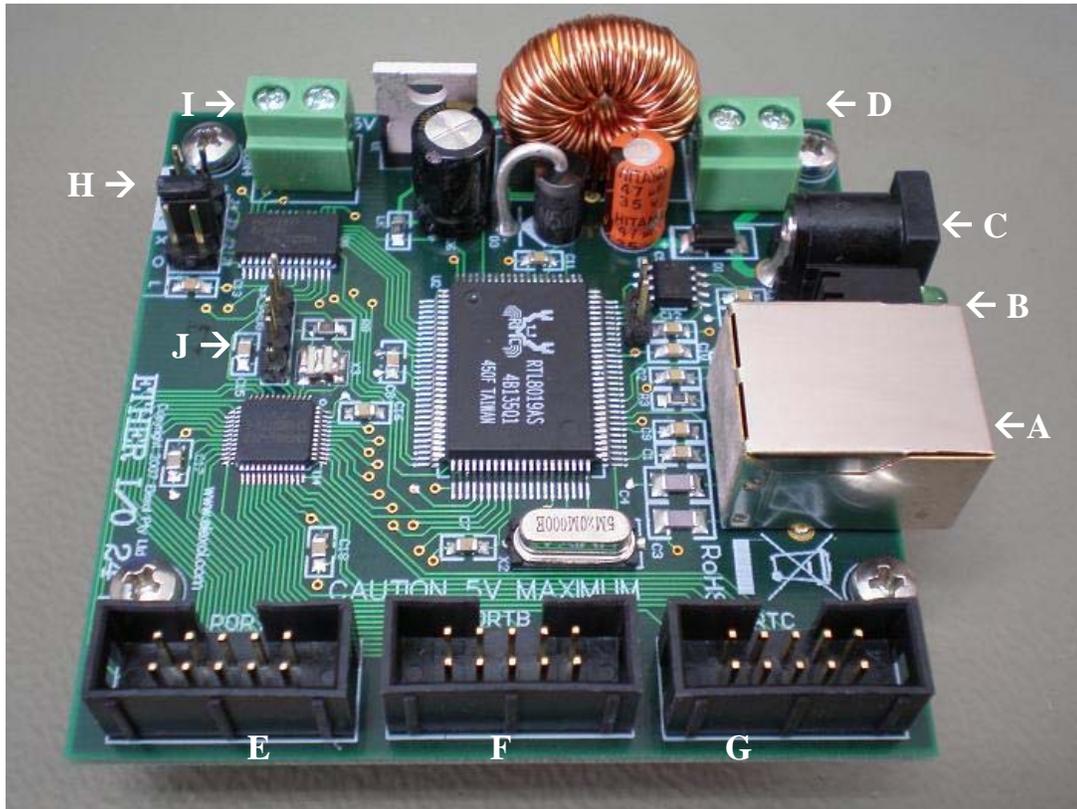# ELEXOL

Ether I/O 24 R

Users Manual

Version 1.1
For Firmware Release 2.0

Ether I/O 24 R Module Hardware

A. Ethernet Connector
B. LED Indicators
C. Power Connector 2.1mm DC Jack Center +
D. Power Connector Screw Terminal
E. Port A Connector
F. Port B Connector
G. Port C Connector
H. CONFIG Jumper Links J1-J4
I. Optional 5V output screw terminal
J. Factory Programming Header

Table of Contents

Contents

When your Ether I/O 24 module arrives you should receive the following items.

1   The Ether I/O 24 Module in a Protective Anti Static Bag
2   2 Jumper Links (please note that these links may be factory fitted to the CONFIG connector on the board)

Please inspect the Module carefully for any damage that may have occurred during shipping or handling. If you do not receive the module in its protective bag, if the Jumper links are missing, or your module appears damaged in any way, please contact your place of purchase immediately.

Please be sure to download the latest support tools and datasheets from the ELEXOL website at www.elexol.com and check that you are using the most up to date versions.

Technical Support and Further Information

For any questions relating to the Ether I/O 24 or if we can assist you with integrating the Ether I/O 24 into your own equipment please contact us by Email at support@elexol.com

Email: support@elexol.com
Ph: +61 755 031202   Fax: +61 755 031206

Elexol Pty Ltd   Elexol Pty Ltd
Unit 1   PO Box 2742
8 Pirelli Street, Southport   Southport Business Centre
Queensland 4215   Queensland 4215
Australia   Australia

Product Use Limitations, Warranty and Quality Statement

This product is not designed, intended, or recommended for use in systems intended to support or sustain life, or for any other application in which the failure of the product could create a situation where personal injury or death may occur and should not be used for those applications.

The Ether I/O 24 is warranted to be free from manufacture defects for a period of 12 months from the date of purchase. Subjecting the device to conditions beyond the Absolute Maximum Ratings listed in this document will invalidate this warranty. As the Ether I/O 24 is a static sensitive device, anti static procedures should be used in its handling.

All Ether I/O 24 units are tested during manufacture and are despatched free of defects.

Elexol is committed to providing products of the highest quality. Should you experience any product quality issues with this product please contact our quality assurance manager at the above address.

Disclaimer

This product and its documentation are provided as-is and no warranty is made or implied as to their suitability for any particular purpose. Elexol Pty Ltd will not accept any claim for damages arising from the use of this product or its documentation. This document provides information on our product and all efforts are made to ensure the accuracy of the information contained within. The specifications of the product are subject to change and continual improvement without notification.

Other than the extent permitted by law and subject to the Trade Practice Act, all and any liability for consequential loss or damage arising from an ELEXOL Ether I/O24 module is hereby limited, at ELEXOL's discretion, to replacement or repair.

Introduction

Thank your for purchasing the Ether I/O 24 Module. We trust that programming and using the Ether I/O 24 will prove to be an ongoing positive experience for you.

Please take a moment to unpack and inspect your module to ensure it is undamaged and is complete with the items listed in the contents section. As the module has several connectors that are critical to its correct operation, please take a moment to familiarize yourself with the Module's layout and connector locations.

Along with the module itself you will require the following items for its operation.

1. A Power Supply 8 - 34V DC, Minimum 1.2Watts (0.1A@12V / 0.05A@24V)
2. A PC equipped with an Ethernet Network Interface and Software to support the TCP/IP protocol suite
3. A Network cable for connection of the Module to your networks hub/switch or a crossover cable for connection directly to the PC
4. One or more 5V operable devices or switches and connecting cables for operation by, or connection to the module. (A range of accessory devices are available from your ELEXOL distributor.)

The module's test and programming software is only operable on a PC with the Windows operating system. Your own software can be written to operate and program the module from any platform that has an Ethernet connection. Full details of the programming and operation of the module are contained in the programming sections of this manual.

UDP/IP packets sent over the network from any device with appropriate software are able to operate the Ether I/O 24. The module has DHCP support to allow it to automatically obtain its IP address from a suitable server device. If a DHCP server is not available, the module may be configured by jumper link to a fixed address to allow for initial programming operations. Once programmed, the device may have any IP address the user assigns.

Three Ports labeled Port A, Port B and Port C are for connection of the module to external digital devices operating at 5V levels. These ports are standard 10 way box headers with 0.1" pitch. Each of these ports has 8 I/O signals and the +5V and Ground lines from the module.   As these groups of 8 signals are treated as a single byte for the efficient use of software resources, it is best to connect devices of the same type to the same port. For Example, If you had LED illuminated switches then it would be recommended to connect your switches to Port A and your LED's to Port B, then to make the switches control the LED's you could simply read the value from Port A and write that value to Port B.

Non-volatile memory on the module can be programmed to allow the module to have a fixed, user assigned, IP address, to each of the ports to power up in a user defined state and/or to allow the module to automatically scan any of it's input lines at a user defined rate, filter the signals digitally and transmit a UDP/IP packet to report any valid input line changes to a user programmed destination address.

The Ether I/O 24 module has been designed and manufactured as a component of a larger system that may consist of the module combined with, your own input and/or output devices, any of our accessory boards connected to other components or modules, other 5V level operated modules and devices by direct connection.

The Ether I/O and all of the accessory modules are built to a 72 mm standard width thereby allowing their easy mounting to a DIN rail by means of commonly available DIN rail mounting hardware. The modules also have mounting holes to allow 3mm or 1/8" mounting hardware to be used. Care should be taken so as not to damage the printed circuit boards when mounting the modules.

Functional Description

The Ether I/O 24 is an integrated, micro-controller based network interface board with 24 digital user I/O lines and an on board switch mode voltage regulator. The module's firmware and hardware enable your devices or other modules to be connected to a generic Ethernet network and controlled or sensed using industry standard protocols. Each of the 24 User I/O lines operates at 5V DC maximum levels and can be independently programmed as, an Input whose state can be remotely sensed via another network device, an Input whose state is internally checked and transmitted when a change occurs, or an output whose state can be remotely controlled by another networked device.

The IP Address of the module is determined either automatically by BOOTP protocol from a DHCP server or is programmed by the user to be at a fixed address. The MAC or Ethernet address of the module is factory programmed and cannot be altered by the user.

Onboard firmware reads the user configuration stored on the module's onboard non-volatile memory and sets the ports to a user configured state at power up. If un-configured, all ports will be set as inputs with the input thresholds set at TTL levels and the pull-up function disabled.

If configured to do so, the module periodically scans any or all of the digital inputs, filters any changes to remove noise and signals a remote unit of the changes. The scan rate can be set from 1 millisecond to 65.5 seconds and the filter can be set to discard any number of unstable readings from 1 up to 255. Each of the 24 signal lines has an independent control bit that controls whether the module detects changes on that line.   Filtering is done on a per port basis with each port filtered as a group. When any line is set as an output its state is not checked for changes.

If a PC controls the module, the Programmer must have access to an UDP/IP socket in order to communicate with the module. The Winsock control in MS Windows operating systems provides for such communication in a simple and easy manner. As other operating systems have different methods of programming network sockets, please consult your operating system's specifications, software and language manuals for details of how to open a UDP/IP socket to communicate with the module.

The programmer should note that the UDP/IP protocol requires an IP address and a Port Number to allow communication with the module. The Port number for communication with the module is 2424 decimal and this port number is used for all UDP/IP communications for module and port programming. Other port numbers are used for the ICMP and BOOTP protocols, however the programmer is advised not to use other ports unless they possess an extensive knowledge of these protocols.

The IP address of the module can be programmed to a fixed address by a windows PC running the ELEXOL Ether I/O 24 Test and Programming utility software (downloadable from the Elexol website). Alternately, the user may program the fixed IP address from their own software by using the EEPROM writing commands. If the Fixed IP address function is not used the module will have a dynamically assigned IP address from the DHCP server. To find the IP address of any module, broadcast a special message to port 2424 and each of the Ether I/O 24 modules will respond, stating their IP address, Ethernet address and firmware version number.

LED Functions

There are 2 LED indicator lights on the Ether I/O 24 module; their operation is as follows.

UPPER LED = NETWORK LINK/ACTIVITY. This LED is illuminated when the module is powered and the network interface has detected a connection. The LED will blink whenever there is activity on the network link.

LOWER LED = VALID COMMAND. This LED illuminates for 0.1 second each time the unit processes a valid command.   When the commands are arriving faster than 10 times per second the LED will be continuously illuminated.

Accessory Boards

In order to connect devices that are not 5V operated or require isolation, ELEXOL has pre-built accessory boards available. All accessory boards are equipped with box headers matching those on the Ether I/O 24 module and are supplied with 30cm long connecting cables. All external connections to the input or output channels of the accessory boards are by screw terminals that will accept cables $0.5 – 2$ mm^2.

Opto Input Board, 8 optically isolated input channels to accept signals from 3-24V AC or DC. Each channel of the board is electrically isolated from all other channels and the I/O module thereby preventing incoming signals from interfering with each other or damaging the I/O module. The board has a box header matching the connector of the I/O module and an indicator LED for each channel.

Relay Output Board, with both N/O and N/C contacts rated to 250V AC or DC at 5Amps. The 8 relays on the board can drive a multitude of devices. Each of the 8 relays on the board has a matching LED to indicate its state.

Connector / LED Board, to provide for easy connection of larger wires to the Ether I/O 24 module or to provide a buffered visual indication of the states of all the signals on a port. The Connector / LED Board can also be used simply as a mimic light board for a remote location or for system diagnosis and debugging.

Switch / Push Button Board, allowing user inputs to be mounted along side the Ether I/O 24 module or for diagnosis and debugging. The Switch / Push Button Board features 8 miniature pushbutton switches and 8 slide switches, with each channel having an indicator LED to show that the board is powered and that the switch or the button is activated.

50 Pin Connector Board, to allow connection of industry standard I/O racks with 50 pin connectors, this board will adapt the three 10 pin box headers on the Ether I/O 24 module to a single 50 pin header with industry standard pin configuration.

Please check our website for further details of other accessory boards that may be released from time to time.

Application Features

The Ether I/O 24 has many unique features that enable its use in a great many applications. This section will describe what makes the module ideal for many real world situations.

Why should I use Ethernet?

Ethernet is the most prolific, most quickly growing and most common network standard in the world. Over 50 million pieces of equipment are already installed on this network. Ethernet is most commonly carried over inexpensive CAT-5 UTP cable; for longer runs or where electrical interference is an issue, fiber optic cable and converters may be used. Electricians and communication system installers are very familiar with Ethernet cable systems and terminations, thus making it easy to find an experienced installer. Ethernet infrastructure devices such as switches, media converters and routers are readily available from numerous manufacturers and vendors.

Industrial Automation and Distributed I/O

High-speed inputs and outputs. When used with a switched Ethernet network, the module, with its high-speed inputs and outputs, provides a reliable and deterministic I/O platform. The latency from command receipt to response transmission is less than 500 microseconds under all circumstances. If short I/O packets are used, the latency will always be less than 200 microseconds. The high speed of the Ethernet interface to the control system will allow for 160,000 inputs to be read per second or 500,000 outputs to be controlled per second using off-the-shelf Ethernet switches and Ether I/O 24 modules. Due to the nature of modern switched Ethernet and full duplex communications the CSMA/CD system that plagued older, non-switched Ethernet networks with collisions, packet loss and indeterminate transfer times, no longer provides an obstacle to deterministic Ethernet communications.

AutoScan Mode. The AutoScan mode of the module allows changes of input signals to be sent to the host without the host having to poll the module. An onboard digital low pass filter allows the module to screen out noisy electrical signals or switch contact noise thereby removing this burden from the system controller. Using this mode also allows one Ether I/O 24 to be connected directly to another Ether I/O 24 module by an Ethernet or Internet link. The states of the ports on one module can be sent to the other module without the need of a host system. Using this mode and two or more Ether I/O 24 modules, several or even hundreds of signals can be sent from one location to another by low cost twisted pair cable up to 100 meters in length. By using readily available fiber-optic converters this distance can be extended dramatically.

Programmable Power-Up State. The module can be programmed to power up with all its ports to a programmed state, thus if a machine needs to have certain devices enabled at power up or if the machine designer desires all lamps to light in a lamp test, it is possible for the module to accomplish this before the main control system is active.

Industrial Automation and Distributed I/O (continued)

Fixed IP. Fixed IP addressing allows each module to be given a specific address that remains constant throughout the life of the machine thereby simplifying machine software design and allowing easy diagnosis of machine wiring faults.

Electrical Isolation. Electrical Isolation within the Ether I/O 24 module is provided by the transformer coupling of the data signals. This means that when the module is used with an Ethernet switch, a severe fault in one section of a machine resulting in high voltage being applied to the module is unlikely to damage other parts of the machine. Most common bus systems do not provide this level of isolation.

In industrial systems there is frequently a need for user controls with indicators to be placed at several points on a machine and the economical nature of the Ether I/O 24 and other Ethernet components make its use more affordable and easier to use than most other systems.

The star or multi linked star wiring of Ethernet makes for very easy fault diagnosis and the onboard LED indicators of the module allow the diagnostician to quickly ascertain whether a module is powered, active or dormant, and whether any Ethernet signals are reaching the module. As all Ethernet switches also have link and activity LED indicators, it is easy to see if any parts of the network are not communicating. This allows for quick diagnosis of any wiring faults in the Ethernet signaling or power distribution systems. As each module has its own serial number, the system's control software is able to verify that all parts of the network and all I/O modules are correctly communicating and notify the operator of the specific location of any malfunction. The low cost per I/O line of the module allows for the monitoring of non-signaling inputs such as power supply rails for advanced diagnostics.

Using Ethernet allows companies to utilize their existing IT infrastructure to convey control information, messages or process data to various points throughout their facilities. The economical nature of the Ether I/O 24 makes this an attractive alternative to other systems.

Home, Office and Building Automation, Distributed control and Internet Connectivity

Because of its low cost per I/O line, the Ether I/O 24 module ideally services the budget sensitive building automation market. Ethernet's low cost infrastructure, cabling and switching systems combined with the ability of the Ethernet network to service other systems within a building make it a single platform multi function network for delivering services throughout a building.

Ethernet networks can be used to deliver Voice Over IP telephony services, Internet and Local Area Network services, Video and Audio on demand as well as control and monitoring of devices or services within the building. With IP becoming the new standard for worldwide communication and most modern buildings being fitted with Ethernet cabling during construction, the Ether I/O 24 can economically share that infrastructure and optimize connectivity.

The Ether I/O 24 in conjunction with the accessory boards can be used to control most electrical devices. Future expansion of the accessory board range will allow additional devices to be serviced by the module.

The falling costs of PC computing power combined with the fact that most new PC hardware is equipped with an Ethernet port make the PC platform combined with Ethernet connectivity ideal for central control of most mid to large scale systems. When linked to the Internet, an automated building can be controlled and monitored from any Internet terminal; Internet enabled mobile phone or other device anywhere on the globe. Wireless connectivity will further enable the occupant to access their entire buildings network facilities without the burden of wires.

The low power consumption of the Ether I/O 24 module will enable the module to reduce the additional cost burden of building automation on the utility bill. In many cases an automated building may have a reduced utility bill through more efficient use of power. Load shedding during peak demand times will allow building designers to purchase energy at lower rates and electricity supply companies to make more efficient use of their infrastructure.

When connected to a router or if the main control PC has Internet access the system is able to request its own maintenance or repair when a malfunction occurs. When a unit has run its scheduled number of hours it can automatically contact the building supervisor or contracted maintenance organization to arrange for service. By automating these services, the chances of maintenance being overlooked and expensive malfunctions and downtime resulting will be minimized. When a fault does occur, downtime can be minimized by prompt, automatic notification to the correct people.

Digital I/O from PCs

Ethernet is now standard on almost all PCs making it an attractive option for the connection of I/O devices. In the past, using Ethernet for I/O has been too expensive or complicated. The Ether I/O 24 now changes that situation with the cost of Ethernet I/O now similar to that of USB solutions. The added advantages of longer cable length, an almost unlimited numbers of nodes, electrical isolation and Internet connectivity, make Ethernet a very attractive option for I/O connection.

No matter what you are connecting, if the requirement is for medium to large numbers of I/O connections the Ether I/O 24 is an ideal solution. Additional modules enable the Ether I/O 24 module to connect to almost any electrical device, and with appropriate sensors or actuators, it can be used to control or monitor almost any simple or complex system.

With wireless Ethernet now coming as standard on many laptop computers and wireless access points becoming more economical, the Ether I/O 24 can be linked wirelessly to a laptop computer by the same hardware that allows that laptop to connect to the Internet or other network services. By connecting the Ether I/O 24 to a wireless network access point, any device that requires monitoring by the network can be connected wirelessly.

Whether you are connecting a special function keyboard, indicator lights or a coffee machine, the Ether I/O 24 will connect your device to your network and from there to your local PC or by Internet, to the world.

Network Enabling your PC linked machine or equipment

If you're designing a new PC linked machine or piece of equipment, imagine the end users delight at finding an Ethernet connection on the back that enables them to place your new device anywhere in their facility that the network reaches and control or monitor it from right from their own PC wherever it is located. Multiple users may even be able to share the device or have scheduled or queued access to it.

The Ether I/O 24 has been designed for easy integration into other equipment. By using the Ether I/O 24 module as the main interface board in your machine, all control functions can be offloaded to a host PC, thus eliminating the need to develop and program an embedded machine controller. The high speed of modern PCs allows complex machine functions to be handled by the PC and the easy software development environment of the PC enables more sophisticated machine operations and user interfaces to be provided without the high development costs and unit costs of embedded controllers. Your new devices' software can now be easily updated or maintained via the web. If a fault were to occur you could access all the machine's functions remotely via the Internet for instant diagnosis.

If higher speed or real time functions are required these can be implemented on your own boards with connection back to the Ether I/O module for communication back to the controlling PC. The Ether I/O 24 can easily communicate with your own boards by synchronous serial or clocked parallel interface whereby a single packet from the host PC can send up to 250 bytes of data to your board.

The Ether I/O 24 has been designed and physically laid out to simplify the task of users wishing to incorporate the Ether I/O 24 as the principal interface element in their product. The Ethernet and power connectors as well as the status LED indicators are aligned down one edge of the board and the internal power outlets and I/O connectors aligned down the adjacent edges. The I/O ports use low cost IDC connection cables and the board provides 500mA at 5V for your own circuitry as well as access to the main incoming power feed for higher power devices. If your device has an internal power supply the board can be fed from inside and the external connector used for powering ancillary external devices or an Ethernet switch.

Differences between Ether I/O 24 and Ether I/O 24R

The Ether I/O 24 R is the next version of the Ether I/O 24.

The End of life notice on the SX52 due to Non RoHS compliance / Packaging issues has forced Elexol to update the Ether I/O 24 to incorporate the Parallax SX48.

The difference between the two versions are as follows
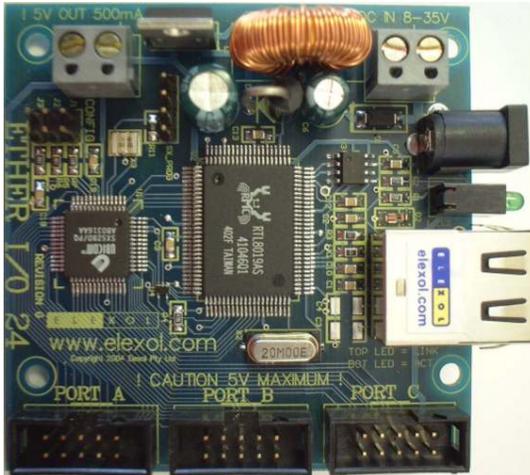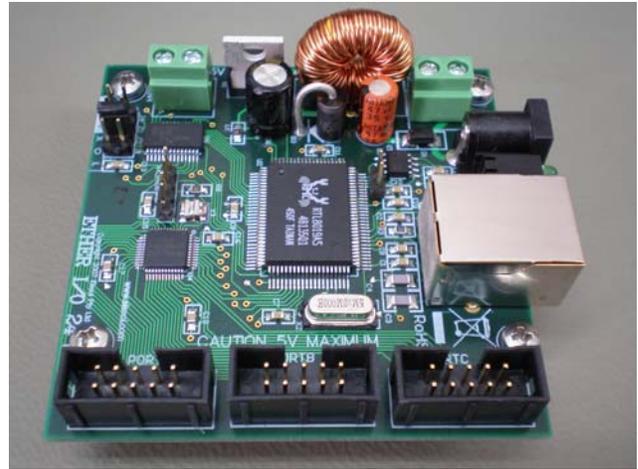
<div align="center">Ether I/O 24   Ether I/O 24R</div>

Ether I/O 24

- Main Controller Ubicom SX52 (52 pin TQFP) End of Life due to Non RoHS
- Mechanicals 72mm x 72mm
- Not RoHS

Ether I/O 24R

Hardware Changes

- Main Controller Parallax SX48 (48 pin TQFP) RoHS Compliant
- Mechanical dimensions identical to Ether I/O 24
- New jumper for fixed IP address of 192.168.0.10
- RoHS Compliant

Firmware Changes

- Added Programmable port number
- Added Programmable subnet mask and gateway address
- Added SPI on PORT A (ability to send multiple bytes)
- Changes in Autoscan feature to perform ARP
- The ' , * and % commands are now redundant.

Electrical Characteristics

Absolute Maximum Ratings

Warning! Exceeding these ratings may cause irreparable damage to the unit.

Parameter	Absolute Maximum Conditions
Storage Temperature	-65°C to +150°C
Ambient Temperature (Power Applied)	0°C to + 70°C
Power Input Voltage	-1.0v to +35.00v DC
DC Input Voltage – Inputs	-0.6v to +5.6v
DC Output Current – Outputs	45mA
DC Output Current – Total all outputs	210mA
Maximum DC current into an input pin	±500uA
Power Dissipation	3W

DC Characteristics (Temperature = 25°C, Power = 24VDC)

Parameter	Conditions	Min	Typ	Max	Units
DC Input		8		35	V
Power Consumption			1.1		W
Humidity Range		0		85	%RH
Logic Low					
TTL		0		0.8	V
CMOS		0		1.5	V
Schmitt Trigger		0		0.75	V
Logic High					
TTL		2.0		5.0	V
CMOS		3.5		5.0	V
Schmitt Trigger		4.25		5.0	V
Input Leakage Current	Vin = 0V or 5V	-3.0		+3.0	µA
Pull-Up Current		200	400	600	µA
Output High Voltage	Load = 14mA	4.3			V
Output Low Voltage	Load = 25mA			0.6	V

Packet Timing Characteristics

Parameter	Conditions	Min	Typ	Max	Units
Response Time	1 Read Command in Packet	80	100	200	µS
Port Write Time	Within one packet		1.5		µS
Port Write Speed	250 writes per packet		500,000		Writes/S
Port Write Time	1 write per packet		100		µS
Port Write Speed	1 write command per packet		10,000		Writes/S
Port Read Speed	1 read per packet		6,500		Reads/S
Port Read Speed	32 reads per packet		160,000		Reads/S

Module Connections

Module I/O Connectors (PORT A, PORT B, PORT C)

All three of the I/O connectors are wired the same

I/O Port Pin Diagram

+5v User Power	I/O Bit 7 (128)
I/O Bit 6 (64)	I/O Bit 5 (32)
I/O Bit 4 (16)	I/O Bit 3 (8)
I/O Bit 2 (4)	I/O Bit 1 (2)
I/O Bit 0 (1)	Ground
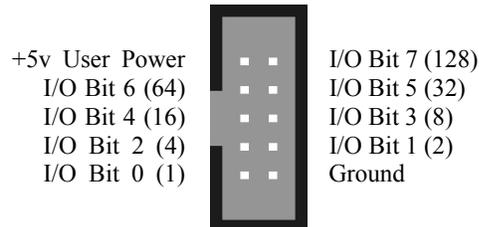
PIN #	SIGNAL	TYPE	DESCRIPTION
1	+5v User	PWR	+5 Power supply out for user circuits
2	I/O 7	I/O	Programmable I/O pin with bit value of 128 or $80
3	I/O 6	I/O	Programmable I/O pin with bit value of 64 or $40
4	I/O 5	I/O	Programmable I/O pin with bit value of 32 or $20
5	I/O 4	I/O	Programmable I/O pin with bit value of 16 or $10
6	I/O 3	I/O	Programmable I/O pin with bit value of 8 or $08
7	I/O 2	I/O	Programmable I/O pin with bit value of 4 or $04
8	I/O 1	I/O	Programmable I/O pin with bit value of 2 or $02
9	I/O 0	I/O	Programmable I/O pin with bit value of 1 or $01
10	Ground	PWR	Ground Pin / Common reference for all signals

Module Configuration Options Connector (CONFIG J1, J2, J3)

The CONFIG connector is a 6 pin arranged as 2 rows of 3 pins. To set an option jumper place one of the 2 supplied links across the 2 rows next to the corresponding label on the circuit board overlay.

JUMPER#	Function
J1	ON Lock EEPROM, OFF allows EEPROM writes
J2	ON DHCP, OFF LOAD IP FROM EEPROM IF PROGRAMMED else DHCP
J3	Fix IP – The module will operate at fixed IP address 10.10.10.10
J4	Fix IP – The module will operate at fixed IP address 192.168.0.10

Jumper J1, when placed will lock the EEPROM memory from any write or erase procedures.

Jumper J2 when placed, will cause the module to power up ignoring the user settings in the EEPROM; DHCP will be enabled and port will be set to 2424. When not placed the user settings will be loaded from the EEPROM if the fixed IP address is enabled this IP address will be loaded else the unit will DHCP and load the port number programmed into the EEPROM. The unit is programmed with default Port Number 2424.

Jumper J3, when placed will disable DHCP and override the EEPROM IP address settings and force the module to used IP address 10.10.10.10 with Port 2424.

Jumper J4, when placed will disable DHCP and override the EEPROM IP address settings and force the module to used IP address 192.168.0.10 with Port 2424.

Module Connections (continued)

Ether I/O 24 Factory Programming Header (SX_PROG)

This header is used for factory programming, **DO NOT USE** this connector for any other purpose.

Power Connectors

The Module has 2 types of power in connector; these connectors are connected directly to each other. The 2.1mm DC jack is mounted overhanging the boards edge in order that the module can be mounted with the network connector, power connector and status LEDs protruding through a case. The second power connector or 5V from the module can then be used to power your own circuitry. If the module is not mounted through a panel either or both of the power in connectors may be used.

5V Out Screw Terminal Block (optional)

This terminal block, if fitted allows for the 5v supply from the onboard regulator to be used to power user circuits and sensors. The maximum current that user circuits may draw from the on board regulator is 500mA; if this current is exceeded then the operation of the board may be adversely affected.

Ethernet Connector

The module is equipped with a standard RJ45 network socket and conforms to the 10 Base-T standard. Only 4 of the 8 wires are used for network interface, 2 as a pair for data sent from the module and 2 as a pair for data being received by the module the other wires are unused at this time.
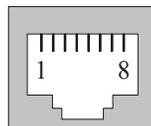
Ethernet Connector

Pin #	Name	Description
1	TXD +	Transmit Data Positive Signal
2	TXD -	Transmit Data Negative Signal
3	RXD +	Receive Data Positive Signal
6	RXD -	Receive Data Negative Signal

Pins 4, 5, 7 and 8 not used

Command Summary

For ease of use we have broken the command set into subgroups based on their function. All commands are shown as their ASCII characters, text in *italics* represent binary 1 byte values.

Values that pertain to port Input, Output or control are shown as *data*, values that pertain to address information are shown as *address* and values that represent 16-bit information are shown as *MSB* and *LSB*. When a byte is required as padding or for future use it is shown as *dummy*. Where a specific byte is required its hexadecimal value is shown with a dollar sign thus *$55*.

The spaces shown are only for clarity and no actual spaces are used in commands sent to the module.

Example Code

All example code is written in Visual Basic with the Winsock control named Winsock1. To execute these commands you will need to Place a Winsock component on your form and set the following option.

Winsock1.Protocol = sckUDPProtocol

Other options will be discussed further in the text

The I/O lines are accessed as 3 ports and each line is controlled by its bit value within the data byte.

Port I/O Commands

Function	Command	Reply
Write Port A	A *data*	
Write Port B	B *data*	
Write Port C	C *data*	
Read Port A	a	A *data*
Read Port B	b	B *data*
Read Port C	c	C *data*

Port Configuration Commands

For each of the three I/O ports there are 4 commands used to set the ports' options. First and most critical of these options is Direction, which can be set as input or output. When set as output, the I/O line will be driven to the last value written to the port. This value can be pre set by writing to the port before writing to the direction register. When set as an output, none of the other configuration commands have any effect. The Pull Up configuration command applies to those lines that are set as inputs, writing a 0 to the corresponding bit applies a pull up resistor to the line so that if it is not driven low it will be pulled to a known high state, this is very useful if sensing contact closures or open collector outputs. The threshold function sets the threshold at which a line reads as high or low. When the corresponding bit is set as 1 then the threshold is set at 1.4V and any voltage above this reads as a high level. When the corresponding threshold bit is set to 0 the threshold is set at 2.5V and any voltage above this reads as a high level. Last of the configuration commands allows the port to read as Schmitt trigger inputs which means that the input line is compared to 2 voltages, 0.75V and 4.25V. When the line's voltage drops below 0.75V it will read as a low until the line's voltage rises above 4.25V at which time the line will read as a high. When the lines' voltage is in between 0.75V and 4.25V, the value will remain stable at its previous level. To enable the Schmitt trigger on any input a 0 must be written to the corresponding bit.

Function	Command		Function	Command
Write Direction A	!A *data*		Write Pull Up A	@A *data*
Write Direction B	!B *data*		Write Pull Up B	@B *data*
Write Direction C	!C *data*		Write Pull Up C	@C *data*

Command Summary (continued)

Write Threshold A	#A *data*	Write Schmitt A	$A *data*
Write Threshold B	#B *data*	Write Schmitt B	$B *data*
Write Threshold C	#C *data*	Write Schmitt C	$C *data*

The EEPROM Reading and Programming Commands

All EEPROM commands must be sent as a single packet with 5 bytes, the module will ignore packets with a size other than 5 bytes.

Read EEPROM word	'R *address dummy dummy*
Write Enable EEPROM	'1 *address $AA $55*
Write Disable EEPROM	'0 *address dummy dummy*
Write EEPROM word	'W *address MSB LSB*
Erase EEPROM word	'E *address $AA $55*

Only the Read EEPROM command generates a response in the form of 'R *address MSB LSB*

A special EEPROM command is used to reboot the module and cause it to load and activate any new settings.

Reboot Module   '@ *dummy $AA $55*

Identification and Information Commands

The module will always respond to a packet containing IO24, 4 bytes in length sent to port 2424.
The response contains the module's six-byte MAC address and a two-byte firmware version number.

Command Set Quick Reference

Table 1, Module Command Set

Command ASCII	Command Hex	Bytes	Data	Function	Response Bytes	Response Identifier	Response Data
A	$41	2	Port_Value	Write Port A	-	-	-
B	$42	2	Port_Value	Write Port B	-	-	-
C	$43	2	Port_Value	Write Port C	-	-	-
a	$61	1	-	Read Port A	2	A	PortA_Value
b	$62	1	-	Read Port B	2	B	PortB_Value
c	$63	1	-	Read Port C	2	C	PortC_Value
!A	$21 $41	3	Direction	Write Port A Direction Register	-	-	-
!B	$21 $42	3	Direction	Write Port B Direction Register	-	-	-
!C	$21 $43	3	Direction	Write Port C Direction Register	-	-	-
IO24	$49 $4F $32 $34	4	-	Identify IO24 Units	12	IO24	6 Byte MAC Address 2 Byte Firmware Version
!a	$21 $61	2	-	Read Port A Direction Register	3	!A	Direction
!b	$21 $62	2	-	Read Port B Direction Register	3	!B	Direction
!c	$21 $63	2	-	Read Port C Direction Register	3	!C	Direction
@A	$40 $41	3	Pull_Up	Write Port A Pull Up Register	-	-	-
@B	$40 $42	3	Pull_Up	Write Port B Pull Up Register	-	-	-
@C	$40 $43	3	Pull_Up	Write Port C Pull Up Register	-	-	-
#A	$23 $41	3	Threshold	Write Port A Threshold Register	-	-	-
#B	$23 $42	3	Threshold	Write Port B Threshold Register	-	-	-
#C	$23 $43	3	Threshold	Write Port C Threshold Register	-	-	-
$A	$24 $41	3	Schmitt	Write Port A Schmitt Trigger Register	-	-	-
$B	$24 $42	3	Schmitt	Write Port B Schmitt Trigger Register	-	-	-
$C	$24 $43	3	Schmitt	Write Port C Schmitt Trigger Register	-	-	-
@a	$40 $61	2	-	Read Port A Pull Up Register	3	@A	Pull_Up
@b	$40 $62	2	-	Read Port B Pull Up Register	3	@B	Pull_Up
@c	$40 $63	2	-	Read Port C Pull Up Register	3	@C	Pull_Up
#a	$23 $61	2	-	Read Port A Threshold Register	3	#A	Threshold
#b	$23 $62	2	-	Read Port B Threshold Register	3	#B	Threshold
#c	$23 $63	2	-	Read Port C Threshold Register	3	#C	Threshold
$a	$24 $61	2	-	Read Port A Schmitt Trigger Register	3	$A	Schmitt
$b	$24 $62	2	-	Read Port B Schmitt Trigger Register	3	$B	Schmitt
$c	$24 $63	2	-	Read Port C Schmitt Trigger Register	3	$C	Schmitt
'R	$27 $52	5	Address NU NU	Read EEPROM Word	4	R	Address MSB LSB
'W	$27 $57	5	Address MSB LSB	Write EEPROM Word	-	-	-
'E	$27 $45	5	Address $AA $55	Erase EEPROM Word	-	-	-
'0	$27 $30	5	NU NU NU	Write Disable EEPROM	-	-	-
'1	$27 $31	5	NU $AA $55	Write Enable EEPROM	-	-	-
'@	$27 $40	5	NU $AA $55	Reset Module	-	-	-
S1A	$53 $31 $41	3	-	Sets up Port A for SPI, sets bits 0-3 to appropriate directions for SPI	3	S1A	-
S0A	$53$30$41	3	-	Disables SPI on Port A, restores direction values and port values to what they were before an S1A command	3	S0A	
SAXX	$53$41	X No. of bytes	X Data Bytes	Sends out the data bytes on Port A via SPI. The number of response bytes is dependent on the number of bytes sent.		SAXX	X No. of bytes + Response Bytes

Bytes values include all Commands and Data sent in the packet
NU represents a value that is *Not Used;* a dummy byte must be included to ensure correct operation
- Means that there is no data or no response, do not insert data bytes
All hex values represented by $xx represent a single byte having this value

Command Set

Write Port A

ASCII Code	Bytes	Data	Function
A	2	*Port-Value*	Writes data to ports output lines. A bit value of 1 sets the corresponding line high and a 0 sets it low

The power up default value for this port is 0

Operation: This command affects any of the eight lines of port A that are set as outputs. The port value is written to the entire port with each of the values bits affecting the corresponding I/O line. To change a single I/O line without affecting the others it is required to store the old value of the port or read its current value before writing a new value with only the corresponding bits changed. This command does not affect any I/O lines that are set as Inputs.

Example: Winsock1.SendData "A" + Chr$(*Value*)

Write Port B

ASCII Code	Bytes	Data	Function
B	2	*Port-Value*	Writes data to ports output lines. A bit value of 1 sets the corresponding line high and a 0 sets it low

The power up default value for this port is 0

Operation: Same as Write Port A
Example: Winsock1.SendData "B" + Chr$(*Value*)

Write Port C

ASCII Code	Bytes	Data	Function
C	2	*Port-Value*	Writes data to ports output lines. A bit value of 1 sets the corresponding line high and a 0 sets it low

The power up default value for this port is 0

Operation: Same as Write Port A
Example: Winsock1.SendData "C" + Chr$(*Value*)

Write Port A Direction Register

ASCII Code	Bytes	Data	Function
!A	3	*Direction*	Writes data to port's direction register. Lines with a corresponding bit value of 0 are set as outputs, lines with a bit value of 1 are set as inputs

The power up default for Direction is 255 setting all lines as inputs

Operation: This command affects all eight lines of port A. The Direction value is written to the entire port with each of the bits in the byte affecting the corresponding I/O line. To change a single I/O line without affecting the others it is necessary to store the old value of the port or read its current value before writing a new value with only the corresponding bits changed. To set the entire port as outputs use Direction = 0 to set all as inputs use Direction = 255 to set 0, 1, 2 and 3 as inputs and 4, 5, 6 and 7 as outputs use Direction = 15.

Example: Winsock1.SendData "!A" + Chr$(*Direction*)

Command Set (continued)

Write Port B Direction Register

ASCII Code	Bytes	Data	Function
!B	3	*Direction*	Writes data to ports direction register. Lines with a corresponding bit value of 0 are set as outputs, lines with a bit value of 1 are set as inputs

The power up default for Direction is 255 setting all lines as inputs

Operation: Same as Write Port A Direction Register
Example: Winsock1.SendData "!B" + Chr$(*Direction*)

Write Port C Direction Register

ASCII Code	Bytes	Data	Function
!C	3	*Direction*	Writes data to ports direction register. Lines with a corresponding bit value of 0 are set as outputs, lines with a bit value of 1 are set as inputs

The power up default for Direction is 255 setting all lines as inputs

Operation: Same as Write Port A Direction Register
Example: Winsock1.SendData "!C" + Chr$(*Direction*)

Write Port A Pull Up Register

ASCII Code	Bytes	Data	Function
@A	3	*Enable*	Writes data to port's direction register. Lines with a corresponding bit value of 0 have their pull up resistors turned on, lines with a bit value of 1 have their pull up resistors turned off

The power up default for Enable is 255; all pull up resistors turned off

Operation: This command affects all eight lines of port A whose direction is set as an input. The Enable Value is written to the entire port with each of the bits in the byte affecting the corresponding I/O line. To change a single I/O line's behaviour without affecting the others it is necessary to store the old value of the register or read its current value before writing a new value with only the corresponding bits changed. To set the entire port with pull up resistors turned on use Enable = 0 to turn all the pull up resistors off use Enable = 255 to set 0, 1, 2 and 3 as on and 4, 5, 6 and 7 as off use Enable = 240.
Example: Winsock1.SendData "@A" + Chr$(*Enable*)

Write Port B Pull Up Register

ASCII Code	Bytes	Data	Function
@B	3	*Enable*	Writes data to port's direction register. Lines with a corresponding bit value of 0 have their pull up resistors turned on, lines with a bit value of 1 have their pull up resistors turned off

The power up default for Enable is 255; all pull up resistors turned off

Operation: Same as Write Port A Pull Up Register
Example: Winsock1.SendData "@B" + Chr$(*Enable*)

Command Set (continued)

Write Port C Pull Up Register

ASCII Code	Bytes	Data	Function
@C	3	*Enable*	Writes data to port's direction register. Lines with a corresponding bit value of 0 have their pull up resistors turned on, lines with a bit value of 1 have their pull up resistors turned off

The power up default for Enable is 255; all pull up resistors turned off

Operation: Same as Write Port A Pull Up Register
Example: Winsock1.SendData "@C" + Chr$(*Enable*)

Write Port A Threshold Register

ASCII Code	Bytes	Data	Function
#A	3	*Select*	Writes data to port's Threshold register. Lines with a corresponding bit value of 0 have input threshold voltage set at 2.5V, lines with a bit value of 1 have their input threshold voltage set to 1.4V

The power up default for Select is 255; all lines have a threshold of 1.4V

Operation: This command affects all eight lines of port A whose direction is set as an input. The Select value is written to the entire port with each of the bits in the byte affecting the corresponding I/O line. To change a single I/O line's behaviour without affecting the others it is necessary to store the old value of the register or read its current value before writing a new value with only the corresponding bits changed. To set the entire port with threshold voltage of 2.5V use Select = 0 to set all the ports input thresholds at 1.4V use Select = 255 to set 0, 1, 2 and 3 at 1.4V and 4, 5, 6 and 7 at 2.5V use Select = 15.
Example: Winsock1.SendData "#A" + Chr$(*Select*)

Write Port B Threshold Register

ASCII Code	Bytes	Data	Function
#B	3	*Select*	Writes data to port's Threshold register. Lines with a corresponding bit value of 0 have input threshold voltage set at 2.5V, lines with bit value of 1 have their input threshold voltage set to 1.4V

The power up default for Select is 255; all lines have a threshold of 1.4V

Operation: Same as Write Port A Threshold Register
Example: Winsock1.SendData "#B" + Chr$(*Select*)

Write Port C Threshold Register

ASCII Code	Bytes	Data	Function
#C	3	*Select*	Writes data to port's Threshold register. Lines with a corresponding bit value of 0 have input threshold voltage set at 2.5V, lines with a bit value of 1 have their input threshold voltage set to 1.4V

The power up default for Select is 255; all lines have a threshold of 1.4V

Operation: Same as Write Port A Threshold Register
Example: Winsock1.SendData "#C" + Chr$(*Select*)

Command Set (continued)

Write Port A Schmitt Trigger Register

ASCII Code	Bytes	Data	Function
$A	3	*Enable*	Writes data to port's Schmitt trigger enable register. Lines with a corresponding bit value of 0 have their Schmitt trigger threshold latches turned on, lines with a bit value of 1 have normal threshold sense inputs

The power up default for Enable is 255; all line Schmitt triggers are disabled

Operation: This command affects all eight lines of port A whose direction is set as an input. The Enable value is written to the entire port with each of the bits in the byte affecting the corresponding I/O line. To change a single I/O line's behaviour without affecting the other lines it is necessary to store the old value of the register or read its current value before writing a new value with only the corresponding bits changed. To set the entire port with Schmitt trigger inputs turned on use Enable = 0 to turn all the Schmitt trigger inputs off use Enable = 255 and to set 0, 1, 2 and 3 as Schmitt trigger inputs and 4, 5, 6 and 7 as normal threshold inputs use Enable = 240.

Example: Winsock1.SendData "$A" + Chr$(*Enable*)

Write Port B Schmitt Trigger Register

ASCII Code	Bytes	Data	Function
$B	3	*Enable*	Writes data to port's Schmitt trigger enable register. Lines with a corresponding bit value of 0 have their Schmitt trigger threshold latches turned on, lines with a bit value of 1 have normal threshold sense inputs

The power up default for Enable is 255; all line Schmitt triggers are disabled

Operation: Same as Write Port A Schmitt Trigger Register
Example: Winsock1.SendData "$B" + Chr$(*Enable*)

Write Port C Schmitt Trigger Register

ASCII Code	Bytes	Data	Function
$C	3	*Enable*	Writes data to port's Schmitt trigger enable register. Lines with a corresponding bit value of 0 have their Schmitt trigger threshold latches turned on, lines with a bit value of 1 have normal threshold sense inputs

The power up default for Enable is 255; all line Schmitt triggers are disabled

Operation: Same as Write Port A Schmitt Trigger Register
Example: Winsock1.SendData "$C" + Chr$(*Enable*)

Read Port A

Command Sent	ASCII Code	Bytes	Data	Function
	a	1	-	Sends the Value of Port A back to the host
Command Reply	ASCII Code	Bytes	Data	
	A	2	*Port-Value*	

Operation: The Value of the 8 lines of Port A is read and sent back to the host. Those pins that are set as outputs are read as though they were inputs and their values sent back in the Port Value Byte.
Example: Winsock1.SendData "a"

Command Set (continued)

Read Port B

Command Sent	ASCII Code	Bytes	Data	Function
	b	1	-	Sends the Value of Port B back to the host
Command Reply	ASCII Code	Bytes	Data	
	B	2	*Port-Value*	

Operation: The Value of the 8 lines of Port B is read and sent back to the host. Those pins that are set as outputs are read as though they were inputs and their values sent back in the Port Value Byte.

Example: Winsock1.SendData "b"

Read Port C

Command Sent	ASCII Code	Bytes	Data	Function
	c	1	-	Sends the Value of Port C back to the host
Command Reply	ASCII Code	Bytes	Data	
	C	2	*Port-Value*	

Operation: The Value of the 8 lines of Port C is read and sent back to the host. Those pins that are set as outputs are read as though they were inputs and their values sent back in the Port Value Byte.

Example: Winsock1.SendData "c"

Read Port A Direction Register

Command Sent	ASCII Code	Bytes	Data	Function
	!a	2	-	Sends the Direction Register value back to the host
Command Reply	ASCII Code	Bytes	Data	
	!A	3	*Register-Value*	

Operation: The Direction Register is read and it's value sent back to the host.
Example: Winsock1.SendData "!a"

Read Port B Direction Register

Command Sent	ASCII Code	Bytes	Data	Function
	!b	2	-	Sends the Direction Register value back to the host
Command Reply	ASCII Code	Bytes	Data	
	!B	3	*Register-Value*	

Operation: The Direction Register is read and it's value sent back to the host.
Example: Winsock1.SendData "!b"

Command Set (continued)

Read Port C Direction Register

Command Sent	ASCII Code	Bytes	Data	Function
	!c	2	-	Sends the Direction Register value back to the host
Command Reply	ASCII Code	Bytes	Data	
	!C	3	*Register-Value*	

Operation: The Direction Register is read and it's value sent back to the host.
Example: Winsock1.SendData "!c"

Read Port A Pull Up Register

Command Sent	ASCII Code	Bytes	Data	Function
	@a	2	-	Sends the Pull Up Register value back to the host
Command Reply	ASCII Code	Bytes	Data	
	@A	3	*Register-Value*	

Operation: The Pull Up Register is read and it's value sent back to the host.
Example: Winsock1.SendData "@a"

Read Port B Pull Up Register

Command Sent	ASCII Code	Bytes	Data	Function
	@b	2	-	Sends the Pull Up Register value back to the host
Command Reply	ASCII Code	Bytes	Data	
	@B	3	*Register-Value*	

Operation: The Pull Up Register is read and it's value sent back to the host.
Example: Winsock1.SendData "@b"

Read Port C Pull Up Register

Command Sent	ASCII Code	Bytes	Data	Function
	@c	2	-	Sends the Pull Up Register value back to the host
Command Reply	ASCII Code	Bytes	Data	
	@C	3	*Register-Value*	

Operation: The Pull Up Register is read and it's value sent back to the host.
Example: Winsock1.SendData "@c"

Command Set (continued)

Read Port A Threshold Up Register

Command Sent	ASCII Code	Bytes	Data	Function
	#a	2	-	Sends the Threshold Register value back to the host
Command Reply	ASCII Code	Bytes	Data	
	#A	3	*Register-Value*	

Operation: The Threshold Register is read and it's value sent back to the host.
Example: Winsock1.SendData "#a"

Read Port B Threshold Up Register

Command Sent	ASCII Code	Bytes	Data	Function
	#b	2	-	Sends the Threshold Register value back to the host
Command Reply	ASCII Code	Bytes	Data	
	#B	3	*Register-Value*	

Operation: The Threshold Register is read and it's value sent back to the host.
Example: Winsock1.SendData "#b"

Read Port C Threshold Up Register

Command Sent	ASCII Code	Bytes	Data	Function
	#c	2	-	Sends the Threshold Register value back to the host
Command Reply	ASCII Code	Bytes	Data	
	#C	3	*Register-Value*	

Operation: The Threshold Register is read and it's value sent back to the host.
Example: Winsock1.SendData "#c"

Read Port A Schmitt Trigger Register

Command Sent	ASCII Code	Bytes	Data	Function
	$a	2	-	Sends the Schmitt Trigger Register value back to the host
Command Reply	ASCII Code	Bytes	Data	
	$A	3	*Register-Value*	

Operation: The Schmitt Trigger Register is read and it's value sent back to the host.
Example: Winsock1.SendData "$a"

Command Set (continued)

Read Port B Schmitt Trigger Register

Command Sent	ASCII Code	Bytes	Data	Function
	$b	2	-	Sends the Schmitt Trigger Register value back to the host
Command Reply	ASCII Code	Bytes	Data	
	$B	3	*Register-Value*	

Operation: The Schmitt Trigger Register is read and it's value sent back to the host.
Example: Winsock1.SendData "$b"

Read Port C Schmitt Trigger Register

Command Sent	ASCII Code	Bytes	Data	Function
	$c	2	-	Sends the Schmitt Trigger Register value back to the host
Command Reply	ASCII Code	Bytes	Data	
	$C	3	*Register-Value*	

Operation: The Schmitt Trigger Register is read and it's value sent back to the host.
Example: Winsock1.SendData "$c"

Identify Ether IO 24 Units

Command Sent	ASCII Code	Bytes	Data	Function
	IO24	4	-	Used to identify/find modules on the network and send specific module information (MAC address)
Command Reply	ASCII Code	Bytes		
			Data	
	IO24	12	6 Bytes being the Modules' MAC Address 2 Bytes being the Modules' Firmware Version	

Operation: This Operation is used to find modules on the network as the module will respond to this command when broadcast. When this command is received, the Module's Information is sent back to the host, the module's IP Address can be obtained from the Source Address of the packet.
Example: Winsock1.SendData "IO24"

Command Set (continued)

Read EEPROM Word

Command Sent	ASCII Code	Bytes	Data	Function
	'R	5	*Address NU NU*	The EEPROM data at *Address* is read and sent back to the host
Command Reply	ASCII Code	Bytes	Data	
	R	4	*Address MSB LSB*	

Operation: The module will read the EEPROM memory at the specified address and send a packet back to the host containing this data.

Special Conditions: All EEPROM functions must be sent as a single 5-byte packet.

Example: Winsock1.SendData "'R" + Chr$(*Address*) + Chr$(0) + Chr$(0)

Reset Module

ASCII Code	Bytes	Data	Function
'@	5	*NU $AA $55*	The Module Resets and reads the EEPROM

Operation: The module reset operation causes all the ports to be set to all inputs or as set up in the EEPROM and all EEPROM settings to be read and activated. When reset command is sent to the unit a 10ms delay should be allowed before sending any other commands to the unit.

Special Conditions: This function must be sent as a single 5-byte packet.

Example: Winsock1.SendData "'@" + Chr$(0) + Chr$(&HAA) + Chr$(&H55)

Write EEPROM Word

ASCII Code	Bytes	Data	Function
'W	5	*Address MSB LSB*	The EEPROM is written with the data *MSB* and *LSB* at the specified *Address*

Operation: The module will write the EEPROM memory at the specified Address with the data contained in the MSB and LSB bytes.

Special Conditions: All EEPROM functions must be sent as a single 5-byte packet. A Write Enable command must be sent before any Write or Erase commands can be performed. The user cannot write addresses 0-4 at any time. The EEPROM cannot be written or erased if the J2 jumper is on.

Example: Winsock1.SendData "'W" + Chr$(*Address*) + Chr$(*MSB*) + Chr$(*LSB*)

Erase EEPROM Word

ASCII Code	Bytes	Data	Function
'E	5	*Address $AA $55*	The EEPROM memory at *Address* is erased

Operation: The module will erase the EEPROM memory at the specified Address.

Special Conditions: All EEPROM functions must be sent as a single 5-byte packet. A Write Enable command must be sent before any Write or Erase commands can be performed. The user cannot write addresses 0-4 at any time. The EEPROM cannot be written or erased if the J2 jumper is on.

Example: Winsock1.SendData "'E" + Chr$(*Address*) + Chr$(&HAA) + Chr$(&H55)

Write Enable EEPROM

ASCII Code	Bytes	Data	Function
'1	5	*NU $AA $55*	The EEPROM memory is Write Enabled

Operation: The module will Write Enable the EEPROM memory allowing Write or Erase Operation to be performed.

Special Conditions: All EEPROM functions must be sent as a single 5-byte packet. A Write Enable command must be sent before any Write or Erase commands can be performed. The user cannot write addresses 0-4 at any time. The EEPROM cannot be written or erased if the J2 jumper is on.

Example: Winsock1.SendData "'1" + Chr$(0) + Chr$(&HAA) + Chr$(&H55)

Command Set (continued)

Write Disable EEPROM

ASCII Code	Bytes	Data	Function
'0	5	*NU NU NU*	The EEPROM memory is Write Disabled

Operation: The module will Write Disable the EEPROM memory preventing any Write or Erase Operations.

Special Conditions: All EEPROM functions must be sent as a single 5-byte packet. A Write Enable command must be sent before any Write or Erase commands can be performed. The user cannot write addresses 0-4 at any time. The EEPROM cannot be written or erased if the J2 jumper is on.

Example: Winsock1.SendData "'0" + Chr$(0) + Chr$(0) + Chr$(0)

Enable SPI Mode PORT A

Command Sent	ASCII Code	Bytes	Data	Function
	S1A	3	-	Sets up Port A for SPI, sets bits 0-2 to appropriate directions for SPI, stores port value and direction registers
Command Reply	ASCII Code	Bytes	Data	
	S1A	3	-	

Operation: Sets up PORTA for SPI mode, it will set the direction register bits 0-2 for the appropriate setting, it also stores the Port Value and direction register of the port before setting up SPI

Example: Winsock1.SendData "S1A"

Disable SPI Mode PORT A

Command Sent	ASCII Code	Bytes	Data	Function
	S0A	3	-	Disables SPI on Port A, and restores port value and direction registers that were before an enable SPI
Command Reply	ASCII Code	Bytes	Data	
	S0A	3	-	

Operation: Disables SPI on PORT A. Restores direction register and port values before SPI was enabled.

.

Example: Winsock1.SendData "S0A"

Send SPI Data on PORT A

Command Sent	ASCII Code	Bytes	Data	Function
	SA	2 + #data bytes + data	*data*	Sends out SPI data on PORT A. The number of data bytes is sent out first followed by the data. The return data will automatically be clocked in and sent back in the command reply.
Command Reply	ASCII Code	Bytes	Data	
	SA	2 + #data bytes + data	*Return data*	

Operation: Sends out SPI data on PORTA. The command consists of SA which tells the firmware that SPI data will be sent on PORTA, then the number of data bytes to be sent out followed by the data bytes. Below are some examples of the commands

Example: Winsock1.SendData "SA" + Chr$(&H2) + Chr$(&HAA) + Chr$(&H55)
Where Chr$(&H2) is the number of data bytes to be sent via SPI and Chr$(&HAA) and Chr$(&H55) are the data bytes sent.

SPI Mode

Notes on SPI

Bit 1 is always Clock and must be set as an output from the SPI to function
Bit 2 is always Serial Data Out (MOSI) and must be set as an output from the SPI to function
Bit 3 is always Serial Data In (MISO) and must be set as an input from the SPI to function
All other port pins act as normal
Setting the port pin as high or low will set the clock as normally high or low before the SPI transaction begins.
Any of the other pins can be used as SS or CE when SPI is used.

Setting up SPI

1. Set PORTA direction register and data register to appropriate values for Chip select pins or other pins that will be used by PORTA.
   Winsock1.SendData "!A" + Chr$(*Direction*)
   Winsock1.SendData "A" + Chr$(Value)

2. Send command Enable SPI on PORTA
   Winsock1.SendData "S1A"
   This will set the direction register of bits 0-2 to the appropriate value for SPI.

3. Sending bytes out via SPI
   To send out SPI data on Port A it requires and "SA" prefix followed by the number of bytes you want to send followed by the data stream. If you wanted to send out the following bytes $FF $AA $55 you would do the following
   Winsock1.SendData "SA" + Chr$(&H3) + Chr$(&HFF) + Chr$(&HAA) + Chr$(&H55)

4. Receiving Bytes from SPI.
   The number of bytes returned is determined by the number of bytes sent. Therefore if more data is expected than what was sent out then Null bytes will have to be sent out. Sending out the Null bytes is required to keep the SPI clock running so that the return data is clocked out from the SPI slave device. There are null bytes sent in the example image below, this can be seen in the last two bytes which are sent (implemented with don't cares) and on the input pin you will have the valid data from the device. The valid data will be sent back in the command response packet.

   Note: you may have to send out leading zero's and don't cares depending on device you are communicating with.   e.g. seven leading zero's then a start bit as per the image below.
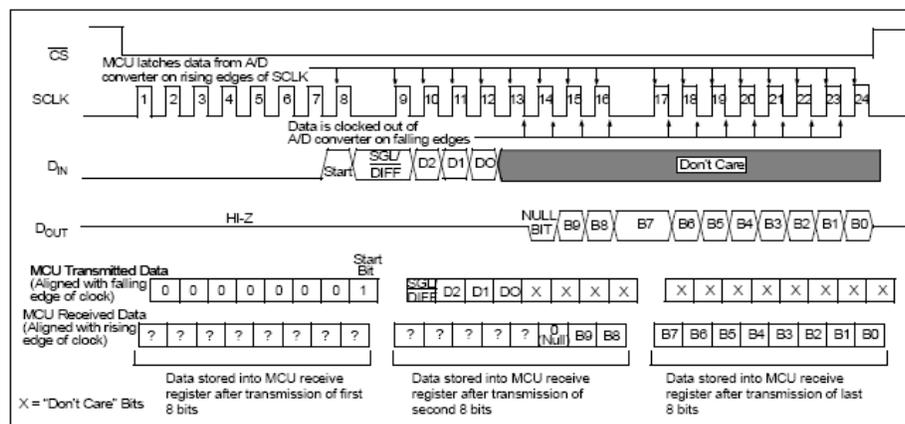
   Example image for SPI communications with the MCP3004/3008 using 8 bit segments taken from the MCP3004./3008 datasheet. (http://www.microchip.com)

EEPROM Memory contents

The EEPROM on the Ether I/O 24R is used to store the board's Serial number and other critical factory settings as well user settings for the module and there is even a spare area where you can store your own data to be kept by the module, even when the module loses power.

The EEPROM chip on the module is 1 Kilobit in size or 1024 bits of memory; this is arranged as 64 words of 16 bits each. The first 5 words as addresses 0 to 4 are not user write-able. Words 5-24 are currently used to store the user settings like fixed IP address, port power-up settings and AutoScan mode settings. Words 25-47 are reserved for future use and words 48-63 are free for your own data storage.

Memory Usage is shown by Byte for Clarity, each word is made up of 2 bytes.

Table 2, EEPROM Memory Usage

Word	MSB Byte	Function	LSB Byte	Function
0-4	1-9	**Reserved (Unwritable)**	0-8	**Reserved (Unwritable)**
5	11	Control Bits 2	10	Control Bits 1
6	13	Fixed IP Address Byte 2	12	Fixed IP Address Byte 1
7	15	Fixed IP Address Byte 4	14	Fixed IP Address Byte 3
8	17	Preset Port A Value	16	Preset Port A Direction
9	19	Preset Port A Pull up	18	Preset Port A Threshold
10	21	Preset Port B Direction	20	Preset Port A Schmitt Trigger
11	23	Preset Port B Threshold	22	Preset Port B Value
12	25	Preset Port B Schmitt Trigger	24	Preset Port B Pull up
13	27	Preset Port C Value	26	Preset Port C Direction
14	29	Preset Port C Pull up	28	Preset Port C Threshold
15	31	Reserved for Future Use	30	Preset Port C Schmitt Trigger
16	33	AutoScan Port B Mask	32	AutoScan Port A Mask
17	35	AutoScan Filter Count	34	AutoScan Port C Mask
18	37	AutoScan Scan Rate MSB	36	AutoScan Scan Rate LSB
19	39	AutoScan Target MAC Address 2	38	AutoScan Target MAC Address 1
20	41	AutoScan Target MAC Address 4	40	AutoScan Target MAC Address 3
21	43	AutoScan Target MAC Address 6	42	AutoScan Target MAC Address 5
22	45	AutoScan Target IP Address 2	44	AutoScan Target IP Address 1
23	47	AutoScan Target IP Address 4	46	AutoScan Target IP Address 3
24	49	AutoScan Target Port MSB	48	AutoScan Target Port LSB
25	51	Subnet Mask IP Address 2	50	Subnet Mask IP Address 1
26	53	Subnet Mask IP Address 4	52	Subnet Mask IP Address 3
27	55	Gateway IP Address 2	54	Gateway IP Address 1
28	57	Gateway IP Address 4	56	Gateway IP Address 3
29	59	Programmable Port MSB	58	Programmable Port LSB

! Words 30 to 47 are reserved for future use and must be left erased to all ones !

Words 48 to 63 are for your own use and may be used for any function you desire.

The Control Bits 1 Location is used to turn on and off the Fixed IP address, Preset Port and AutoScan mode functions. When the EEPROM is blank it reads all ones i.e. each blank word reads 65535 or $FFFF. Because of this we use a 0 bit value to turn a function on. The currently used bits are bit 0, which is used to enable the Fixed IP address, Bit 1, which is used to enable the Preset Port function and Bit 2, which is used to enable the AutoScan function. All the remaining bits should be left as ones for future compatibility as the firmware is upgraded and additional functions added.

Setting the Control Bits 1 byte to 255 turns off all three functions. The Fixed IP function has a bit value of 1, the Preset Port function a bit value of 2 and the AutoScan function a bit value of 4. Simply subtract the bit values of all the functions you wish to enable from 255 to calculate the value to write to the Control Bits 1 Location.

Basic Programming

Please Note: All example code in this section is written in visual basic code using the Winsock control for opening a UDP socket for communication with the Ether I/O 24 module. As most programming languages have access to network sockets, please refer to your languages manuals for similar socket functions.

1. Reading data from the Module

There are two different ways to read data from the socket, polling and interrupt. Polling is best for simple programs as each part of the program must complete before the next part can begin. However, if you have set up the AutoScan feature then the interrupt method should be used.

In order to receive data from the module we must bind the port and then check the Winsock1.BytesReceived value for incoming data. When using the polled mode it is recommended that each data-waiting loop has a timeout as otherwise a missed packet or unplugged module will cause your program to stop responding. This is best achieved by using the Timer to exit the data-waiting loop after a timeout period has elapsed.

The interrupt method is recommended for more advanced programmers as it allows a function to complete and the program to return to idle while waiting for the data to arrive thereby saving CPU power, as the program will not have to continually poll the socket for incoming data. This method is accomplished by using the Winsock1_DataArrival event.

In the following examples we will show both methods where applicable, or just the simple method if there is no reason to use the interrupt method.

2. Working with individual bits

As each I/O line is one of a group of 8, we must know how to change these lines one at a time. To work with these ports as bits we must use the logical OR and logical AND instructions. To make a line high we take the port's current value and OR the bit value of the line we wish to change. To make it low we AND the inverse of the bit value with the port. To generate the inverse value of the bit we use the NOT function. It is recommended to make a global variable for each of the port values as such.

```
Global PortAValue, PortBValue, PortCValue As Byte
```

Thus if we wish to make high the I/O 4 line of port A, we must take the current value of port A and OR it with 8, which is bit value of I/O 4. The code for this is as follows.

```
PortAValue = PortAValue Or 8
Winsock1.SendData "A" + Chr$(PortAValue)
```

To make the same I/O line low again we would use AND with NOT to set the line low.

```
PortAValue = PortAValue And Not 8
Winsock1.SendData "A" + Chr$(PortAValue)
```

If the value of the port is not known and you don't want to change the whole port you must read the ports value before writing it again.

To convert a line number 1-8 to a bit value $1 - 128$ we use twos exponent with the exponent number being one less than the line number.

```
BitValue = 2 ^ (Line – 1)
```

Basic Programming (continued)

3. Finding a module's IP address

If we aren't using a Fixed IP scheme then we need to find the IP address that the DHCP server has allocated to the Ether I/O 24 module. To find the module's IP address on our local network we must open a UDP socket and listen on that socket for a reply from the module, then broadcast an **Identify Ether IO 24 Units** command over the network and wait for any Ether I/O 24 modules to respond. To accomplish this we must add a Winsock component to our program's form and then at the start of our program we need to place this code in the Form Load() module. This code will broadcast the IO24, **Identify Ether IO 24 Units** command over the network and wait for the first Ether I/O 24 module to respond, following which the Data is read and the IP Address is printed to the debug window.

```
Private Sub Form_Load()
Winsock1.Protocol = sckUDPProtocol          ' Set the protocol to UDP
Winsock1.RemoteHost = "255.255.255.255"     ' Set the remote address to broadcast
Winsock1.RemotePort = 2424                   ' Set the port to 2424 for the Ether I/O 24 module
Winsock1.Bind                                ' Bind the local port to receive data
Winsock1.SendData "IO24"                     ' Send the Identify command to all IO24 units
While Winsock1.BytesReceived = 0             ' Wait for data to Arrive
Wend                                         ' Loop the Wait
N = Winsock1.BytesReceived                   ' How many bytes arrived
Winsock1.GetData a$, vbString, N             ' Read them into a buffer
Debug.Print Winsock1.RemoteHostIP            ' Tell the user the IP Address
End Sub
```

If using the Interrupt mode, rather than waiting for the data to arrive back from the module, we use the Winsock1_DataArrival Module to process the data when it arrives thereby freeing the program to do other tasks while waiting for a response. Because we have bound the port with the Winsock1.bind property, when data arrives it will automatically run the Winsock1_DataArrival event module. To do something useful with this we put a small piece of code there to print the module's IP address to the debug window.

```
Private Sub Form_Load()
Winsock1.Protocol = sckUDPProtocol          ' Set the protocol to UDP
Winsock1.RemoteHost = "255.255.255.255"     ' Set the remote address to broadcast
Winsock1.RemotePort = 2424                   ' Set the port to 2424 for the Ether I/O 24 module
Winsock1.Bind                                ' Bind the local port to receive data
Winsock1.SendData "IO24"                     ' Send the Identify command to all IO24 units
End Sub


Private Sub Winsock1_DataArrival(ByVal bytesTotal As Long)
Winsock1.GetData a$, vbString, bytesTotal     ' Get the data from the buffer
Debug.Print Winsock1.RemoteHostIP             ' Show the IP address of the unit
End Sub
```

Now we know the IP address of our Ether I/O 24 module we can execute other commands to set up the ports for Input or Output of signals.

Basic Programming (continued)

4. Basic set up and writing to the ports

To set up the ports for output or basic input there are only 2 registers per port that need writing or reading. The first of these is the Direction register and the second is the Port register that sets whether the output pin is high or low. To set the direction register we write to the module with a Port Letter Prefix of !, telling the module that the information is for the Direction Register, Then we tell the module which port A, B or C we want to write to and finally we tell the module what value to place in this register.

For example, to set Port A as all outputs we would write the following code.

```
Winsock1.SendData "!A" + Chr$(0)
```

To then Set all of the Port A lines low we would write.

```
Winsock1.SendData "A" + Chr$(0)
```

To save code space and network traffic we combine these two commands into a single command.

```
Winsock1.SendData "!A" + Chr$(0) + "A" + Chr$(0)
```

5. Reading the Ports

To read the ports is a little more complex as we have to send a read command to the module and then process the reply. This can be done by the module waiting for the reply or by using the Winsock1_DataArrival event similar to what we did in the Finding the IP address code. To read Port A we transmit a **Read Port A** command and then process the response. The first example code does this by waiting for the data to arrive back from the module.

```
Winsock1.SendData "a"                    ' Send the Read Port A Command
While Winsock1.BytesReceived = 0         ' Wait for data to Arrive
Wend                                     ' Loop the Wait
N = Winsock1.BytesReceived               ' How many bytes arrived
Winsock1.GetData a$, vbString, N         ' Read them into a buffer
If N = 2 And Left$(A$, 1) = "A" Then     ' Confirm that the Data packet is valid
   PortA = Asc(Right$(A$, 1))            ' Store the value into a register
   Debug.Print Str$(PortA)               ' Tell the user the value read
   End If
```

Using the Winsock1_DataArrival event to read a port.

```
Winsock1.SendData "a"                    ' Send the Read Port A Command
```

```
Private Sub Winsock1_DataArrival(ByVal bytesTotal As Long)
Winsock1.GetData a$, vbString, bytesTotal       ' Get the data from the buffer
If bytesTotal = 2 And Left$(A$, 1) = "A" Then   ' Confirm that the Data packet is valid
   PortA = Asc(Right$(A$, 1))                    ' Store the value into a register
   Debug.Print Str$(PortA)                       ' Tell the user the value read
   End If
End Sub
```

Obviously in a real program we would not just print these results to a debug window, we would actually use them to do something useful. To illustrate this point and the operation of the module and it's programming, we have produced a series of application demonstration programs on output, input, EEPROM programming, using the AutoScan functions and interfacing to Clocked Serial devices.

Basic Programming (continued)

6. Programming the Input options

All 24 lines of the module can be independently programmed with 6 different input options or be set as an output. These input options are listed below with a brief description of each.

TTL Input: An input signal voltage below 1.4V reads as a low, above 1.4V reads as high
CMOS Input: An input signal voltage below 2.5V reads as a low, above 2.5V reads as high
Schmitt Trigger: An input signal below 0.25V reads as low, above 4.25V reads as high with any value in between these values having no effect on the inputs status.
Pull Up: Any of the three modes listed can be enhanced by turning on an internal pull-up resistor within the module thus removing the need for an external pull-up resistor. This feature allows any contact or open collector output device to be connected to the module without additional external components.

Three registers for each port, The Threshold Register, Schmitt Trigger Register and Pull-UP Register, control the six different input modes. The Threshold Register sets whether the input threshold voltage is 1.4V or 2.5V: if the corresponding bit in the register is high then the threshold voltage is 1.4V and if low, the threshold voltage is 2.5V. The Schmitt Trigger Register overrides the threshold register when the corresponding bit is low, enabling the Schmitt Trigger input state. Finally, the Pull-Up register enables the corresponding pull-up resistor when the bit is low. The default power-up state for all registers is all high bit (TTL Input, No Pull-Up) unless the EEPROM has been programmed to set them up differently.

There are 9 commands that correspond to the 3 input option registers for the three ports. The use of these commands is exactly the same as the direction register except that the port letter prefix changes.

If for example we had 8 switches on Port C and we wanted these switches to be read by the module we would probably choose to use the CMOS input mode with the Pull-Up resistors turned on. To achieve this we would program the Threshold Register to all low bits and the Pull-Up register to all low bits with the Schmitt Trigger Register as all high bits and the Direction register as all high bits.

```
PortC_Direction = 255
PortC_PullUp = 0
PortC_Threshold = 0
PortC_SchmittTrigger = 255
Winsock1.SendData "!C" + Chr$(PortC_Direction) + "@C" + Chr$(PortC_PullUp) + "#C" + Chr$(PortC_Threshold) + "$C"
+ Chr$(PortC_SchmittTrigger)
```

Once again, we have combined the four commands into a single packet thereby saving network traffic. Whenever possible, combine all your set-up commands into a single data packet whereby processing overhead will be lower and the module will be initialised more quickly.

Basic Programming (continued)

7. Using the EEPROM

The Ether I/O 24 module EEPROM is programmed using several fixed length packets. Packet length is always 5 bytes with the first byte being the ' character. Where all 5 bytes are not specifically needed, such as a Read command where only the command and address need to be sent to the module, the packet is padded with bytes whose value is 0. There are five different EEPROM specific and 1 special command in the EEPROM set that resets the module. Because this command is used to activate the data stored into the EEPROM it is placed in the EEPROM command set.

The five EEPROM commands are Read, Erase, Write, Write Enable and Write Disable. We will show examples of how each command is used and describe when to use it.

The EEPROM Read command is used whenever we want to know the values stored in the EEPROM of the module. The read command will respond with a 4-byte packet containing the R from the Read command the address and the data from the EEPROM.

```
Dim ReadDone As Boolean
ReadDone = False
TimeOut = Timer + 0.2
Winsock1.SendData "'R" + Chr$(EEAddress) + Chr$(0) + Chr$(0)
While ReadDone = False And Timer < TimeOut
If Winsock1.BytesReceived <> 0 Then
   Winsock1.GetData a$, vbString
   If Len(a$) = 4 And Left$(a$, 2) = "R" + Chr$(EEAddress) Then
      EEData(EEAddress) = Asc(Mid$(a$, 4, 1)) + Asc(Mid$(a$, 3, 1)) * 256!
      ReadDone = True
      End If
   End If
Wend
If ReadDone = False Then Call MsgBox("No Response from Module", vbCritical, "Module Error"): Exit Sub
```

The EEPROM Write command is used to write a byte to the EEPROM memory. It is recommended to read the EEPROM in the module after any write command to ensure the data is correct after the write cycle. Allow a 10ms delay between writing the EEPROM word and reading it, this gives time for the EEPROM to correctly program the memory location.

```
Winsock1.SendData "'W" + Chr$(EEAddress) + Chr$(EEDataMSB) + Chr$(EEDataLSB)
```

This code writes the Values in EEDataMSB and EEDataLSB to the EEPROM memory at location EEAddress. It should be noted that this command will not work unless an EEPROM Write Enable command has first been sent. This command should be followed by an EEPROM Read Command to verify that the data has been written to the EEPROM successfully.

Erasing an EEPROM memory location is accomplished by using the EEPROM Erase Command. The only data that is needed for the Erase function is the Address to erase so we use the extra 2 bytes to the command to confirm that the command is valid.

```
Winsock1.SendData "'E" + Chr$(EEAddress) + Chr$(&HAA) + Chr$(&H55)
```

As with the Write command an EEPROM Write Enable Command must be sent before the Erase command will work. The code for executing an EEPROM Write Enable command is as follows.

```
Winsock1.SendData "'1" + Chr$(0) + Chr$(&HAA) + Chr$(&H55)
```

After the Write or Erase commands are complete you should protect the EEPROM from accidental writes by sending the EEPROM Write Disable Command. Here is the code for the write disable command.

```
Winsock1.SendData "'0" + Chr$(0) + Chr$(0) + Chr$(0)
```

These commands can be used to configure the module as described in the Advanced Programming section or to read and write the User data area in the EEPROM memory.

Advanced Programming

This section will cover the several features of the module that require the EEPROM to be programmed for them to operate.

Programming the module to have a fixed IP address

Having a fixed IP address is ideally suited to industrial control and is essential when using the module with a router and a routing table.

To calculate the values to write to the Fixed IP words take the IP address you wish to use and split it into 2 halves, the first half contains bytes 1 and 2 and the second half contains 3 and 4.

Lets say we want to use a Fixed IP address for the module of 10.0.0.100, the 10.0 become the word at address 6 and the value at word address 7. Conveniently the IP address is already split into byte values so we can use these directly in our programming function to program the EEPROM.

To Program the first Word of our IP address into the EEPROM we simply take our IP address and use the first byte as the LSB of our programming command and the second byte as the MSB of our programming command. Thus our commands to program the EEPROM look like this.

Winsock1.SendData "'W'" + Chr$(6) + Chr$(0) + Chr$(10)

  EEPROM Write Command **Address** **MSB** **LSB**

Winsock1.SendData "'W'" + Chr$(7) + Chr$(100) + Chr$(0)

  EEPROM Write Command **Address** **MSB** **LSB**

Keep in mind that to program this to our EEPROM we must have sent the Write Enable command to the module first and ensure that the J2 jumper link is not placed.

To then enable the Fixed IP function we have to write to word 5 with bit 0 of the LSB as a zero.

It is recommended to read the EEPROM value at Address 5, ensure that bit-0 is cleared and then write this value back to the EEPROM.

Please refer to the EEPROM read and write examples in the previous section for example code.

Programming the module's ports default power up state

At power up or reset the module will read it's EEPROM memory, if bit 1 in EEPROM address five is cleared then it will read the EEPROM memory addresses from 8 to 15 and write these values to the Port Registers. By writing to the appropriate addresses in EEPROM memory it is possible to have any port set to any state by default. The ports are set up by the EEPROM settings the same as by the commands sent from the host.

For Example, let's say we want Port A to power up as all Inputs and Port B to power up as all outputs with lines 1 to 4 at low and the other lines at high. We would need to ensure that the Port A Direction register was all high and the Port B direction register was all low bits, then set the Port B value register as appropriate for the desired output.

Advanced Programming (continued)
Programming the module's ports default power up state

By referring to the EEPROM memory use, Table 2 on Page 31, we can see that the Address we need to set up for the Port A direction register is the MSB of Address 8. Being that we don't care about the LSB of Address 8 and we want the MSB to be all ones we can program address 8 with the values MSB = 255 and LSB = 255.

For Port B the direction register is in the LSB of address 11 and the Port Value is in the MSB of Address 10.

As LSB of Address 10 is the Port A Schmitt Trigger value and as we want these turned off we can program this with 255, the Value for Port B is binary 11110000, which translates to 240 decimal.

The MSB of address 11 is the Port B Pull-Up register and as we don't care about this value if the port is set as outputs we can just write 255 to the MSB and 0 to the LSB to make Port B all outputs.

The last step is to program bit1 of address 5 as a low, to do this we read the existing value of address 5 use the And and Not functions to affect just the one bit, then write this new value back to address 5.

Programming the AutoScan Mode

The AutoScan mode will allow the module to originate communication with a remote device or another Ether I/O 24 module. This mode is very useful as it allows your software the freedom not to have to poll the module to check the state of the inputs.

To allow the module to communicate through a router the Gateway address, Subnet mask, IP address and Port Number needs to be programmed into the module. When autoscan is enabled the address of the target device will be compared to the subnet mask to see if the address falls within the Local network. If the address is within the local network the packet will be automatically sent to the address of the target device on the programmed port number. If the address is outside the local network the gateway will be arped to get the MAC address and then the packet will be sent to the Gateway with the IP Address of the target device.

Mask bits are used to allow some of the input pins of the module to toggle without generating messages from the module. Any input whose corresponding mask bit is low, is ignored by the AutoScan function. The Scan Rate is a 16-bit value which is used to divide the scan rate of the AutoScan mode from it's base rate of 1,000 scans per second down to a user programmed rate from 500 scans per second to one scan per 65.5 seconds. When set at 1 the scan rate is 1,000 scans per second, it is 500 scans per second when the value is 2 and so on. You simply divide the 1,000 per second rate by your desired scan rate to find the value for address 18. The filter value which is in the MSB of Address 17 is used to count the number of identical reads that are required before a port value is considered valid and sent to the target device. When set at 0 the filter is turned off, when set at 1 the port must read the same for two scans to be considered valid and sent to the target. Higher numbers simply increase the number of identical reads required before the value is considered valid.

Care should be taken when using high filter values with slow scan rates, as the reporting time for a change under these conditions can be over 4 hours.

Resetting the Module

To activate the new settings that have been programmed into the EEPROM memory you must reset the module. This is accomplished using the Reset Module Command. The command must have that data as shown in the programming example.

```
Winsock1.SendData "@" + Chr$(0) + Chr$(&HAA) + Chr$(&H55)
```

All of these settings can be programmed using the Ether I/O 24 Test Programming Utility available for download from the Elexol website.

Programmable Port Number

The port number is now programmable so that it can be changed from the default setting of 2424. It should be noted that when the jumpers are placed on the Ether I/O 24 the unit will always default to 2424. The Programmable Port number can be programmed like any other EEPROM value and will become active after a reboot condition.

Glossary

BOOTP	Boot Protocol if the protocol used by DHCP for the module to get a Dynamic IP address.
IP	Internet Protocol is the protocol used for all Internet messages and it allows a Packet to travel from its Source to its Destination via a Router or even hundreds of Routers distributed around the world.
UDP	User Datagram Protocol is a standard protocol allowing for the Connectionless exchange of data between 2 points on a IP network.
ICMP	Internet Control Message Packet is a protocol used to send messages to control the network and also used to send a Ping request and reply.
Ping	The common name given to a program that sends an ICMP Echo message to another network device and times how long the reply takes to return.
DHCP	Dynamic Host Control Protocol, A standard way to assign an IP address to a device, most small routers have an DHCP Server that will assign the next available address to any device on the network that requests one. This saves the user from having to keep track of the Address of each network device and manually assign it an address.
Port Number	Each device on an IP network has an IP Address, this would allow for only 1 link to be formed with another device without the use of a sub addressing scheme, this scheme is referred to as the port number. The numbers from 1 to 1024 are reserved for special functions and protocols such as DHCP or ICMP. The Ether I/O 24 uses port 2424 as it is easy to remember.
IP Address	An address like a post box number assigned to every network device in order for massages, called packets, to be sent from one device to another. The IP address usually takes the form of 4 numbers between 0 and 255 separated by dots. (e.g. 192.168.1.1)
MAC	Media Access Control which is the actual message format sent over an Ethernet cable. This format is used to contain an IP packet and allow simple hardware devices called switches to route packets around a local area network based on the physical address of the hardware and not a logically assigned IP address.
MAC Address	The physical address of a network device that is assigned at manufacture time and allows the IP address to be assigned dynamically by allowing devices to differentiate each other at the hardware level.
IEEE	Institute of Electrical and Electronic Engineers, The organization responsible for defining standards that will allow different devices to talk to each other, regardless of their specific manufacturer or country of origin.

Document Revision History

Ether I/O 24R User Manual Version 1.0 – Initial document created 20th March 2008

Ether I/O 24R User Manual Version 1.1 – Updated 9th December 2009
- Updated Temperature ratings

Firmware Revision History

Ether I/O 24 Firmware Version 1.0
- Initial firmware release

Ether I/O 24 Firmware Version 1.1
- DHCP checksum fixed works on all IP addr ,
- Fix added to accept broadcast from 255.255.255.255 to x.x.x.255

Ether I/O 24 Firmware Version 1.2
- 8Mhz version of board
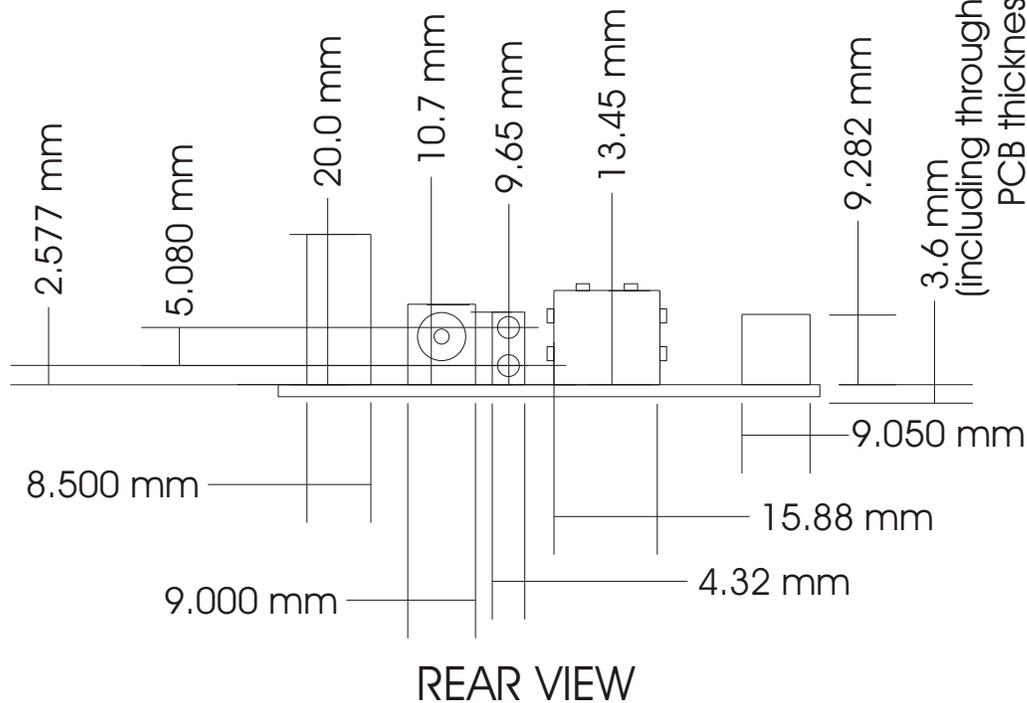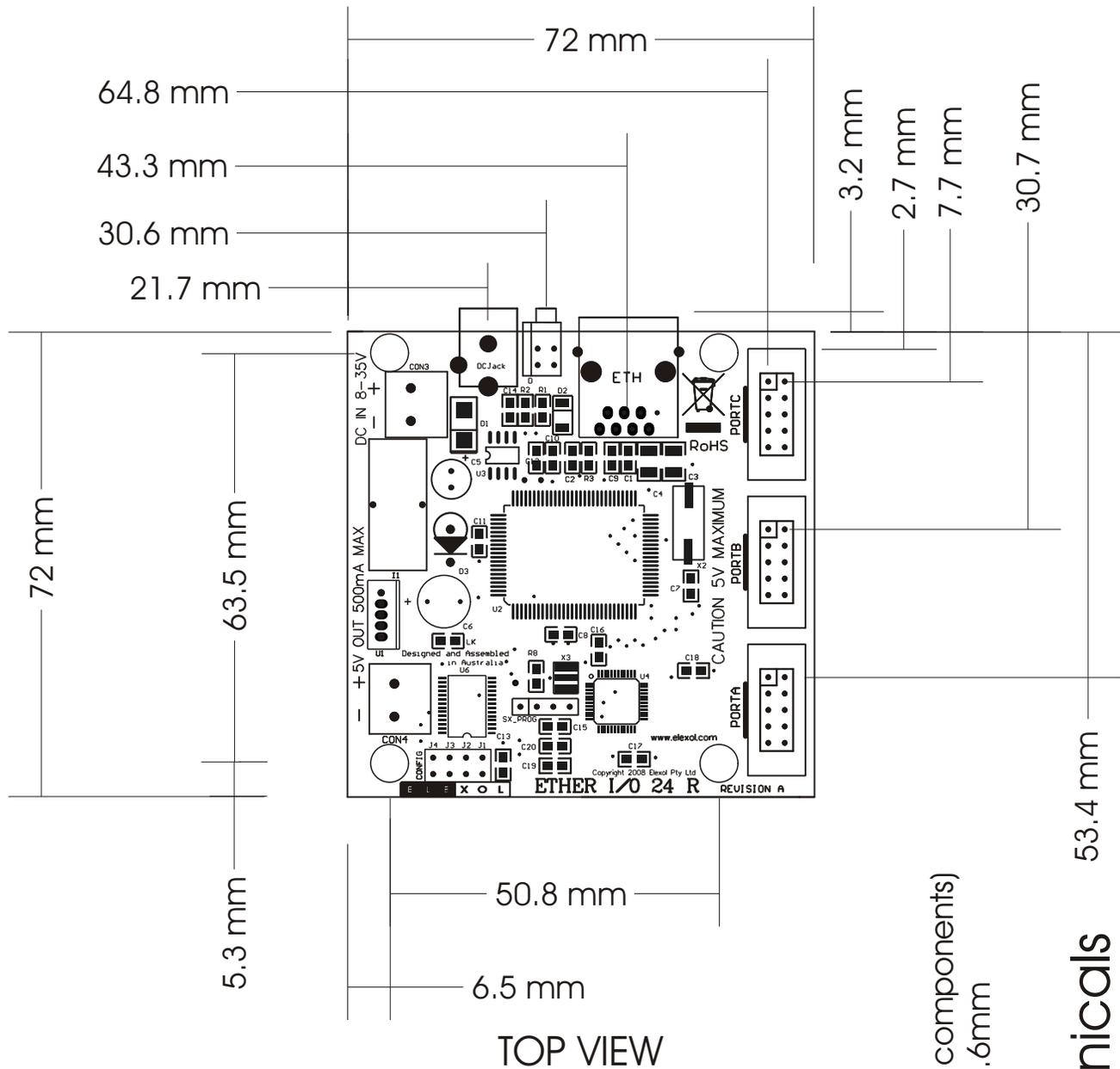
Ether I/O 24 Firmware Version 1.3
- Fix added so that the unit on occasion doesn't stop responding when a TCP packet is sent to it

Ether I/O 24 Firmware Version 1.4
- Fix added so that the unit IO24 Command doesn't have to be sent from broadcast packet

Ether I/O 24 Firmware Version 2.0
- Adapted firmware to run the new hardware, as well as adding new functionality of programmable port number, SPI and updating Autoscan features

TOP VIEW

72 mm
64.8 mm
43.3 mm
30.6 mm
21.7 mm

3.2 mm
2.7 mm
7.7 mm
30.7 mm

72 mm
63.5 mm
5.3 mm
53.4 mm

50.8 mm
6.5 mm

DC IN 8-35V
CON3
DCJack
ETH
RoHS
PORTC
+5V OUT 500mA MAX
CAUTION 5V MAXIMUM
PORTB
PORTA
Designed and Assembled in Australia
SX_PROG
CON4
CONFIG
J4 J3 J2 J1
E L E X O L
ETHER I/O 24 R
REVISION A
www.elexol.com
Copyright 2008 Elexol Pty Ltd

REAR VIEW

2.577 mm
5.080 mm
20.0 mm
10.7 mm
9.65 mm
13.45 mm
9.282 mm
3.6 mm
(including through/hole components)
PCB thickness = 1.6mm
9.050 mm
8.500 mm
15.88 mm
9.000 mm
4.32 mm

Ether I/O 24 R Mechanicals