period, a Cold Start is attempted.

Warm Start within Downtime else Do Not Start -

The downtime, i.e. time for which power has been off is checked against the maximum acceptable downtime as defined in the **OK_Down_Time** parameter of the PcsSTATE function block. If the power has been off for less than this period, a warm start is attempted. If it is not successful or if the downtime is greater than the acceptable period, the program is not run.

Selecting a start-up strategy

Care should be taken to select the appropriate strategy. This depends very much on the behaviour of the plant and process which the PC3000 is controlling.

Examples:

If the PC3000 is controlling a system with mechanisms that may assume undefined positions after a power-fail and for which manual resetting may be necessary, **Do Not Start-up** will be the safest strategy.

If the controlled plant can always be re-started from the same defined state on power-up, then select the **Cold Start** strategy.

Heat treatment applications will typically want to continue control after the resumption of power. For example, if a furnace is still hot, it may be possible to continue to run the User Program from where power was lost. The **Warm Start within Downtime else Cold Start** strategy will be appropriate, with the **OK_Down_Time** parameter set to the period for which the cooling of the furnace is tolerable.

Chapter 3

REAL TIME SYSTEM STATE INFORMATION

Contents

Overview

Information about the state of the PC3000 real time system is available by reading certain system parameters via the Default Communications. For further information refer to the PC3000 Communications Overview document.

The following information is presented on screens on the Programming Station :

Firmware Version

e.g. 2.10 - this gives the version number of the LCM system software.

User Program Name

e.g. ProcessA - the name of the currently loaded program.

User Program Size

Size of the loaded User Program in bytes. This is the total memory require ment for the executable program.

Maximum Program Size

The maximum memory space available for loading a User Program.

User Program Status

This is normally set to “OK” for a valid program. However the following errors may be reported :

“No User Program Loaded” see note below

“PC3000 Still in Load Mode” see note below

“Bad User Program Checksum”

“User Program Header Error 1”

“User Program Header Error 2”

Other system error numbers may be reported.

Note : All errors except “OK” imply that the User Program is corrupted and should be downloaded again.

User Program Source Size

This gives the size in bytes for the Source Code of the executable program if it has been downloaded from the Programming Station. Note, that it is optional whether the Source Code is downloaded.

Maximum Source Size

The maximum memory space available for holding Source Code.

Status at Start-up

e.g. System cold started - this defines the type of start-up made last time the system powered up.

Start-up Strategy

The currently selected start-up strategy. (See paragraph on Start-up Strategy in Chapter 2)

System error log

The system provides an error log which records all internal system errors and significant system events. The log is organised in reverse chronological order, i.e. the latest error is always inserted at the top of the list. The log can hold up to 40 entries. If the log is full, the oldest error is removed, to enable a new entry to be added to the top of the log. Each error is recorded with a time stamp, an error code and two diagnostic information fields.

See Appendix B, System Errors for further details.

The error log is cleared when a User Program is down-loaded.

System errors are reported if the I/O modules required by the loaded User Program are not fitted. During commissioning a new system, a large number of errors may be logged because of this reason. The **Reg_IO_Fail** parameter in the PcsSTATE function block can be used to enable and disable the registration of I/O failures. I/O error registration can be disabled during commissioning but should be enabled when the system is running normally.

If the system is functioning normally, the only system errors that should be seen in the log are :

400 - Power failure
499 - Power recovery (a system event)

The User Program can monitor the number of system errors by testing the **Sys_Alarms** parameter in the PcsSTATE function block. The number should not normally change while the program is running. The system error count includes all

errors reported since the User Program was loaded and not the number in the log.

The system error log can be read via the Default Communications by reading certain system parameters, see PC3000 Communications Overview document.

PcsSTATE function block

This function block provides an interface between the system manager and the User Program. Every program is built with one instance of this block. Refer to the PC3000 Function Block Reference for a full description of this block.

> **Note:** PcsSTATE is not updated unless the program is running, i.e. PC3000 is in RUNNING or SEQ_HELD operating modes.

The principal parameters are :

Start_Strat - defines the currently selected start-up strategy, see the Modes of Operation Chapter.

Last_Down_Tm - the period the power was off the last time.

Last_Start-Up - the type of start-up that occurred the last time the system powered up.

Reg_io_Fail - normally this is set to “YES” to record I/O module errors in the system error log. Set to “NO” will inhibit logging I/O errors.

Module_Fault - records the position of the first faulty I/O channel found by rack number, module number and channel number. If set to zero, no faulty I/O channels are present.

Battery_Cond - shows the condition of the battery for the non-volatile memory. This can have values Good, Low and Faulty. If set to low, the battery should be changed.

HW_link - shows the value of the hardware links on the LCM board used to set the address for PC3000 communications. See the PC3000 Hardware Reference and PC3000 User Guide for further details.

Sys_Alarms - records the total number of system errors logged since the current User Program was loaded.

The parameters **Seq_Mode** and **HW_fault** are not currently supported and are provided for future enhancement.

Chapter 4

REAL TIME CLOCK

Contents

Overview

The LCM is equipped with a Real Time Clock (RTC) for time stamping and general time management within a User Program. Access to the Real Time Clock is provided by the RT_Clock function block; each User Program is built with one instance of this block.

The RTC is maintained by the non-volatile memory battery when the power is off. It is accurate to +/- 5 minutes per year. Adjustments to the calendar for leap years is provided.

> **Note:** The RTC can only be set with the current time via the Programming Station when the PC3000 is running with a valid User Program.

The RTC may be set as follows:

1. Run the user program by setting the PC3000 mode to Run.
2. Select the RT_Clock Function Block from the SYSTEM Class.
3. Display the Function Blocks parameter screen.
4. Enter the required Date and Time into the Preset_DT parameter. The time is represented as a 24 hour clock.
5. Change the Preset_Clk parameter from Tock (0) to Tick (1) in order to 'clock' the new date and time into the RTC.
6. Reset the Preset_Clk parameter from Tick (0) to Tock (0).

Any changes made to Preset_DT or Preset_Clk will be ignored if the program is not in the Run state.

Chapter 5

SYSTEM WATCHDOG and FAULT RECOVERY

Contents

Overview

The LCM is equipped with hardware watchdog that must be repeatedly triggered by the system software within 9 ms otherwise the watchdog forces a total system reset. This is provided to detect the exceptional situation where the system has malfunctioned. For example, if the system software runs in an endless loop or executes with corrupt data, the system reset will cause the PC3000 to re-initialise as if it was recovering from a power failure. The start-up will then follow the current start-up strategy. A watchdog reset is recorded in the system error log as error number 401.

There is also a watchdog timer on each task of the User Program. This is fully described in the paragraphs on the Real Time Task Scheduler. If a task fails to complete within an period defined by the task watchdog, error 804 is logged and a system reset is actioned, in the same manner as a hardware watchdog detected failure.

Detection of repeated watchdog resets

There are further checks within the system fault recovery software to ensure that the system is running in a stable state; this is regarded as running without watchdog resets for longer than 30 seconds. If the system is unable to stabilise after 10 watchdog resets and subsequent restarts, the User Program is aborted by being cleared and the PC3000 is forced into the IDLE mode. In which case, an error 404 is recorded in the system error log.

Note: A repeated watchdog reset may occur if the User Program has been built on a Programming Station which does not have the correct version of the Function Block library installed.

WARNING

I/O State after forced IDLE Mode

If repeated watchdog resets result in the PC3000 aborting the User Program and entering the IDLE mode, the physical outputs may be in an indeterminate state. The system will attempt to set all digital outputs to their low state, e.g. relays driven by DO12_RLY channels will be off, analogue outputs will set output levels to zero. These values are not guaranteed and such output values may not be safe for the current plant state. For safety, it is recommended that the fail safe relay such as on the DO11_RFS is used. This can be used to detect a watchdog reset and therefore can be wired to inhibit critical outputs such as heaters etc.

For further details on the fail safe relay, refer to the DO11_RFS module or the fail safe relay in power supply module in the PC3000 Hardware Reference.

Chapter 6

I/O SUB-SYSTEM

Contents

Overview

The PC3000 I/O subsystem comprises the physical I/O modules, the communication bus between the Local Controller and the I/O modules via a device described as the Input Output Concentrator (IOC).

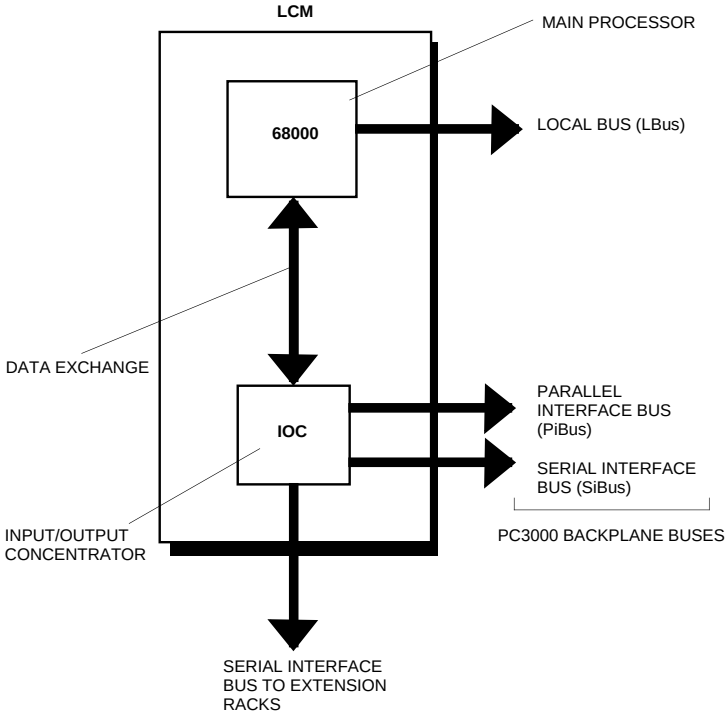

Figure 6-1 A simplified view of the PC3000 Main Processor/IOC Interface

The function of the IOC is to act as a data concentrator for data sent to or received from the I/O modules. It offloads the task of I/O message handling from the main processor in the Local Controller Module (LCM).

It distributes data to, and collects data from the I/O modules according to an ordered list provided by the main processor. Data is sent or collected at pre-defined intervals ensuring that all I/O transactions are fully deterministic.

Communication between the IOC and the main processor is co-ordinated at a pre-defined ‘rendezvous’ time.

Once the data has been gathered it is stored in an internal buffer. This data is used to update the I/O Function Blocks outputs when they next execute. For more information on the relationship between physical I/O, Function Block execution etc. consult the paragraph ‘Performance Considerations’page 6-8 and Chapter 7 Real Time Scheduler.

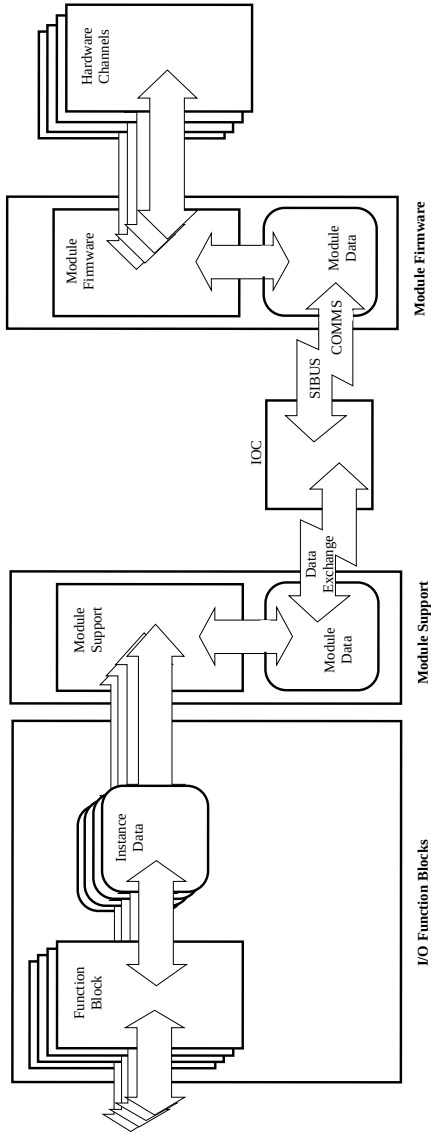

Figure 6-2 Analogue module example

I/O Function Blocks such as Analog_In or Debounce_In are automatically created during the process of ‘Hardware Definition’. This is the User Program development phase which is associated with the assignment of the physical I/O modules to specific rack slots. Following module assignment, the channels are named. The Function Blocks created as a result of this process may then be manipulated in the same way as any other PC3000 Function Block. The flow of data between function blocks and hardware channels is shown in figure 6-2.

These Function Blocks provide an interface to, and a view of the physical I/O. The process of ‘Hardware Definition’ is covered by the User Guide relating to your Programming Station. Refer to PC3000 Programming Station User Guide or PC3000 Microcell User Guide for further details.

I/O communications

Data is sent to, or collected from the I/O modules at pre-defined times. This occurs on a regular ‘heartbeat’, referred to as the system tick.

In a simple PC3000 configuration there would be two such ticks; one is associated with the digital I/O and the other the analogue I/O. The analogue I/O tick is a multiple of the digital I/O tick. Each tick is split up into a number of time ‘slices’ with each slice having a pre-defined purpose.

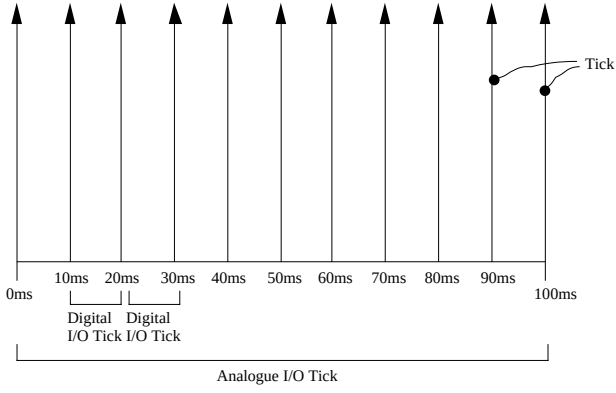

Figure 6-3 Relationship between digital and analogue I/O tick

The value of certain I/O process parameters, that may change rapidly, are required every tick. For instance, a digital input process value **digin.PV** should be updated every digital I/O tick. Regular updates of such parameters are necessary to ensure deterministic operation. However, many configuration parameters rarely require updating on a regular basis.(e.g. a parameter that defines the linearisation type of an

analogue input channel). Once set, these parameters are rarely changed. In the event of a change, such configuration parameters are sent to the I/O module using spare time allocated in every time slice.

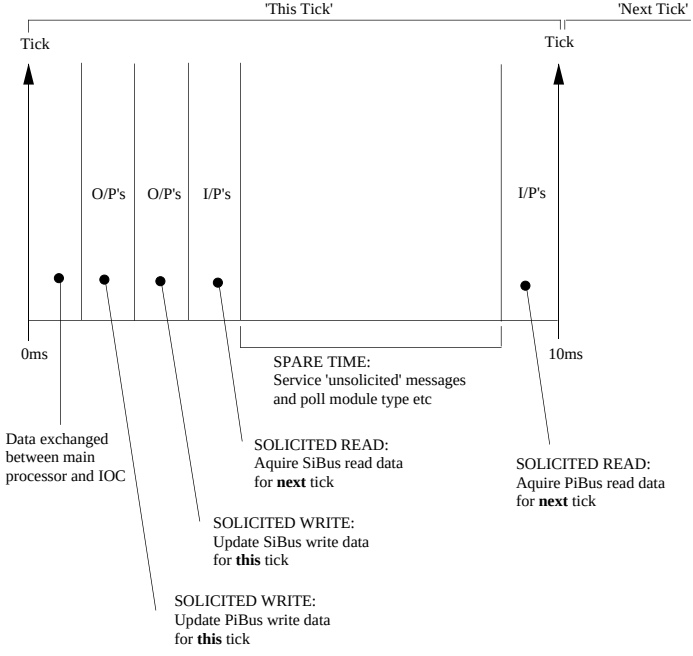

Figure 6-4 How the tick is split up

Parameters that require regular updates are accessed using solicited messages. In contrast configuration parameters are accessed using unsolicited messages since they are rarely changed.

Solicited messages are fixed in time when the program is compiled. Unsolicited messages are non-deterministic, in that they are scheduled or fitted in, according to available spare time. This spare time is reduced as the number of I/O channels and hence the number of solicited messages increases.

This spare time is also used to ‘poll’ the I/O modules on a regular basis in order to establish the type and status of the I/O modules fitted to the rack.

All parameters such as I/O input and output values are updated as solicited parameters; other configuration or diagnostic parameters are handled using unsolicited messages.

Whilst all fast digital I/O is updated every tick, it is not possible to update all analogue I/O at the same rate. In a typical system the analogue I/O will run at a rate, say, 10 times slower than the digital I/O. The amount of analogue I/O in a typical system would take longer to collect than a single digital I/O tick. As an example, a 24 loop system, with a digital I/O tick of 10ms and an analogue I/O tick of 100ms would require 24ms to gather the analogue data.

As a result, analogue I/O is distributed evenly across all digital I/O ticks. In the previous example, one-tenth of the analogue I/O is gathered in each tick.

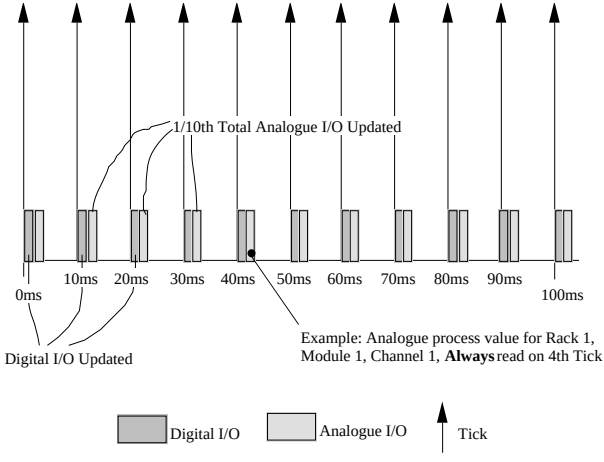

Figure 6-5 I/O Update

In this example, there are 10 digital I/O ticks to every single analogue I/O tick. The analogue data gathered on the ‘nth’ tick will always be the same. This ensures deterministic operation of the analogue I/O, i.e. a given I/O point is processed using constant sample rate.

Types of I/O communications

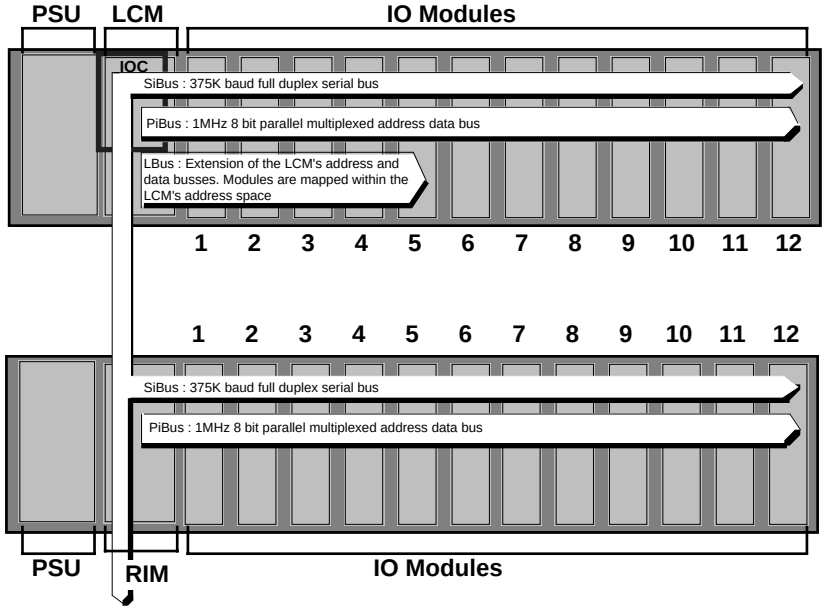

Figure 6-6 PC3000 Buses

SiBus Communications

The PC3000 Serial Interface Bus, is a two-wire, asynchronous communication scheme with the I/O Concentrator acting as the master and I/O modules responding as slaves. The Bus operates at 375KBaud. It is used to communicate with intelligent I/O modules such as analogue inputs and outputs where local input/output processing is carried out by on-board micro-controllers.

It is also used to communicate with the Rack Interface Modules which reside in extension racks.

All communications transactions are checksummed to ensure robust communications and a re-try strategy is provided to ensure that messages are not lost in environments with high radio frequency interference (RFI) noise.

PiBus Communications

The PC3000 Parallel Interface Bus, is an 8 bit multiplexed address/data bus used to communicate with digital input and output modules. All data and addresses transmitted on the bus are hardware parity checked to ensure robust communications. The bus operates at a data rate of 1Mbit/s.

The bus is controlled by the I/O Concentrator directly in the case of the main rack, or by the Rack Interface Module, for an extension rack.

LBus Communications

The PC3000 Local Bus, is a high speed parallel address, data and control bus. It is reserved for use with specialist modules such as communications, network interfaces and high speed I/O functions. Unlike, SiBus or PiBus communications, I/O messages passed via this bus are not controlled by the I/O Concentrator. The main processor maintains responsibility for I/O update.

Communications between Racks

The Serial Interface Bus (SiBus), is connected to all extension racks by the extension rack interface on the Local Controller Module and is 'daisy-chained' between racks via Rack Interface Modules. Within the main rack, all digital I/O which communicates by means of the Parallel Interface Bus (PiBus), is accessed directly by the IOC.

In extension racks, all digital I/O communicating on PiBus, is accessed by the Rack Interface Module. The IOC is responsible for 'packing' data intended for 'remote' digital output modules located in the extension racks, for subsequent 'unpacking' by the Rack Interface Module. Data read from the remote digital input modules is unpacked by the IOC.

Digital I/O modules located in extension racks are updated at the same tick rate as digital I/O in the main rack.

Performance Considerations

Bandwidth Considerations

From the definition of the I/O required, it is possible to calculate the spare I/O bandwidth, and hence determine suitable analogue and digital task rates.

The I/O bandwidth usage is calculated from a number fixed overheads, and specific overheads which are dependent on the I/O channels being used.

Fixed Overheads

There is a fixed overhead of 250µS per extension rack containing digital input modules, plus 250µS per extension rack containing digital output modules. This

is the basic overhead in communicating with the Rack Interface Module. In addition, there is a 35µS overhead per external rack containing up to 8 digital modules, and a 70µS overhead per external rack containing more than 8 digital modules.

Channel I/O Usage

Modules can be split into their generic types of analogue or digital and inputs or outputs. For instance, a Digital Input 14 channel contact closure module is treated as a digital input.

The I/O bandwidth usage of each type of module is thus:

4 Channel Analogue Input -

500µS per channel used

4 Channel Analogue Output -

500µS per channel used

14 Channel Digital Input -

105µS per module, if in main rack

70µS per module, if in extension rack

12 Channel Digital Output -

105µS per module, if in main rack

70µS per module, if in extension rack

Note: The time used by analogue modules depends on the number of channels in use, but the time used by digital modules is fixed.

Simple Example Calculation

The following worked example explains how to apply these figures in a simple system.

The I/O configuration for the system is as follows:

Rack 1	**Rack 2**
AI4	AI4
AO4	AO4
DI14	DI14
DO12	DO12

All channels of the analogue modules are used.

The analogue I/O runs on the 100ms update rate, the digital on a 10ms update rate

Consider the analogue modules first.

There are 16 channels of analogue I/O (four for each module).

Each channel uses 500µS

Total analogue time is 16x500µS = 8000µS per analogue tick

The time used per digital tick is therefore:

8000µSx10mS(digital rate)/100mS(analogue rate) = 800µS per digital tick.

The main rack digital modules each take 105µS per tick.

Main rack time usage is 2x105 = 210µS

The overheads for the extension rack digital I/O can be broken down into fixed overheads and module specific overheads.

The fixed overheads are 250µS for a digital output in extension rack plus 250µS for a digital input in an extension rack, and 35µS for 2 digital modules in an extension rack.

Overheads are therefore 250 + 250 + 35 = 535µS

The input and output modules in extension racks each take 70µS per module. There are two modules, hence there is an extra 140µS time used.

The total time used by the extension rack digital I/O is therefore 535+140=675µS per tick.

The total time used per tick is:

800µS (Analogue)

+210µS (Main rack digital)

+675µS (Extension rack digitals) =1685µS per 10000µS tick.

As can be seen from the I/O bandwidth calculations, there are fixed overheads associated with placing digital modules in extension racks. There is a separate overhead for inputs and for outputs.

Reducing I/O communications bandwidth usage

When using a large number of digital inputs or outputs, it is better to place only inputs in one rack, and only outputs in another rack.

As a simple example assume there are 12 digital inputs and 12 digital outputs in extension racks. The difference in bandwidth usage for a system with inputs and outputs in different racks, and inputs and outputs mixed between racks can be calculated. For either case, the time used per module is a constant of 70µS per module, giving a total of 1680µS. Each rack also uses 70µS as it contains more than 8 digital modules.

This gives an overhead of 1750µS.

If the inputs and outputs are in different racks, each rack has a fixed overhead of 250µS for either inputs and outputs giving total bandwidth used as 2250µS. If inputs and outputs are mixed in both racks, each rack has a fixed overhead of 500µS for containing both inputs and outputs, giving total bandwidth used as 2750µS, or 20% more time used.

Rule:

For large systems I/O Bandwidth may be optimised by placing inputs in one rack and outputs in another rack.

I/O latency

The time between the state of a physical input sensor changing and the time taken for the PC3000 control to system react is defined as the I/O Latency.

The data presented here provides an indication of the expected input/output latency, or delay time, for various input and output combinations.

All data is based on a two task system with task rates of 10ms and 100ms.

In each case, the input I/O function block (i.e. the **.**PV parameter) is directly soft-wired within the User Program to the output I/O function block (i.e. the **.**PV parameter).

Digital inputs/outputs

The timing for a single digital input soft wired to a digital output is given in figure 6-7.

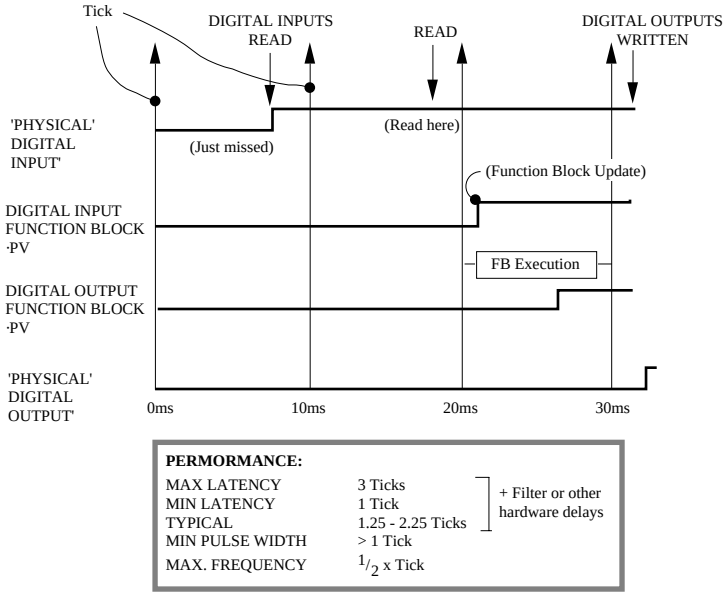

Figure 6-7 Example of digital and output function blocks 'wired' together

Actual measured latency times are as follows:

Digital task interval [ms]	Average latency time [ms]	Min latency time [ms]	Max latency time [ms]
5	12	9	15
10	20	15	25
20	35	25	45
50	80	60	100

Analogue inputs/outputs

The timing for an analogue input/output pair is represented below. The analogue input module includes a four stage rolling average filter which imposes a delay on the signal. Additionally, the sampling rate of 100ms imposes an additional 100ms delay giving 450ms plus the associated execution delays.

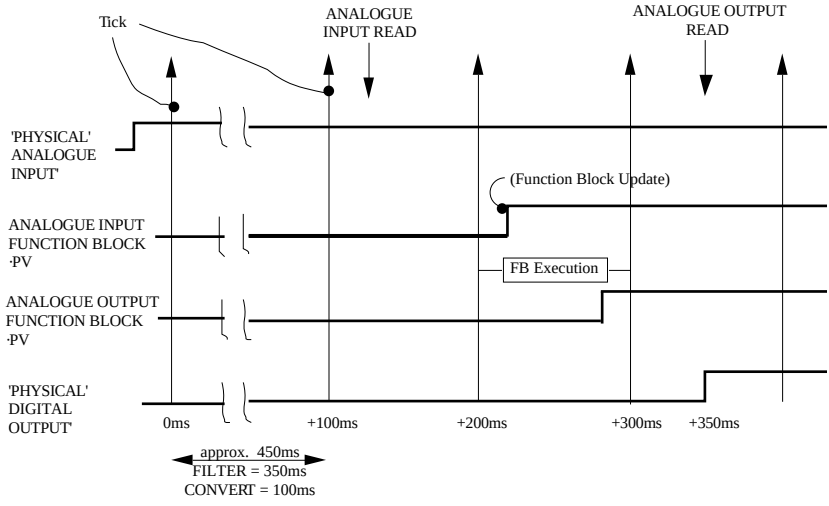

Figure 6-8 Example of analogue input and output function blocks 'wired' together

Actual measured latency times are as follows:

Analogue task interval [ms]	Average latency time [ms]
100	800
200	1400

Composite timing

The following timings cover mixed analogue/digital applications. Data is based on time measured from change of state at input to corresponding change at output.

Digital input to analogue output

Actual measured latency time, digital input to output (0 to 100%)

Example assumes change of Process_Val following a change in state at the digital input.

Analogue task interval [ms]	Digital task interval [ms]	Average latency time [ms]	Min latency time [ms]	Max latency time [ms]
100	10	150	100	200
100	20	140	90	190
100	50	110	60	160

Analogue input to digital output

Actual measured latency time, analogue input to digital output. Example assumes change of digital output state following input exceeding 50% of full scale.

Analogue task interval [ms]	Digital task interval [ms]	Average latency time [ms]	Min latency time [ms]	Max latency time [ms]
100	10	650	600	700
100	20	700	650	750
100	50	800	750	850
200	10	1000	900	1100
500	10	900	700	1100

Timing for fast analogue input and output function blocks

Times are similar to those associated with digital I/O since this module runs in the digital function block task. All measurements are based on the cut off frequency of the filter set at 160Hz.

Analogue task interval [ms]	Digital task interval [ms]	Average latency time [ms]	Min latency time [ms]
100	68	20	115
10	20	15	25

I/O error codes

The following system errors are associated with the PC3000 I/O system. For a full list of system errors refer to Appendix B.

Error code	Error description	Field 1	Field 2
305 & 307	**Error** Communication with an I/O module has failed after 3 retries. **Cause** This most likely occurs when a module is removed or inserted but may also occur if a module resets, possibly due to its local watchdog. It may also be caused by RFI noise within the rack or on a multi-rack system in the inter-rack connections. A repeated failure from the same module is likely to indicate a hardware fault. **Solution** If a single module is consistently giving errors then it is likely there is a module failure and it should be replaced. Random errors indicate RFI noise so screening and cabling may need to be improved.	Module address [1]	Diagnostic information

Note 1. The first digit (from the left) is the rack number (1 to 8) and the following two digits are the slot number (1 to 12).

Error code	Error description	Field 1	Field 2
308	**Error** Digital I/O system over-loaded. **Cause** On initialisation, the I/O handler detected that there would not be sufficent bandwidth on the parallel bus to communicate with the number of modules defined in the User Program. **Solution** Use less digital I/O or extend the interval of the fastest task.	0	0
309	**Error** Analogue I/O system overloaded. **Cause** On initialisation the I/O handler detected that there would not be sufficient bandwidth on the serial bus to communicate with the number of modules defined in the User Program. **Solution** Use less analog I/O or extend the interval of the slower IOH (See Appendix C, paragraph Allocation of handlers to task. Appendix C, Scheduler overheads, performance and limitations).[2]	0	0

Note 2. A time proportioned digital output may be classified as analogue in this context.

Chapter 7

REAL TIME TASK SCHEDULER

Contents

Overview

The multi-tasking system supported by the Real Time Task Scheduler (referred to as the Scheduler) provides flexible control over the execution of function blocks and sequences in the User Program. A User Program can be configured to have between two and seven tasks. Tasking is controlled and configured using Task function blocks which also provide detailed diagnostic information on task behaviour and support for failure detection using a watchdog facility.

Multi-tasking is provided for the following reasons :

To allow processor execution of certain function blocks within the User Program to execute at a rate that matches the required responsiveness of the system.

For example, a counter function block wired to a digital input used to count pulses from a position sensor may need to be scanned every 5ms.

To allow function blocks that do not require fast execution, to be run in slower tasks so that processor time can be used more effectively.

For example, some PID function blocks for temperature control where there are long time constants, may be run in a 500ms task instead of the default 100ms task.

If specific problems are encountered when using multi-tasking refer to paragraphs on the Typical Tasking Problems and Solutions and Rules and guidance for Multi-Tasking, within this Chapter.

For further details on the Scheduler behaviour refer to Appendix C, Scheduler Overheads, Performance and Limitations. Typical Task configurations are given in Appendix D.

Task scheduling

The following paragraphs give an overview on Scheduler operation and task behaviour.

Tasks

A Task is a group of operations which are regularly repeated when a User Program is running. For example a group of function blocks, their wiring and associated Sequential Function Charts (SFCs) can be configured to run in a designated task. Each task has associated parameters which control its priority and the rate at which it is repeated.

For example, a User Program can be configured to have a fast task that executes once every 10ms and a slower task that executes every 100ms. In this case, every function

block instance in the User Program and the associated wiring will be allocated to either of these two tasks. The Sequential Function Charts (SFCs) will typically be allocated to the slower 100ms task.

Scheduler

The Real Time Task Scheduler is associated with system software within the PC3000 which selects which task is to run at any particular time.

Figure 7-1 shows an example of scheduling performed on a three task User Program.

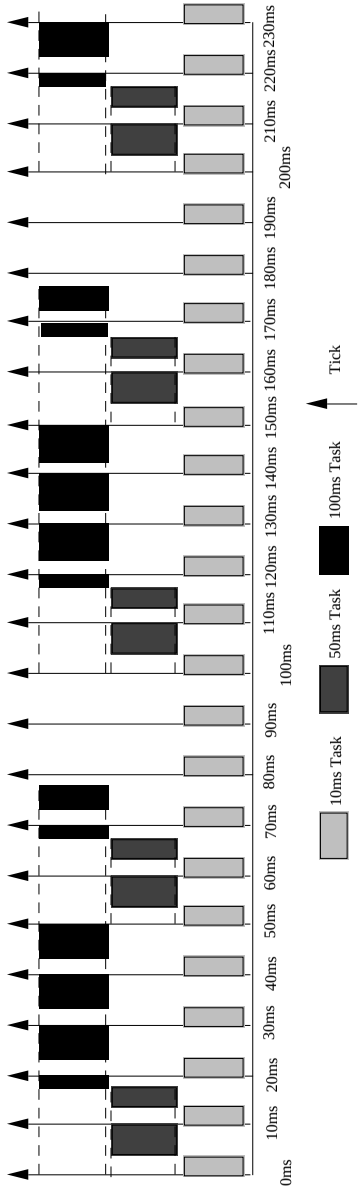

Figure 7-1 The scheduling of three tasks

Operation of the scheduler

The shaded regions in figure 7-1 show which task is running at any given time. The dotted lines show where a task is 'ready' but not actually running, i.e. the task's execution has been held up while a higher priority task is run. The Scheduler allows interruption of a lower priority task by a higher priority task using a real-time scheduling technique known as "pre-emptive scheduling".

The vertical arrows in figure 7-1 indicate timing points at which the execution of the tasks is interrupted. At these points, known as the basic tick , the Scheduler decides which task should run next. The highest priority task is always the most frequently re-scheduled task and its periodicity defines when the basic tick occurs. In figure 7-1 the basic tick rate is 10ms.

> **Note:** With the PC3000 Scheduler strategy, any task can be interrupted by a higher priority task. No task can interrupt the highest priority task.

After each basic tick, the Scheduler begins running the highest priority task. This may interrupt a lower priority task in which case, the lower priority task's "context" is stored so that it may be resumed from the same point later. In the figure 7-1 example, after every fifth basic tick, the 50ms task is scheduled (made 'ready' to run) and similarly after every tenth basic tick, the 100ms task is scheduled. Being scheduled implies that at the next available opportunity they will be run. When the 10ms task is completed the Scheduler is invoked and it then selects the next lower priority task which is 'ready' (if there is one). For example, on the first tick in figure 7-1, (shown at 0 ms), the 10ms task is run. Once it has completed, the 50ms task takes over for the rest of the tick.

Similarly when the 50ms task is completed (after 18ms), the 100ms task is run. When no other tasks are 'ready' and the running task completes, the Scheduler simply idles until the next basic tick (78ms, 84ms and 94ms.).

Using default tasks

By default, the Programming Station always configures a User Program to execute in two tasks, 10ms and 100ms which are controlled by associated Task Function Blocks. You are advised to use the default tasks if the User Program does not have any special performance or loading requirements.

Refer to the PC3000 Function Block Reference for further details on the Task Function Block.

The default task function blocks

The default Task Function Blocks for tasks designated Task_1 and Task_2, are shown in figure 7-2 with their associated parameter cold start values.

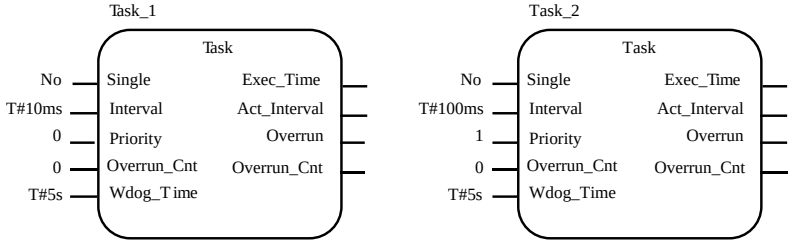

Figure 7-2 The default task function blocks

Default allocation of function blocks to tasks

Each function block instance within a PC3000 User Program is assigned to a task controlled by a designated Task Function Block.

Function block instances may be allocated to any task. By default, when function block instances are created, they are allocated to Task_1 or Task_2 corresponding to the default 10ms and 100ms tasks respectively.

> **Note:** The allocation of function blocks to tasks in earlier releases of PC3000 which did not support multi-tasking, was fixed for each function block type. For example the Digital_Out function block instances were allocated to the 10ms task and the Analog_Out function block instances allocated to the 100ms task.

Overrunning

A task is said to overrun if it cannot be scheduled at the requested interval. When a task overruns the Scheduler must degrade the priority of all other tasks so that the task execution completes. Figure 7-3 shows the scheduling of a two task User Program. The Scheduler attempts to maintain the ratio between the number of executions of the different priority tasks in this situation.

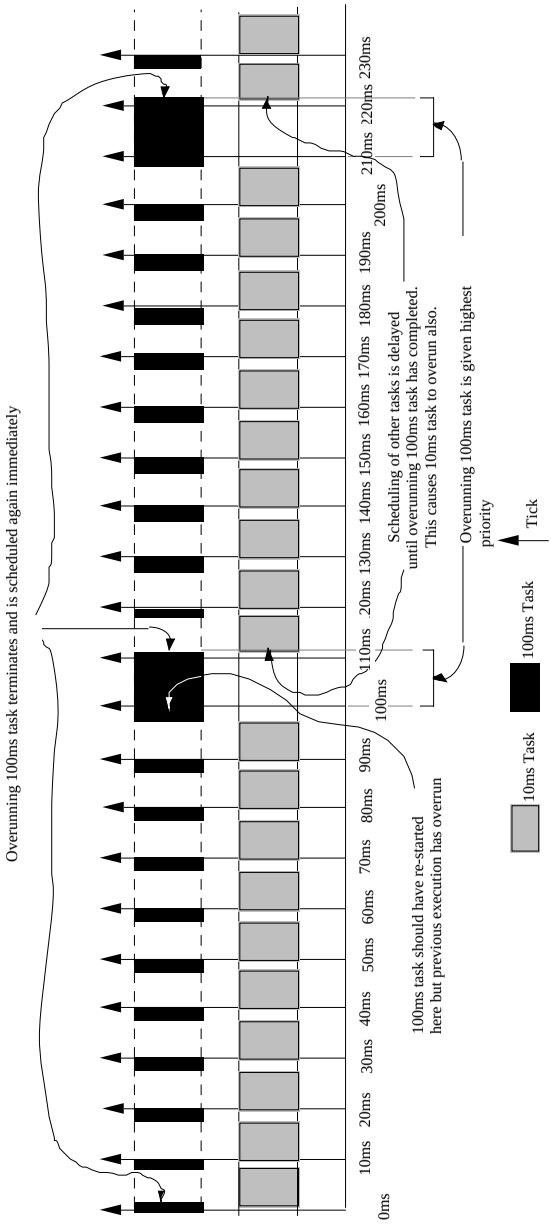

Figure 7-3 Scheduling of overrunning tasks

Task degradation when overrunning

If a task has not completed when it is scheduled to begin running again, the Scheduler automatically increases its priority above all other tasks. This has the effect of making the Scheduler ignore basic ticks until that task has completed. Once the overrunning task has completed, it is then immediately re-scheduled but at its normal priority. The ratio of the number of executions of different tasks is always maintained by this strategy.

> **Note:** In the figure 7-3 example the 10ms task overruns once per execution of the 100ms task because it must wait for the 100ms task to complete.

Function blocks such as PID will adjust the sample time used for internal calculations to account for the task interval being longer due to overrunning.

The timing of STEP durations i.e. the T parameter, will be adjusted to account for task overruns to maintain timing accuracy.

Caution

Overrunning should be regarded as an exceptional situation and User Programs should be designed to avoid overruns where possible. Control accuracy and performance will be degraded if a User Program continually executes with overruns.

Task function block diagnostics

Task function blocks provide parameters to monitor system loading by the User Program. The PC3000 Function Block Reference has further details on the Task Function Block.

The Act_Interval parameter

The **Act_Interval** output from the Task function block indicates the measured interval which was achieved by the system for the last execution of the task.

Normally this would be equal to the **Interval** requested but if the PC3000 is unable to execute all the function blocks in the required time, it may be extended. The **Exec_Time** parameter can be used to identify which task is causing the main processor loading, see paragraph The Exec_Time Parameter.

The **Act_Interval** is measured in terms of basic ticks so it will always be a multiple of the fastest tasks interval. The **Act_Interval** is measured as the difference between the tick in which it was scheduled for this execution and the tick in which it was scheduled for its last execution.

The Overrun and Overrun_Cnt parameters

The **Overrun** parameter is Yes (i.e.1) when the **Act_Interval** is greater than the **Interval**, i.e. the task is overrunning. The **Overrun_Cnt** is a count of the number of executions of the task which have overrun. The overrun count can be cleared at any time by writing 0 to this parameter from the Sequential Function Charts.

The Exec_Time parameter

The **Exec_Time** output shows the execution time of the task in integer milliseconds and can be a useful diagnostic parameter to show the system loading, i.e. which task is taking a high percentage of processor time and is causing a particular User Program to overrun.

The value of **Exec_Time** as a proportion of the **Interval** is known as the processor utilization. The following equation can be used to obtain the percentage of processor utilization for a task.

$$\text{processor utilisation} = \frac{\text{Exec\_Time} \times 100\%}{\text{Interval}}$$

This only calculates the processor utilization for the execution of the given task. It does not include scheduling overheads etc. or loading due to executing other tasks. It does however include the execution of any communications or Real Time Clock interrupt service routines which may interrupt the tasks' normal execution and any system handlers which are run for that task, see Appendix C Scheduler Overheads, Performance and Limitations. Adding all the tasks' processor utilizations together gives a good indication of overall system loading. If this exceeds 100% then the PC3000 will certainly overrun. Below 100% the PC3000 may still overrun because of the overheads of scheduling.

Caution

It is recommended that the total processor utilization for all tasks should not exceed 80% and ideally should be less than 50%.

Keeping the total processor utilisation below 50% will allow time for scheduling overheads and some margin of error for variations in processing time from execution to execution. The percentage error in this calculation will be significant when the **Exec_Time** is small (because the measurement is only to the nearest millisecond). In User Programs which are close to overrunning it may be useful to take this error into consideration by taking the highest value for **Exec_Time.**

Changing task parameters

The following parameters modify the task behaviour; **Interval**, **Priority** and **Wdog_Time**. These configuration parameters can be changed at any time but they only have an effect on the task at the time when the User Program first begins running, i.e. during the INITIALISING operating mode, see paragraph Modes of Operation. Because the configuration parameters can be unexpectedly reset, due to a watchdog or power failure, it is essential to build the task configuration parameter values into the User Program as cold start values.

It is possible to change these parameters when the User Program is in RESET mode, (mode will change to HALTED after any parameter is modified) and run the program in order to experiment with different values.

> **Note:** The **Single** input parameter of the Task function block is not yet implemented and is included for future expansion; it should be left at its default value.

When an attempt is made to run a User Program with an illegal task configuration the PC3000 will remain in the RESET or HALTED mode and an error will be logged in the system error log. A list of the errors which can be logged by the scheduling system are shown in paragraph Scheduler System Errors.

Changing the default intervals

The task **Interval** can be changed to either slow a task re-execution rate in order to avoid task overruns or increased to improve system responsiveness.

Rules when changing task intervals

When changing the task intervals note the following:

1. The fastest task cannot be any faster than 5ms or slower than 65ms.
2. The next fastest task cannot be any slower than 20 times the Interval of the fastest task.
3. Each task's interval must be an integer multiple of the next fastest task's interval. In a two task User Program, the slowest task's interval should be an integer multiple of the faster task, e.g. 100ms and 10ms.
4. PIM2, Analog_In, Analog_Out, PI_Smpl_Ctr, Analog_Out and T_Prop_Out function blocks should be run with a minimum Interval of 100ms or greater.

Recommendations for changing task intervals

When changing task intervals it is necessary to consider the processor utilization. Consider the two default tasks summarised in the following table:

Interval	Exec_Time	Processor Utilisation
10ms	4ms	40%
100ms	53ms	53%
	Total	93%

Table 7-1 An overloaded user program

The example in table shows a User Program which is probably overrunning and is certainly not leaving sufficient margin of free processing time for scheduling overheads or task execution time variation. To improve the program execution, the total processor utilization by the tasks should be reduced to less than 80%. This can be done by increasing the interval time of either or both of the tasks. tables 7-2 to 7-4 show the effect of slowing down the faster task (table 7-2), slowing down the slower task (table 7-3) and slowing down both tasks (table 7-4).

Interval	Exec_Time	Processor Utilisation
20ms	4ms	20%
100ms	53ms	53%
	Total	73%

Table 7-2 Slowing down the faster task to stop overrunning

Interval	Exec_Time	Processor Utilisation
10ms	4ms	40%
130ms	53ms	38%
	Total	78%

Table 7-3 Slowing down the slower task to stop overrunning

Interval	Exec_Time	Processor Utilisation
12ms	4ms	33%
120ms	53ms	44%
	Total	77%

Table 7-4 Slowing down both tasks to stop overrunning

Any of the above strategies would stop the User Program overrunning and provide a reasonable margin for execution time variations.

Task configuration restrictions

By slowing down the faster task (table 7-2) a restriction is introduced by rule 3 in paragraph Rules and guidance for Multi-Tasking. The slower tasks interval must be an integer multiple of the faster tasks interval (as well as being >5ms and <65ms because of being the fastest task). If the 100ms tasks interval is to remain unchanged then the interval for the fastest task may only be 5ms, 10ms, 20ms, 25ms or 50ms, i.e. a factor of 100ms. If these values are not suitable then the slower task interval should be changed. For example, if the fastest task interval chosen was 16ms then the slower task interval could be changed to 96ms or 112ms.

Compare the task processor utilizations before and after slowing down the faster task (tables 7-1 and 7-2).

When the task intervals are 10ms and 100ms, the task processor utilization is reasonably balanced at 40% and 53% respectively. After slowing down the faster task, the processor utilizations are 20% and 53% which is not so evenly balanced. It may be necessary to keep the slower task at 100ms because of limitations imposed by the process. In which case this would have to be the overriding consideration. Where possible however, keeping the processor utilization evenly spread across the tasks does reduce the likelihood of the User Program overrunning by giving equal margins for variation in each tasks execution time. In the example, task Interval values 10ms and 130ms (table 7-3) is an ideal configuration in this respect.

In applications where there is no significant restriction set by the process on interval times, it is good practice to have plenty of margin in task execution times. In this example, Interval times of 20ms and 200ms would be reasonable. This gives margins for task execution time variation and allows for some expansion of the User Program without extending the intervals.

> **Note:** It is possible to set both tasks to the same interval e.g. 65ms. In which case, there may be delays introduced when wiring is used between the two tasks. For further details see paragraph Parameter passing between Tasks, in Appendix C, Scheduler Overheads, Performance and Limitations.

Allocating function blocks and sequencing to other tasks

Changing the task intervals as described in the last paragraph may be sufficient to alleviate problems of overrunning or insufficient response time in some simple applications. In other cases, it may be necessary to modify the scheduling of the User Program by changing the execution of selected instances of function blocks from their default task to an alternative task.

Allocating function blocks to tasks

It is possible to change the function block task allocations from the PC3000 Programming Station (refer to the PC3000 Programming Station User Guide or PC3000/ Microcell User Guide). The function block instance to task allocation is fixed when the User Program is built and cannot be manipulated in any way after the User Program has been downloaded into the PC3000.

Changing the sequential function charts task allocation

Changing the task allocation of any Step or Macro function block instance will change the task for the execution of all Sequential Function Charts (SFCs) in the User Program.

Note: All Sequential Function Charts execute in the same task.

Guidance on changing function block task allocations

The execution overheads of different tasks can be modified by allocating selected function block instances to run in different tasks. This may be necessary to alleviate overrunning problems by slowing down tasks with unnecessarily fast execution and to improve control of the system by speeding up critical control.

When allocating function blocks to tasks, you are advised to group function block instances which need to run at similar rates. This is a useful tactic because execution rates usually depend on the process being controlled and not on the functionality of the blocks used to control the process. For example, a Digital_In function block, which is used to detect pulses for a counting application, may need to be run with a 10ms interval in order to catch every pulse. In contrast, where an input is connected to a ‘START’ button, a 100ms interval would normally be quite adequate.

Figure 7-4 shows a simple example of instances of the Digital_In function block allocated to different tasks.

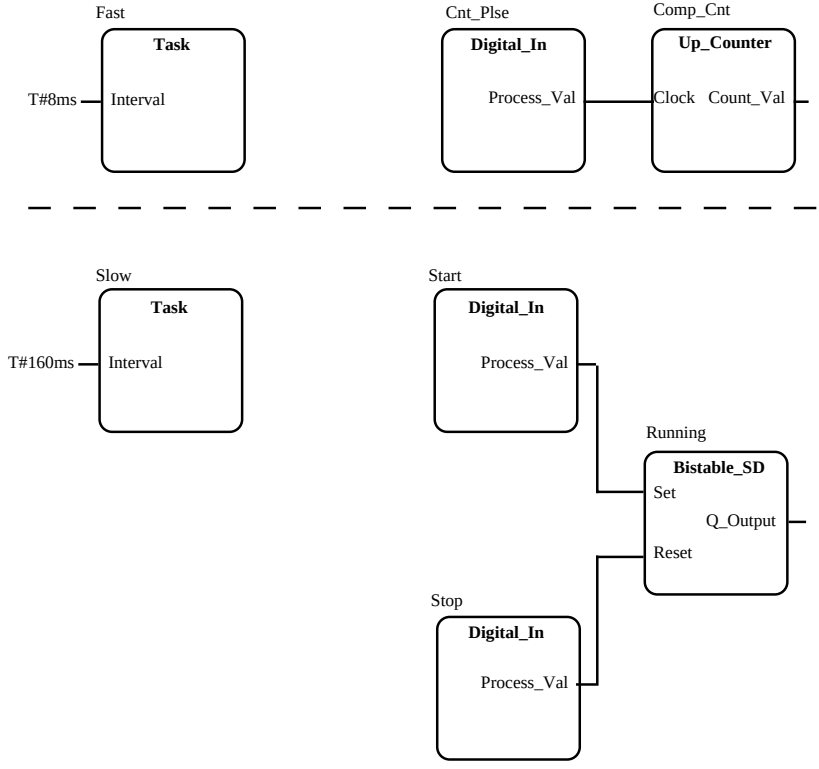

Figure 7-4 Example using the same function block type in different tasks.

When changing task allocations, the processor utilizations for each of the tasks should be balanced. By varying both the function block task allocations and task intervals, it is possible to design a balanced multi-tasking User Program.

As stated in paragraph Changing the Default Intervals, it is advisable to leave sufficient free processor execution time when designing a User Program to minimise

the risk of overruns. Evenly spreading the loading across the tasks will also minimise the risk of overruns.

Avoid allocating function blocks or wiring which involve calculations with decimal floating point parameters (i.e. REALs) in a fast task. Each floating point calculation takes approximately 70µs which can be a significant overhead if the task interval is, for example, 5ms.

The minimum interval for Analog_In, Analog_Out, T_Prop_Out , PIM2 and PI_Smpl_Ctr function blocks is 100ms. If faster analogue inputs or outputs are required then Fast_An_In, I_Fast_An_I or Fast_An_Out can be used with the appropriate module up to a 5ms Interval. Avoid allocating too many Fast_An_In or Fast_An_Out function blocks with task intervals faster than 50ms because they perform floating point calculations which use up processor execution time.

Adding more tasks

Generally most User Programs will only require two tasks but the PC3000 can support up to seven, if required.

New tasks are added simply by creating new Task function block instances. When new Task function blocks are added, the **Interval** and **Priority** default values are set according to the number of tasks, as shown in table 7-5.

Task instance	Interval	Priority
1st	10ms	0
2nd	100ms	1
3rd	500ms	2
4th	1s	3
5th	5s	4
6th	10s	5
7th	30s	6

Table 7-5 Default task intervals and priorities

Rules and guidance for introducing more tasks

In table 7-5 the task **Priorities** are in the same order as the task **Intervals**. This type of configuration is known as **Rate Monotonic**.

Caution

In order to avoid unexpected side effects do not change the task intervals or priorities such that they are no longer in accending order.

For example, task configurations as summarised in tables 7-6 and 7-7 which are rate monotonic will function correctly, yet the task configuration as shown in table 7-8 which is not rate monotonic, may operate incorrectly. In table 7-7 a third task has been introduced and set to 50ms interval but in order to maintain the priority ordering in line with the interval order, the priorities of tasks 2 and 3 have been swapped. Further details of the problems of non-rate monotonic task configurations can be found in paragraph Changing Priorities, in Appendix C, Scheduler Overheads, Performance and Limitations.

Task	Interval	Priority
Task_1	10ms	0
Task_2	100ms	1
Task_3	500ms	2

Table 7-6 Example, three task configuration

Task	Interval	Priority
Task_1	10ms	0
Task_2	100ms	2
Task_3	50ms	1

Table 7-7 Example, alternative three task configuration

Task	Interval	Priority
Task_1	10ms	0
Task_2	100ms	1
Task_3	50ms	2

Table 7-8 Example, non-rate monotonic configuration

Caution

A User Program should not be configured using too many tasks because each new task will introduce memory and execution time overheads.

Task watchdog

The watchdog facilities built into the PC3000 are designed to detect program mal-functions resulting from faulty task configuration or internal data corruption. The latter should be a rare occurrence caused by some form of excessive external electromagnetic or radio frequency interference (RFI). Consult the PC3000 Hardware Reference for details on installation guidelines to avoid RFI. Once a watchdog has been detected, the PC3000 can take action to re-initialise and continue running.

Task watchdog operation

The user task watchdog is configured by the **Wdog_Time** parameter on the task function block. This parameter is only read when the User Program begins running. Altering the parameter once the User Program is running will have no effect. The **Wdog_Time** defines a time by which a task should finish execution after it has been scheduled to run, i.e. it defines a task finish deadline.

If the task is not scheduled in this period or fails to finish, the PC3000 system will be forced to reset.

A system error 804 see paragraph Scheduler System Errors, is logged indicating which task has run beyond the watchdog period. During User Program re-initialisation, other system errors may be added to the system error log.

Disabling the task watchdog

It may be useful to completely disable a user task's watchdog during User Program commissioning. This can be done by setting the **Wdog_Time** to 0ms.

Caution

Extreme care should be taken when disabling the task watchdog. It will remove vital protection from the task making certain task failures very difficult to detect.

Typical tasking problems and solutions

The most frequently encountered problems with multi-tasking User Programs together with suggested solutions, giving references to relevant paragraphs of this manual, are:

The User Program will not run - The cause is probably an illegal task configuration. Check the system error log which should show the reason for the illegal task configuration. Refer to paragraph, Scheduler System Errors for a list of all system errors associated with the multi-tasking system.

Unexpected User Program Watchdog Resets- Check that the **Wdog_Time** in the task function blocks are not too short and extend durations if necessary, see paragraph Task Watchdog.

Tasks are not Rate Monotonic- If the tasks are not rate monotonic then overrunning and watchdog resets are likely to occur, see paragraph Rules and guidance for introducing more tasks. If it is necessary to have non-rate monotonic tasks then extend the task intervals in order to stop the overrunning, see paragraph Changing Priorities in Appendix C, Scheduler Overheads, Performance and Limitations.

Unexpected result from a Wiring or SFC Calculation- If a pair (or more) of parameters from another task are referenced, they may not be coherent values (e.g. from the same execution of that task). For more details see paragraph Changing the default Intervals and Appendix C, Scheduler Overheads, Performance and Limitations.

I/O Fails to operate correctly- Some I/O will not operate at speeds faster than 100ms. Check in the PC3000 Hardware Reference whether the associated function blocks are being run at a rate which is too fast; see page 7-9, Rules when changing Task Intervals.

Task stops operating and PC3000 does not Watchdog - Check the **Wdog_Time** parameter of the task function blocks. It is probably too long to detect a mal-function in reasonable time and should be shortened. Alternatively if the **Wdog_Time** is 0ms then the task watchdog has been turned off. Unless commissioning a User Program, the task watchdog should always be on. For more details see page 7-16, Task Watchdog.

User Program overruns although processor utilization is low- This situation will occur when tasks are not rate monotonic. See paragraph Rules and Guidance for introducing more Tasks and paragraph Changing Priorities, Appendix C Scheduler Overheads, Performance and Limitations.

Task Interval, Priority Or Wdog_Time seem to be incorrect- The **Interval**, **Priority** and **Wdog_Time** parameters of the task function block are only read when the User Program begins running; see page 7-9, Changing Task parameters.

PC3000 does not re-start after a Watchdog reset- The PC3000 may fail to re-start after a watchdog or power failure even though it has a suitable start-up strategy

because of an illegal task configuration. It is important to ensure that the cold start values of a User Program are valid or this problem will recur.

Unexpectedly large Exec_Time for a Task- The execution time for a task may be due to overheads arising from the System Handlers which are automatically assigned to tasks by the Scheduler. For full details of these see Appendix C, Scheduler Overheads, Performance and Limitations.

Unexpectedly Long Propagation Delays- Propagation delays can be introduced when passing parameters between tasks, see Appendix C, Scheduler Overheads, Performance and Limitations.

Rules and guidance for multi-tasking

1. The interval of the fastest task must be greater than or equal to 5ms and less than or equal to 65ms.

2. The interval of the second fastest task must be no more than 20 times longer than the interval of the fastest task.

3. Each task's interval must be an integer multiple of the interval of the next fastest task.

4. The total of all the task processor utilizations should be less than 80% and preferably less than 50% .

5. The values of the configuration parameters of a task function block (**Interval**, **Priority** and **Wdog_Time**) should be built into the User Program as cold start values except while experimenting with different task configurations.

6. The **Single** parameter of the Task function block is for future enhancement and should not be changed.

7. Sequential Function Charts (SFCs) can only execute in a single task.

8. The allocation of function blocks to tasks is fixed when the User Program is built on the Programming Station, i.e. it is not possible to change task allocations once a User Program has been down-loaded into the PC3000.

9. Limits of execution intervals for individual function blocks should be considered when changing the default values.

   For example, Analog_In, PI_Smpl_Ctr, Analog_Out and T_Prop_Out function blocks should be run with a minimum Interval of 100ms.

   Note: Most analogue inputs and outputs should not be run in a task with an interval less than 100ms. Check the PC3000 Hardware reference for details.

10. Avoid calculations using floating point (i.e. REAL values) in fast tasks; floating point calculations can be a significant overhead and may cause overrunning to occur.

11. There should be at least two and no more than seven tasks.

12. The names of tasks can be changed by using the normal function block renaming facilities.

13. The fastest task should be the highest Priority.

14. Having task priorities which are not in the same order as the task intervals (i.e. non-rate monotonic) may cause unexpected side effects.

15. It is possible to have two tasks with the same interval and/or priority if required.

16. A User Program will not begin running if its task configuration is illegal. The PC3000 will remain RESET or HALTED and an error will be logged in the system error log. (See Appendix B System Errors)

17. Function block wiring is always evaluated in the same task as the destination function block.

18. In function block wiring, there is no guarantee that the value of parameters from function blocks allocated to other tasks are coherent i.e. all come from the same task execution.

Scheduler system errors

The following errors may be logged by the Scheduler in the system error log.

For a full list of System Errors see Appendix B.

Where task numbers are specified in Field 1 or Field 2, the numbers refer to the position in the function block instance list, the top of the list being task number 1, the second from the top task 2 etc.

Error Number	Description	Field 1	Field 2
800	**TOO MANY TASKS** There are too many tasks defined. The maximum number is 7.	n/a	n/a
801	**FASTEST TASK TOO SLOW** The fastest task's interval is too slow. The fastest task interval must be no slower than 65ms.	Fastest task number	n/a
802	**FASTEST TASK NOT HIGHEST PRIORITY** The fastest task is not the highest priority. The fastest task must have a higher priority (lower number) than any other task.	Fastest task number	n/a
803	**TASKS NOT SIMPLY PERIODIC** The tasks are not simply periodic. Each task's interval must be an integer multiple of the next fastest tasks interval.	Slower task number	Faster task number
804	**USER TASK WATCHDOG** A task has not completed within the specified **Wdog_Time** .	Task number	Program counter

Chapter 8

MEMORY USAGE

Contents

Overview

The LCM on-board memory that is fitted to the mother board, is used to hold the compiled and built User Program and associated Source Code if required. If more memory is required, extra memory can be fitted to slots 1 and 2 using RAM daughter boards. The memory regions used in the LCM are depicted in figure 8-1.

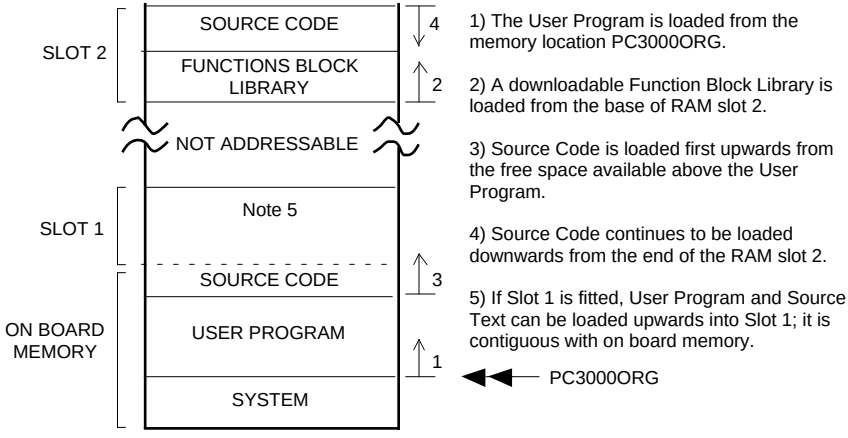

Figure 8-1 Memory (RAM) layout

The LCM non-volatile memory (RAM) is used for holding the following :

User program	Always downloaded from Programming Station
Source Code	Optionally downloaded from Programming Station
Additional Function Block Library	

Further details, on the available memory sizes and on fitting daughter boards ,are given in the PC3000 Hardware Reference.

The User Program is always loaded from a base address defined by symbol PC3000ORG. This varies for different releases of the PC3000 system software (firmware) and is normally set-up automatically by the PC3000 installation software.

> **Note:** If the PC3000 firmware is changed on-site, and the Programming Station software is not re-installed, it is possible to override the original PC3000ORG value by defining the new value in the

AUTOEXEC.BAT file. Release notes will give details on how to do this. If new Program Station software is installed, the PC3000ORG symbol should be removed from AUTOEXEC.BAT.

Downloading source code

The Source Code for the User Program can optionally be loaded and retained in non-volatile memory. It can be uploaded (copied) to the Program Station at any time, for example, if there is reason to view or modify the User Program. It is cleared if a different version of the User Program is downloaded. The Source Code is loaded into memory starting from the end of the User Program area. Default Communications as described in the Overview paragraph, provides parameters to read and write the Source Code.

Note: The Source Code holds a complete definition of the User Program including all function block wiring in Structured Text, program comments, Sequential Function Charts and hardware I/O configuration in a compact binary format. On the PS and Microcell Programming Stations, this information is held in a file with the extension **.cfg**. It is not possible to re-create a particular User Program unless the Source Code is retained either in LCM non-volatile memory or in a PC file.

WARNING

It is advised that a backup copy of the User Program .cfg file is retained on disk because there is always a possibility that Source Code held in non-volatile memory is lost, for example, by removing the LCM and accidentally disconnecting the battery.

Extending on-board memory

By adding a memory daughter board to slot 1, extra space for a larger User Program and Source Code can be provided.

Downloadable function blocks

The Programming Station provides facilities to build User Programs with function blocks from an additional selected Function Block Library, i.e. in addition to the standard library which is built into the system software. If an additional library is required, a memory daughter board should be fitted to slot 2 to receive the downloaded library. The downloaded function block library will be retained in non-volatile memory until either, a different library is downloaded or it has to be overwritten to make room for downloaded Source Code (see next paragraph).

Source code in slot 2 memory

Source Code will overflow into the top of the memory in slot 2 and grow downwards if there is no further free space above the User Program area. If the Source Code cannot fit in the remaining space in slot 2, the Source Code download is aborted. If the function block library is required by the current User Program, it will not be overwritten by the Source Code. However, if the function block library is not required, the Source Code will overwrite it, if the memory space is needed.

Contents (continued)

APPENDIX A TERMINOLOGY

Basic Tick The regular timing pulse within the PC3000 which is used as the basis for timing all scheduling.

COH Communications Handler, software responsible for executing Default Communications.

Coherent Parameter values are said to be coherent when they originate from the same task execution.

Cold Start A User Program has a cold start when all function block parameter values are reset to their original values as defined on the Programming Station.

FBH Function Block Handler, software responsible for executing Function Blocks.

Function Block Instance This refers to the association of a function block to task allocation with a specified task.

Indivisibly A task executes indivisibly when it is not interrupted by the execution of another task.

Interrupt A signal into the CPU from external hardware. For example the 1 second tick of a Real Time Clock which causes an interrupt service routine to be called within the 68000 processor of the Local Controller Module.

Interrupt Service A routine which is called when an interrupt occurs, in order to deal with the source of the interrupt.

Interval The period at which a task is scheduled to run is known as the task interval.

I/O An abbreviation for Input/Output, values read in from sensors and written out to actuators.

IOC Input/Output Concentrator - a processor on board the Local Controller Module responsible for the exchange of I/O values within the I/O System.

IOH I/O Handlers, software responsible for transferringI/O values to and from the PC3000 I/O system, via the Input/Output Concentrator.

Overrun A task is said to have overrun if the actual interval attained by the scheduler is greater than that requested. i.e. **Act_Interval>Interval**.

Pre-emptive A 'pre-emptive' Scheduler is one which will interrupt a lower priority task, which is already running, in order to allow a higher priority task to run.

Priority The priority of a task is a parameter used to determine which task should be run when a number of tasks are ready to run. In PC3000, 0 is the highest priority and 6 the lowest.

Processor Utilization Processor utilization is the percentage of processor execution time used by a specific task.

Rate Monotonic A rate monotonic task configuration is a configuration where the task priority is based on its interval. The shortest interval task has the highest priority with longer interval tasks having successively lower priorities.

RAM Random Access Memory. The memory within a computer which is used to store programs and data.

Real Time Clock The clock within the PC3000 which keeps track of the date and time.

RFI Radio Frequency Interference - this refers to radiated noise which may disturb the operation of a control system or communications link.

Scheduled A task is said to be scheduled when it is made ready to run.

Scheduler The Scheduler is a the part of the PC3000 which controls the execution of tasks on a periodic basis.

SEH Sequence Handler, software responsible for executing SFCs.

SFC Sequential Function Chart, a graphical representation of a sequence of steps and transitions, see PC3000 User Guide.

Simply Periodic Tasks are said to be simply periodic if each task (except the fastest) has an interval which is an integer multiple of the next faster task, eg. task intervals 10ms, 50ms, 100ms, 200ms.

Task A task is a group of function block instances (and optionally Sequential Function Charts) which are executed on a periodic basis by the Scheduler.

Task Function Block The task function block is used to control and monitor the scheduling of the task in which it is contained. Each task contains and is controlled by one task function block.

Task Ready A task is said to be 'ready' when it is due to be run. This does not necessarily mean that the task will immediately start to run because the pre-emptive Scheduler may make it wait until a higher priority task ihas finished running.

Source Code This contains diagnostic and textual information information required by the PC3000 Programming Station to re-create a particular User Program.

User Program The compiled form of the program created on either the PS or Microcell Programming Stations that embodies the control strategy for a particular production process and can be downloaded into the PC3000.

Warm Start A User Program has a warm start when all function blocks continue execution with parameter values retained from the time they last ceased execution, such as when power failed. For example, the **Process_Value** parameter of a PID will be set to the last value read from the plant.

Watchdog A watchdog is a device for detecting when a program mal-functions.

APPENDIX B SYSTEM ERRORS

This appendix lists system errors which may be detected by the PC3000 system software and are either registered in the system error log or returned as an 'EE' number via EI Bisync communications.

The system error log on the PC3000 stores the last 40 errors which have occurred within the PC3000, see paragraph System Error Log, in the Chapter Real Time System State Information.

System errors may be triggered by faults within the User Program, e.g. an illegal arithmetic operation such as dividing a floating point value by zero, or may indicate that there are major faults within the system. Faults in hardware modules may need to be investigated by Eurotherm Service personnel.

Power fail/watchdog errors

There are two event codes which indicate when the system is powered down (400) and when the system is re-started (499). Radio frequency interference (RFI) noise, intermittent hardware faults or possibly a software problem may cause the PC3000 system to detect a failure by means of a watchdog. There are two watchdog codes - 401 and 804; error 804 indicates that a task watchdog has triggered a system reset, error 401 indicates that the PC3000 system has failed to re-trigger the hardware watchdog.

Error 804 followed by a 401 is normal, as the task watchdog uses the hardware watchdog to restart the system. Immediately following the watchdog codes there will be a system re-start event (499) indicating that an attempt has been made to clear the fault. An occasional watchdog reset caused by system noise may be tolerable in some systems as a suitable start-up strategy can be chosen to re-start the User Program within seconds. If re-sets initiated by the watchdog occur frequently or repeatedly the reason should be investigated. This may imply that RFI disturbances should be reduced.

Mathematic operation errors

Errors in mathematics operations can occur within the function blocks, wiring and sequence program of the User Program. In integer mathematics operations, only division by zero is detected as an error. In floating point mathematics operations more detail is given in field 1 of the system error log to identify the type of mathematics function that caused the error (see Table 2). The three errors codes 500, 501 and 502 indicate different types of mathematics operation error (see Table 1).

System errors which are not referred to in the following tables, indicate a PC3000 internal system failure, and should be reported to Customer Services at Eurotherm Controls.

Table 1 Error codes

Error Code	Error Description	Field 1	Field 2
101	**Error** The User Program checksum has failed **Cause** (1) This may be due to corruption of the system by RFI noise etc. (2) This may be caused by downloading a user program which has been built with PC3000ORG set to the wrong value. Refer to Chapter 8; Memory Usaage. (3) A user program which was built on a system with downloadable function blocks installed may have been down-loaded by a system which does not have the same, or any, downloadable function blocks installed. (4) The user program, may be too large to fit in the available RAM in the PC3000. **Solution** (1) Re-download the User Program. If the error persists this probably indicates a problem with the build or download process. (2) Verify that the PC3000 is correctly set on the programming station then re-build the user program. (3) Install the correct downloadable function block library on the Programming Station. (4) If the user program is too large install more RAM in the PC3000 LCM.	Disagnostic information	User program checksum

Error Code	Error Description	Field 1	Field 2
305 & 307	**Error** Communication with an I/O module has failed after 3 retries **Cause** This most likely occurs when a module is removed or inserted but may also occur if a module resets, possibly due to its local watchdog. It may also be caused by RFI noise within the rack or on a multi-rack system in the inter-rack connections. A repeated failure from the same module is likely to indicate a hardware fault. **Solution** If a single module is consistently giving errors then it is likely there is a module failure and it should be replaced. Random errors indicate a RFI noise so screening and cabling may need to be improved.	Module address [1]	Diagnostic information

Note 1. The first digit (from the left) is the rack number (1 to 8) and the following two digits are the slot number (1 to 12).

Error Code	Error Description	Field 1	Field 2
308	**Error** Digital I/O system overloaded. **Cause** On initialisation, the I/O handler detected that there would not be sufficient band-width on the parallel bus to communicate with the number of modules defined in the User Program. **Solution** Use less digital I/O or extend the interval of the fastest task.	0	0
309	**Error** Analog I/O system overloaded. **Cause** On initialisation the I/O handler detected that there would not be sufficient band-width on the serial bus to communicate with the number of modules defined in the User Program. **Solution** Use less analog I/O or extend the interval of the slower IOH (See Appendix C, paragraph Allocation of Handlers to Tasks. Appendix C, Scheduler Overheads, Performance and Limitations).	0	0

Error Code	Error Description	Field 1	Field 2
400	**Error** The system shut down **Cause** Power failure was detected **Solution** If unexpected, investigate reason for power failure.	0	0
401	**Error** The system hardware watchdog has timed out. **Cause** This may be caused by RFI noise, an intermittent hardware fault, or a software mal-function. **Solution** If the error seldom occurs, it is probably caused by RFI noise or an intermittent hardware fault. If it is caused by a software fault, the error is likely to be repeatable and the diagnostic field will contain the same or a similar value, each time.	Diagnostic information	0
403	**Error** User Program cleared on command **Cause** The user has issued a Clear User Program command over Default Communications.	0	0

Error Code	Error Description	Field 1	Field 2
404	**Event** User Program cleared after start-up has been aborted. **Cause** The system has failed to carry on running for 30 seconds after ten attempts to re-start, and the system has therefore cleared the User Program. **Solution** The most frequent cause is due to the User Program being built with the wrong function block installation. Check that firmware version and function blocks are compatible.	Disagnostic information	0
499	**Event** The system was powered up	0	0
500	**Error** A mathematics operation on a floating point, (REAL) has resulted in an infinite result. **Cause** A mathematics operation such as division by zero has been performed which gave an infinite result. Once an infinity has occurred it is likely that the infinity will propogate throughout many mathematics operations causing many occurrences of the error. **Solutions** Investigate the User Program	Maths operation code [3]	Diagnostic information

Note 3. The Mathematics Operation Codes are listed in Table 2.

Error Code	Error Description	Field 1	Field 2
501	**Error** A floating point, (REAL) mathematics operation was performed with an illegal operand **Cause** A mathematics operation has been performed on a value which is not sensible such as ASIN (IN:=2). Such an operation will result in an IEEE NAN (similar to an infinity) and it is likely that this will propagate throughout many mathematics operations causing many errors to be reported. **Solution** Investigate the User Program	Maths operation code [3]	Diagnostic information
502	**Error** A mathematics operation on a floating point, (REAL) has resulted in an overflow. **Cause** A mathematics operation has been performed which results in a value which is too large to be represented. Such an operation will result in an IEEE infinity and it is likely that this will propagate throughout many mathematics operations causing a number of errors to be reported. **Solution** Investigate the User Program	Maths operation code [3]	Diagnostic information

Note 3. The Mathematics Operation Codes are listed in Table 2.

Error Code	Error Description	Field 1	Field 2
510	**Error** A mathematics operation on a floatng point (REAL) number has resulted in a error. (LCM-Plus only) **Cause** A mathematics operation such as division by zero has been performed which caused an error. Once an error has occured it is likely that an infinite or undefined result will propogate further causing many occurrences of the error. **Solution** Investigate the User Program to identify causes for the error. Try running the application without the sequence program (User program mode Seq-Held) to determine whethe the error is in the function blocks/wiring or the sequence program.	Diagnostic information [4]	Program counter
550	**Error** Integer division by zero **Cause** An attempt has been made to divide an integer by zero. **Solution** Investigate the User Program.	0	Diagnostic information

Note 4. Mathematics Error Diagnostic Information see Table 4.

Error Code	Error Description	Field 1	Field 2
800	**Error** Too many task function blocks. **Cause** The limit of seven task function blocks has been exceeded. **Solution** Reduce the number of tasks in the User Program.	0	0
801	**Error** Fastest task is too slow **Cause** The fastest task has a requested interval of greater than 65ms. **Solution** Reduce the interval of the fastest task	Task	0
802	**Error** Fastest task not have the highest priority **Cause** The fastest task does not have the highest priority. **Solution** Change the task priorities	Task	0
803	**Error** Task is not periodic. **Cause** The requested task interval is not an integer multiple of the next fastest. **Solution** Examine the task interval.	Number of the next faster task	Task

Error Code	Error Description	Field 1	Field 2
804	**Error** User task watchdog **Cause** A task has not completed within its watchdog time. **Solution** Adjust the task interval times and/or the watchdog times of the task function blocks.	Task	Program counter

Table 2 Mathematics operation codes

These codes are used to identify the mathematics operation errors and are supplied in the diagnostic fields of the system errors 500, 501 and 502.

Operation codes	Maths operation description
1	double to long conversion
2	double to single conversion
3	single to long conversion
11	single precision addition
12	single precision subtraction
13	single precision multiplication
14	single precision division
21	double precision addition
22	double precision subtraction
23	double precision multiplication
24	double precision division
25	double precision sqrt
26	double precision 1n
27	double precision log
28	double precision exp [1]
29	double precision sin
30	double precision cos
31	double precision tan
32	double precision asin
33	double precision acos
34	double precision atan
35	double precision mod
36	double precision expt [2]

Note 1. Natural exponent

Note 2. Exponentiation

EI-Bisync communications errors

The information is included here for diagnostic purposes and may be useful if a custom interface to PC3000 is being developed, for example, for a supervisory computer.

For further information on using Default Communications, refer to the PC3000 Communications Overview document.

The EE parameter is manually read by an EI Bisync communications link master device, such as a supervisory system, in response to receiving a Not Acknowledged (NAK) message. The EE parameter contains sufficient information to define the cause of the communications error.

The EE parameter is used to report error types to the supervisory computer. The data is a four digit hexadecimal number whose value indicates the type of error that has occurred. The EE parameter always contains the error code applicable to the last communications error detected by the PC3000 and consists of three fields, Error Code, Instrument Category and Error Category.

ERROR CODE								**INSTRUMENT CATEGORY**				**ERROR CATEGORY**			
Bit 15			Bit 12	Bit 11			Bit 8	Bit 7			Bit 4	Bit 3			Bit 0
Digit A				Digit B				Digit C				Digit D			

Error codes

The set of error codes which can be read from PC3000 as part of the EE parameter are defined in table 3.

Error code (Hex)	Cause of error
General errors	
00	No error
01	Invalid Mnemonic
02	Checksum Error
03	Line Error - e.g. parity, overrun, framing error
04	Read attempted on write only parameter
05	Write attepted on read only parameter
06	Invalid logical unit/channel number combination
07	Invalid data format
08	Data out of range
PC3000 Specific Errors	
20	Unsupported parameter type
21	Unable to encode data
22	Composite parameter not yet supported
23	Record separator (RS) expected
24	Too many elements
25	Structure nesting incorrect
26	Internal failure
File System Errors	
81	Too many open files
82	Unsupported file system
83	Incorrect file identity
84	Unknown open mode

Error code	Cause of error
85	No error
86	No space for directory
87	Not enough file storage
88	File does not exist
89	File exists
8A	Too many files
8B	File already open
8C	File already open for write
8D	File not open for read
8E	File not open for write
8F	No more file space
90	File storage unformatted
91	File name incxorrect

Instrument category number

This will always be read back as hexadecimal F.

Table 4 LCM-Plus Maths Error Diagnostic Information

The LCM-Plus math system error provides some diagnostic information in Field 1. The number in Field 1 should be converted into 32 binary digits. This conversion may be done using the numeric conversion utility built into the PS tools (Alt H). A one in any digit position can be interpreted as follows-

Category	Bit Pos.	Meaning	
Accrued Exception Flags	0	0	This bit always zero
	1	0	This bit always zero
	2	0	This bit always zero
	3	INEX	An inexact result has occured since the LCM first ran. An inexact result exception occurs when the result of a calculation cannot be represented exacltly in the binary floating point format. This exception is disabled in the PC3000 and inexact results will be rounded towards the nearest value and, in the case of a tie, to the nearest even result.
	4	DZ	A divide by zero has occurred since the LCM first ran.
	5	UNFL	An underflow has occurred since the LCM first ran. An underflow occurs when the result of a calculation is approximately $<1.2x10^{-38}$ and $>-1.2x10^{-38}$. Underflow exceptions are disabled on the PC3000 and all underflowed results will be rounded.
	6	OVFL	An overflow has occurred since the LCM first ran
	7	IOP	A BSUN, SNAN or OPERR has occurred since the LCM first ran
Exception Flags	8	INEX1	This type of exception is disabled in PC3000
	9	INEX2	This type of exception is disabled in PC3000
	10	DZ	A floating point division by zero has occurred
	11	UNFL	This type of exception is disabled in PC3000
	12	OVFL	An arithmetic operation has yielded a result too large to be represented in the floating point format, e.g. approximately $>3.4x10^{38}$ or $<-3.4 \times 10^{38}$
	13	OPERR	An illegal value has been used as the input to a function, e.g. ACOS(-10.0)
	14	SNAN	An illegal number (SNAN) has been used in an arithmetic operation
	15	BSUN	A comparison has been made between two numbers one or more of which was a NAN
Exception vector number	16-27	0000 1100 0000	Branch/Set on unordered A comparison was made between two values, at least vector one of which was a NAN.
		0000 1100 1000	Divide by zero exception A division by zero or LOG(0.0) has been performed.
		0000 1101 0000	Operand error exception An operation was performed where the function has no meaning for the given input e.g.ACOS(10.0)
		0000 1101 0100	Overflow exception An arithmetic operation has resulted in a value which is too large to be represented in the floating point format, e.g. approximately $>3.4x10^{38}$ or $< -3.4x10^{38}$
Exception	281-31	0000	Pre-Instruction Exception An arithmetic or conditional instruction frame type has been initiated when an exception was pending from a previously executed, concurrent instruction.
		1001	Mid-Instruction Exception A move from a floating point register to memory has caused an exception.

Note: The IEEE standard for binary floating-point arithmetic (IEEE P754 standard) defines a value for floating point numbers called NAN. This stands for Not-A-Number and is the result returned for functions such as ACOS(12.0) where no mathematically interpretation exists.

Error categories

The following error categories are defined.

Table 5 Error categories

Error category Number	Error type
	Recoverable errors
0	No error found
1	Character orientated error (UART) e.g. parity
2	Message data error e.g. checksum error
	Non-recoverable Errors
7	Invalid message e.g. unknown mnemonic
8	Invalid message content e.g. attemping to write to read only parameter.

A recoverable error is defined as any error that can be corrected by re-transmission of the same message, i.e. a supervisory computer does not have to take any other corrective action other than to resend the message.

PC3000 Programming station system error reporting

If the Programming Station receives a “not acknowledged” (NAK) message from the PC3000, it reads the EE error parameter. The error code is then normally displayed as a textual message.

If the Programming Station is unable to interpret the EE code, the error code is displayed as a four digit number as “Error XXXX”. The number is created by adding 5000 to the high byte of the EE code. For example, an error code of 81F7 would appear on the Programming Station as “Error 5129” (note, 81 in hexadecimal is 129 decimal).

APPENDIX C SCHEDULER OVERHEADS, PERFORMANCE AND LIMITATIONS

This appendix provides further detailed information on the behaviour of the Real Time Task Scheduler which will help you understand the system overheads, performance and limitations of the Scheduler.

Processing of communications, I/O, function blocks and Sequential Function Charts is all controlled by software components called handlers. These are normally run to perform specific services to support task execution and therefore add to the task processor utilization. Some understanding of the handlers may be useful when considering scheduling and performance problems. This appendix also outlines the purpose of the handlers and gives detail of their operation.

Function of handlers

Default Communications Handler (COH) :-

The COH provides communications on all LCM ports which have not been defined by a running User Program for use with a specific communications protocol, see paragraph Default Communications, in the Chapter 1 Overview.

I/O Handlers (IOH) :-

There are two I/O handlers, one for fast I/O and another for slower I/O; referred to in the following text as the Fast IOH and Slow IOH . In general digital I/O is handled by the fast I/O handler and analogue I/O by the slower I/O handler. The I/O handlers control the transfer of data between I/O modules and the LCM.

Function Block Handler (FBH) :-

The function block handler controls the execution of all the function blocks within a task.

Sequence Handler (SEH):-

The sequence handler controls the execution of the Sequential Function Charts (SFCs) within the User Program.

Allocation of handlers to tasks

The handlers are allocated to tasks as follows:

FBH

The FBH is run in every task .

SEH

The SEH is executed in the task which contains the SFCs.

COH

The COH is allocated to the fastest task and does not add any significant processing overhead if there are no communications active on any ports. However, it may be necessary to move the communications execution into a slower task if there is significant communications activity. This can be achieved by associating an EI_Bisync_S to the port and then allocating the function block to a slower task.

The communications driver function block will then be run by the FBH in the task to which it has been allocated.

IOH

The Fast IOH always runs in the fastest task because all digital I/O transactions with the modules occur at the basic tick interval.

The fastest task should have the highest priority because the I/O modules should be read and written soon after the system tick.

The Scheduler will select a task to run the Slow IOH depending on the I/O channel function blocks associated with the task and the task interval.

The following function blocks determine which task is used to run the Slow IOH :

Analog_In
Analog_Out
T_Prop_Out
PIM2
PI_Smpl_Ctr

Table A Function Blocks associated with the Slow IOH

Note: The task interval allocated to function blocks in table A should not be shorter than 100ms, see Chapter Real Time Task Scheduler, paragraph Rules when changing Task Intervals.

The Slow IOH is normally allocated to the fastest task which contains one or more instances of the function blocks given in table A.

However, there is a limitation that the Slow IOH should run in a task with an interval no longer than 20 times that of the Fast IOH.

Otherwise, the Slow IOH is assigned to the slowest task with an interval less than or equal to 20 times the interval of the Fast IOH.

If there are no instances of the I/O function blocks in table A and therefore no associated task, the Slow IOH will run with the Fast IOH in the fastest task.

Sometimes the overheads of running the Slow IOH on the fastest task can be significant. In which case, the Scheduler can be forced to run the Slow IOH in a slower task, by creating an instance of a T_Prop_Out channel on a spare digital output. If this is not possible, a dummy PIM2 function block can be generated with an empty module address. This will cause the Slow IOH to run in the slower task allocated to these dummy channels. An error will be logged for the PIM2 function block, indicating that there is a missing module however, in this case, this error can be ignored.

Handler execution order

When writing User Programs some understanding of the order of execution of the handlers can be useful.

Both Fast and Slow I/O handlers are split into Post-tick and Pre-tick routines, to ensure that the delay between processing inputs and outputs is minimised. The inputs are read in by the Post-tick IOH. The function blocks which may use these inputs are then executed by the FBH. Finally the outputs are updated by the Pre-tick IOH. This ordering ensures that an output which is soft wired to an input will propagate within one execution of the task.

In each task, the order of execution can be summarised as follows:

1) Post-tick fast IOH (if fast IOH is in this task)
2) Post-tick slower IOH (if slower IOH is in this task)
3) FBH (function blocks assigned to this task only)
4) COH (if COH is in this task)
5) Pre-tick fast IOH (if fast IOH is in this task)
6) SEH (if Sequential Function Charts is in this task)
7) Pre-tick slower IOH (if slower IOH is in this task)

Changing task priorities

The priority of a task is governed by the task function block's **Priority** parameter and is used to govern which task is to be executed when the Scheduler has two or more tasks ready to run. 0 represents the highest priority and 6 the lowest. In the Chapter Real Time Task Scheduler, only rate monotonic task configurations were considered, i.e. the fastest task has the highest priority, with slower tasks having progressively lower priorities.

Tasks with the same interval

Varying task priorities can benefit situations where it is necessary to have two or more tasks with equal interval which should be executed in the correct order. A configuration with tasks of equal interval, is valid.

> **Note:** The priority of equal interval tasks can be in any order so long as faster tasks have higher priority and slower tasks have lower priority in order to remain rate monotonic ordering.

Non-rate monotonic task configurations

Figure 1 shows the scheduling of a non-rate monotonic task configuration. In the example, the 100ms task has been given a priority which is higher than that the 50ms task (the 100 ms task's priority value is lower). Setting the priorities in this way ensures that the 50ms task is not executed until the 100ms task has finished executing.

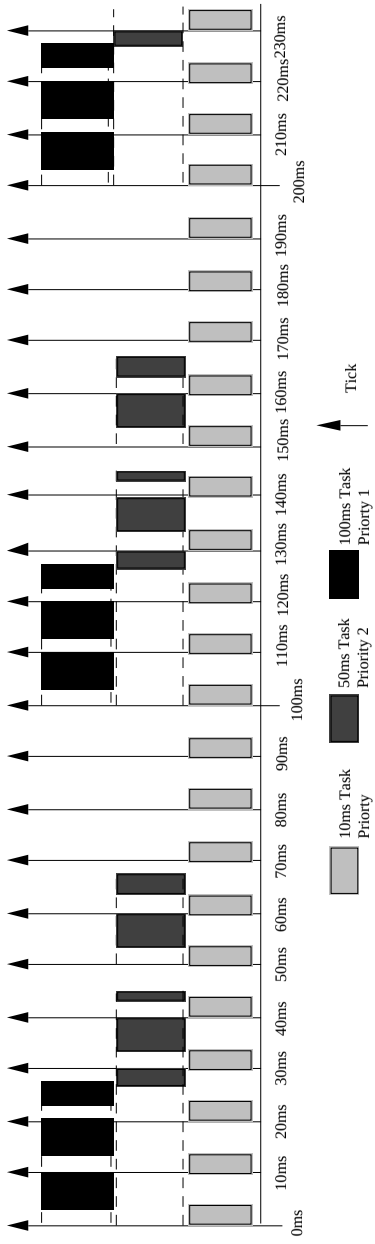

Figure 1 Scheduling with complex (non-rate monotronic) priority configuration

Problems with overrunning using non-rate monotonic tasks

In figure 1 the first execution of the 50ms task following the 100ms task is close to overrunning. There is always a high risk of overrunning when using non-rate monotonic task configurations. Figure 2 shows the effect of an overrunning non-rate monotonic User Program.

Unlike a rate monotonic task configuration, when overrunning occurs there may still be some time during the execution when the PC3000 is idling.

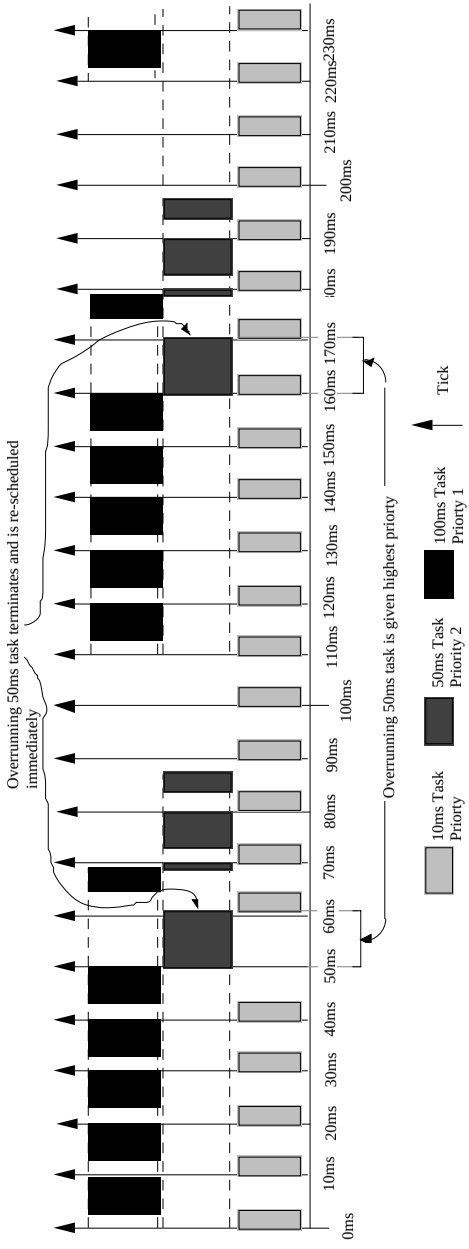

Figure 2 Overrunning caused by priority configuration

Parameter passing between tasks

When parameters are passed between tasks it may be important to know which tasks are associated with processing the source and destination parameters.

The following points should be noted about inter-task parameter passing:

1. There is no buffering of parameters between tasks, i.e. there is no guarantee that two parameters from the same task referred to from another task will be from the same task execution, i.e. the parameter values may not be coherent.

2. Evaluation of soft wiring is executed in the destination task.

3. The writing of parameter values is indivisible for example, the writing of the result of a wiring expression, say a 100 character long string, into the function block parameter to which it is wired, cannot be interrupted by any other task. This ensures that a partially updated string would never be read by any task.

Coherency limitations

The following example shown in figure 3 demonstrates the effect of passing parameters between tasks without having coherent values.

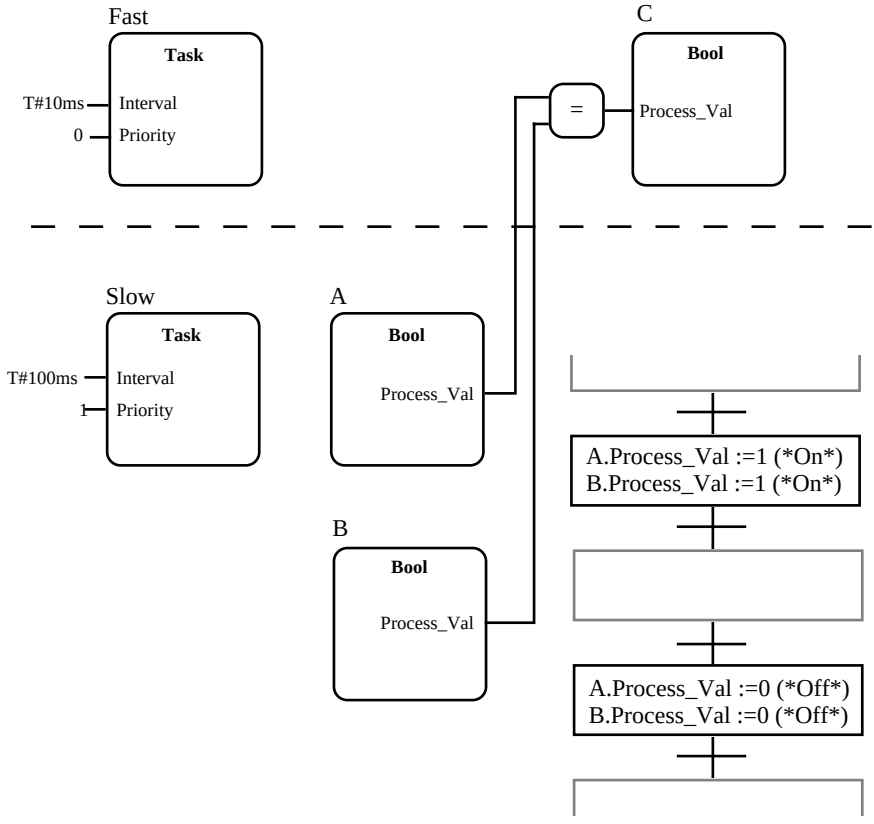

Figure 3 An example of inter task parameter passing incoherence

The wiring:

```
C.Process_Val:= (A.Process_Val = B.Process_Val);
```

is evaluated by the Fast (10ms) task. It might be assumed that C.Process_Val would always be 1 i.e. On; this is not always the case. In the example, the priority of the

Fast (10ms) task is highest which implies that the wiring will be evaluated indivisibly (without interruption from the Slow (100ms) task). However, the Fast task can interrupt the assignments to `A.Process_Val` and `B.Process_Val` in the Slow task.

This gives rise to the possibility of `C.Process_Val` being 0 i.e. Off. Table 5 shows that the possible values of C when the Fast task is executed and interrupts the 100ms task.

SFC	AProcess_Val	B.Process_Val	C.Process_Val
	0(*Off*)	0(*Off*)	1(*On*)[1]
A.Process_Val:=1 (*On*)[2]			
	1(*On*)	0(*Off*)	0(*Off*)[1]
B.Process_Val:=1(*On*)[2]			
	1(*On*)	0(*On*)	0(*On*)[1]
A.Process_Val:=0(*Off*)[2]			
	1(*Off*)	0(*On*)	0(*Off*)[1]
B.Process_Val:=0(*Off*)[2]			
	1(*Off*)	0(*Off*)	0(*On*)[1]

Time

Note 1. Fast (10ms) Task executes

Note 2. Slow (100ms) Task executes

Table 5 Example Of 100ms sequential function chart task interrupted by 10ms wiring task

.

Avoiding problems with parameter incoherence

Parameter value incoherent can be avoided using the following techniques:

Avoid using expressions which require coherent values from another task, such as edges (changes in digital (BOOL) values) which occur at the same time in the same task and are then fed into a single function block in another task.

In the previous example, the BOOL function block instance C can be moved to the Slow task. Because the BOOL function blocks would then all be in the same task, the values of A and B will be coherent when C is evaluated.

Where an input is wired to an expression which includes parameters from more than one task, it may be necessary to have a user variable function block instance in each task (i.e. other than the destination task). These would then be wired to hold the value of a partial result by accessing parameters in the same task, as shown in Figure 4.

This technique is only applicable where there is just one partial result per task.

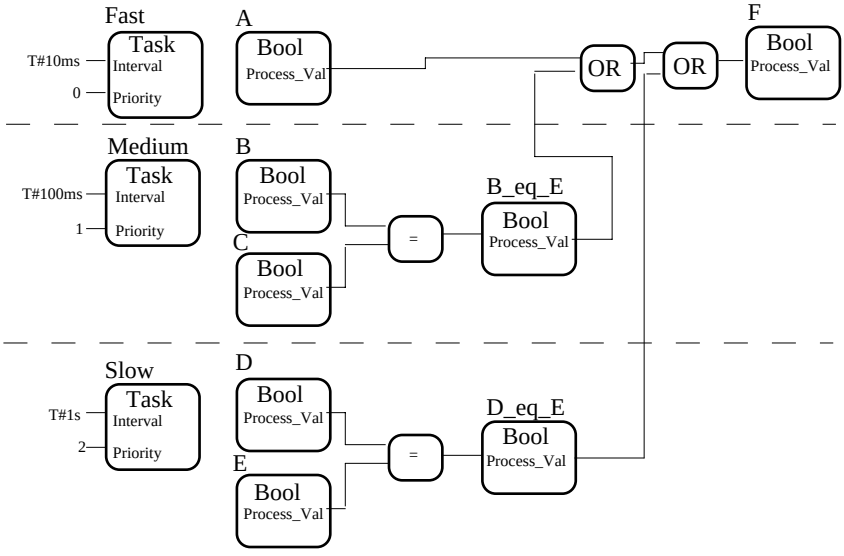

Figure 4 Retaining coherence by using partial results

Delays caused by inter-task parameter passing

Parameter values can be passed between tasks either explicitly through wiring, Sequential Function Chart Step assignments or implicitly through connection to I/O modules via the I/O handlers. These inter-task parameter links can, in some cases cause extended propagation delays. Although not usually important, the following paragraphs provide details on some of the effects which may occur.

Parameter passing between function blocks in different tasks

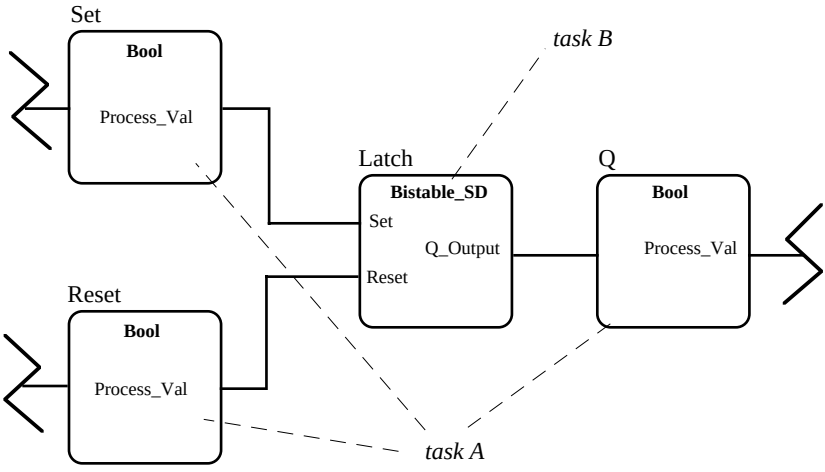

Figure 5 Example of typical chained wiring

If the function block instances are in different tasks, as shown in Figure 5, it is possible to have a propagation delay longer than the execution times of the blocks. This is due to the task priorities modifying the order of execution.

Consider the situation where the instances Set, Reset and Q are in task A with priority 2 and the instance Latch is in another task B of priority 1. To simplify the example, consider the case where both tasks have the same interval.

The higher priority task B is executed first. This task evaluates the inputs **Latch**. **Set** and **Latch.Reset** first, taking the values of **Set.Process_Val** and **Reset.Process_Val** from the last execution of the Bool function block instances Set and Reset. The Latch function block then executes to produce the **Q_Output**. Once the higher priority task has executed, the Bool function blocks in the lower priority task A are processed. **Set.Process_Val** and **Reset.Process_Val** are assigned their new values and **Q.Process_Val** is set to the new value of **Latch.Q_Output**. (Where these values come from/go to is not relevant to this example.)

To summarise, a delay of one execution cycle occurs between the changing of Set.Process_Val and/or Reset.Process_Val and the corresponding **Latch.Q_Output** and **Q.Process_Val** being assigned.

If the intervals are different, the delay calculation becomes more complex. If the interval of the higher priority task is longer than that of the lower priority task then

the effect is equivalent to one (high priority task) execution cycle delay in the Set and Reset function block instances. If the interval of the lower priority task is longer than that of the higher priority task then the effect is equivalent to one (low priority task) execution cycle delay in updating the Q function block instance.

Delays in writing to function blocks from SFCs

Because the Sequential Function Charts (SFCs) are executed (by the SEH) at the end of the task to which they are assigned, delays may occur that are similar to those for function block wiring.

Care must be taken when assigning a value to an input of a function block from the sequence (SFC) and then taking action on the output of that function block. For example the following SFC will not work.

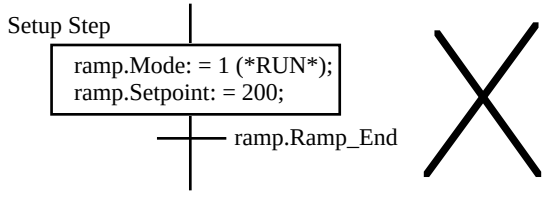

Figure 6 Example of incorrect control of a function block from an SFC

In the example shown in Figure 6 the function block intstance **ramp** may not have executed by the time **ramp. Ramp_End** is tested by the sequence handler (SEH) and subsequently to **ramp. Mode** and ramp. Setpoint being written. If this is the case and ramp.Ramp_End was true prior to the step executing then execution of the sequence will move on even though the **ramp** has not reached 200.

The simplest method of circumventing this problem is to test for `ramp.Output = ramp.Setpoint` as in Figure 7. Another method which may be more appropriate in other, similar situations would be to add a time delay equal to the interval of the function blocks task, in order to ensure that the block executes.

e.g. `ramp.Ramp_End AND Setup.Time>=T#100ms` where ramp is in a 100ms task

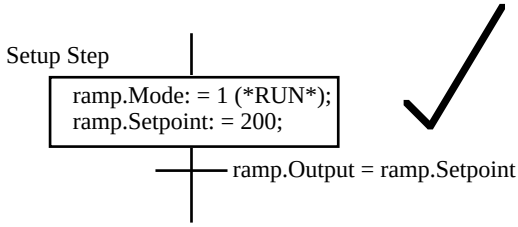

Figure 7 Example of correct control of a function block from SFC

Delays in connection with I/O

The Fast and Slow I/O handlers, see paragraph Allocation of Handlers to Tasks, transfer the I/O values between the I/O channel function blocks and their associated I/O modules. Because these are assigned to tasks in a similar way to function blocks, there is an associated propagation delay.

Each I/O Handler is processed in two phases by Post-tick and Pre-tick routines to optimise I/O processing time. In tasks that contain an I/O handler, the Post-tick routine of the IOH is executed first to fetch inputs, the function blocks are then executed, followed by the Pre-tick routine to update the outputs. This ensures that values can be propagated from inputs to outputs in one task execution cycle.

Because of inter-task parameter passing, a delay is unavoidable when evaluating I/O values in a task which is not associated with the appropriate I/O handler. Consider an example where the Fast I/O handler is associated with a 10ms task and wiring connects Digital_In to a Digital_Out function blocks both running in a 100ms task of lower priority. In this case the propagation delay from digital input to output would be between 10ms best case and 120ms worse case, depending on exactly when the input value changes.

> **Note:** Although the wiring between the Digital_In to Digital_Out is evaluated during the 100ms task, the output will not be written out to the digital I/O module until the next execution of the Fast IOH.

APPENDIX D EXAMPLE TASK CONFIGURATIONS

These example task configurations are provided to assist with the initial selection of tasks suitable for different types of process application.

Example 1

Task Name	Interval	Priority
Fast	5ms	0
Normal	100ms	1
Slow	500ms	2

These tasks will be suitable for an application that requires fast digital responses but also requires slower analogue function blocks.

The Fast task is used for a small selected set of digital function blocks and digital I/O channel function blocks that are required to have fast response times.

The Normal task can be used for PID control loops that have a time constants typically greater than 10 seconds, e.g. those controlling temperature. Digital function blocks concerned with slower acting devices and the Sequential Function Charts are also run in this task.

The Slow task is used to run analogue function blocks used for devices with slow time constants, i.e. greater than 50 seconds or for which the monitoring of rapid value changes is not important.

> **Note:** In this case, the fast IOH will be processed in the Fast task, and the slow IOH will run in the Normal task.

Example 2

Task Name	Interval	Priority
Fast	5ms	0
Normal	100ms	1
Slow	500ms	2

These tasks will be suitable for an application having a large number of digital and analogue function blocks but where response times slower than provided by the default tasks are acceptable. The Slow task is used to run a communications function block responsible for issuing a log message to a printer.

The Fast task is used for all digital function blocks and digital I/O channel function blocks.

The Normal task can be used for all analogue function blocks and the Sequential Function Chart execution.

The Slow task is used to run a communications function block, such as Raw_Comms, to issue a log message to a printer. In the case of the Raw_Comms block, two executions of the block are needed to transmit a message, i.e. a message will be logged every 10 seconds.

> **Note:** In this example, the fast IOH will be processed in the Fast task, and the slow IOH will run in the Normal task.

Example 3

Task Name	Interval	Priority
Fast 1	20ms	0
Fast 2	20ms	1
Normal	200ms	2

In this example, there are two fast tasks of different priority. These are required because there is a need for both fast digital responses and high speed communications.

The Fast1 task is used for all digital function blocks and digital I/O channel function blocks.

The Fast2 task is allocated to all communications function blocks.

The Normal task is used to run all analogue function blocks and the Sequential Function Charts.

Where a User Program requires both fast communications and digital functions, it is advisable to run these in separate tasks.

> **Note:** In this example, the fast IOH will be processed in the Fast1 task, and the slow IOH will run in the Normal task.