Colter Group

FMT/BIS Training Manual

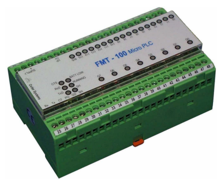
Colter Group Ltd

Unit 7 Zone C, Chelmsford Road Industrial Estate, Dunmow, Essex, CM6 1HD
Tel: 01371 876887 Fax: 01371 875638 Email: sales@coltergroup.co.uk

Contents

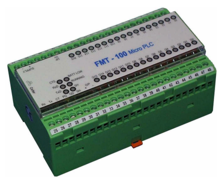
FMT/BIS Products

FMT-100A and FMT-100B



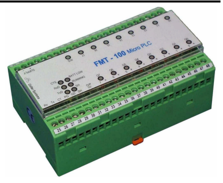
- 16 inputs
- 8 relay outputs

FMT-100C and FMT-100D



- 16 inputs
- 16 outputs, PNP 100 mA

FMT-100E and FMT-100F

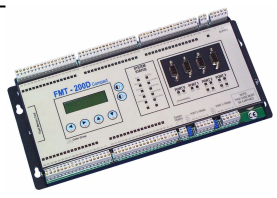


- 8 inputs
- 8 outputs, PNP 500 mA

FMT-100J

- 4 analogue inputs, 8-bit 0-10v, 0-4v, or 0-20 mA
- 4 analogue outputs, 8-bit 0-10V and 0-20 mA with JACO
- 8 digital inputs
- 5 digital outputs, PNP 100 mA

FMT-200D



- 48 inputs, 24v bipolar
- 32 outputs, PNP 500 mA

FMT-200J

- 8 analogue inputs, 12-bit 0-10v, 0-4v, or 0-20 mA
- 4 analogue outputs, 12-bit 0-20 mA
- 32 inputs, 24v bipolar
- 24 outputs, PNP 500 mA

BIS-100

- Plug in Fieldbus Module
- 4 inputs, 24v bipolar
- 4 outputs, Relay

FMT-400

- Expandable rack system with plug-in cards
- Up to 1024 digital in and 1024 digital out
- Up to 256 analogue in and 256 analogue out
- Up to 8 communication ports - 6 as RS485
- Up to 4 fieldbus modules
- High speed input module
- Plug-in Flash card module

Controller Features

- Opto-isolation on all analogue and digital inputs and outputs.
- Real Time Clock.
- Communications.
  - FMT-100
    One programming/general purpose RS232
  - FMT-200
    One programming/general purpose RS232
    One general purpose RS232
    Two general purpose RS232 or RS485
  - FMT-400
    One programming/general purpose RS232
    One general purpose RS232
    Up to 6 general purpose RS232 or RS485
    Up to 4 Fieldbus modules
  - BIS-100
    One programming/general purpose RS232
    One general purpose RS232
    Two general purpose RS232 or RS485
    One Fieldbus modules
- PID
- Built-in Communication Protocols:
  Modbus RTU
  Mitsubishi 'Protocol 1'
  Linkline and Linkline Plus
- LED indication for:
  All digital inputs and outputs
  System healthy
  Serial port activity
  Running
- High speed features:
  Incremental pulse encoder
  Up/down counter
  Fast edge catching
  Interrupt driven instruction module
- Data Storage:
  Battery backed RAM (all FMT's)
  Plug in Flash Card (FMT-200 and 400)
- Display and Function Keys (FMT-200 and FMT-400):
  Use under program control for error messages, changing set points etc.
  View and clear system errors and warnings
  Set date and time
  Set programming station number
  Set or clear I/O forces
  Set monitor mode
  View analogue inputs

Facilities

- Inputs
- Outputs
- Analogue Inputs
- Analogue Outputs
- Flags
- Counters 0 - 99,999,999
- Timers 0 - 99 hours 59 minutes 59.99 seconds
- 16-Bit (0 - 65,535) and 32-Bit registers (0 - 4,294,967,295)
- Floating Point (9.2e+18 to 5.4e-20 positive or -9.2e+18 to -2.7e-20)
- Text strings
- Internal registers:

  Information: scan time, divide remainder.

  Date and Time

  Facility preservation during power down

  High speed options

  Communication Status
- Internal Flags:

  Clock pulses

  Errors and warnings

  Communication status
- Allocated RAM
- Flash Cards

  Data Storage

  Program Storage

Internal Operation

- What is a Project?

- What is a Module?

  Ladder Modules

  Instruction Modules

- Principles of Operation

  Operation with Ladder modules only

  Operation with Instruction modules only

  Operation with both Ladder and Instruction modules

- Optimising Performance

  I/O update time.

  I/O used.

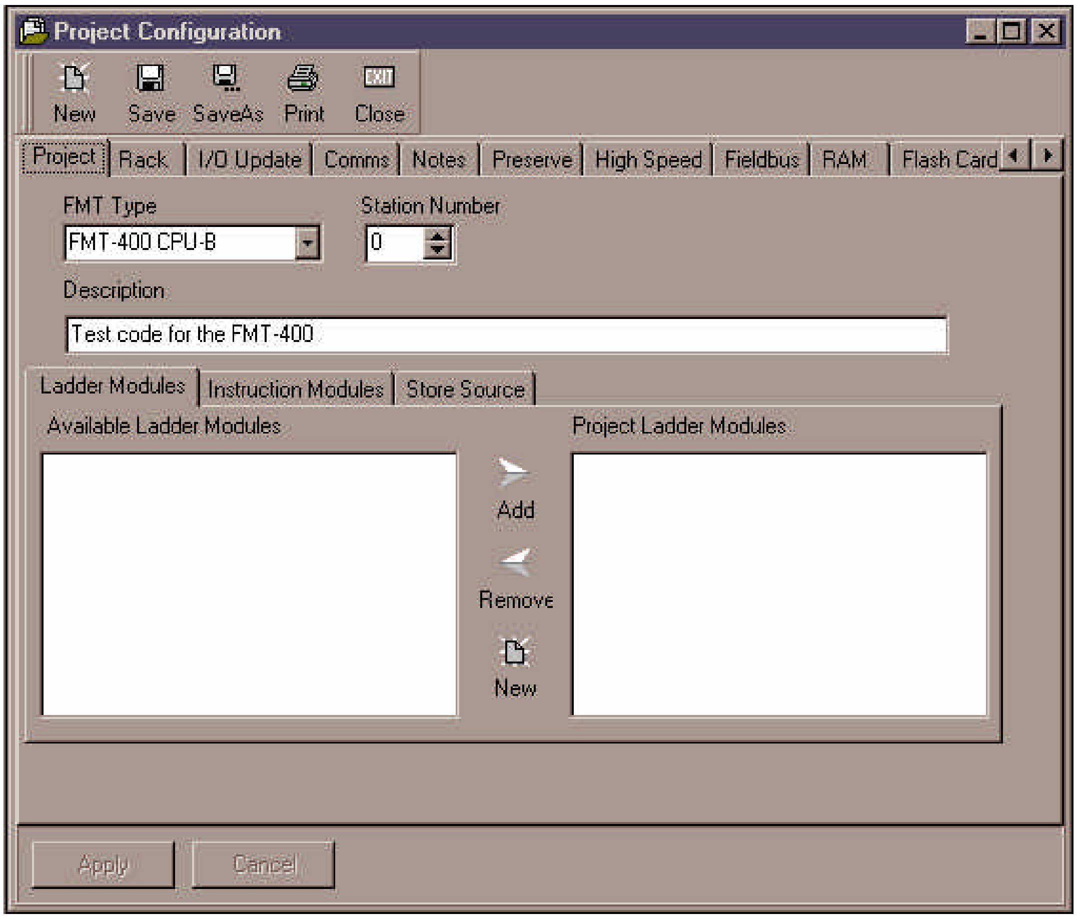
Flex32 programming package

Project Configuration

The project configuration form is where most of the details for the project are entered. Some configuration can be made without requiring a full compile and download; these are handled under the On-Line adjustments.

The FMT type, Station number and Description are entered when the project was created, but may be edited from the 'Project' page of the main Project Configuration screen.

The available modules, both Ladder and Instruction, can be viewed or added to/removed from the project on the tabbed pages on the project pages. Source Store downloads the entire project (including comments, symbol names, and source code) into the non-volatile memory of the FMT.



FMT Type: FMT-400 CPU-B

Station Number: 0

Description: Test code for the FMT-400

Available Ladder Modules

Project Ladder Modules

Other pages on this form are for configuring:

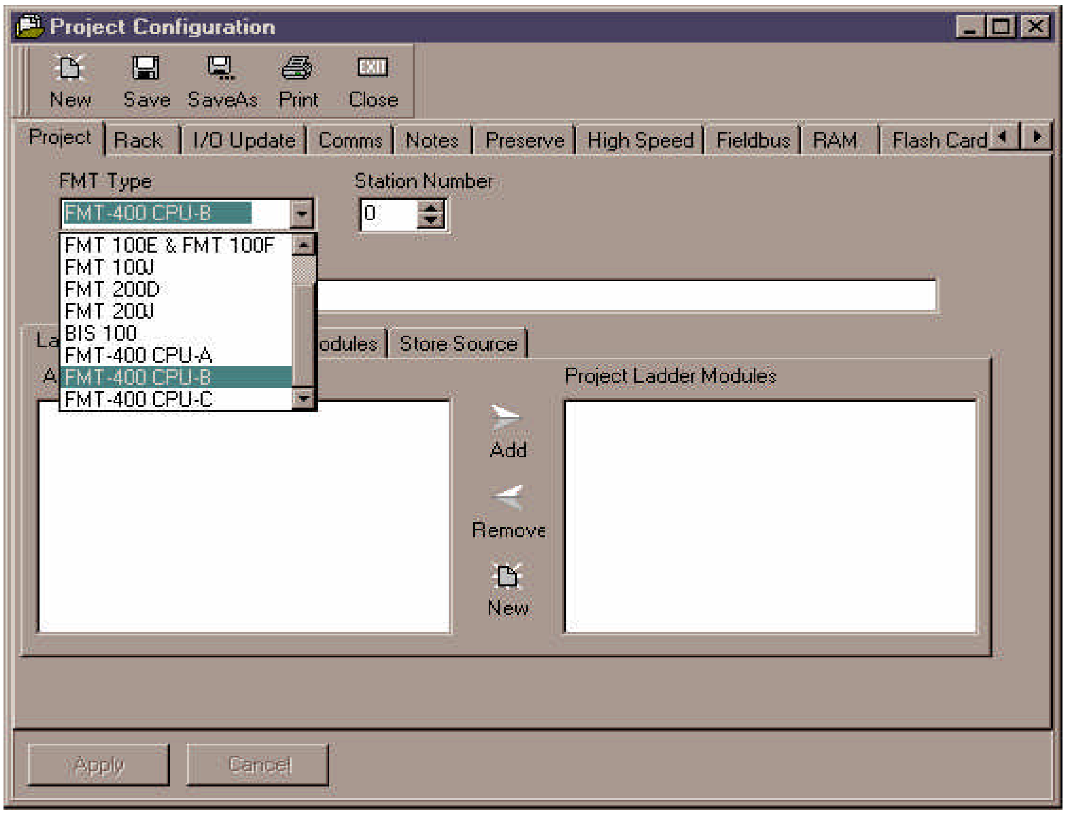
Rack: Configuration of modules within an FMT-400 rack.

The FMT-400 modular controller requires the arrangement of modules within a rack to be configured as part of the project.

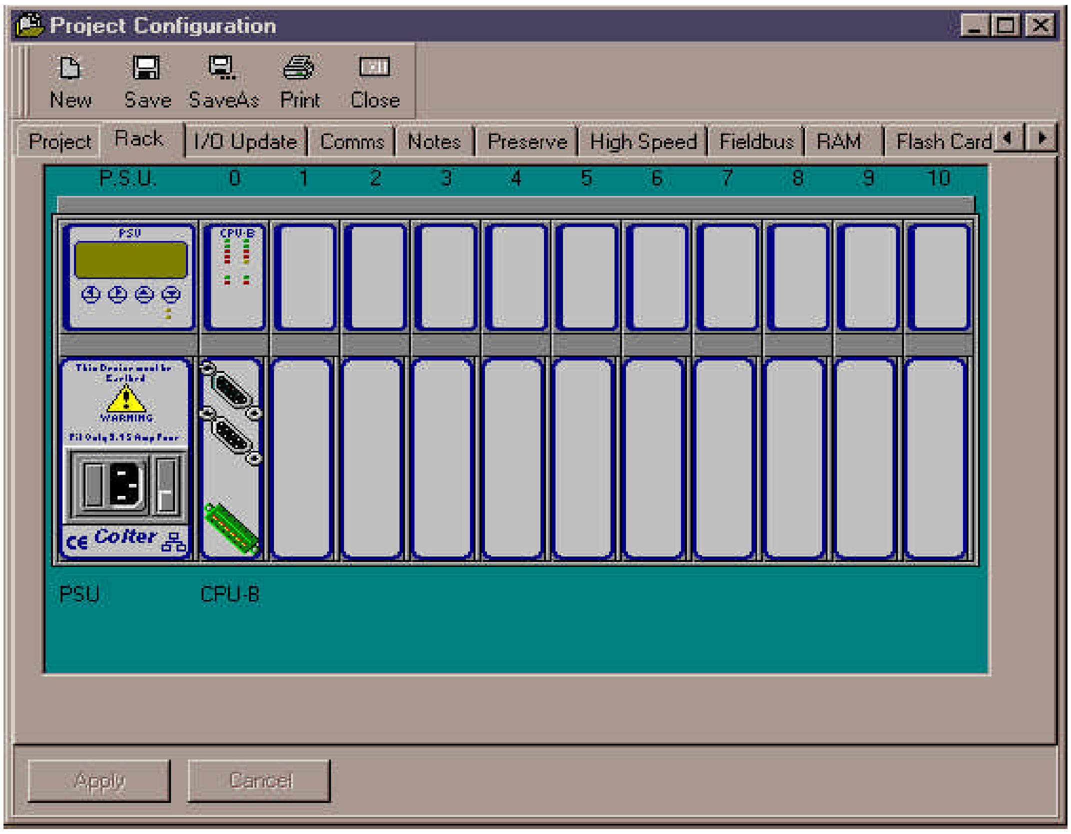
The FMT-400 system comprises of the Rack (Full or Half rack), the CPU module of your choice, the power supply and the I/O modules of your choice. After deciding where the Modules should go in the rack they should be physically placed in their positions (with the power turned OFF).

After the modules have been placed in position you can now begin configuring the FMT-400 using Flex32. The following screen-shots highlight the key points of configuring the FMT-400:

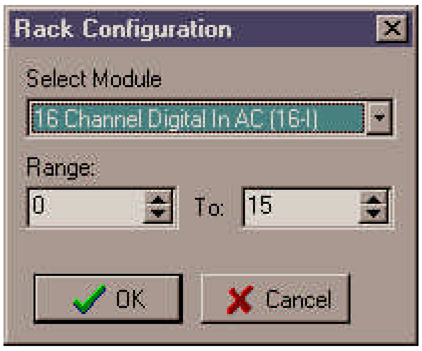
The CPU type that is to be used should first be selected from the ‘FMT Type’ drop down menu within the ‘Project’ page of the Flex32 project configuration screen....



The rack configuration page is selected by clicking on the ‘Rack’ tab of the project configuration screen....

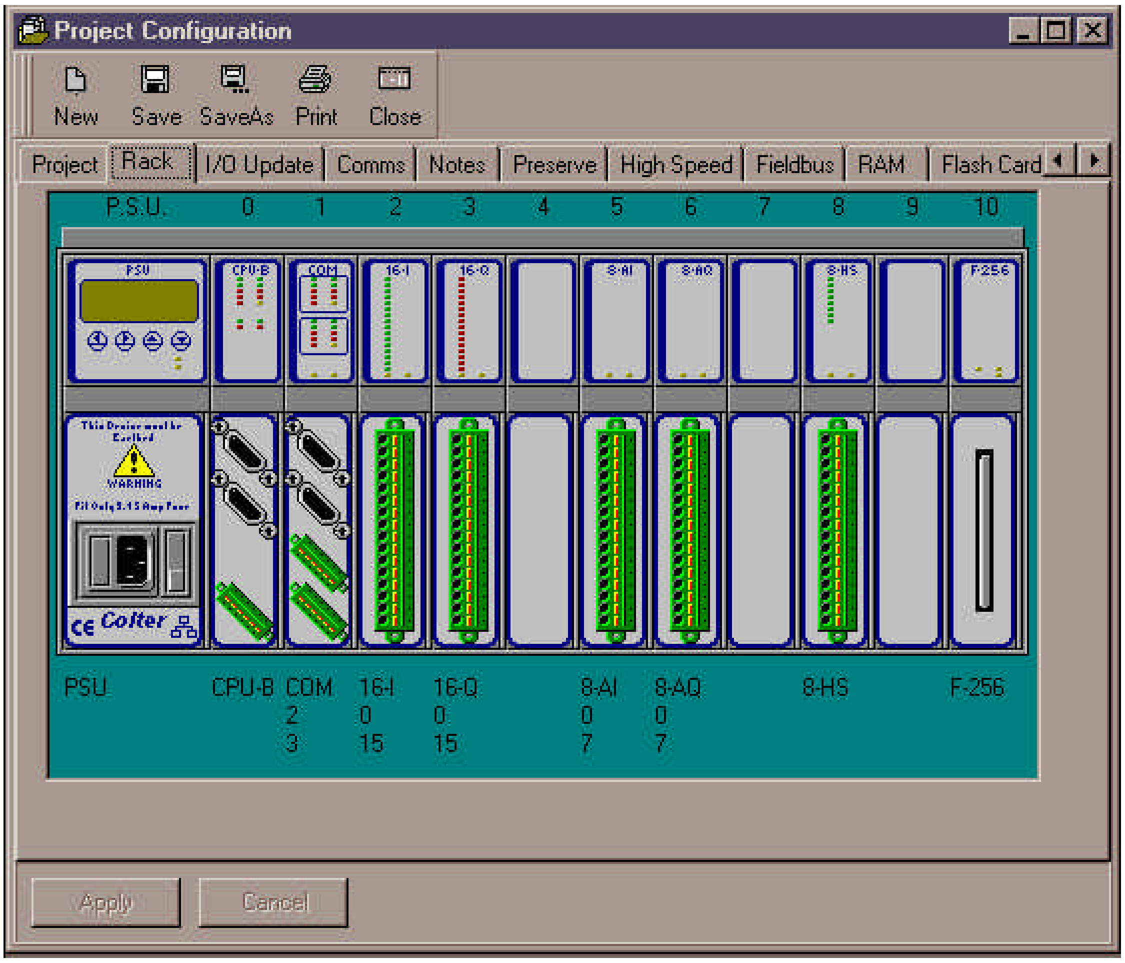


To change the type or rack click on the edge of the rack at the left hand side of the picture. To change a module, click on the picture of that module. The Rack Configuration Dialog will appear....



Select the new element form the drop down list, then enter the range (if required) in the boxes. The example above shows a 16 channel input card configured as I0 to I15.

Continue in this way until the rack is complete...



PSU	CPU-B	COM	16-I	16-Q		8-AI	8-AQ		8-HS		F-256
		2	0	0		0	0				
		3	15	15		7	7				

After the rack configuration is downloaded to the FMT as part of the compile and download process. If the firmware detects a discrepancy between the actual hardware and the downloaded configuration then an appropriate warning is raised.

<u>I/O update:</u> Set the intervals at which the firmware updates the digital and analogue inputs and outputs

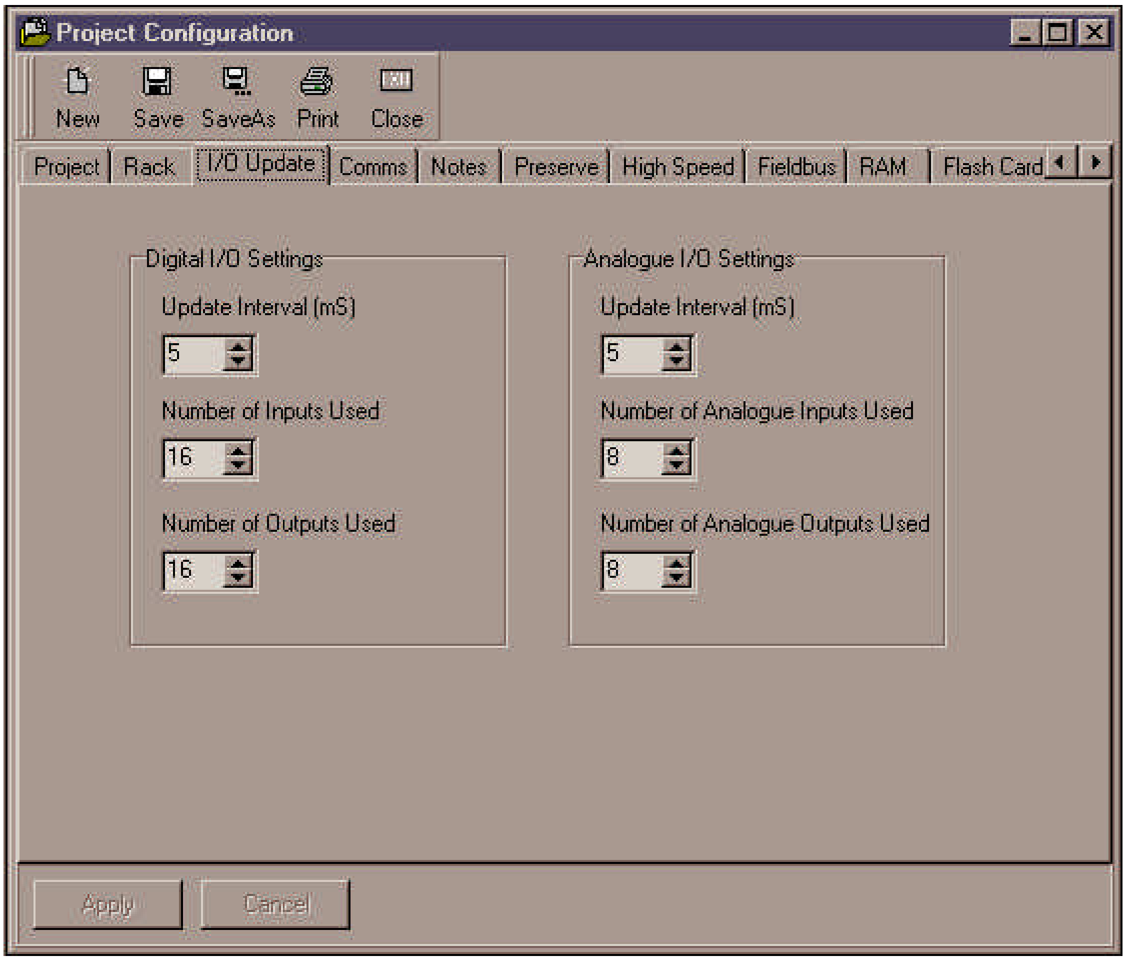
These settings control how often the firmware update the I/O hardware.

Digital and Analogue Update Intervals:

If your project uses any ladder modules the the I/O update is handled at the end of every ladder scan and these settings have no effect.
If your project only uses instruction language modules then you can set the number of milliseconds between I/O updates depending on the nature of your application. The more often you update the I/O the slower the code will execute. Setting the I/O interval to 0 will update the I/O in between every line of code, this gives the fastest I/O update but the slowest code execution.

Number of Inputs and Outputs Used:

When set to zero the firmware will update all the analogue and digital I/O. In some applications where I/O is not used you can set the number of each facility you wish to be processed; other facilities outside of this range will be ignored and seen by the program as always off.

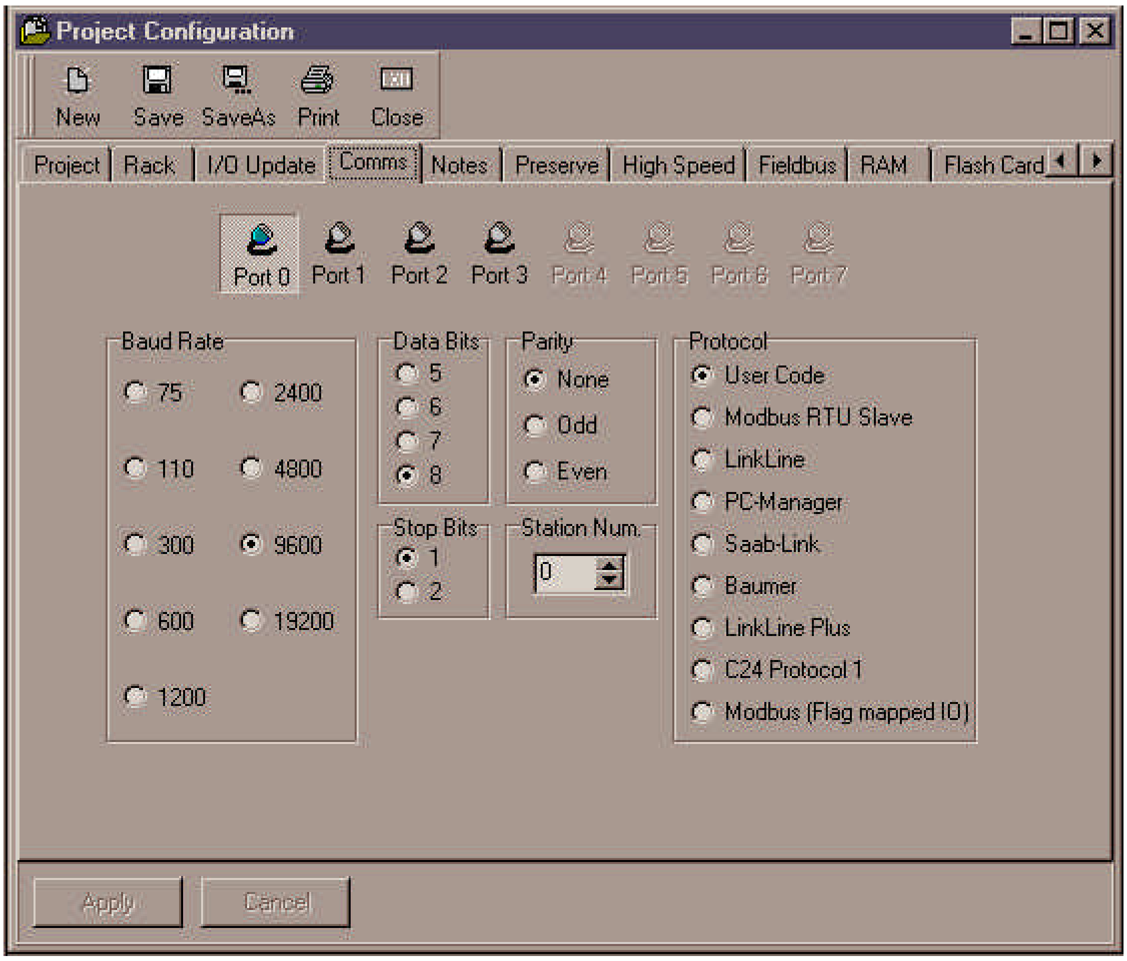


Digital I/O Settings
Update Interval (mS) 5
Number of Inputs Used 16
Number of Outputs Used 16

Analogue I/O Settings
Update Interval (mS) 5
Number of Analogue Inputs Used 8
Number of Analogue Outputs Used 8

Comms: The settings of the FMT communication ports.

All FMT and BIS controllers have serial communication ports; This screen is where you can set up the communications parameters for each of the comms ports on the controller you are using.



Baud Rate: 75 110 300 600 1200 2400 4800 [✓ 9600] 19200

Data Bits: 5 6 7 [✓ 8]

Parity: [✓ None] Odd Even

Stop Bits: [✓ 1] 2

Station Num. 0

Protocol: [✓ User Code] Modbus RTU Slave LinkLine PC-Manager Saab-Link Baumer LinkLine Plus C24 Protocol 1 Modbus (Flag mapped IO)

Select the port you wish to configure by clicking on the buttons labelled Port0 to Port7. Only the buttons for the ports available on the controller you are using will be enabled.

Select the baud rate, number of data bits, number of stop bits, and parity from the radio button boxes.

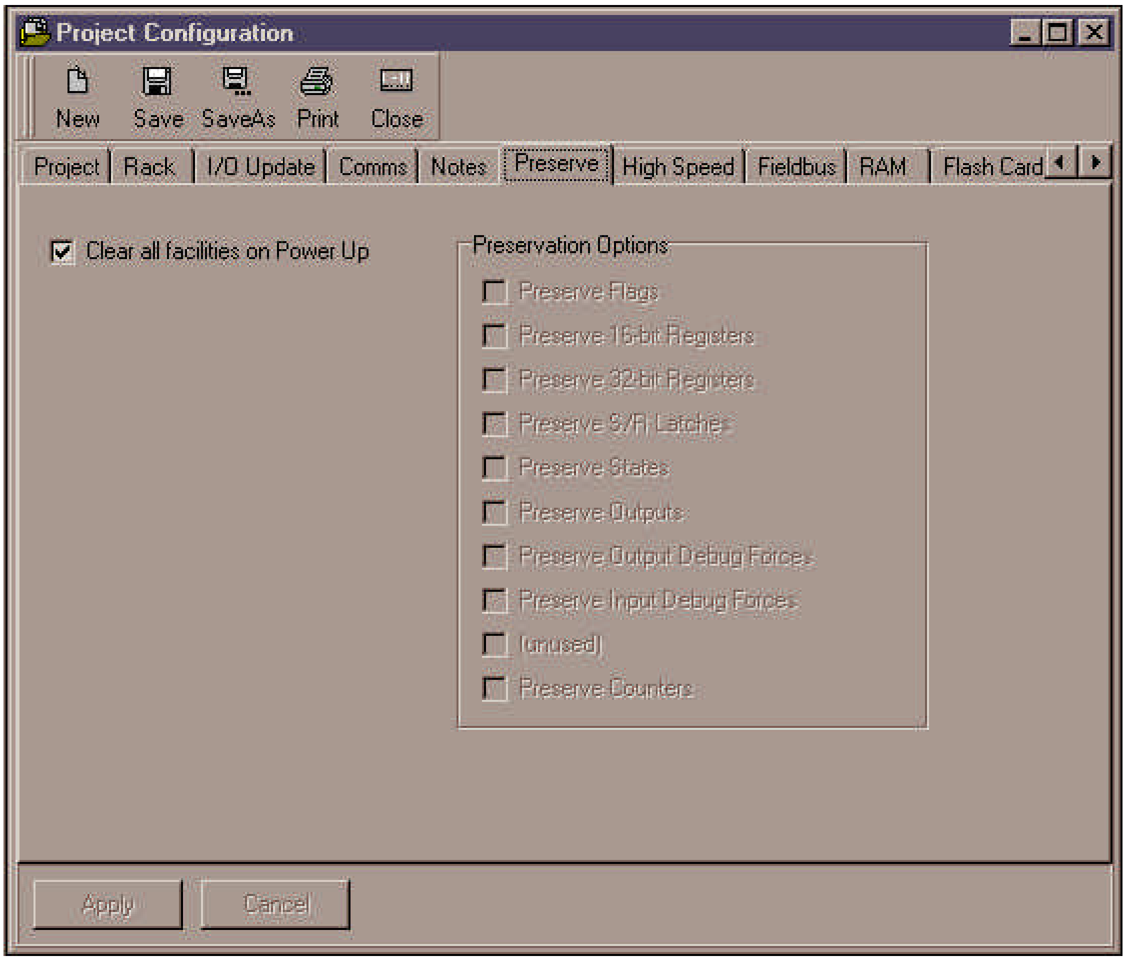
Select the protocol from the right hand radio button box; If you with to drive the communication port using the serial-in and text commands from either the Ladder or Instruction modules then select 'User Code', otherwise select one of the build in protocols from the list.

Notes: Project notes editor.

You can make helpful notes about your project using the Project Notes feature of Flex32.
To make notes about you project:

Click on the 'Notes' page of the Project Configuration Screen. Type your notes in the available space. Your project notes will be saved when you next save the project configuration (click the 'Save' button on the Project Configuration Screen) or save the entire project (click 'Save Project' from the file drop down menu).

Preserve: Select which FMT facilities are preserved and which are cleared on start-up



[✓] Clear all facilities on Power Up

Preservation Options

- [ ] Preserve Flags
- [ ] Preserve 16-bit Registers
- [ ] Preserve 32-bit Registers
- [ ] Preserve S/R Latches
- [ ] Preserve States
- [ ] Preserve Outputs
- [ ] Preserve Output Debug Forces
- [ ] Preserve Input Debug Forces
- [ ] (unused)
- [ ] Preserve Counters

High Speed: Configure the FMT high speed inputs for counters, encoders etc.

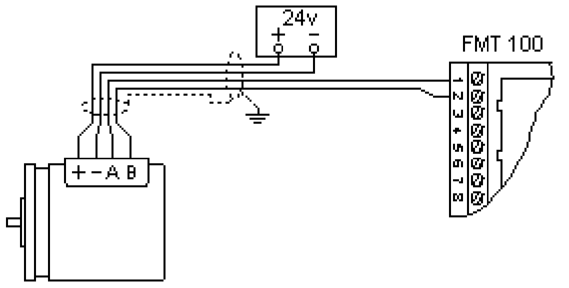
The FMT-100 and FMT-200 hardware include circuitry to process high speed events on selected inputs. These inputs can be configured to implement one of the following high speed facilities.

Incremental Pulse Encoder.
High Speed Counter.
Fast Edge Catching
Event driven user instruction program

The FMT-100's support two high speed inputs (I0 and I1) using W0 as a 32-bit counter to hold the total. The FMT-200's support eight inputs (I0 - I7) using W0 - W3 as counters.
The operation of the high speed features is set-up in the 'High Speed' page of the project configuration screen.

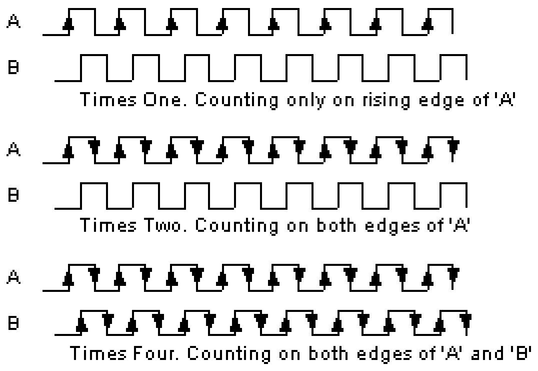
Incremental pulse encoder

The diagram below shows how to connect an incremental pulse encoder (PNP outputs) to an FMT-100. Note the use of screened cable to avoid false pulses from electrical noise.



There are three options for the resolution with which the FMT will count pulses from a particular encoder.

'Times one' will count once per encoder cycle, i.e. 100 counts per revolution for a 100 p.p.r. encoder.
'Times two' will count twice for every cycle, i.e. 200 counts per revolution for a 100 p.p.r. encoder.
'Times four' will count four times for every cycle, i.e. 400 counts per revolution for a 100 p.p.r. encoder.


A
B
Times One. Counting only on rising edge of 'A'
A
B
Times Two. Counting on both edges of 'A'
A
B
Times Four. Counting on both edges of 'A' and 'B'

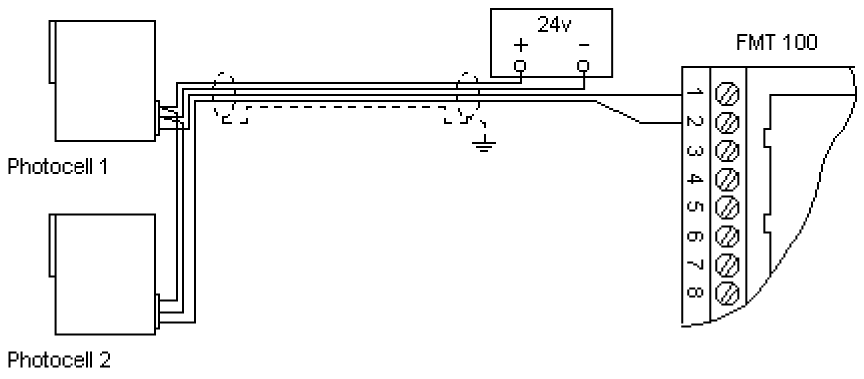
In all cases above, the current total is stored in 32-bit register W0. The value in W0 can be read by the application programme at any time. You can also clear or pre-set the count at any time by moving a number to W0 with a move function.

High speed counter

The high speed counter option is similar to the encoder setting but intended for general purpose counting. This option can be used to count pulses which are too fast to be reliably counted within the normal loop code. The 32-bit registers W0 to W3 are used to hold the counter value.
There are three high speed options for counters. The examples below are for the first channel using I0 - I1.

Count up only. Each rising edge on input I0 will increase the total in W0 by one. Input I1 is unused.
Count down only. Each rising edge on input I0 will decrease the total in W0 by one. Input I1 is unused.
Count up and count down. Each rising edge on input I0 will increase the total in W0 by one. Each rising edge on input I1 will decrease the total in W0 by one

In all cases above the current total is stored in 32-bit register W0. The value in W0 can be read by the application programme at any time. You can also clear or pre-set the count at any time by moving a number to W0 with a move function.



Fast pulse catching
In normal operation the code you write for the FMT can only respond to pulses which are longer than the loop time of the program. For example, if the loop time is 10mS your program will not reliably respond to input signals unless they are at least 10mS long.
To cope with shorter pulses you can use the high speed features to catch pulses and guarantee they are seen by at least one application program loop. You can select from the following options...

Catch high speed positive edges on I0 only, I1 unused
Catch high speed positive edges on I0 and I1

Note: The maximum number of lines of code that can be executed is 20, an error will occur for more than this. Commands that cause execution to stop i.e. wait_for type commands are permitted but not recommended, execution will be attempted 20 times before an error occurs.

Input interrupt instruction code
A new feature with FLEX32 as part of the instruction language is module execution when an input comes on. A module is written using the normal suite of instruction commands but the final statement must be a END_INT command. In addition to selecting the correct setting of the high speed input in the 'High Speed' page of the project configuration screen it is also necessary to select the operating mode of the module for input interrupt from the control menu option, for further details please see the Module Control help.

Note: The maximum number of lines of code that can be executed is 20, an error will occur for more than this. Commands that cause execution to stop i.e. wait_for type commands are permitted but not recommended, execution will be attempted 20 times before an error occurs.

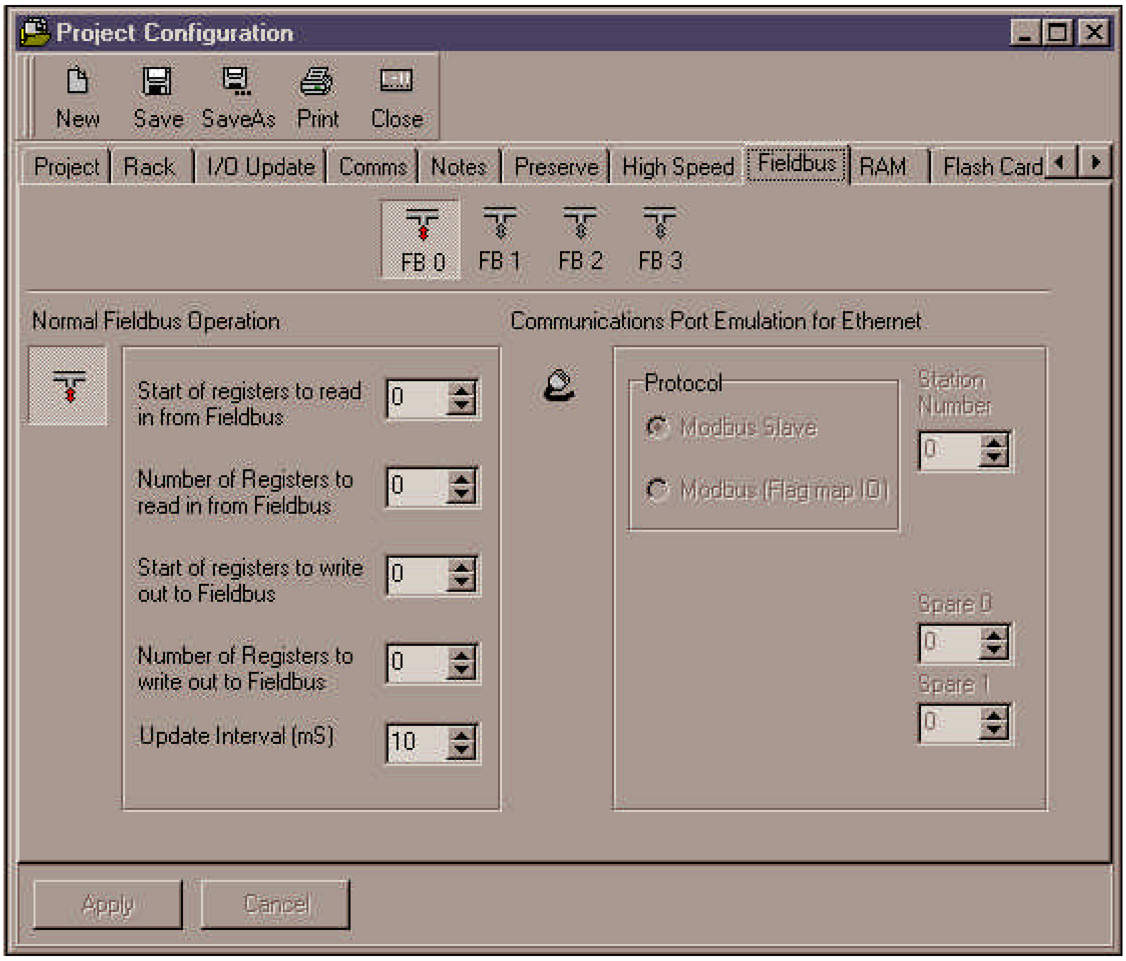
Maximum values
The following maximum values apply to the high speed features on both FMT-100 and FMT-200 products...

Maximum frequency for counting 10kHz total.
Maximum frequency for encoder 2.5kHz total
Minimum pulse time 50µs.

NOTE: Selecting a high speed option does not affect the normal operation of the input.

Fieldbus: Configure the BIS-100/FMT-400 fieldbus modules

BIS-100 and FMT-400 controllers can be fitted with fieldbus modules for communicating with any of the standard fieldbus technologies.



Normal Fieldbus Operation

Start of registers to read in from Fieldbus	0
Number of Registers to read in from Fieldbus	0
Start of registers to write out to Fieldbus	0
Number of Registers to write out to Fieldbus	0
Update Interval (mS)	10

Communications Port Emulation for Ethernet

Protocol

[✓ Modbus Slave]

Modbus (Flag map (0))

Station Number: 0

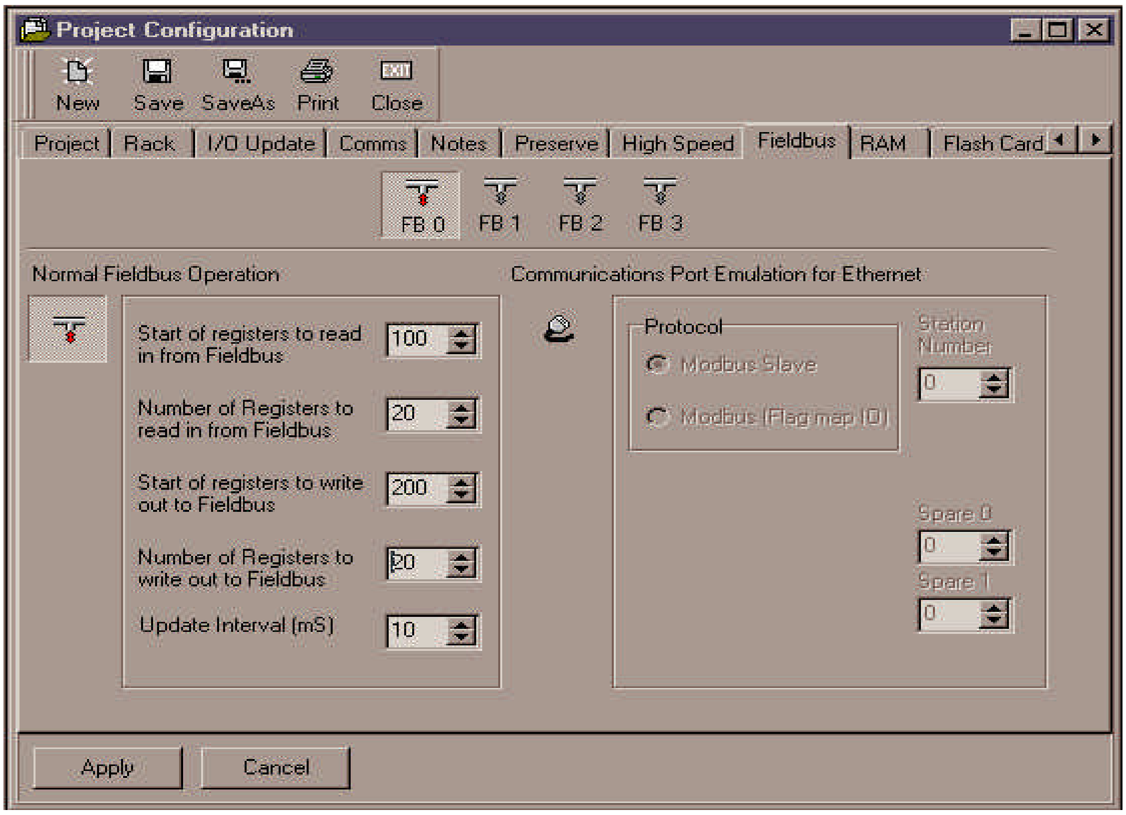
Spare 0: 0

Spare 1: 0

Select the fieldbus port you wish to configure from the buttons labelled FB0 to FB3. Only the buttons for the ports which exist on the controller you are using will be enabled.

For all fieldbus modules except Ethernet you should select 'Normal Fieldbus Operation'. In this mode blocks of data are transferred between the fieldbus module and FMT/BIS 16-bit registers. Select the start number and size of the block of registers to be written to from the fieldbus in the first two edit boxes, and the start number and size of the block of FMT/BIS registers to be written out to the fieldbus. You can also select the interval in milliseconds between transfers to/from the fieldbus unit.

In the example below, registers R100 to R119 are written to with data from the fieldbus every 10ms. Registers R200 to R209 are read and sent out to the fieldbus module every 10ms.


Project Configuration

New Save SaveAs Print Close

Project | Rack | I/O Update | Comms | Notes | Preserve | High Speed | Fieldbus | RAM | Flash Card

FB 0 FB 1 FB 2 FB 3

Normal Fieldbus Operation

Start of registers to read in from Fieldbus	100
Number of Registers to read in from Fieldbus	20
Start of registers to write out to Fieldbus	200
Number of Registers to write out to Fieldbus	20
Update Interval (mS)	10

Communications Port Emulation for Ethernet

Protocol: Modbus Slave / Modbus (Flag map IO)

Station Number: 0

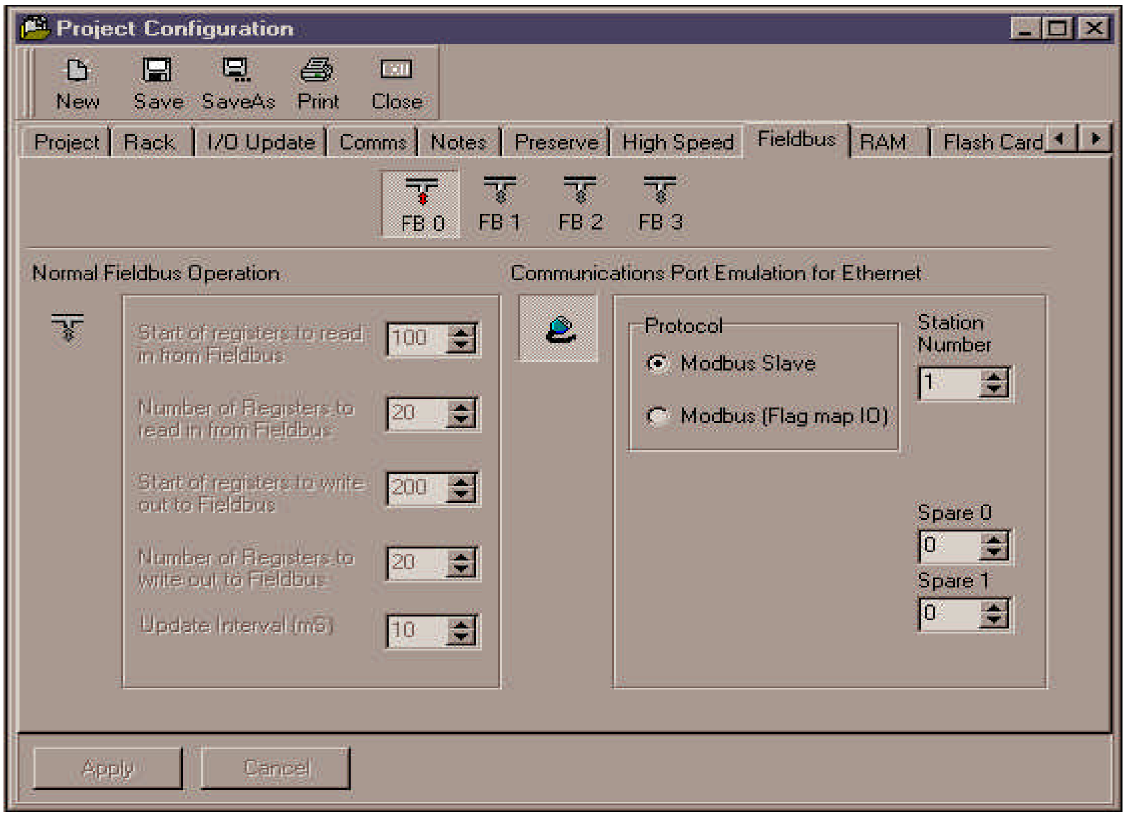
Spare 0: 0

Spare 1: 0

Apply Cancel

For Ethernet modules only you can select 'Comms Port Emulation for Ethernet'. This option uses the FMT firmware to handle Modbus/TCP messages and is the preferred solution for most ethernet applications.

In the example below, the fieldbus module is set up to use the firmware to process modbus/TCP commands as Slave Address 1


Project Configuration

New Save SaveAs Print Close

Project | Rack | I/O Update | Comms | Notes | Preserve | High Speed | Fieldbus | RAM | Flash Card

FB 0 FB 1 FB 2 FB 3

Normal Fieldbus Operation

Start of registers to read in from Fieldbus	100
Number of Registers to read in from Fieldbus	20
Start of registers to write out to Fieldbus	200
Number of Registers to write out to Fieldbus	20
Update Interval (mS)	10

Communications Port Emulation for Ethernet

Protocol: [✓ Modbus Slave] / Modbus (Flag map IO)

Station Number: 1

Spare 0: 0

Spare 1: 0

Apply Cancel

RAM: Define the amount of allocated RAM

The FMT contains battery backed memory which is used to store your program. If you wish you can use some of this RAM to hold data instead of program code, but obviously the maximum size of your program will be smaller. This 'allocated RAM' is accessed by the functions 'ram_erase', '
ram_read', and 'ram_write'.

Sufficient RAM (in the form of 16-bit words) must be allocated for your applications storage using the RAM page on the Project Configuration screen.

Notes:

Allocated RAM is battery backed and will not be cleared unless you call 'ram_erase()'.

For large blocks of allocated RAM the erase time can be significant - approximately 150mS for 100,000 16-bit words.

Flash Card: Define the use for the FMT-200/FMT-400 plug in flash card.

Some members of the FMT family have a socket to accept a plug in Flash Card. This card can be used to store either a program or data.

There are three modes, these are:

Programme storage.
Programme storage with auto-update.
User Data store.

To set the mode select the 'Flash Card' page from the project configuration screen.

Programme storage: In this mode your program that is present in the FMT's internal flash memory is downloaded to the Flash Card. When the Flash Card is inserted into the FMT's Flash Card socket and the FMT is then powered up, then the programme that is present in the Flash Card will automatically run on the FMT but it will not be transferred to the internal flash. If the FMT is powered up next time with the Flash Card no longer present then the program that was in the Flash Card will no longer be present in the FMT.

Programme storage with auto-update: In this mode your program that is present in the FMT's internal flash memory is downloaded to the Flash Card. When the flash card is inserted into the FMT's Flash Card socket and the FMT is then powered up, then the program that is present in the Flash Card will automatically run on the FMT, it will also be copied into the FMT's internal flash so that the program that was present on the Flash Card will now be present in the FMT when it is next powered up even if the Flash Card is no longer present.

User Data store: In this mode the Flash Card is used for data storage (eg. data logging). However the Flash Card can still be used for programme storage in this mode, in order to use it the programme should be downloaded to the Flash Card as normal. In order to use the program when the Flash Card is inserted into the FMT's Flash Card socket, keys F1 and F2 should be held down while the FMT is next powered up. The FMT will then display the message 'Run card program' (press F4 to run the program), the FMT will then display the message 'Update int flash' (press F3 to do this). Only select 'Update int flash' if you want to copy the programme that is in the Flash Card into the internal flash. If however you only want to run the programme then press F1, F2 or F4 when 'Update int flash' is displayed.

Note: To use the flash_erase, flash_read and flash_write functions you must set the card mode to 'User Data store'.

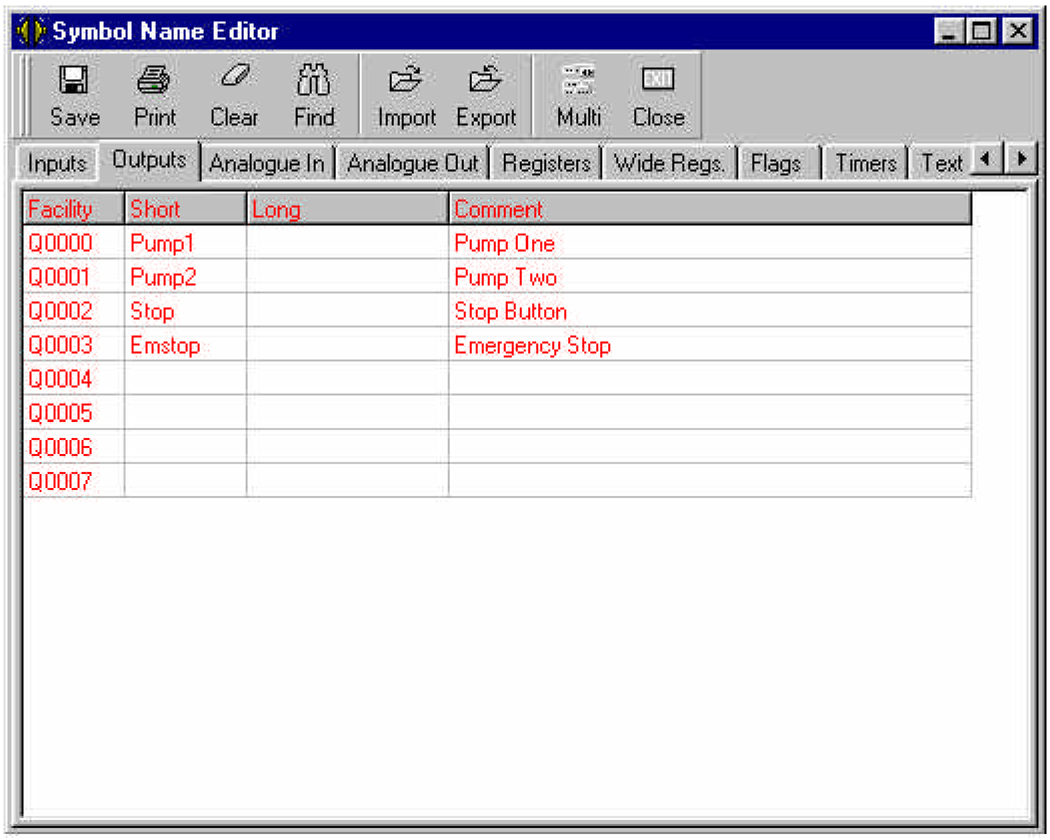
Symbol Name editor

Any facility can be given a symbol name.

Symbol names can be very useful when writing a program. You can assign symbol names to any facility in the FMT. To assign symbol names to facilities you should use the Symbol Name Editor.
Open the editor (if it is not already open), you will see various pages that you can select from each with different facilities on them eg. Inputs, Outputs, Registers. When you are on the page that you require you can assign a symbol name to the facility that you want to name. You should write this name in the short name column of the relevant facility if the symbol name is being used in ladder code (up to a maximum of 6 characters) or if the symbol name is being used in instruction language you should write the name in the long name column of the relevant facility (up to a maximum of 12 characters). Note that short symbol names can be used in the instruction language but long symbol names can not be used in ladder code. Long names and comments are optional.

When you have assigned a symbol name to a facility then type this instead of the facility. For instance if you assign the name 'pump' to output Q0. Then instead of typing TURN_ON (Q0) you would type TURN_ON (pump).

The Symbol name editor:



Facility	Short	Long	Comment
Q0000	Pump1		Pump One
Q0001	Pump2		Pump Two
Q0002	Stop		Stop Button
Q0003	Emstop		Emergency Stop
Q0004			
Q0005			
Q0006			
Q0007			

There are various buttons along the toolbar of the Symbol Name Editor from left to right these are:

'Save'. Saves the symbol names.

'Print'. Prints the symbol names

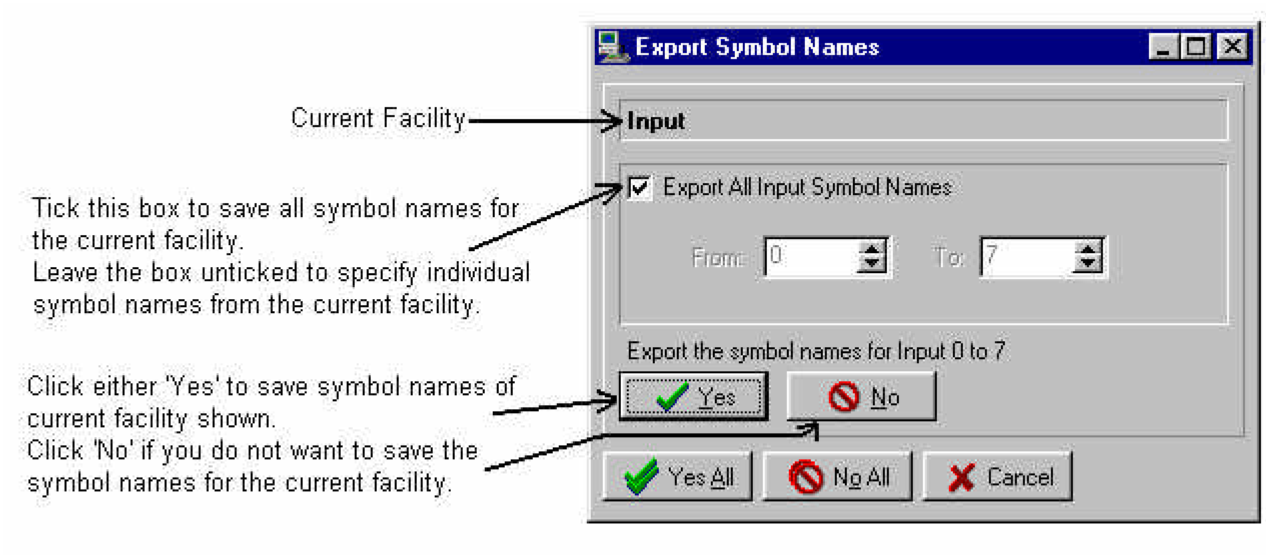
'Clear'. Clears the entry that you are working on.

'Find'. This will prompt you to enter a symbol name to find from all the symbol names that have been entered.

'Import'. This button will enable you to load previously saved symbol name files, the file extensions are *.def, for the default symbol name file which comes with Flex32 and is in the Flex32 root directory and *.txt for a exported symbol name file of your own creation.

'Export'. This button enables you to export you symbol names currently in use to a text file, you can chose which directory to export the files to. This feature is useful if you wish to use your symbol names from your current project in another project.

When you have entered the file name you wish to use then you will be presented with the box below:


Export Symbol Names
Current Facility
Input
Tick this box to save all symbol names for the current facility.
Leave the box unticked to specify individual symbol names from the current facility.
Export All Input Symbol Names
From: 0
To: 7
Export the symbol names for Input 0 to 7
Click either 'Yes' to save symbol names of current facility shown.
Click 'No' if you do not want to save the symbol names for the current facility.
Yes
No
Yes All
No All
Cancel

You will see that there are several choices, you can either chose to save the individual facility symbol names if you click 'Yes' you will then move on to the next facility (Input, Output etc.). If you click 'No' then the symbol names for the current facility shown will not be saved. You can also chose which symbol names that are saved by un ticking the 'Export All xxxx Symbol Names' box (where xxxx represents the current facility). You can then specify which names are saved eg. Inputs from 0 to 7.

Clicking 'Yes All' will save all symbol names of all facilities. Clicking 'No All' will not save any symbol names. Clicking 'Cancel' will cancel the export

'Load'. When clicked will prompt you for a text file containing symbol names.

'Multi'. Toggles between all facility pages and one row of pages which have to be scrolled along in order to access the other pages.

'Close'. Clicking on this button will close the Symbol Name Editor.

Text String Editor

Text Strings are pre-entered in the text string editor and sent out using the Text function.

Text strings are used to store pre-defined messages for sending out the serial ports with the text command. The text strings are entered using the text string editor and downloaded as part of the program.
Text strings can include control characters for including registers, timers, counters and also data and time in your text strings.

To enter a text string you must use the Text String Editor. This is a window in the Flex32 package. You will see that it resembles a table with the text string numbers down the lefthand side. The text strings can be assigned a long and short name in the symbol name editor and these will be shown alongside the text string if it has been assigned with a name. You can enter your text string in the column marked 'Text....'. For correct formatting of the text string please see 'Text String Format'.

There are various buttons along the toolbar of the Text String Editor from left to right these are:

'Save'. Saves the text strings.

'Control'. Click on this button to select from various ASCII control characters to put in your text string. The character you select will be put into you text string immediately after the current cursor position.

'ASCII'. Click on this button to place an ASCII character in your text string. The two question marks that appear after the '#' should be replaced with the hex value of the ASCII character that you require. Please see the ASCII Table if you need to know the value for you chosen character.

'Value'. Click on this button to place the value of a 16-bit register, or the value of a 32-bit register, or the value of a timer, or the value of a counter. You will be prompted to select the formatting for the value that is to be printed. When you have selected what value and formatting you want then you will see four question marks appear after the text string code. You need to replace these question marks with the register, timer or counter that you want.

'Buffer'. Click on this button to print a string of characters using the data in the registers starting at R????. If using fixed length buffer denoted by '%SR????:??' the length of the string will be :?? characters. If using terminated buffer, denoted by '%SR????#??', the string will continue until a terminating character is met or until 250 characters have been sent. The terminating character is the character whose ASCII value is #?? hex.

'Insert'. Click on this button to link another text string into the present one. This will allow you to cascade several text strings together. Type the number of the text string that you wish to cascade in place of the question marks which appear.

'DateTime'. Clicking on this button will allow you to insert the command for date/time into your text string. You will need to chose from the various formatting options that you have the choice of.

'Check'. Clicking on this button enables you to place a checksum calculation on you text string. Click on 'Start Calculation' to start the checksum calculation from the current cursor position. Click on end checksum to end the checksum calculation on the data. The calculation will be performed on the data between the start and end checksum calculation commands. You should then place a print checksum command which will print the checksum in the format that you chose from the various formats that you are presented with.

'Display'. When clicking on this button you will be presented with formatting options for your text string which will be useful if you are printing you text string to the FMT-200's or FMT-400's built in display.

'Close'. Clicking on this button will enable you to close the Text String Editor.

Ladder module editor

The Ladder Editor is used to write project instruction modules in FLEX 32 utilizing the Ladder logic language.

Using the Ladder Module editor:

Starting a new Project and Module:

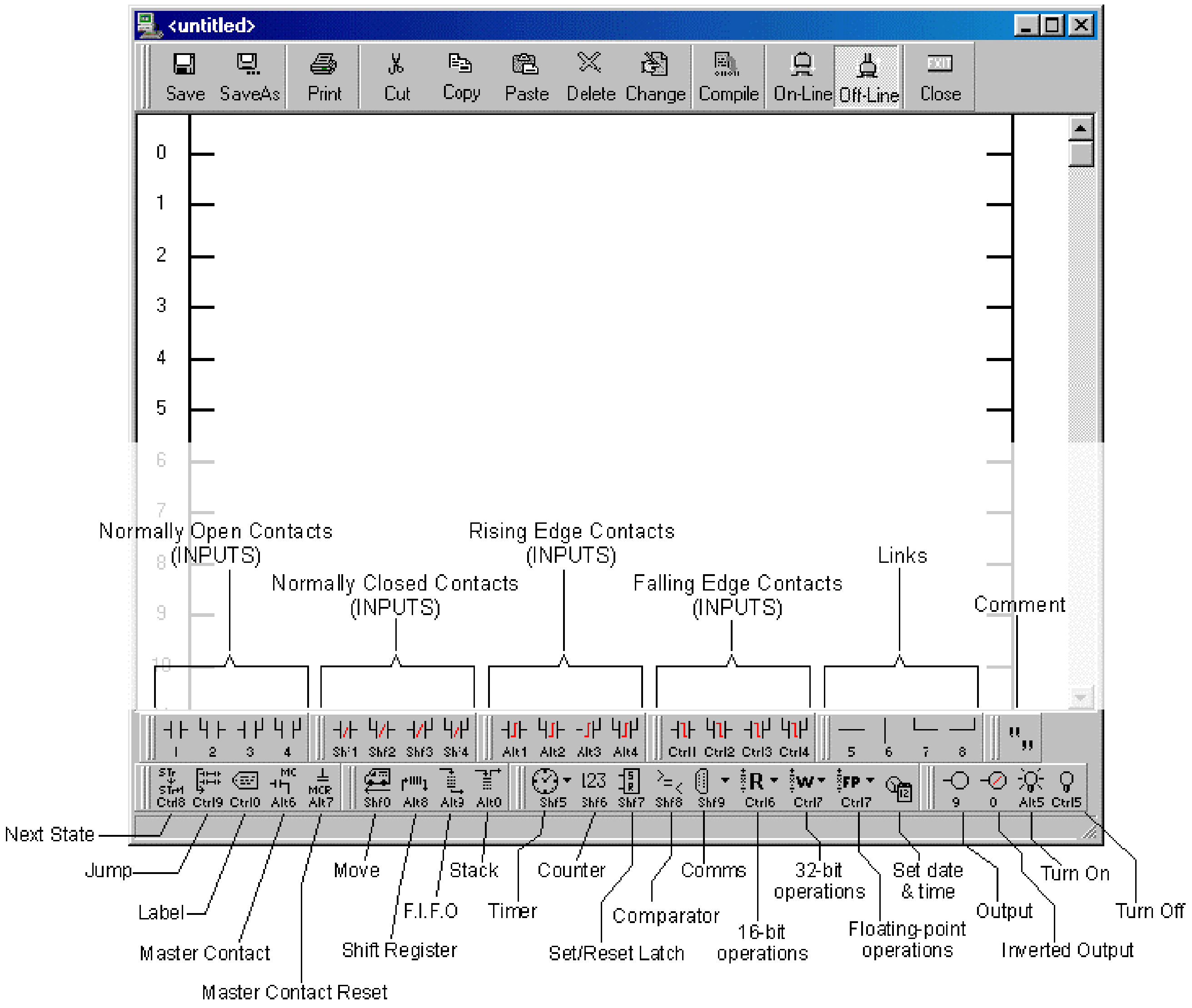
1. If you need to start a new project then do so by selecting 'New Project' from the File menu. Follow the on screen instructions that you are presented with.

2. From the project configuration screen left click on the 'New' button which is situated between 'Available Ladder Modules' and 'Project Ladder Modules'.

3. After entering a suitable name for your module you will enter the module editor:

Entering Ladder code into your Ladder module:

One you are in the Ladder module editor you can enter facilities in your Ladder logic program. To enter ladder code you should position your cursor on the ladder rung where you wish to enter you code. Now click on the facility that you wish to add, facilities are all presented as buttons at the bottom of the Ladder editor. When you click on a facility, you will be required to complete some details about the facility that you are placing in the ladder code for example if the facility was an output you will be required to complete some detail about which output you are going to use. After you have filled in the required details then the facility will be placed where your cursor is positioned.



The Tool Bar:

Starting with the lefthand side of tool bar in the instruction editor the buttons will be described in some more detail:

'Save'. When clicked on with the lefthand button of your mouse this button will save you module at the stage that it is at when you click the button. You must first give you module a name before you use the save command. This is done using the...

...'Save As' button. When this is clicked on you will be prompted to enter a name for you module.

'Print'. This button will be enabled when you have just edited something in your module for example if you have just deleted a part of your module but now realize you shouldn't have then click on this and what you have just deleted will be reinstated.

'Cut'. To use this button you must first highlight some Ladder code to cut, do this as you would in a word processor, now click on the 'Cut' button. The highlighted Ladder code will be cut to the clipboard.

'Copy'. To use this button you must first highlight some Ladder code to copy, do this as you would in a word processor, now click on the 'Copy' button. The highlighted Ladder code will be copied to the clipboard.

'Paste'. To use this button there must be some Lader code in the clipboard. Position the cursor where you want the pasted Ladder code to start and click on the 'Paste' button. The Ladder code in the clipboard will be pasted into the current cursor position.

'Change'. To use this button you must first have a facility currently highlighted in your ladder code. When this button is clicked you will be able to change the parameters of the facility currently highlighted.

'Compile'. When this button is clicked on your Ladder code that you have entered into the Ladder editor will be compiled into a form of code that is common to both ladder and instruction modules (this code is unseen by the user but is used in the download process to the FMT). The Instruction module is also checked to make sure that it is acceptable to the FMT and that the source code does not have any errors present.

'On-Line'. When this button is clicked on and you are connected to the FMT which has your project downloaded to it then the current state of the ladder elements will be shown with elements that are 'on' being highlighted in yellow.

'Off-Line'. If you are 'On-Line' then you will be taken 'Off-Line' when this button is clicked on. You must be **'Off-Line'** in order to make changes to your instruction module.

The Main Menu:

'Edit' drop-down menu. From this menu you can select from 'Undo', 'Redo' (this has the opposite effect of undo, if you have just undone something that you wish you had not, then click on Redo), 'Cut', 'Copy', 'Paste' and 'Delete' these work in the same way as described in the Tool Bar of the Ladder Editor. You can also select whether 'Show Symbol Names' is selected. If it is then names assigned to facilities will be shown. If is is not then the actual facility will be displayed.

'Search' drop-down menu. This does not function in the Ladder editor.

'Bookmark' drop-down menu. This does not function in the Ladder editor.

More Useful Features....

When editing your module if you click the right-hand mouse button while you have a facility highlighted you will be presented with a list of five options:

'Change' To use this button you must first have a facility currently highlighted in your ladder code. When this button is clicked you will be able to change the parameters of the facility currently highlighted.

'Normally Open' This only works when an input (contact) is highlighted if you click 'normally open' then the input currently highlighted will be changed to a normally open contact.

'Normally Closed' This only works when an input (contact) is highlighted if you click 'normally closed' then the input currently highlighted will be changed to a normally closed contact.

'Edge Up' This only works when an input (contact) is highlighted if you click 'Edge Up' then the input currently highlighted will be changed to a rising edge triggered contact.

'Edge Down' This only works when an input (contact) is highlighted if you click 'Edge Down' then the input currently highlighted will be changed to a falling edge triggered contact.

Instruction module editor

The Instruction Editor is used to write project instruction modules in FLEX 32 utilizing the text based instruction language.
The editor works in much the same way as a basic word processor terms of entering text, functions such as Cut & Paste and Find & Replace etc. are available.

Using the Instruction Module editor:

Starting a new Project and Module:

1. If you need to start a new project then do so by selecting 'New Project' from the File menu. Follow the on screen instructions that you are presented with.

2. From the project configuration screen left click on the 'New' button which is situated between 'Available Instruction Modules' and 'Project Instruction Modules'.

3. After entering a suitable name for your module you will enter the module editor

Entering code into your instruction module:

One you are in the Instruction Module editor you can enter in your program text in much the same way as you would on a word processor. When you are entering functions and facilities etc into the editor you will see that they are highlighted in different colours for example outputs e.g. Q1 are displayed in red and text string numbers e.g. TX12 are highlighted in grey this will enable you to follow your code more easily.

If you type in something incorrectly the word you have just typed in will appear underlined. This enables mistakes to be easily seen.

There are various useful editing functions available in the instruction editor, these are available as buttons that can be clicked on the tool bar of the instruction module editor screen. Some useful editing functions are also available from the main menu these being under the menus 'Edit', 'Search' and 'Bookmark'.

The Tool Bar:

Starting with the lefthand side of tool bar in the instruction editor the buttons will be described in some more detail:

'Save'. When clicked on with the lefthand button of your mouse this button will save you module at the stage that it is at when you click the button. You must first give you module a name before you use the save command. This is done using the...

...'Save As' button. When this is clicked on you will be prompted to enter a name for you module.

'Undo'. This button will be enabled when you have just edited something in your module for example if you have just deleted a part of your module but now realize you shouldn't have then click on this and what you have just deleted will be reinstated.

'Redo'. This has the opposite effect of undo. If you have just undone something that you wish you had not, then click on Redo.

'Cut'. To use this button you must first highlight some text to cut, do this as you would in a normal word processor, now click on the 'Cut' button. The highlighted text will be cut to the clipboard.

'Copy'. To use this button you must first highlight some text to copy, do this as you would in a normal word processor, now click on the 'Copy' button. The highlighted text will be copied to the clipboard.

'Paste'. To use this button there must be some text in the clipboard. Position the cursor where you want the pasted text to start and click on the 'Paste' button. The text in the clipboard will be pasted into the current cursor position.

'Delete'. Clicking on this button will delete any text that is currently highlighted.

'Indent'. To use this button first highlight a line of text to indent then click on the 'Indent' button.

'Unind.'. This button performs the opposite of the 'Indent' button ie. it remove the indents from any text that you highlight.

'Font'. This when clicked on will enable you to change select different fonts and to enable you to change the properties of the font that you are using.

'Compile'. When this button is clicked on your source code that you have entered into the Instruction Editor will be compiled into a form of code that is common to both ladder and instruction modules (this code is unseen by the user but is used in the download process to the FMT). The Instruction module is also checked to make sure that it is acceptable to the FMT and that the source code does not have any errors present.

'Control'. When this button is clicked on you will be presented with four ways in which your instruction module code can be executed, for more information please see Module Control.

'On-Line'. When this button is clicked on and you are connected to the FMT which has your project downloaded to it then you will be able to see the FMT stepping through your code, monitor registers etc.

'Off-Line'. If you are 'On-Line' then you will be taken 'Off-Line' when this button is clicked on. You must be 'Off-Line' in order to make changes to your instruction module.

The Main Menu:

'Edit' drop-down menu. From this menu you can select from 'Undo', 'Redo', 'Cut', 'Copy', 'Paste' and 'Delete' these work in the same way as described in the Tool Bar of the Instruction Editor.

'Search' drop-down menu. From this menu you can select from 'Find', 'Replace' and 'Search Again'.

'Find'. Prompts you to enter a string of characters to be searched for, then searches the current module for matching text.

'Replace'. Prompts you to enter a string of characters to be searched for, then a second set with which to replace the original.

'Search Again'. Repeats the last search operation with the same search data,case sensitivity and direction. The search starts from the current cursor position.

You can set the following options to optimize the search and replace operation for your exact needs.

Direction:
The direction can be set to search either up (towards the beginning of the module) or down (towards the end of the module). You should select which direction you want.

Match whole word only:
If this option is selected then only the whole word will be matched. If this is not selected then the search string could be entered as just part of a word eg. par would match with part and particular if match whole word is not selected.

Match case:
The search can be set up to be case sensitive or not. If you select case sensitive, 'Colter' will not match with 'colter' or 'COLTER'. If you do not select this option then 'Colter' will match 'colter' and 'COLTER'.

Replace All:

You can have the search and replace operation find and replace all matching texts through the module if you select this.

Find Next:
Clicking on this will search for the next matching text after the one that has just been found.

'Bookmark' drop-down menu. From this menu you can set bookmarks in your instruction module so that you can jump to and from them.

You can set a bookmark in you module by clicking on the line of text that you wish to bookmark, then select 'Set' from the Bookmark drop-down menu, select the bookmark number that you want and the bookmark will be placed for you next to your selected line of text.

To jump to a bookmark select 'Goto' from the Bookmark drop-down menu then select the bookmark number that you wish to go to. The cursor will be placed on the book marked line.

More Useful Features:

When editing you module if you click the right-hand mouse button while pointing over you code you will be presented with a list of three options:

'Symbol Names...' Clicking on this will present you with a list of symbol names used in you project and what facility the symbol name has been assigned to. Clicking in the symbol names column and typing a name will show you which facility is associated with the name you have just entered (or nearest name if the one that you typed does not exist). Clicking in the functions column and typing a function will show you what name is associated with the function that you have just typed (or nearest function if the one that you just typed does not exist).

Double clicking on a symbol name in the list will place this name in your module after the current cursor position.

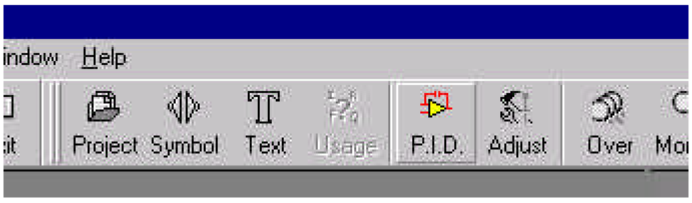
'Functions...' Clicking on this will present you with a list of function that can be used in you instruction module. You can scroll up and down the list and double clicking on a function will place it in your module after the current cursor position, leaving you to fill in the appropriate detail eg. registers, constants.

'Keywords' Clicking on this will present you with a list of keywords, for example, else, if, until. Double clicking on a keyword will place it in your instruction module after the current cursor position.

If the **'Usage'** button on the main toolbar is clicked then all the facilities that are used in your project will be shown in a list. The facilities are listed along with their short and long symbol names. The multiple number of the facility being used is also shown along with the module name that the facility is used in, the function that controls the facility is shown and also whether the facility is being read or written to by the function.

PID configuration

Click the **'P.I.D.'** button on the main toolbar to be shown all the available facilities for configuring PID.

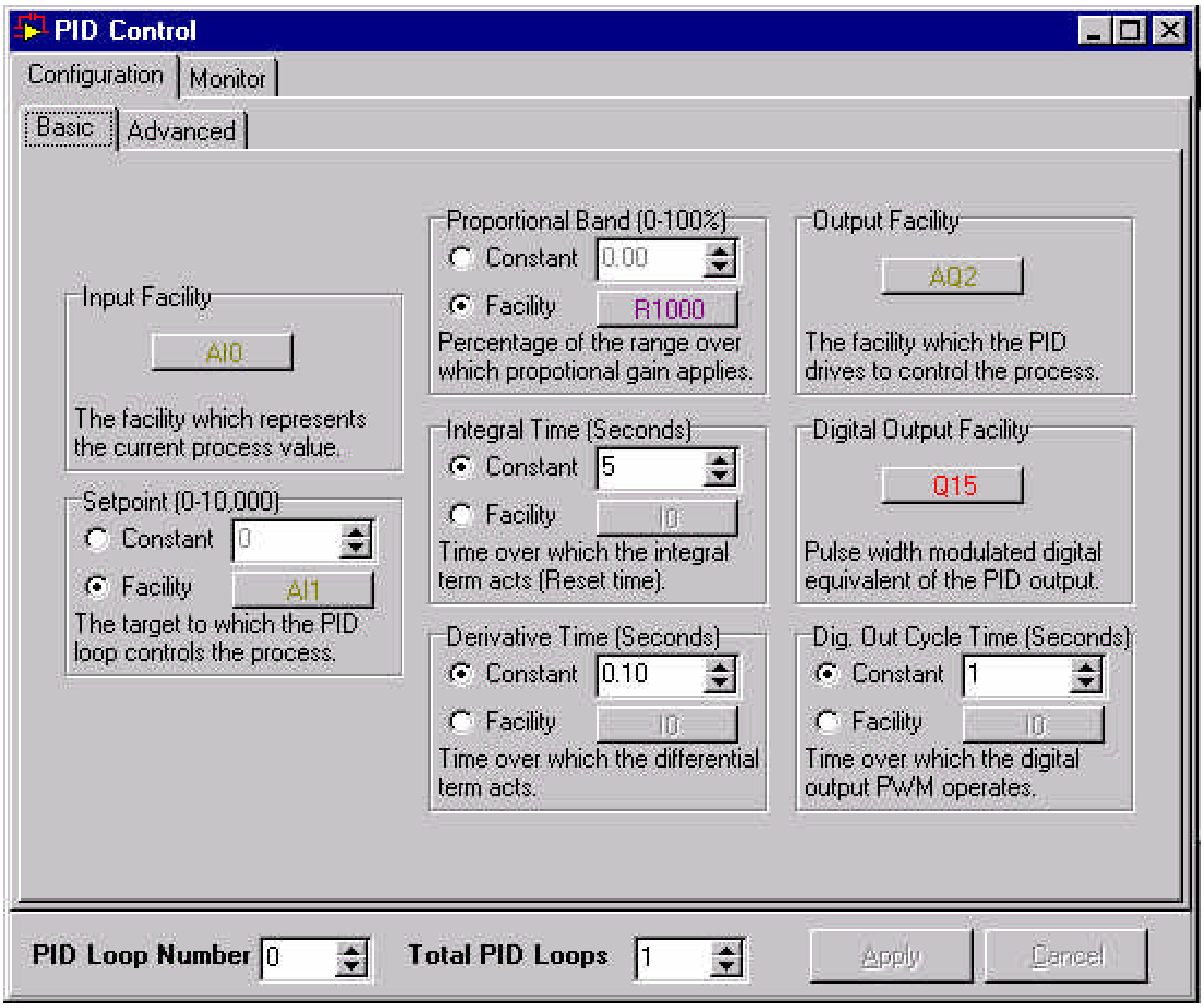


The maximum number of PID loops that can be supported by FMT/BIS hardware is as follows:

FMT-100………………………..1
FMT-200………………………..16

BIS-100……………………………16
FMT-400 CPU-A………………..8
FMT-400 CPU-B………………..16
FMT-400 CPU-C………………..64

Basic:


PID Control

Configuration | Monitor

Basic | Advanced

Input Facility: AI0 — The facility which represents the current process value.

Setpoint (0-10,000): ( ) Constant 0 / (●) Facility AI1 — The target to which the PID loop controls the process.

Proportional Band (0-100%): ( ) Constant 0.00 / (●) Facility R1000 — Percentage of the range over which propotional gain applies.

Integral Time (Seconds): (●) Constant 5 / ( ) Facility I0 — Time over which the integral term acts (Reset time).

Derivative Time (Seconds): (●) Constant 0.10 / ( ) Facility I0 — Time over which the differential term acts.

Output Facility: AQ2 — The facility which the PID drives to control the process.

Digital Output Facility: Q15 — Pulse width modulated digital equivalent of the PID output.

Dig. Out Cycle Time (Seconds): (●) Constant 1 / ( ) Facility I0 — Time over which the digital output PWM operates.

PID Loop Number 0 — Total PID Loops 1

First select the total PID loops to be configured, and also the loop number. Each loop will require to be configured separately. When all of the parameters have been completed then click the ‘Apply’ button.

'Input Facility': Click on this button to input the current process value, which can be an analogue input (AI), analogue output (AQ) or register (R). The process value has a range of 0-10000, is volatile and user program adjustable.

'Setpoint': Selecting ‘Constant’ allows a fixed value to be entered via the up/down arrows within the range 0-10000. If ‘Facility’ is used then click on the button to input either an analogue input (AI), analogue output (AQ) or a register (R), again choosing within the range 0-10000. The process setpoint is non-volatile, preserved through power down, and user program and online adjustable.

'Proportional Band': This can be either a constant or facility in the same manner as ‘Setpoint’. The range is 0.00-100.00%, non-volatile, preserved through power down, and user program and online adjustable. **A value of zero will disable the PID.**

'Integral Time': This can be either a constant or facility in the same manner as ‘Setpoint’. The range is 0-3000 seconds, non-volatile, preserved through power down, and user program and online adjustable. **A value of zero will disable this function.**

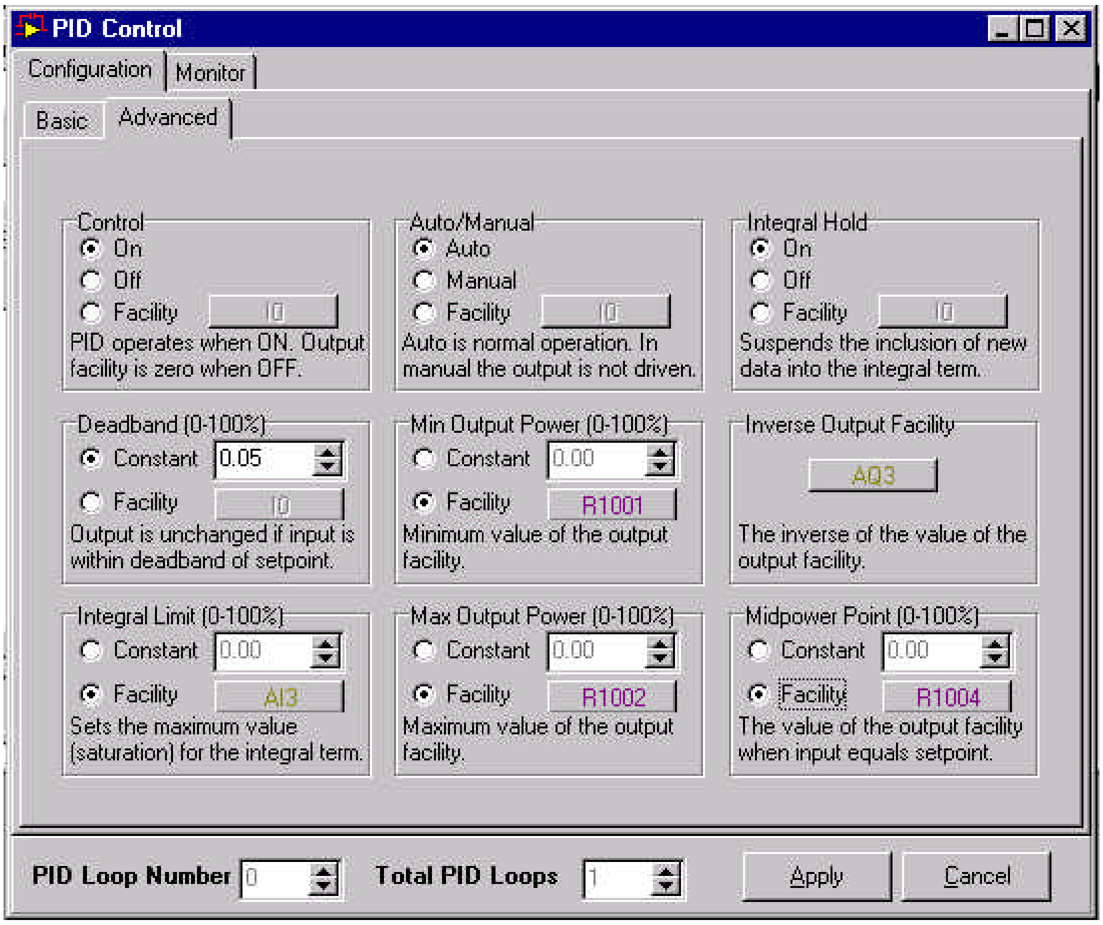
'Derivative Time': This can be either a constant or facility in the same manner as 'Setpoint'. The range is 0.00-65.00 seconds, non-volatile, preserved through power down, and user program and online adjustable. **A value of zero will disable this function.**

'Output Facility': Click on this button to select the output which the PID drives to control the process. This can be either an analogue output (AQ) or register (R), and has a range of 0-10000, is volatile, and user program and on-line adjustable.

'Digital Output Facility': Click on this button to select the digital output for PMW output. This can be a digital output (Q) or a flag (F) and is volatile and has no other facilities.

'Digital Output Cycle Time': This can be either a constant or facility in the same manner as 'Setpoint'. The range is 1-600 seconds, non-volatile, preserved through power down, and user program and online adjustable. **A value of zero will disable output operation.**

Advanced:



Control: [✓ On] Off Facility I0
PID operates when ON. Output facility is zero when OFF.

Auto/Manual: [✓ Auto] Manual Facility I0
Auto is normal operation. In manual the output is not driven.

Integral Hold: [✓ On] Off Facility I0
Suspends the inclusion of new data into the integral term.

Deadband (0-100%): [✓ Constant] 0.05 Facility I0
Output is unchanged if input is within deadband of setpoint.

Min Output Power (0-100%): Constant 0.00 [✓ Facility] R1001
Minimum value of the output facility.

Inverse Output Facility: AQ3
The inverse of the value of the output facility.

Integral Limit (0-100%): Constant 0.00 [✓ Facility] AI3
Sets the maximum value (saturation) for the integral term.

Max Output Power (0-100%): Constant 0.00 [✓ Facility] R1002
Maximum value of the output facility.

Midpower Point (0-100%): Constant 0.00 [✓ Facility] R1004
The value of the output facility when input equals setpoint.

PID Loop Number 0 Total PID Loops 1

'Control': This can either be fixed or a facility. Click on the button to choose the facility, which can either be a digital input (I), digital output (Q) or a flag (F). PID process is enabled when this selection is 'ON', and disabled when 'OFF'.

'Auto/Manual': This can either be fixed or a facility. Click on the button to choose the facility, which can either be a digital input (I), digital output (Q) or a flag (F). PID process is enabled when this selection is 'AUTO', and when 'Manual' disabled, but will keep tracking.

'Integral Hold': This can either be fixed or a facility. Click on the button to choose the facility, which can either be a digital input (I), digital output (Q) or a flag (F). When this selection is 'OFF' it suspends the inclusion of new data into the integral term.

'Deadband': Selecting 'Constant' allows a fixed value to be entered via the up/down arrows within the range 0.00-100.00%. If 'Facility' is used then click on the button to input either an analogue input (AI), analogue output (AQ) or a register (R), again choosing within the range 0.00-100.00%. The process setpoint is non-volatile, preserved through power down, and user program and online adjustable.

'Integral (Saturation) Limit': This can be either a constant or facility in the same manner as 'Deadband'. The range is 0.00-100.00%, non-volatile, preserved through power down, and user program and online adjustable. **A value of 100% has no effect.**

'Min Output Power': This can be either a constant or facility in the same manner as 'Deadband'. The range is 0-10000, non-volatile and preserved through power down. **A value of zero has no effect.**

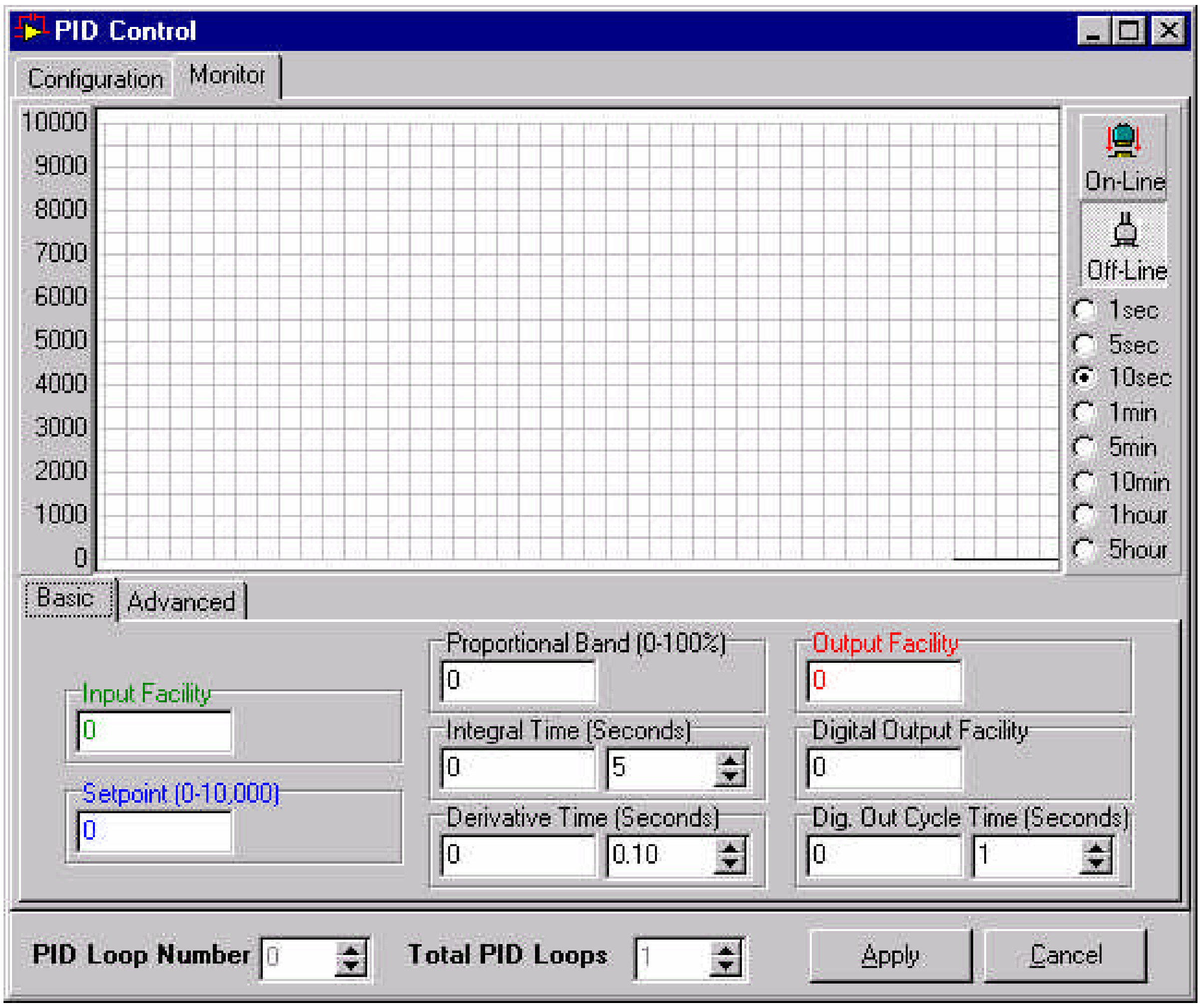
'Min Output Power': This can be either a constant or facility in the same manner as 'Deadband'. The range is 0-10000, non-volatile and preserved through power down. **A value of 10000 has no effect.**

'Inverse Output Facility': Click on this button to select the inverse output to which the PID drives to control the process. This can be either an analogue output (AQ) or register (R), and has a range of 0-10000, is volatile, and user program and on-line adjustable.

'Midpower Point': This can be either a constant or facility in the same manner as 'Deadband'. The range is 0.00-100.00%.

Monitor/Basic:

Click 'On-Line' to monitor PID control using the time scale selected from the right hand column.



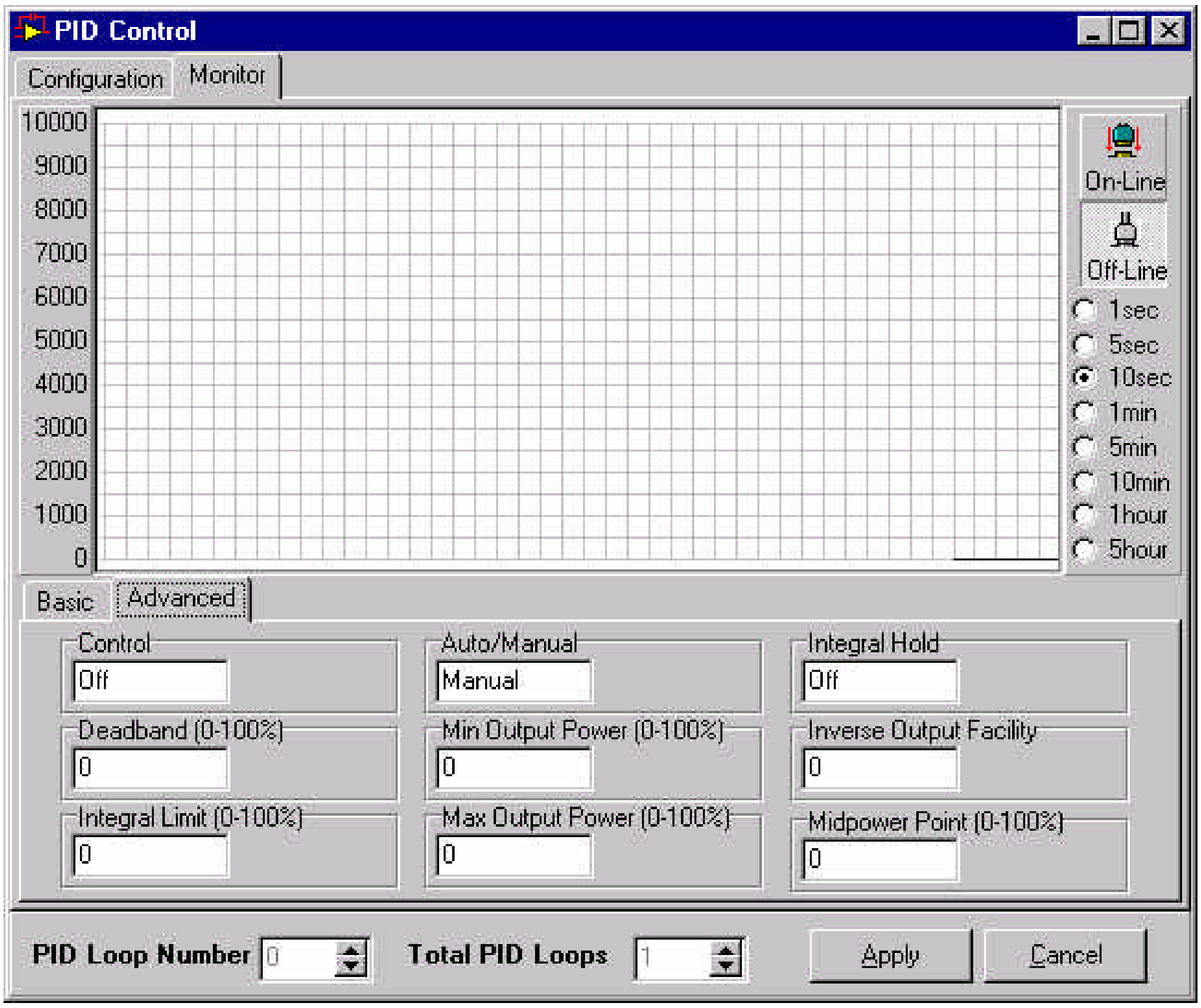
Time scale: 1sec, 5sec, [✓ 10sec], 1min, 5min, 10min, 1hour, 5hour

Input Facility: 0
Setpoint (0-10,000): 0
Proportional Band (0-100%): 0
Integral Time (Seconds): 0 5
Derivative Time (Seconds): 0 0.10
Output Facility: 0
Digital Output Facility: 0
Dig. Out Cycle Time (Seconds): 0 1
PID Loop Number: 0
Total PID Loops: 1

Selecting ‘Basic’ shows the values of the parameters setup in the basic configuration page. If constants have been selected, then their values can be altered by means of the up/down arrows. The resultant change can then be observed on the monitor screen.
Facilities can be altered by using the ‘Monitor’ box from the icon on the main tool bar.

Monitor/Advanced:

Click ‘On-Line’ to monitor PID control using the time scale selected from the right hand column.

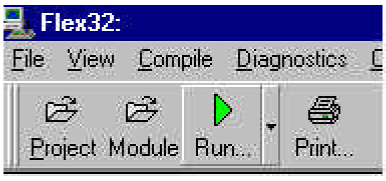


Control: Off
Auto/Manual: Manual
Integral Hold: Off
Deadband (0-100%): 0
Min Output Power (0-100%): 0
Inverse Output Facility: 0
Integral Limit (0-100%): 0
Max Output Power (0-100%): 0
Midpower Point (0-100%): 0
PID Loop Number: 0
Total PID Loops: 1
Time scale: [✓ 10sec]

Selecting ‘Advanced’ shows the values of the parameters setup in the advanced configuration page. If constants have been selected, then their values can be altered by means of the up/down arrows. The resultant change can then be observed on the monitor screen.
Facilities can be altered by using the ‘Monitor’ box from the icon on the main tool bar.

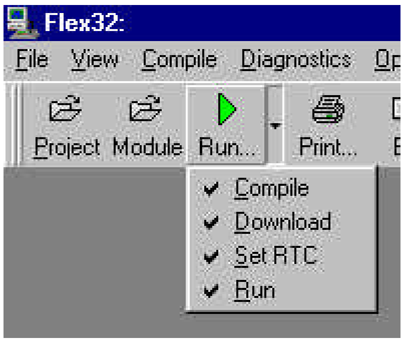
Project compile and download; the 'Run' button.

To download your project to your FMT / BIS click on the Run button on the Flex32 toolbar:



When the Run button is clicked all the modules currently selected in your project will be compiled and downloaded to you FMT / BIS.

If you click on the small down arrow by the side of the Run button you will be presented with a list of options:



The options presented are: Compile, Download, Set RTC and Run:

'Compile': If this option is unticked then the previously compiled *.dld file will be downloaded to the FMT / BIS. This is useful if for example you wish to provide a customer with a set of down loadable, compiled files (see note below) but do not wish to provide them with the source code (instruction language and ladder modules).

'Download': If this option is un-ticked then the project will not be downloaded to the FMT / BIS when the Run button is clicked.

'Set RTC': If this option is un-ticked then the FMT / BIS's clock will not be set to your PC's clock.

'Run' If this option is un-ticked then the project will not run when it is downloaded to the FMT / BIS. It can be made to Run later however, by clicking on the Run button in the monitor window or by downloading your project again with the 'Run' option ticked.

Note:
The files needed for a compiled project download without the source code are the files found in your project directory with the following file extensions:

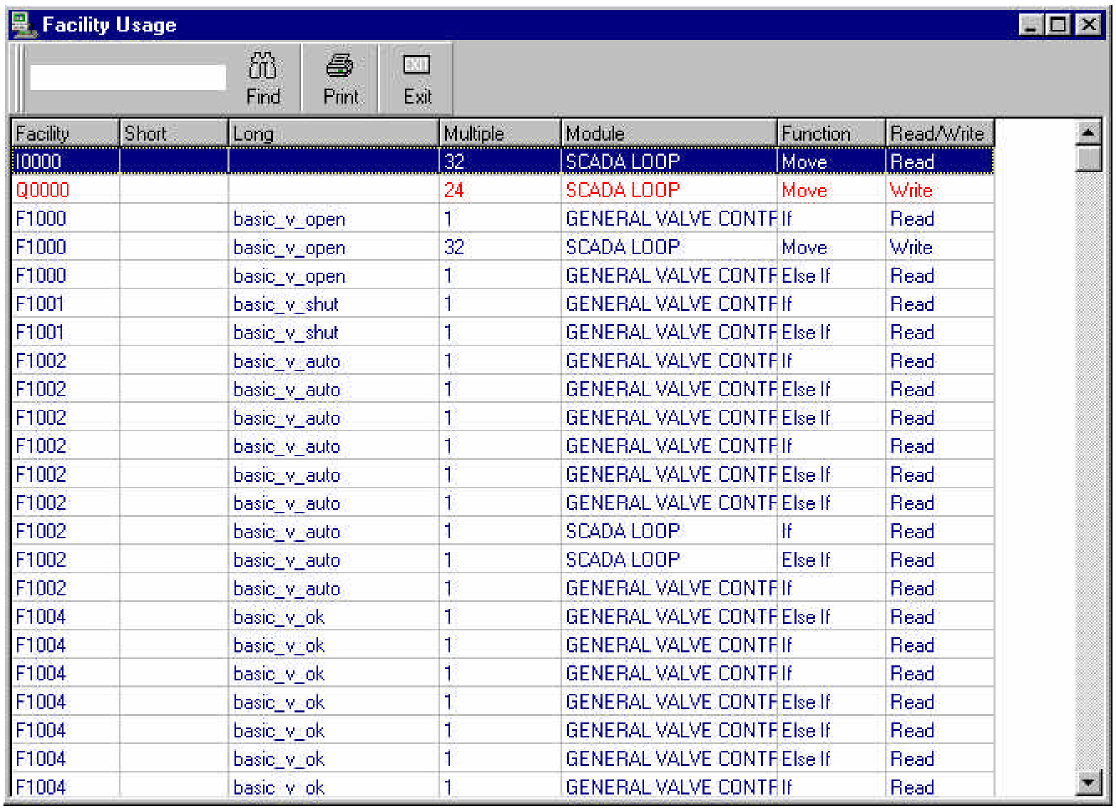
xxxxx.prj	- the main project file
xxxxx.sym	- the symbol name file
xxxxx.idx	- the symbol name index file
xxxxx.tsf	- the text string file
xxxxx.alt	- the 'Alert' code file
xxxxx.dld	- the main download file
*.mcb	- the module control block for each instruction module

(where xxxxx is your project name).

Facility Usage

If the 'Usage' button on the main toolbar is clicked then all the facilities that are used in your project will be shown in a list. The facilities are listed along with their short and long symbol names. The multiple number of the facility being used is also shown along with the module name that the facility is used in, the function that controls the facility is shown and also whether the facility is being read or written to by the function.

Facility Usage is only available once a **'Compile'** or **'Test Compile'** has been performed.



Facility	Short	Long	Multiple	Module	Function	Read/Write
I0000			32	SCADA LOOP	Move	Read
Q0000			24	SCADA LOOP	Move	Write
F1000		basic_v_open	1	GENERAL VALVE CONTF	If	Read
F1000		basic_v_open	32	SCADA LOOP	Move	Write
F1000		basic_v_open	1	GENERAL VALVE CONTF	Else If	Read
F1001		basic_v_shut	1	GENERAL VALVE CONTF	If	Read
F1001		basic_v_shut	1	GENERAL VALVE CONTF	Else If	Read
F1002		basic_v_auto	1	GENERAL VALVE CONTF	If	Read
F1002		basic_v_auto	1	GENERAL VALVE CONTF	Else If	Read
F1002		basic_v_auto	1	GENERAL VALVE CONTF	Else If	Read
F1002		basic_v_auto	1	GENERAL VALVE CONTF	If	Read
F1002		basic_v_auto	1	GENERAL VALVE CONTF	Else If	Read
F1002		basic_v_auto	1	GENERAL VALVE CONTF	Else If	Read
F1002		basic_v_auto	1	SCADA LOOP	If	Read
F1002		basic_v_auto	1	SCADA LOOP	Else If	Read
F1002		basic_v_auto	1	GENERAL VALVE CONTF	If	Read
F1004		basic_v_ok	1	GENERAL VALVE CONTF	Else If	Read
F1004		basic_v_ok	1	GENERAL VALVE CONTF	If	Read
F1004		basic_v_ok	1	GENERAL VALVE CONTF	If	Read
F1004		basic_v_ok	1	GENERAL VALVE CONTF	Else If	Read
F1004		basic_v_ok	1	GENERAL VALVE CONTF	Else If	Read
F1004		basic_v_ok	1	GENERAL VALVE CONTF	Else If	Read
F1004		basic_v_ok	1	GENERAL VALVE CONTF	If	Read

Debugging screens

Information
Alarms

The **FMT** CPU's generate alarms when they detect that something is wrong. These can be viewed from the Alarm screen .

Alarms are categorized as Errors, Warnings or Information.

· Errors are serious and cause the program to stop; all outputs are turned off and the error LED illuminates.
· Warnings cause no effect on the operation of the user program but indicate something is wrong; The warning LED illuminates.
· Information level alarms are provided to help you find possible errors in your application and do not cause the error or warning LEDs to illuminate.

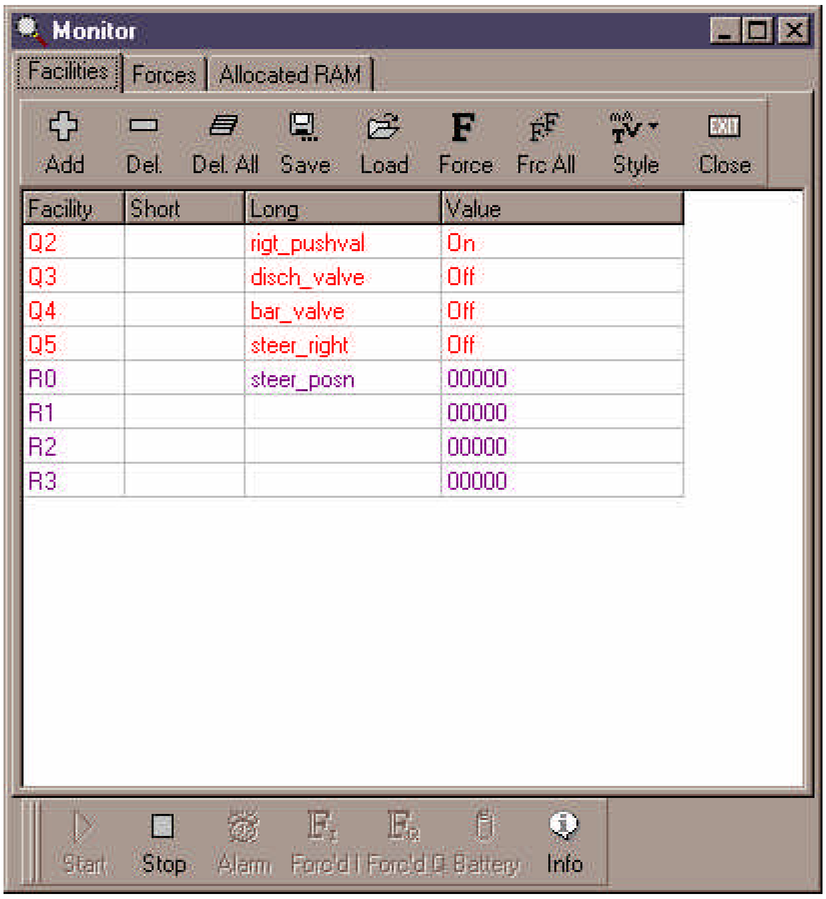
A list of these errors and warnings can be found under '**Errors and Warnings'** in Flex32/Help/Contents.

Monitor

The Monitor window allows you to monitor all FMT facilities and to force inputs and outputs.

The monitor can be very useful because of it's ability to monitor the facilities in the FMT this can enable you to see what is happening to, for example, data in a register.

To view and edit the monitor click on the 'Monitor' button on the main toolbar. The monitor window will then appear:

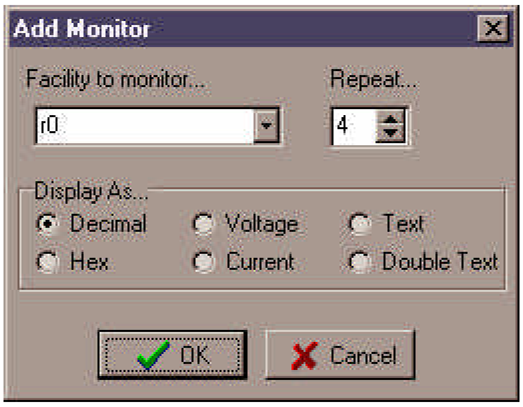


Facility	Short	Long	Value
Q2		rigt_pushval	On
Q3		disch_valve	Off
Q4		bar_valve	Off
Q5		steer_right	Off
R0		steer_posn	00000
R1			00000
R2			00000
R3			00000

When the monitor window appears you will be presented with three pages, Facilities, Forces and Allocated RAM.

Facilities monitor page:

'Add'. Clicking on this button will enable you to add a facility to monitor, you will be presented with a box in which to enter the data about the facility that you wish to monitor:



Facility to monitor... r0

Repeat... 4

Display As... [✓ Decimal] Voltage Text Hex Current Double Text

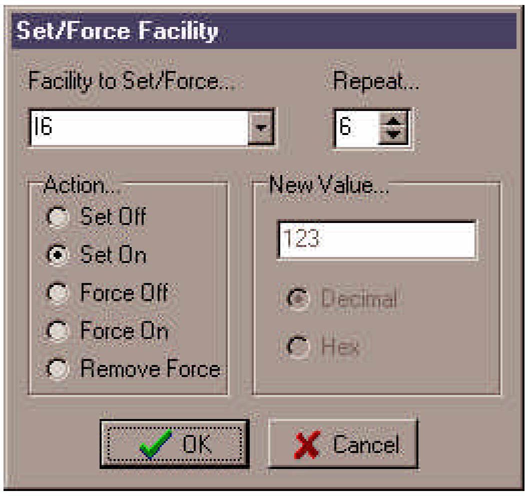
Enter the facility that you wish to monitor in the 'Facility to monitor' box. The repeat number to be entered will determined the amount of facilities to monitor eg. a repeat number of 5 entered when the facility to be monitored is q0 would mean that q0 to q4 would be displayed in the facility monitor.
Clicking on the various options in the 'Display As...' section will enable you to display the data in your facility in the ways listed. (Voltage and current displays are only really applicable to analogue inputs and outputs).

'Del'. Clicking on this button will remove the facility that has been highlighted from the those being monitored. To highlight and remove multiple facilities hold down the left mouse button over the facility to highlight and press the space bar to select, while still holding down the mouse button point to the next facility to select and press down the space bar to select it. Keep selecting facilities in this manner until you do not want to select any more, then release the mouse button and click on 'Del'.

'Del All'. Clicking on this button will clear all the facilities from the list.

'Save'. Clicking on this button will provide you with an opportunity to save your facility monitor name setup. You will be prompted to enter a name for the setup to be saved as. The setup will be saved as 'xxxx.msu' (the x's being whatever you type in for the file name) in your project directory.

'Load'. Clicking on this button will enable you to load a monitor setup that has been previously saved. You can either load a monitor setup from you current project directory, or you can load monitor setups from other projects by going to their directory. If no monitor setups are listed in you project directory when you click 'Load' then non have been saved previously.


Set/Force Facility

Facility to Set/Force... I6

Repeat... 6

Action...
( ) Set Off
(●) Set On
( ) Force Off
( ) Force On
( ) Remove Force

New Value... 123
(●) Decimal
( ) Hex

'Force'. Clicking on this button will provide you with the opportunity to set/force input or outputs on/off. When a facility is set then it is still under program control. eg if an output is set on then the net time the program in the FMT turns the output off then it will be able to do so. When a facility is forced then the program can not control it any more eg. if an output is forced on then the program can not turn it off until the force is cleared.

'Frc All'. Clicking on this will enable all the inputs and outputs that have been selected (for multiple selection see the 'Rem' section of this help) to be set/forced on or off. All facilities that have been forced can also have the forces cleared. Note: Be careful to ensure that nothing has been selected other than inputs and/or outputs otherwise, for example, numbers may be forced into registers if these happened to be selected which could lead to unpredictable results..

'Style'. Selecting this will enable the value in the facility currently highlighted to be displayed in decimal, hexadecimal, voltage, current or text (text takes the value and displays the ASCII character which corresponds with it).

'Close'. Clicking on this will close the monitor window.

'Start'. Starts user program execution if it has been stopped.

'Stop'. Stops user program execution if it is currently running.

'Alarm'. Selecting this will enable you to view any errors or warnings on the FMT.

'Forc'd I'. This button will be 'illuminated' if there are any inputs currently forced on. Clicking on it will enable the force to be removed.

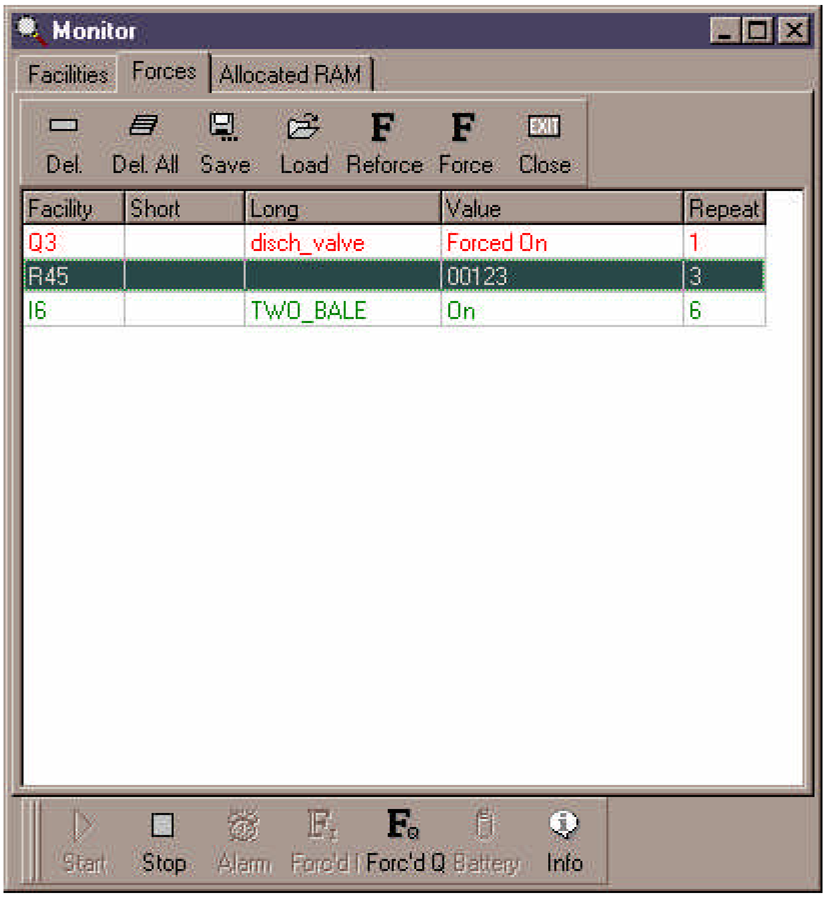
'Forc'd Q'. This button will be 'illuminated' if there are any outputs currently forced on. Clicking on it will enable the force to be removed.

'Battery'. This button will illuminate if the battery in the FMT currently connected is running low.

'Info'. Clicking on this button will provide you with information about the FMT that is currently connected eg. FMT type, firmware version etc.

Forces monitor page:

Any time a facility is forced or set to a new value it is recorded on the Forces page of the monitor form. This list of forces can be re-executed or saved to disk for later use.



Facility	Short	Long	Value	Repeat
Q3		disch_valve	Forced On	1
R45			00123	3
I6		TWO_BALE	On	6

'Del'. Clicking on this button will remove the force that has been highlighted from the list. To highlight and remove multiple forces hold down the left mouse button over the force to highlight and press the space bar to select. Keep selecting facilities in this manner until you do not want to select any more, then release the mouse button and click on 'Del'.

'Del All'. Clicking on this button will clear all the forces from the list.

'Save'. Clicking on this button will provide you with an opportunity to save your force list. You will be prompted to enter a name for the setup to be saved as. The setup will be saved as 'xxxx.mff' (the x's being whatever you type in for the file name) in your project directory.

'Load'. Clicking on this button will enable you to load a force list that has been previously saved. You can either load a setup from you current project directory, or you can load monitor setups from other projects by going to their directory.

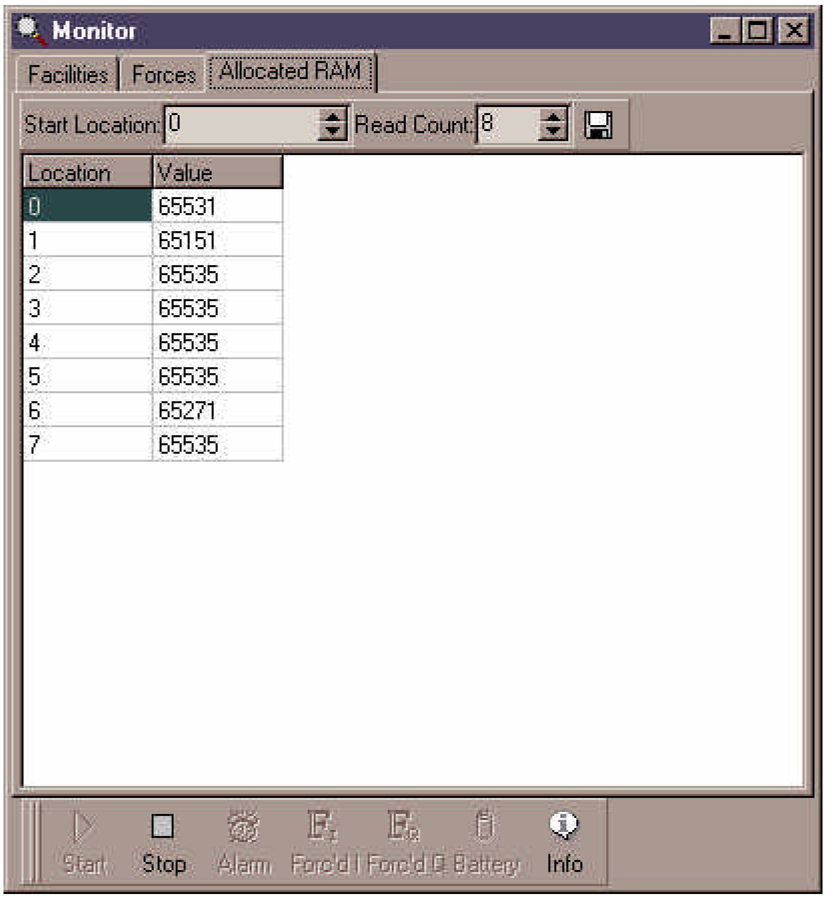
'Reforce'. Re-sends all the force commands current shown on the force list.

'Force'. Clicking on this button will provide you with the opportunity to set/force input or outputs on/off. This functions in exactly the same way as the Force button on the Facilities page.

'Close'. Clicking on this will close the monitor window.

Allocated RAM monitor page:

Clicking on this page will enable you to view the contents of any Allocated RAM you may have in the FMT.



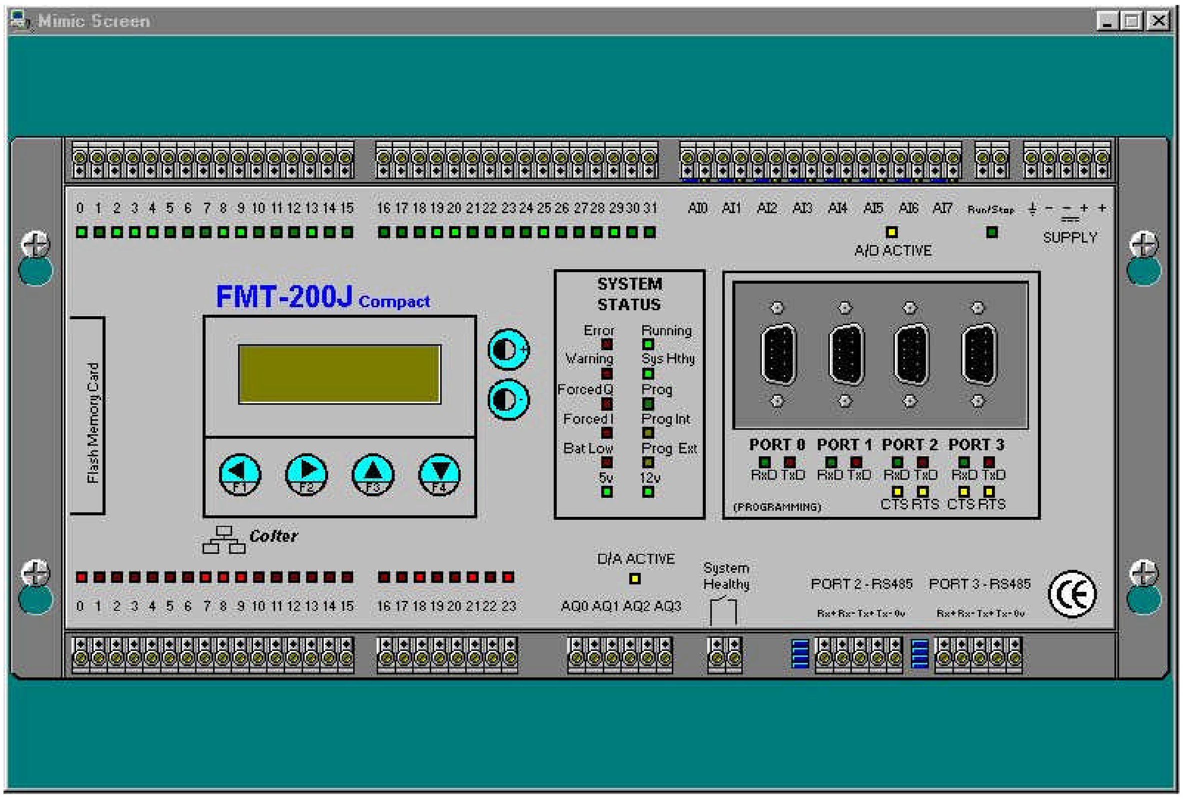
Start Location: 0 Read Count: 8

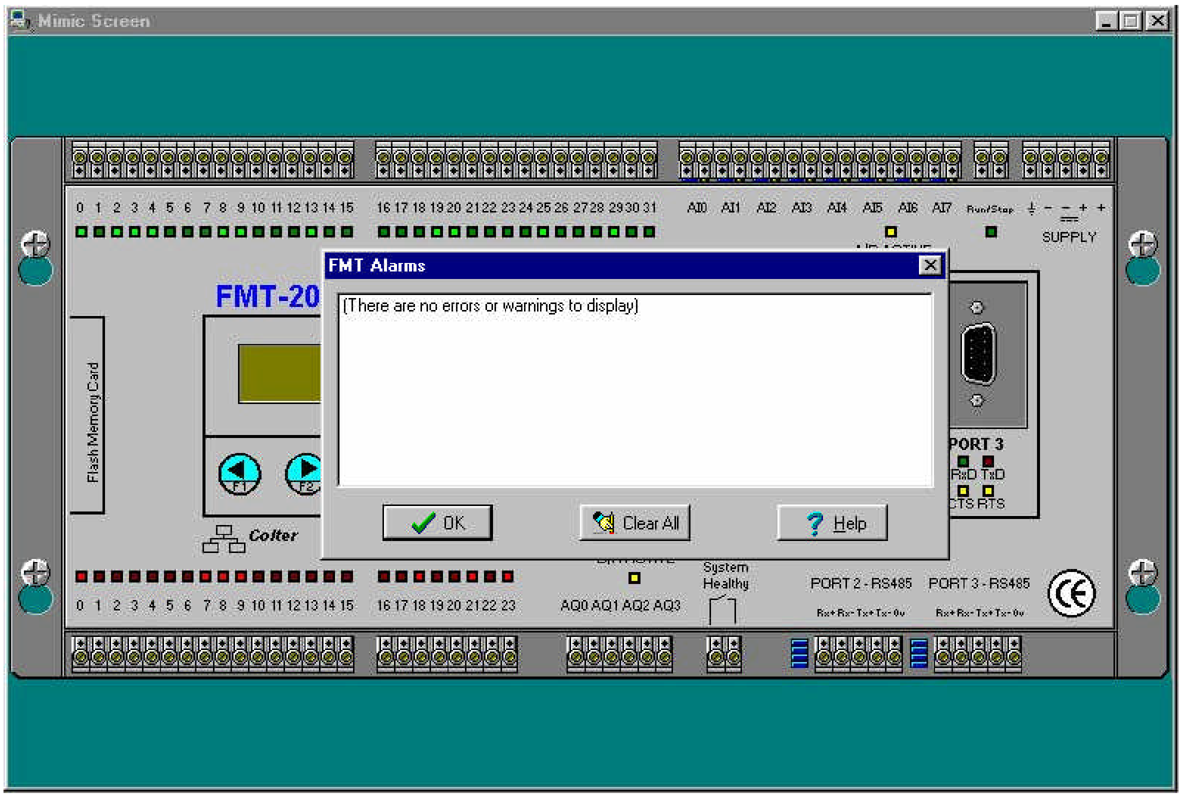
Location	Value
0	65531
1	65151
2	65535
3	65535
4	65535
5	65535
6	65271
7	65535

The 'Start Location' box allows you to enter the location that you wish to start reading the value of 16-bit words in allocated RAM. The 'Read Count' box enables you to specify how many 16-bit words you wish to view. For instance if there were sixty 16-bit words in allocated RAM and you wished to view the first twenty five then you would set the Start Location to 0 and the Read Count to 30, this would show you the contents of the 16-bit words 0 to 29.

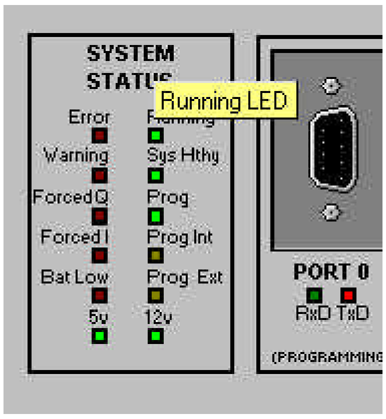
Clicking on the disk icon will save all the allocated RAM to a text file on disk in a suitable format for importing into a spreadsheet or database

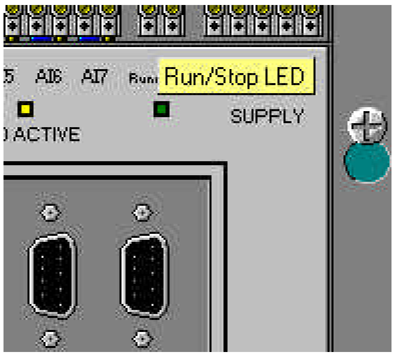
Mimic





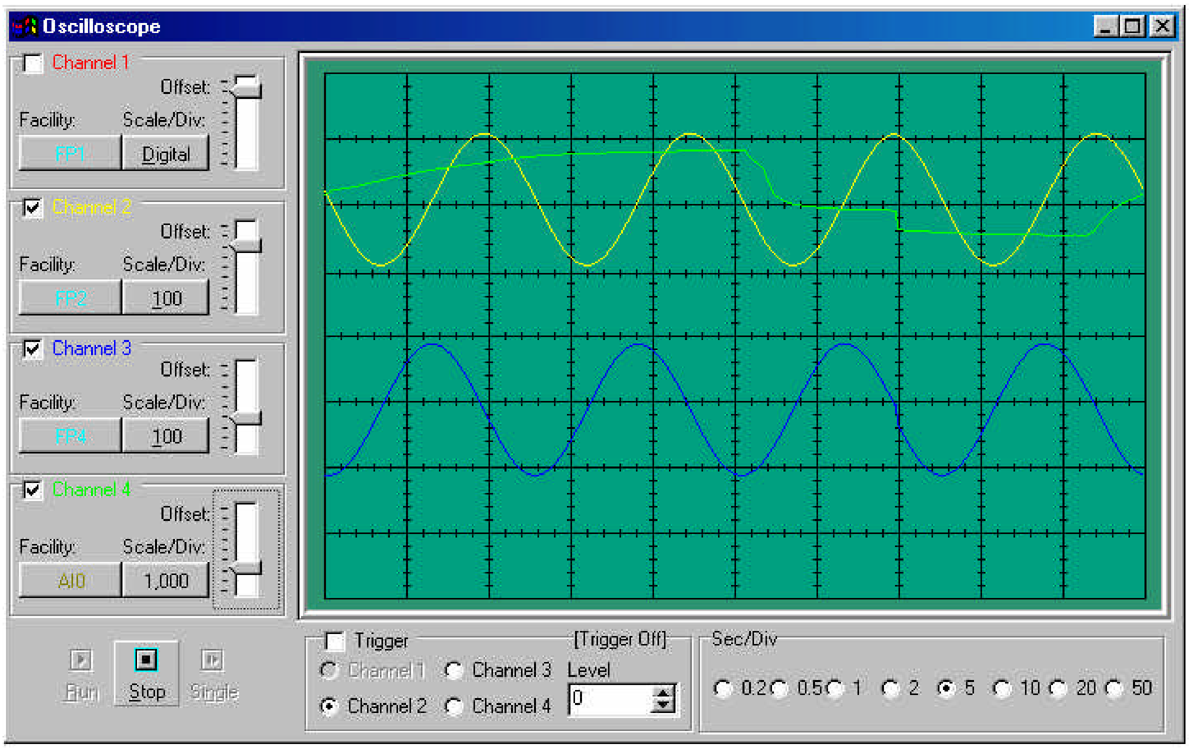
(There are no errors or warnings to display)





Oscilloscope

The Oscilloscope may be selected by clicking on the 'Scope' button on the Flex32 toolbar.
The Oscilloscope is usefull for debugging and monitoring facilities' magnitude over a period of time.



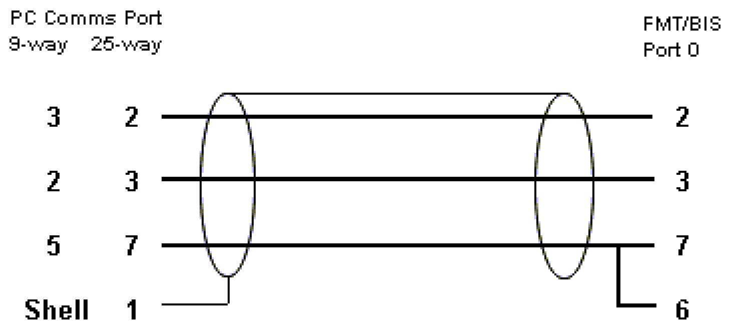
Click on various part of the Oscilloscope window shown below to find out how it functions

Programming Lead

The programming lead (Order Code: FMT PROGRAMMING LEAD) is used to connect your PC to any of the range of FMT controllers.

Note: For FMT-100's a Serial Port Adapter (Order Code: FMT-374) is required so that the programming lead can be connected to the FMT 100.

To make your own programming lead follow the connection diagram below.



The correct cable to use is 4 core non-twisted screened cable specified suitable for RS232 communications.

Please Note: The maximum specified distance for RS232 communications is 15 metres.

Note:

For PC’s without an RS232 (COM) port, then a USB to RS232 converter will be required.

You will need to know the port number ‘Windows’ has configured your adapter to, which must be in the range COM1-4.

Flex32 can only support COM1-4.

Help

Flex32 help file
www.coltergroup.co.uk

Ladder Functions

- Inputs: on, off, edge, fast edge.
- Outputs: on, off.
- Timers: on-delay, off-delay, pulse.
- Counters: count-up, count-down, pre-set, clear.
- Comparators: greater, less-than, equal to.
- 16-Bit, 32-Bit, and Floating Point operations:

  Add, Subtract, Multiply, Divide, Square, Square Root.

  And, Or, Exclusive Or. Negate, Logical shift, Rotate.

  Binary to BCD, BCD to Binary.

  Sine, Cosine, Tangent, Arctan, Exp, Ln, Log, and Power (Floating point only)
- Move:

  inputs, outputs, and flags to/from registers.

  analogue inputs and outputs to/from registers.

  16-Bit or 32-Bit registers to/from 16-Bit or 32-Bit registers.

- Data Handling with flags and registers:

  Shift Register.

  Stack.

  FIFO.

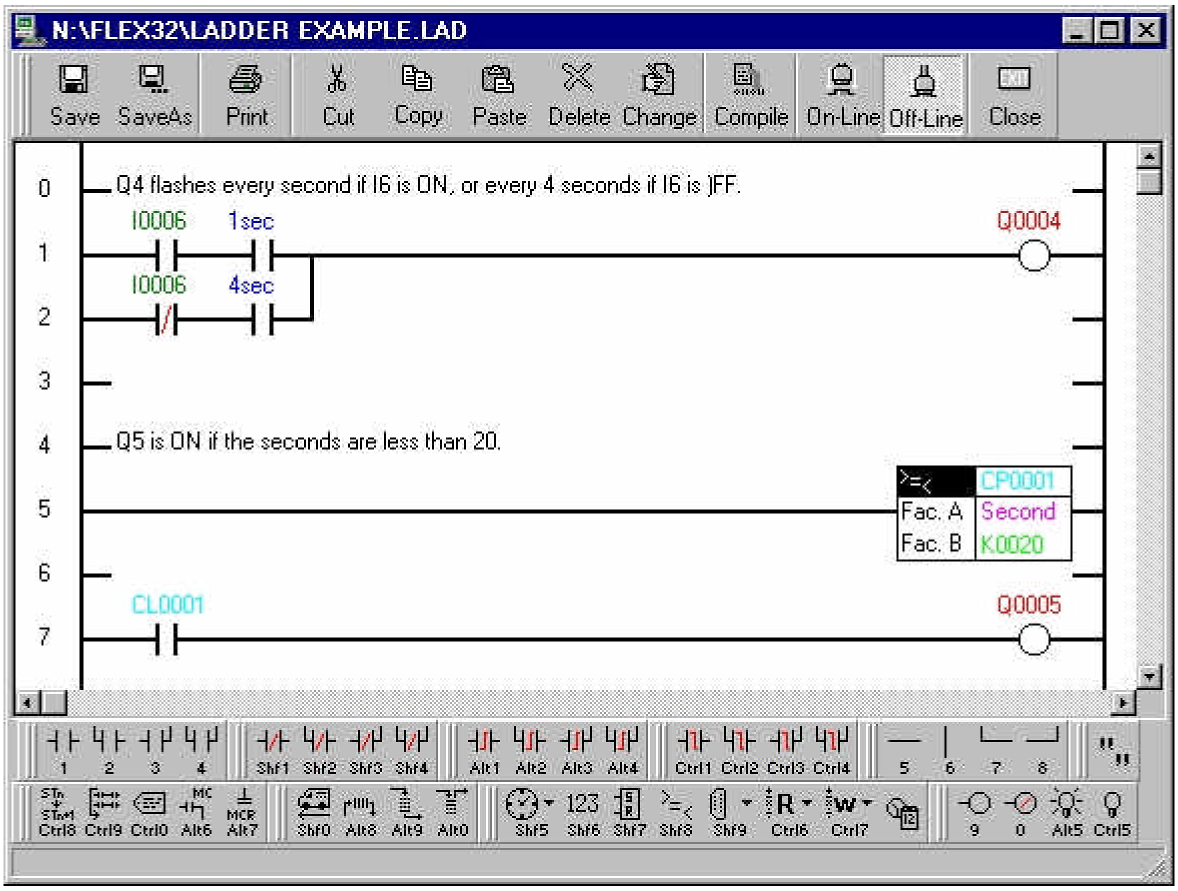
- Serial Communications:

  Send out Text string.

  Receive number, text, data.

- Compare received text with stored text.

Ladder Example


N:\FLEX32\LADDER EXAMPLE.LAD

Save SaveAs Print Cut Copy Paste Delete Change Compile On-Line Off-Line Close

```
0  Q4 flashes every second if I6 is ON, or every 4 seconds if I6 is )FF.
1  I0006  1sec                                             Q0004
2  I0006  4sec
3
4  Q5 is ON if the seconds are less than 20.
5                                           >=<     CP0001
                                            Fac. A  Second
                                            Fac. B  K0020
6
7  CL0001                                                  Q0005
```

Enter the above example in a ladder module, download, and test in debug...

Instruction Language

- Keywords
    - Wait_For I3
    - If I5

        * code only executed when I5 is on…

      Else_If I7

        * code only executed when I5 is off but I7 is on…

      Else

        * code only executed when I5 is off and I7 is off…

      End_if
    - Repeat 3

        * this code executed 3 times…

      End_Repeat
    - While AI0 < 750

        * this code executed all the time the value of AI0 is less than 750

      End_While
    - Do

        * this code executed continually until the value of R24 is greater then 40

      Until R24 > 40
    - For R100 = 0 to 99

        * this code executed 100 times with R100 equal to 0,1,2….99

      Next
    - Alert ErrorSub when I6 = 0

- Conditions, used with some keywords e.g.
    - If R3 = 100
    - While I4 OR (I3 and F4)

- Functions e.g.
    - Turn_on(q0)
    - add(1,r1,r1)

- Subroutines
    - Call MySub
    - Sub MySub

        * Code here is the subroutine

      End_Sub

Instruction Functions

- Turn On/off
  - Turn_On
  - Turn_Off
  - Fast_on
  - Fast_off
- Timers
  - On_Delay
  - Off_Delay
- 16-bit Operations
  - Add
  - Subt
  - Mul
  - Div
  - Bit_And
  - Bit_Or
  - Bit_Xor
  - Square
  - Square_root
  - Neg
  - Lsl
  - Lsr
  - Rol
  - Ror
  - Binary2Bcd
  - Bcd2Binary
- Move
  - Move
- Shift Register
  - Init_Shift
  - Clock_Shift
- FIFO
  - Init_Fifo
  - Fifo_In
  - Fifo_Out
- Stack
  - Init_Stack
  - Stack_Push
  - Stack_Pop
- 32-Bit Operations
  - Add_l
  - Subt_l
  - Mul_l
  - Div_l
  - Bit_And_l
  - Bit_Or_l
  - Bit_Xor_l
  - Square_l
  - Square_root_l
  - Neg_l
  - Lsl_l
  - Lsr_l
  - Rol_l
  - Ror_l
  - Ror_l
- Text
  - Text
- Get Number
  - Get_Number
- Serial In
  - Serial_In
  - Close_Port
  - Compare_Text
- RAM storage
  - Ram_Erase
  - Ram_Read
  - Ram_Write
- Flash Card Storage
  - Flash_Erase
  - Flash_Read
  - Flash_Write
- Miscellaneous
  - Set_RTC
  - Buzz
- Floating Point Operations
  - Add_f
  - Subt_f
  - Mul_f
  - Div_f
  - Square_f
  - Square_root_f
  - Sin_f
  - Cos_f
  - Tan_f
  - Arctan_f
  - Exp_f
  - Ln_f
  - Log_f
  - Power_f

Instruction Example

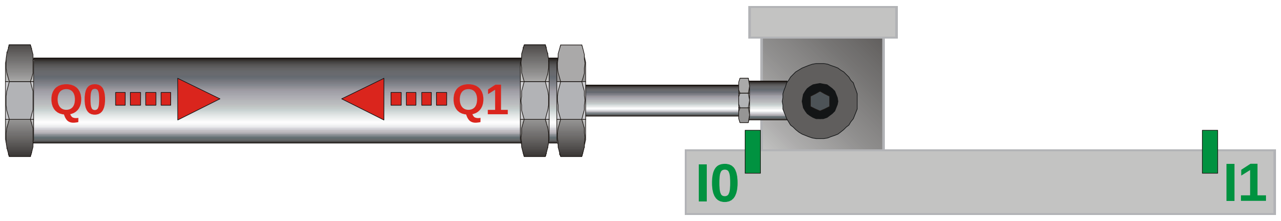
```
label loop

   * wait for the 'Go' button to be pressed and released...
   wait_for I0
   wait_not I0

   * move the cylinder to the end...
   turn_on(q0)
   wait_for I1
   * ... and back...
   turn_off(q0)

goto loop
```

Additional Examples

Example 1

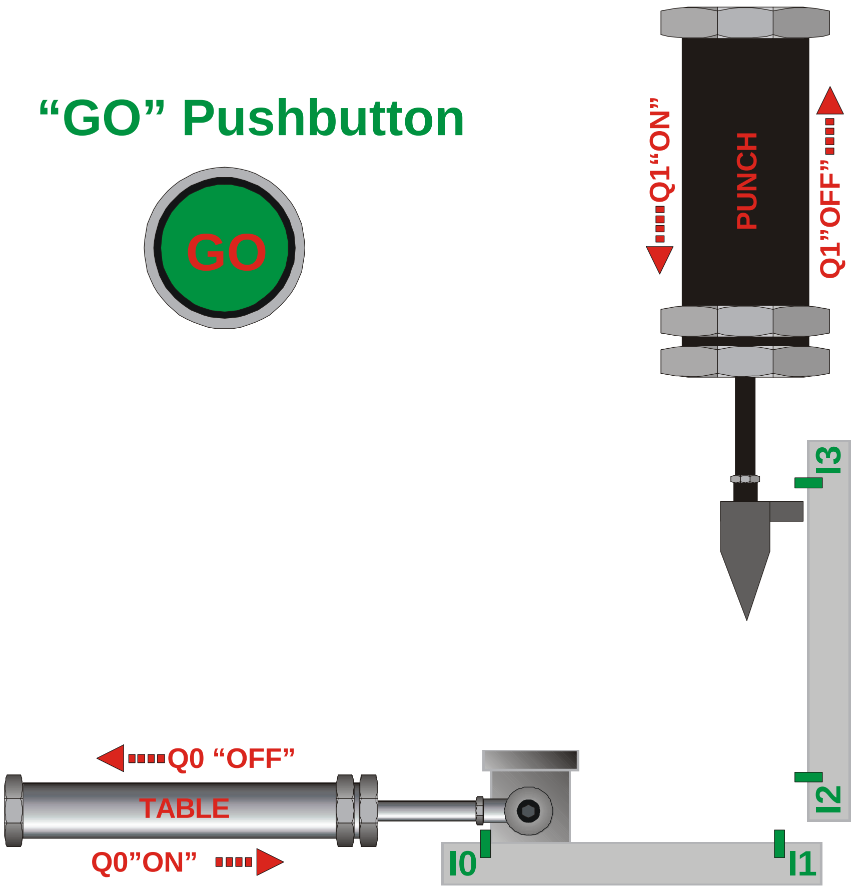
Inputs:	I0	Left limit
	I1	Right limit
Outputs:	Q0	Move Right
	Q1	Move Left

The target is moved left until the left limit is reached, waits for 2 seconds, then moves right until the right limit is reached. Repeat continuously.



Example 2

Inputs:	I0	Left Limit
	I1	Right Limit
	I2	Bottom Limit
	I3	Top Limit
	I4	'Go' button
Outputs:	Q0	Table Cylinder: Off = left, On = Right
	Q1	Punch Cylinder: Off = up, On = Down



When the 'Go' button is pressed, Q0 drives the table to be under the punch. After a 0.5 second delay the punch comes down and then retracts. The table can then return.

Add a 'pause' button that can be pressed anywhere in the cycle.

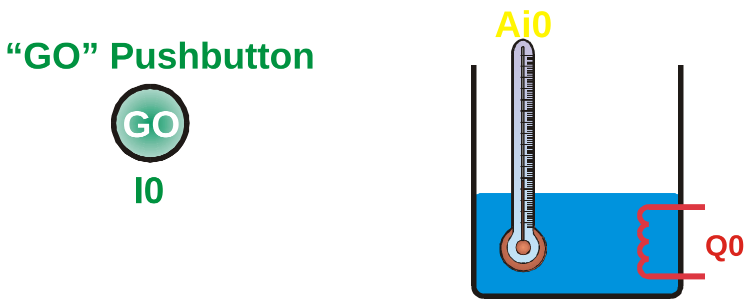
Add a display showing the total number of parts produced.

Add a facility to raise an alarm if any movement takes longer than 2 seconds.

Example 3

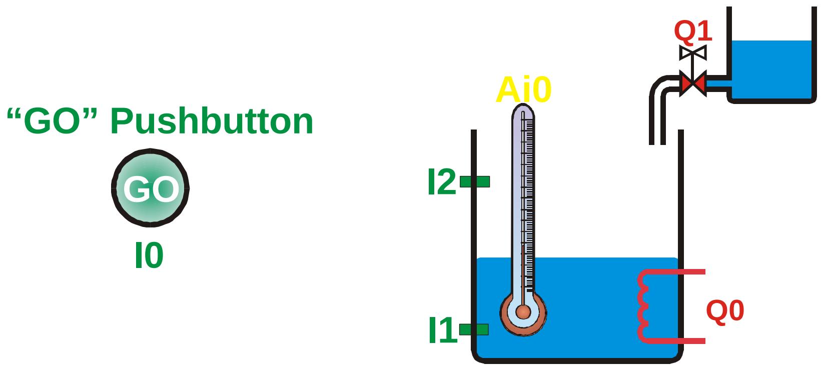
Inputs:	`AI0`	Water Temperature
	`I0`	Run Button
Outputs:	`Q0`	Heating Element

While the 'Run' input is on, the water should be heated until a set temperature is reached when the heater is switched off.

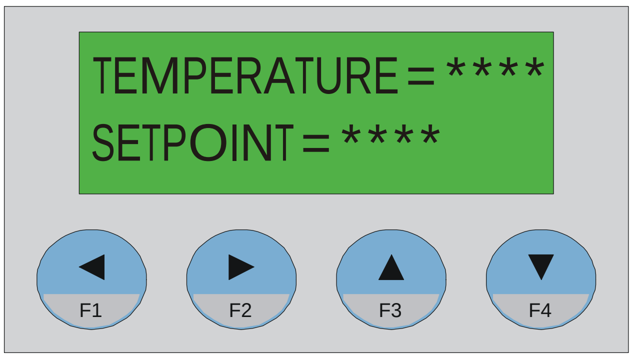


Add a header tank to feed the water heater:

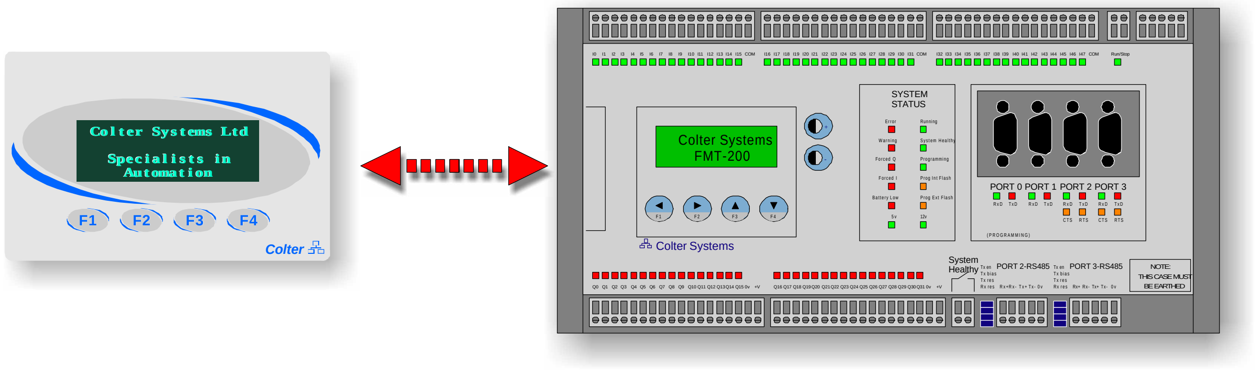
Inputs:	`I1`	Low Water Level
	`I2`	High Water Level
Outputs:	`Q1`	Water Valve



Add a display of the current temperature and the set-point. Allow the user to adjust the set-point using the 'up' and 'down' function keys.



Example 4



Write a program that decodes the key presses and then displays which buttons are pressed. Use a Flag to show the state of each key.

<u>HMI Operation.</u>

When a function key is pressed or released a three character message is transmitted by the HMI:

Start Character:	ASCII <STX>
Data:	ASCII ‘0’ to ‘?’ representing the pattern of keys pressed
Terminating Character:	ASCII <CR>.

Examples of the data byte that is sent back for various key presses is shown below:

Key F1 pressed:	ASCII ‘1’ = 31 (Hex), 00110001 (binary)
Key F3 pressed:	ASCII ‘4’ = 34 (Hex), 00110100 (binary)
Key F1 and F4 pressed at the same time:	ASCII ‘9’ = 39 (Hex), 00111001 (binary)
Keys F1, F2, F3 and F4: all pressed:	ASCII ‘?’ = 3F (Hex), 00111111 (binary)

Set the communications port to:

Baud Rate:	9600
Data bits:	8
Parity	No Parity

Example 5

Inputs:	`I0`	Run Button
	`I1`	Stop Button
	`I2`	Pause Button
	`I3`	Increment Set-Point
	`I4`	Decrement Set-Point

A machine is controlled by the buttons described above. Use allocated RAM to create a data-log that records the actions of the operator.

Allow the contents of the log to be viewed on the display using the function buttons to scroll through the records.

Use the Allocated RAM page of the FLEX32 monitor form to extract the logged data to a text file.

Modify the code to store the log on a plug in flash card instead of allocated RAM. Allow for the card to be erased by the operator.

Example 6

Execute module on input interrupt:

Enter the following program into a new module called “**conveyor**”, this module will mimic a conveyor motor/encoder combination:

```
Q0 = Brake On/Off
Q1 = Conveyor Slow/Fast
Q3 = Wrap and remove
I2 = Stop/Start
I3 = Detector
W0 = Encoder count  (Encoder ; 1 pulse = 0.1mm)
```

```
******************************************************************************
*                                                                            *
* Training course (worked) Example 5                          Date: 03/09/01 *
* Module: Example 6 Conveyor                                  Version 1.00   *
* Description: pseudo Conveyor                                Engineer: JMG  *
*                                                                            *
******************************************************************************

*************************** Code Entry Point *********************************

****************************** CONVEYOR **************************************

label start          * start of code

wait_for I2

if q0 and q1         * if brake is released and motor is in high speed
   add_l(30,w0,w0)   * run encoder 30x faster

   else_if q0 and q1 = 0    * if brake is released and motor is in slow speed
   add_l(1,w0,w0)           * run encoder at single speed

end_if

goto start
```

This example represents an automatic wrapping machine. The machine is started and the conveyor belt is set at “fast” speed, when a parcel is detected by the detector it needs to slow down the conveyor, increment by a further 200mm and then apply the brake. The parcel is then wrapped and removed by activating “Q3” whilst sounding the bleeper and then 2 seconds later the conveyor is started up again in “fast” and waits for the next package.

Use an interrupt driven module to detect the package and slow the conveyor.

Write the current operation and encoder value to the display.

Note:-

> **In interrupt mode the entire module will be executed at the exact moment that the specified input comes on. It should be remembered that executing large sections of code on an input interrupt will reduce the capacity of the FMT to process the other code within the project. You are limited to executing 20 steps of code in one interrupt before the firmware will raise an 'Input Interrupt overrun' error.**

Example 7

Serial Communications

This exercise requires two FMTs to communicate with each other. The object is to read the first 16 inputs from one FMT and display their state on the other FMTs outputs.

Method 1. Linkline

Connect both FMT Port 2s together. Set Port 2 to be Linkline with one FMT as station 0 and the other as station 1. Use the first register of each station’s Linkline block to transfer its inputs to the other stations.

Method 2. Modbus

You will need to connect Port 2 on each FMT to Port 3 on the other. Set up Port 3 as a modbus slave; 9600 baud, 8 data bits, 1 stop bit, no parity. Set Port 2 to user code. Write an application module to use the Modbus_Master function to read the state of the other FMTs inputs.

Method 3. Custom serial protocol

You will need to connect Port 2 on each FMT to Port 3 on the other. Set up both Port 2 and Port 3 for user code at 9600 baud, 8 data bits, 1 stop bit, no parity.

Use the Text function to send the following string out of port 2;

<stx>D<Data1><Data0><checksum><etx>

where <stx> and <etx> are ASCII control characters,

Data1 is a byte representing the state of inputs 8 to 15

Data0 is a byte representing the state of inputs 0 to 7

Checksum is a one byte additive checksum

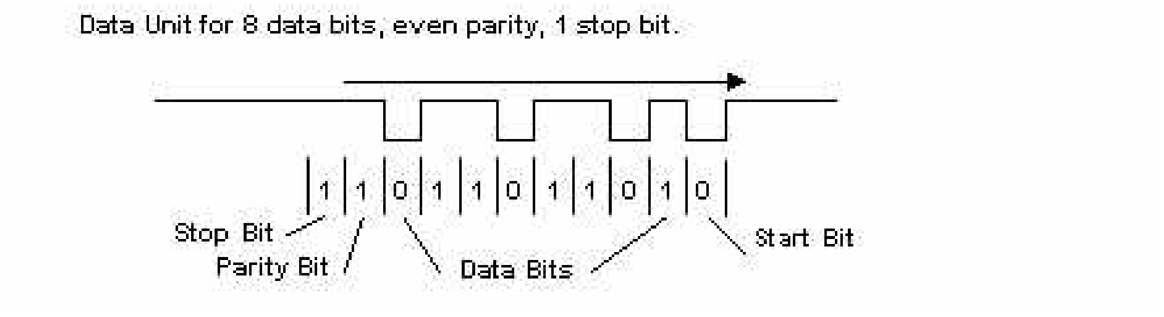
Use the Serial-In function to receive the same format string from the other FMT on Port 3.

Appendix A - Serial Communications

Communication Parameters...

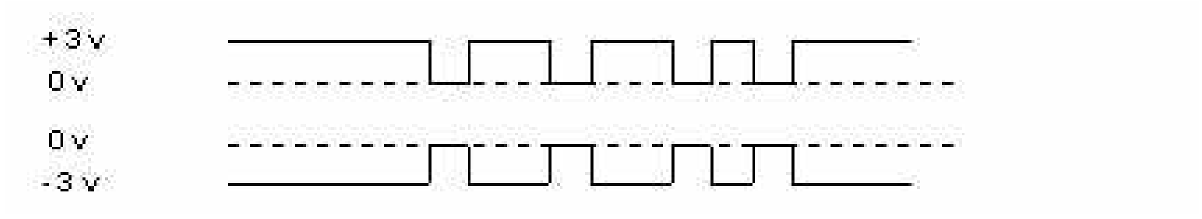
Baud Rate (Speed)	75,110,300,600,1200,2400,4800,9600,19200,38400, 57600
Data Bits	5,6,7,8
Parity (error checking)	None, Odd, Even, (Mark, Space)
Stop Bits	1, 2, (1.5)

Data for one character...

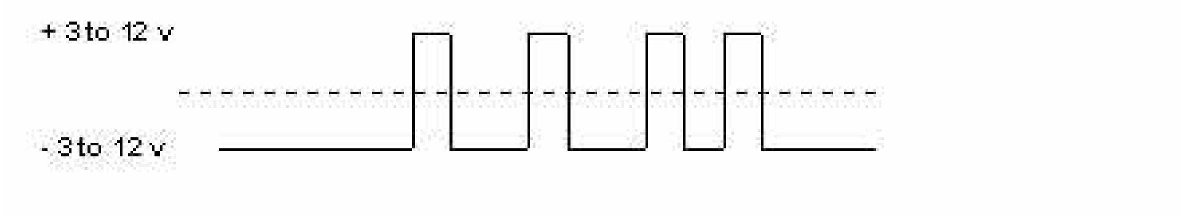


Electrical signals

RS232



RS485



Appendix B – ASCII Table

Dec	Hex	Abbr	Dec	Hex	Chr	Dec	Hex	Chr	Dec	Hex	Chr
0	0	nul	32	20		64	40	@	96	60	`
1	1	soh	33	21	!	65	41	A	97	61	a
2	2	stx	34	22	"	66	42	B	98	62	b
3	3	etx	35	23	#	67	43	C	99	63	c
4	4	eot	36	24	$	68	44	D	100	64	d
5	5	enq	37	25	%	69	45	E	101	65	e
6	6	ack	38	26	&	70	46	F	102	66	f
7	7	bell	39	27	'	71	47	G	103	67	g
8	8	bs	40	28	(	72	48	H	104	68	h
9	9	tab	41	29	)	73	49	I	105	69	i
10	A	lf	42	2A	*	74	4A	J	106	6A	j
11	B	vt	43	2B	+	75	4B	K	107	6B	k
12	C	Ff	44	2C	,	76	4C	L	108	6C	l
13	D	Cr	45	2D	-	77	4D	M	109	6D	m
14	E	So	46	2E	.	78	4E	N	110	6E	n
15	F	Si	47	2F	/	79	4F	O	111	6F	o
16	10	Dle	48	30	0	80	50	P	112	70	p
17	11	dc1	49	31	1	81	51	Q	113	71	q
18	12	dc2	50	32	2	82	52	R	114	72	r
19	13	dc3	51	33	3	83	53	S	115	73	s
20	14	dc4	52	34	4	84	54	T	116	74	t
21	15	Nak	53	35	5	85	55	U	117	75	u
22	16	Syn	54	36	6	86	56	V	118	76	v
23	17	Etb	55	37	7	87	57	W	119	77	w
24	18	Can	56	38	8	88	58	X	120	78	x
25	19	Em	57	39	9	89	59	Y	121	79	y
26	1A	Sub	58	3A	:	90	5A	Z	122	7A	z
27	1B	Esc	59	3B	;	91	5B	[	123	7B	{
28	1C	Fs	60	3C	<	92	5C	\	124	7C	|
29	1D	Gs	61	3D	=	93	5D	]	125	7D	}
30	1E	Rs	62	3E	>	94	5E	^	126	7E	~
31	1F	Us	63	3F	?	95	5F	_	127	7F	del

Appendix C – Ziegler – Nichols Closed Loop Tuning

The Ziegler-Nichols Closed Loop method is one of the more common methods used to tune control loops. It was first introduced in a paper published in 1942v by J.G. Ziegler and N.B. Nichols, both of whom at the time worked for Taylor Instrumentation companies of Rochester, NY.

The open loop method is useful for most process control loops. To use the method the loop is tested with the controller in automatic. The Closed Loop method determines the gain at which a loop with proportional only control will oscillate, and then derives the controller gain, reset, and derivative values from the gain at which the oscillations are sustained and the period of oscillation at that gain.

The ZN Closed Loop method should produce tuning parameters which will obtain quarter wave decay. This is considered good tuning but is not necessarily optimum tuning.

Steps

- ♦ Ensure that the process is “lined out” with the loop to be tuned in automatic with a gain low enough to prevent oscillation.
- ♦ Increase the gain in steps of one-half the previous gain. After each increase, if there is no oscillation change the setpoint slightly in order to trigger any oscillation.
- ♦ Adjust gain so that the oscillation is sustained, that is, continues at the same amplitude. If the oscillation is increasing, decrease the gain sightly. If it is decreasing, increase the gain slightly.
- ♦ Make note of the gain which causes sustained oscillations and the period of oscillation. These are the “ultimate Gain” (GU) and the “Ultimate Period” (PU) respectively.
- ♦ Calculate the tuning for the following set of equations. Use the set which corresponds with the desired configuration: P only, PI, or PID.

Tuning Equations

♦ P Only:	Gain = 0.5 GU
♦ PI:	Gain = 0.45 GU, Reset = 1.2 x PU
♦ PID:	Gain = 0.6 GU, Reset = 2 x PU, Derivative = PU/8

Taken from *Process Control Solutions*

Example 1 – Worked Answer

This example contains 1 module

```
*****************************************************************************
*                                                                           *
* Training course (worked) Example 1                         Date: 03/09/01 *
* Module: Example 1                                          Version 1.00   *
* Description: Simple Conveyor                               Engineer: JMG  *
*                                                                           *
*****************************************************************************

*************************** Code Entry Point *********************************

*************************** SIMPLE CONVEYOR **********************************
label start * start of code

turn_on(Q0)          *move target to the right
wait_for I1          *wait for target to reach end
turn_off(q0)         *Turn off conveyor
On_Delay(t0,0,0,2,0) *turn on 2 sec timer
wait_for(t0)         *Wait for 2 seconds

turn_on(Q1)          *move target to the left
wait_for I0          *wait for target to reach end
turn_off(q1)         *Turn off conveyor
On_Delay(t0,0,0,2,0) *turn on 2 sec timer
wait_for(t0)         *Wait for 2 seconds


goto start
```

Example 2 – Worked Answer

This example contains 3 modules

Module1

```
******************************************************************************
*                                                                            *
* Training course (worked) Example 2                             Date: 29/08/01 *
* Module: Example 2 Main                                         Version 1.00   *
* Description: Main process control                              Engineer: JMG  *
*                                                                            *
******************************************************************************

*************************** Alert Functions **********************************

alert fred when I5   *pause if I5 comes on

*************************** Code Entry Point *********************************

label start * start of code

wait_for I4          *start button
turn_on(Q0)          *move target to the right
wait_for I1          *wait for target to reach end
On_Delay(t0,0,0,0,5) *turn on 0.5 sec timer
wait_for(t0)         *Wait for 2 seconds
turn_on(Q1)          *move punch downwards
wait_for I2          *wait for punch to hit bottom
turn_on(F0)          *flag to say punch was successful
add(1,r0,r0)         *count of how many products produced
turn_off(Q1)         *retract punch
wait_for I3          *wait for punch to fully retract
turn_off(Q0)         *retract table
wait_for I0          *wait for table to be fully retracted


goto start           *loop back to start


************************ Code for Subroutine ********************************

sub fred

wait_not I5          *continue cycle if I5 goes off

end_sub
```

Module2

```
****************************************************************************
*                                                                          *
* Training course (worked) Example 2                          Date: 29/08/01 *
* Module: Example 2 Display                                   Version 1.00   *
* Description: Prints Process count to display                Engineer: JMG  *
*                                                                          *
****************************************************************************

*************************** Code Entry Point ********************************

label start * start of code

if f0              * product has been made
   text(tx0,8)     * print total to display
   turn_off(f0)    * reset flag for next time

end_if

goto start
```

```
******************************************************************************
*                                                                            *
* Training course (worked) Example 2                         Date: 29/08/01 *
* Module: Example 2 Alarm                                    Version 1.00   *
* Description: Monitor process duration                      Engineer: JMG  *
*                                                                            *
******************************************************************************

*************************** Code Entry Point *********************************

name TIME_OUT  25            * timeout set to 5 seconds...
move(0,r1,2)                 * reset alarm counts

label start                  * start of code

* this loop is executed 5 times a second...
wait_for 200ms
wait_not 200ms

* monitor table...
if (q0 and I1=0) or (q0=0 and I0=0)
   * if table is moving, but end stop isn't reached...
   add(1,r1,r1)
else
   * all ok, clear the count...
   move(0,r1,1)
end_if

* monitor punch...
if (q1 and I2=0) or (q1=0 and I3=0)
   * if punch is moving, but end stop isn't reached...
   add(1,r2,r2)
else
   * all ok, clear the count...
   move(0,r2,1)
end_if

* check all axis for error...
if (r1 > TIME_OUT) or (r2 > TIME_OUT)
   turn_on(q6)
else
   turn_off(q6)
end_if


*loop back to start
goto start
```

Example 3 – Worked Answer

This example contains 2 modules

Module 1

```
*****************************************************************************
*                                                                           *
* Training course (worked) Example 3                            Date: 30/08/01 *
* Module: Example 3 Main                                        Version 1.00   *
* Description: Main process control                             Engineer: JMG  *
*                                                                           *
*****************************************************************************

*************************** Code Entry Point *********************************

************************ TEMPERATURE MONITORING ******************************
label start          * start of code

* move(5000,r0,1)      *make setpoint 5000

if I0 and ai0 <r0    *"GO" is on and temperature is less than setpoint
   turn_on(Q0)       *turn on heater

   else
      turn_off(Q0)   *otherwise turn the heater off

end_if

********************** WATER LEVEL MONITORING ********************************

if_not I1            *water level is low, turn on header tank
   turn_on(Q1)
end_if

if I2                *water level is too high, turn off header tank
   turn_off(Q1)
end_if


goto start           *loop back to start
```

```
*****************************************************************************
*                                                                           *
* Training course (worked) Example 3                       Date: 30/08/01 *
* Module: Example 3 display                                Version 1.00   *
* Description: Main process control                        Engineer: JMG  *
*                                                                           *
*****************************************************************************

*************************** Code Entry Point *********************************

************************ TEMPERATURE MONITORING ******************************
label start             * start of code

if r2 <> ai0  or key-f3 or key-f4   * update the display when the temperature
changes
   move(ai0,r2,1)
   wait_for P8free                  * or the setpoint is changed
   text(tx1,8)
end_if


if key-f3               * setpoint is being incremented
   wait_not key-f3
   add(100,r0,r0)       * increment setpoint by 100
   text(tx1,8)
end_if

if key-f4               * setpoint is being decremented
   wait_not key-f4
   subt(100,r0,r0)      * decrement setpoint by 100
   text(tx1,8)
end_if


goto start              *loop back to start
```

Example 4 – Worked Answer

This example contains 2 modules

Module 1

```
*****************************************************************************
*                                                                           *
* Training course (worked) Example 4                        Date: 30/08/01 *
* Module: Example 4 Main                                    Version 1.00   *
* Description: Colter HMI Connection                        Engineer: JMG  *
*                                                                           *
*****************************************************************************

*************************** Code Entry Point *********************************

************************ HMI DECODE KEYS *************************************

turn_on(f100)


label start          * start of code

do
   serial_in(r0,10,x0d,9999,x0003,3)
** Serial_In(buffer,size,terminator,timeout,mode,port)

until p3-ok or p3-err   *until port returns a "comms ok" or comms "error"


if r0 = x02          *check for valid string...

   move(r1,r100,1)   *move key value into another register
   turn_on(f100)     *flag to say valid value in R100
end_if

*loop back to start
goto start
```

Module 2

```
******************************************************************************
*                                                                            *
* Training course (worked) Example 4                        Date: 30/08/01 *
* Module: Example 4 Display                                 Version 1.00   *
* Description: Colter HMI Connection                        Engineer: JMG  *
*                                                                            *
******************************************************************************

*************************** Code Entry Point *********************************
On_Delay(t3,0,0,5,0) * wait for display to get it's marbles together
wait_for t3
turn_on(f100)          * when display is powered up a value will appear on screen

************************ HMI DISPLAY KEY PRESSES *****************************
label start            * start of code

wait_for f100          * Wait for valid value
move(r100,f0,4)        * move key presses out to flags
                       * Key1 = F0, Key2 = F1.....etc
wait_for p3free        * make sure port is not in use
text(tx5,3)            * move to start of first line

wait_for p3free        * make sure port is not in use
if f0                  * if flag0 was set then key1 was pressed
   text(tx2,3)         * so send text saying *ON**

   else
      text(tx3,3)      * otherwise send text saying *OFF*
end_if

wait_for p3free        * make sure port is not in use
if f1                  * if flag1 was set then key1 was pressed
   text(tx2,3)         * so send text saying *ON**

   else
      text(tx3,3)      * otherwise send text saying *OFF*
end_if

wait_for p3free        * make sure port is not in use
if f2                  * if flag2 was set then key1 was pressed
   text(tx2,3)         * so send text saying *ON**

   else
      text(tx3,3)      * otherwise send text saying *OFF*
end_if

wait_for p3free        * make sure port is not in use
if f3                  * if flag3 was set then key1 was pressed
   text(tx2,3)         * so send text saying *ON**

   else
      text(tx3,3)      * otherwise send text saying *OFF*
end_if
turn_off(f100)         * turn off flag to say display written to
                       * waiting for next key press


goto start             *loop back to start
```

Example 5 – Worked Answer

This example contains 2 modules

Module 1

```
******************************************************************************
*                                                                            *
* Training course (worked) Example 5                          Date: 31/08/01 *
* Module: Example 5 MAIN                                      Version 1.00   *
* Description: Colter HMI Connection                          Engineer: JMG  *
*                                                                            *
******************************************************************************

*************************** Code Entry Point *********************************

ram_erase()
move(0,POINTER1,2)

******************* KEY PRESS STORED IN ALLOCATED RAM ************************

label start          * start of code

wait_for I0 or I1 or I2 or I3 or I4 * wait for any input


if I0                            * if I0 came on put a 1 in allocated RAM
   move(1,r101,1)                * and then increment the RAM_WRITE pointer
   add_l(1,POINTER1,POINTER1)
   wait_not I0                   * wait for Input 0 to go off

   else_if I1
      move(2,r101,1)
      add_l(1,POINTER1,POINTER1) *etc
      wait_not I1

      else_if I2
         move(3,r101,1)
         add_l(1,POINTER1,POINTER1) *etc
         wait_not I2

         else_if I3
            move(4,r101,1)
            add_l(1,POINTER1,POINTER1) *etc
            wait_not I3

            else_if I4
               move(5,r101,1)
               add_l(1,POINTER1,POINTER1) *etc
               wait_not I4
end_if

call RAM_WRITE_SUB               * call subroutine that writes the RAM location


goto start           *loop back to start

****************************** SUBROUTINES ***********************************
sub RAM_WRITE_SUB

* write the value of the input pressed registers to the allocated ram
Ram_Write(r101,POINTER1,1)

end_sub
```

```
*****************************************************************************
*                                                                           *
* Training course (worked) Example 5                          Date: 31/08/01 *
* Module: Example 5 RAM READ                                  Version 1.00   *
* Description: Colter HMI Connection                          Engineer: JMG  *
*                                                                           *
*****************************************************************************

*************************** Code Entry Point *********************************

************************** READ ALLOCATED RAM ********************************

label start          * start of code

wait_for key-f3 or key-f4

if key-f3
   add_l(1,POINTER2,POINTER2)
   add(1,r103,r103)
   wait_not key-f3

   else_if key-f4
      subt_l(1,POINTER2,POINTER2)
      subt(1,r103,r103)
      wait_not key-f4

end_if

call RAM_READ_SUB
text(tx10,8)


goto start           *loop back to start

****************************** SUBROUTINES ***********************************


sub RAM_READ_SUB

* read the data back
Ram_Read(POINTER2,r102,1)    *Ram_Read(pointer,destination,number)

end_sub
```

Example 6 – Worked Answer

This example contains 2 modules + (pseudo conveyor module)

Module 1

```
*****************************************************************************
*                                                                           *
* Training course (worked) Example 5                          Date: 03/09/01 *
* Module: Example 6 main                                      Version 1.00   *
* Description: Process code                                   Engineer: JMG  *
*                                                                           *
*****************************************************************************

label start        * start of code

wait_for I2        * wait for the operator to press the start button…
turn_on(q0)        * …start motor in fast…
turn_on(q1)

wait_for f0        * wait for the interrupt module to see the mark

wait_for w0 >= w1 * wait for encoder count to equal setpoint + encoder

fast_off(q0)       * put brake on
turn_off(f0)       * clear flag…

turn_on(q3)        * start wrapper…

repeat 4           * wait for 2 seconds sounding the buzzer…
   Buzz(25)
   on_delay(t0,0,0,0,50)
   wait_for t0
end_repeat

turn_off(q3)   *stop wrapper...

goto start
```

Module 2

```
*****************************************************************************
*                                                                           *
* Training course (worked) Example 6                          Date: 03/09/01 *
* Module: Example 6 INTERUPT                                  Version 1.00   *
* Description: Process code                                   Engineer: JMG  *
*                                                                           *
*****************************************************************************

*************************** Code Entry Point *********************************

move(w0,w1,1)            * save the encoder value at the registration point
turn_off(q1)             * …select slow speed…
add_l(2000,w1,w1)        * …add set-point to encoder to find stop point…
turn_on(f0)              * …indicate we have detected the mark to the main module

end_int
```