ETHER I/O 24 DIP R Digital I/O Module

The ETHER I/O 24 DIP R shown in Diagram 1 is the Dual In-Line Package equivalent of our existing Ether I/O 24 R.

The Ether I/O 24 is an integrated, micro-controller based network interface board with 24 digital user I/O lines. The module's firmware and hardware enable your devices or other modules to be connected to a generic Ethernet network and controlled using a command sent over the network via UDP. 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 outputs of the module are able to source or sink up to 30mA per I/O, up to a maximum 50mA per port, to allow for direct connection to a variety of devices.

The ETHER I/O 24 DIP R Module

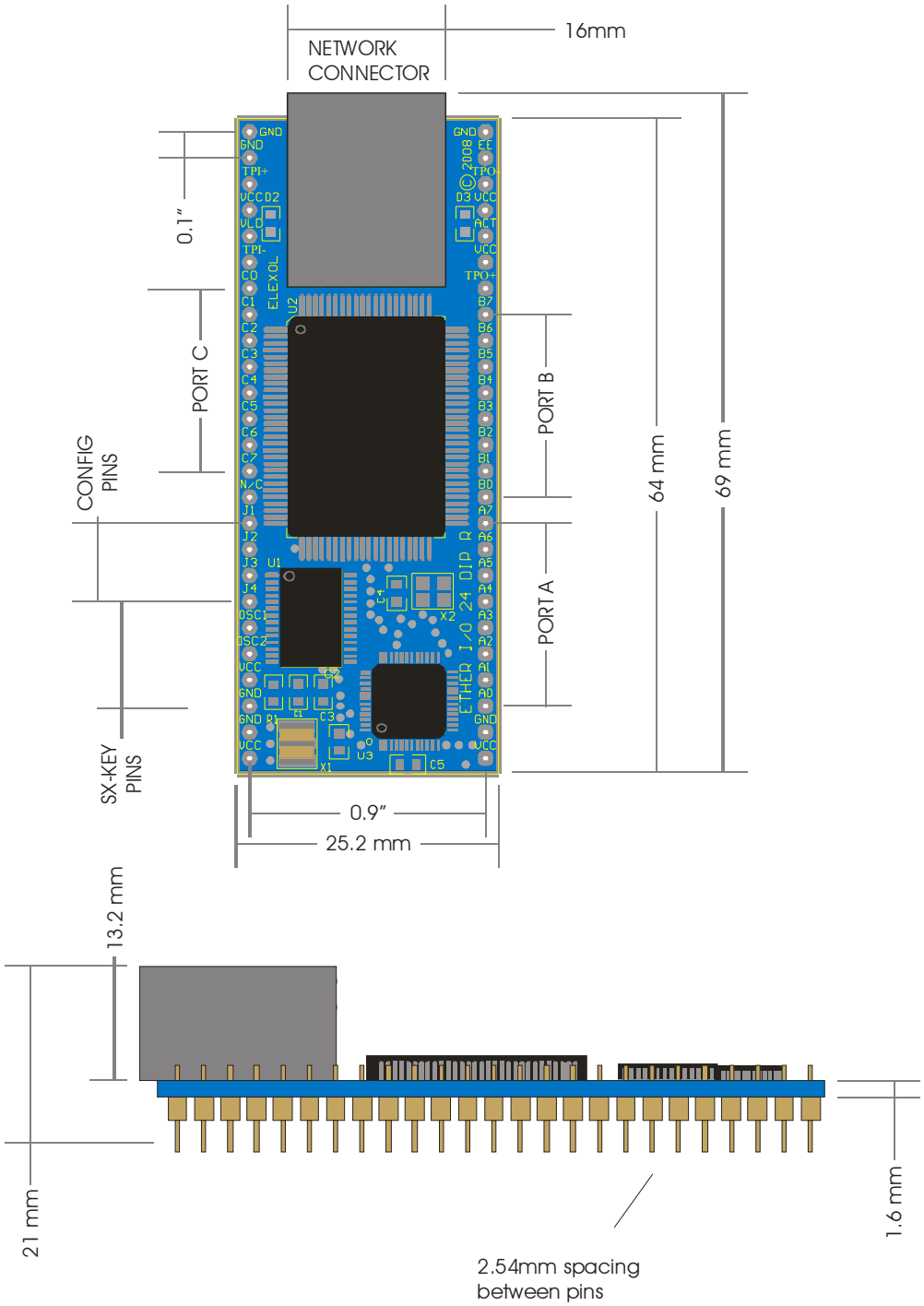
Diagram 1

MODULE FEATURES

- Supports ARP, BOOTP, DHCP, ICMP and UDP/IP Protocols
- Industry standard 10BaseT Ethernet Interface with an industry standard RJ-45 Connector
- 24 independently programmable signal lines with configurable CMOS, TTL or Schmitt Trigger thresholds and programmable pull-ups per line
- 50-pin Dual In-Line Package Ideal for prototyping
- Compact module Fits into a standard 50-pin 900mil IC Socket (69mm x 25.2mm x 21mm)
- Advanced configuration allows the modules to automatically scan the input ports and transmit changes directly to another ETHER I/O 24 module without host connection or to any Internet Port by router connection
- On board EEPROM allows all ports to power up in a user programmable state
- Programmable Fixed IP or Dynamic IP assignment from a DHCP server
- Small packet size and connectionless protocol allows for Real Time sensing and control
- Can be connected to a wireless network gateway or access point for wireless operation

Module Layout

Mechanicals



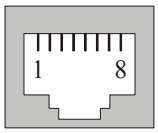
Physical Dimensions

Length	69mm
Width	25.2mm
Height	21mm (not in IC Socket)

Module Connectors

Ethernet Connector

The module is equipped with a standard RJ45 network socket and conforms to the 10 Base-T standards. 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 # on Connector	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

Ether I/O 24 DIP R Pin out Tables

Power Pins for the Ether I/O 24 DIP R

PIN #	SIGNAL	TYPE	DESCRIPTION
1	GND	PWR	Device – Ground Supply Pin
2			
23			
24			
27			
50			
4	VCC	PWR	Device – Power Supply Pin 4.5 to 5V VCC supply to Ether I/O 24 DIP R circuitry.
22			
25			
26			
45			
47			

Configuration Pins for the Ether I/O 24 DIP R

PIN #	SIGNAL	TYPE	DESCRIPTION
16	J1	IN	Tie to GND to lock the EEPROM memory from any write or erase procedures.
17	J2	IN	Tie to GND 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 tied to GND 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.
18	J3	IN	Tie to GND 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.
19	J4	IN	Tie to GND 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.
49	EE	OUT	Used to short out EEPROM for configuration purposes

SX Programming Header Pins

PIN #	SIGNAL	TYPE	DESCRIPTION
20	OSC1	IN	Crystal Oscillator Input – external clock source input
21	OSC2	OUT	Crystal Oscillator Output -
22	VCC	PWR	Device – Power Supply Pin 4.5 to 5V VCC supply to Ether I/O 24 DIP R circuitry.
23	GND	PWR	Device – Ground Supply Pin

I/O Connections (PORT A, B, C)

Listed below are the connections for the I/O connections on the Ether I/O 24 DIP R.

PORT A (Bi-Directional Data Bus pins)

PIN #	SIGNAL	TYPE	DESCRIPTION
35	A7	I/O	Programmable I/O pin with bit value of 128
34	A6	I/O	Programmable I/O pin with bit value of 64
33	A5	I/O	Programmable I/O pin with bit value of 32
32	A4	I/O	Programmable I/O pin with bit value of 16
31	A3	I/O	Programmable I/O pin with bit value of 8
30	A2	I/O	Programmable I/O pin with bit value of 4
29	A1	I/O	Programmable I/O pin with bit value of 2
28	A0	I/O	Programmable I/O pin with bit value of 1

PORT B (Bi-Directional Data Bus pins)

PIN #	SIGNAL	TYPE	DESCRIPTION
43	B7	I/O	Programmable I/O pin with bit value of 128
42	B6	I/O	Programmable I/O pin with bit value of 64
41	B5	I/O	Programmable I/O pin with bit value of 32
40	B4	I/O	Programmable I/O pin with bit value of 16
39	B3	I/O	Programmable I/O pin with bit value of 8
38	B2	I/O	Programmable I/O pin with bit value of 4
37	B1	I/O	Programmable I/O pin with bit value of 2
36	B0	I/O	Programmable I/O pin with bit value of 1

PORT C (Bi-Directional Data Bus pins)

PIN #	SIGNAL	TYPE	DESCRIPTION
14	C7	I/O	Programmable I/O pin with bit value of 128
13	C6	I/O	Programmable I/O pin with bit value of 64
12	C5	I/O	Programmable I/O pin with bit value of 32
11	C4	I/O	Programmable I/O pin with bit value of 16
10	C3	I/O	Programmable I/O pin with bit value of 8
9	C2	I/O	Programmable I/O pin with bit value of 4
8	C1	I/O	Programmable I/O pin with bit value of 2
7	C0	I/O	Programmable I/O pin with bit value of 1

LED / Ethernet Signals / No Connect Pins

PIN #	SIGNAL	TYPE	DESCRIPTION
5	VLD	IN	This pin will go active low when a valid command is received by the module. This pin is primarily used to function the valid command LED (D2)
46	ACT	IN	This pin is active low when the module is powered and the network interface is detected. The pin will go high when ever there is activity on the network.
3	TPI+	IN	This TP input pair receives the 10 M/bit/s differential Manchester encoded data from the twisted pair wire.
6	TPI-	IN	
44	TPO+	OUT	This TP output pair, outputs Manchester encoded signals which have been pre-distorted to prevent overcharge on the twisted pair wire.
48	TPO-	OUT	
15	N/C	N/C	No Connect

Module Hardware Functions

LED Functions

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

D3 = 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.

D2 = 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.

Configuration Pin Functions (J1,J2,J3,J4)

To set the Configuration of the Ether I/O 24 DIP R you will need to tie the appropriate pin to ground. The table below outlines what each configuration pin will do when tied to ground.

Pin Designator	Function
J1	Tie to GND to Lock EEPROM, J1 N/C allows EEPROM writes
J2	Tie to GND DHCP, N/C LOAD IP FROM EEPROM IF PROGRAMMED else DHCP
J3	Tie to GND to Fix IP – The module will operate at fixed IP address 10.10.10.10
J4	Tie to GND to Fix IP – The module will operate at fixed IP address 192.168.0.10

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

Jumper J2 when tied to ground 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 tied to ground 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 tied to ground 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 tied to ground 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.

Port A,B and C Configuration Functions

There are a number of different functions that can be implemented on the I/O port. Each of these functions will be outlined below and the command sets to work with these functions will be outlined in the following section of the datasheet.

I/O

The I/O configuration allows you set the direction and value register of the port pins. When writing to the direction register, setting the corresponding register bit to '1' will set the equivalent port pin to an input, if the register bit is set to '0' the port pin will become an output. The value register sets the pin to either a high or low depending on the state of the direction register. Setting a value register bit to '1' will set the pin in a high state (5V) and a '0' will set the pin to (0V) when the direction of the pin is set to output.

Pull Up

The Pull Up configuration 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

Threshold

The threshold function sets the threshold at which an I/O line reads as high or low. When the corresponding register 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.

Schmitt

Schmitt trigger inputs 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 line's 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

AutoScan

The AutoScan mode on the Ether I/O 24 DIP R allows the module to originate communications with a remote device or another Ether I/O 24 module without being polled. This mode is very useful as it allows the application software freedom to perform other tasks instead of constantly polling the module to check the state of the inputs.

In autoscan, a pin state must be set for bit test enable in autoscan mode and a change on the pin must occur before the data is sent to the target device. If you have an input toggling then this will send a packet every toggle depending on the toggle speed and the autoscan parameters (scan rate and scan filter).

SPI

SPI or Serial Peripheral Interface allows clocked serial communications between the Ether I/O 24 module and other SPI device such as A/D, D/A, GPIO expanders, sensors, Real time clocks, etc. Note this function is only implemented onto PORT A

Module Communication Interface

Communication Protocol via UDP

The functions of the module are controlled by a command set that is sent via UDP/IP packets. These UDP/IP packets can be sent out from the PC via application software or from other Ether I/O module if using Autoscan.

This section will cover in detail the command summary followed by the Command set used by the Ether I/O module.

Command Summary

For ease of use the command set has been broken into subgroups based on their function.

The commands are displayed as followed:

- All commands are shown as 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*
- 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 requires its hexadecimal value it is shown as follows e.g. 0x*55*.
- The spaces shown are only for clarity and no actual spaces are used in commands sent to the module.
- 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	Reply
Write Direction Port A	!A *data*	-
Write Direction Port B	!B *data*	-
Write Direction Port C	!C *data*	-
Write Pull Up Port A	@A *data*	-
Write Pull Up Port B	@B *data*	-
Write Pull Up Port C	@C *data*	-
Write Threshold Port A	#A *data*	-
Write Threshold Port B	#B *data*	-
Write Threshold Port C	#C *data*	-
Write Schmitt Port A	$A *data*	-
Write Schmitt Port B	$B *data*	-
Write Schmitt Port 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.

Function	Command	Reply
Read EEPROM word	'R *address dummy dummy*	'R *address MSB LSB*
Write Enable EEPROM	'1 *address 0xAA 0x55*	-
Write Disable EEPROM	'0 *address dummy dummy*	-
Write EEPROM word	'W *address MSB LSB*	-
Erase EEPROM word	'E *address 0xAA 0x55*	-
Reboot Module	'@ *dummy 0xAA 0x55*	

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.

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
IO24	0x49 0x4F 0x32 0x34	4	-	Identify IO24 Units	12	IO24	6 Byte MAC Address 2 Byte Firmware Version
A	0x41	2	Port Value	Write Port A	-	-	-
B	0x42	2	Port Value	Write Port B	-	-	-
C	0x43	2	Port Value	Write Port C	-	-	-
a	0x61	1	-	Read Port A	2	A	PortA_Value
b	0x62	1	-	Read Port B	2	B	PortB_Value
c	0x63	1	-	Read Port C	2	C	PortC_Value
!A	0x21 0x41	3	Direction	Write Port A Direction Register	-	-	-
!B	0x21 0x42	3	Direction	Write Port B Direction Register	-	-	-
!C	0x21 0x43	3	Direction	Write Port C Direction Register	-	-	-
!a	0x21 0x61	2	-	Read Port A Direction Register	3	!A	Direction
!b	0x21 0x62	2	-	Read Port B Direction Register	3	!B	Direction
!c	0x21 0x63	2	-	Read Port C Direction Register	3	!C	Direction
@A	0x40 0x41	3	Pull-Up	Write Port A Pull Up Register	-	-	-
@B	0x40 0x42	3	Pull-Up	Write Port B Pull Up Register	-	-	-
@C	0x40 0x43	3	Pull-Up	Write Port C Pull Up Register	-	-	-
@a	0x40 0x61	2	-	Read Port A Pull Up Register	3	@A	Pull-Up
@b	0x40 0x62	2	-	Read Port B Pull Up Register	3	@B	Pull-Up
@c	0x40 0x63	2	-	Read Port C Pull Up Register	3	@C	Pull-Up
#A	0x23 0x41	3	Threshold	Write Port A Threshold Register	-	-	-
#B	0x23 0x42	3	Threshold	Write Port B Threshold Register	-	-	-
#C	0x23 0x43	3	Threshold	Write Port C Threshold Register	-	-	-
#a	0x23 0x61	2	-	Read Port A Threshold Register	3	#A	Threshold
#b	0x23 0x62	2	-	Read Port B Threshold Register	3	#B	Threshold
#c	0x23 0x63	2	-	Read Port C Threshold Register	3	#C	Threshold
$A	0x24 0x41	3	Schmitt	Write Port A Schmitt Trigger Register	-	-	-
$B	0x24 0x42	3	Schmitt	Write Port B Schmitt Trigger Register	-	-	-
$C	0x24 0x43	3	Schmitt	Write Port C Schmitt Trigger Register	-	-	-
$a	0x24 0x61	2	-	Read Port A Schmitt Trigger Register	3	$A	Schmitt
$b	0x24 0x62	2	-	Read Port B Schmitt Trigger Register	3	$B	Schmitt
$c	0x24 0x63	2	-	Read Port C Schmitt Trigger Register	3	$C	Schmitt
‘R	0x27 0x52	5	Address *NU NU*	Read EEPROM Word	4	R	Address MSB LSB
‘W	0x27 0x57	5	Address MSB LSB	Write EEPROM Word	-	-	-
‘E	0x27 0x45	5	Address 0xAA 0x55	Erase EEPROM Word	-	-	-
‘0	0x27 0x30	5	*NU NU NU*	Write Disable EEPROM	-	-	-
‘1	0x27 0x31	5	*NU* 0xAA 0x55	Write Enable EEPROM	-	-	-
‘@	0x27 0x40	5	*NU* 0xAA 0x55	Reset Module	-	-	-
S1A	0x53 0x31 0x41	3	-	Sets up Port A for SPI, sets bits 0-3 to appropriate directions for SPI	3	S1A	-
S0A	0x53 0x30 0x41	3	-	Disables SPI on Port A, restores direction values and port values to what they were before an S1A command	3	S0A	
SAXX	0x53 0x41	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

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.

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.

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 operation as Write Port A Register but implemented on Port B

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 operation as Write Port A Register but implemented on Port C

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 I/O 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.

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 I/O 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.

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 I/O 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.

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.

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 operation as Write Port A Direction Register but implemented on Port B

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: operation as Write Port A Direction Register but implemented on Port C

Command Set (continued)

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 of Port A is read and its value sent back to the host.

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 of Port B is read and its value sent back to the host.

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 of Port C is read and its value sent back to the host.

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.

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 operation as Write Port A Pull Up Register but implemented on Port B

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 operation as Write Port A Pull Up Register but implemented on Port B

Read Port A Pull Up Register

	ASCII Code	Bytes	Data	Function
Command Sent	@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 of Port A is read and its value sent back to the host.

Read Port B Pull Up Register

	ASCII Code	Bytes	Data	Function
Command Sent	@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 of Port B is read and its value sent back to the host

Read Port C Pull Up Register

	ASCII Code	Bytes	Data	Function
Command Sent	@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 of Port C is read and its value sent back to the host

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.

Command Set (continued)

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 operation as Write Port A Threshold Register but implemented on Port B

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 operation as Write Port A Threshold Register but implemented on Port C

Read Port A Threshold Register

	ASCII Code	Bytes	Data	Function
Command Sent	#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 of Port A is read and its value sent back to the host.

Read Port B Threshold Register

	ASCII Code	Bytes	Data	Function
Command Sent	#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 of Port B is read and its value sent back to the host.

Read Port C Threshold Register

	ASCII Code	Bytes	Data	Function
Command Sent	#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 of Port C is read and its value sent back to the host.

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.

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 operation as Write Port A Schmitt Trigger Register but implemented on Port B

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 operation as Write Port A Schmitt Trigger Register but implemented on Port C

Read Port A Schmitt Trigger Register

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

Operation: The Schmitt Trigger Register of Port A is read and it's value sent back to the host.

Read Port B Schmitt Trigger Register

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

Operation: The Schmitt Trigger Register of Port B is read and it's value sent back to the host.

Command Set (continued)

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 of Port C is read and it's value sent back to the host.

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.

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 J1 pin is tied to GND.

Erase EEPROM Word

ASCII Code	Bytes	Data	Function
'E	5	*Address 0xAA 0x55*	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 J1 jumper is on.

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 J1 pin is tied to GND.

Command Set (continued)

Write Enable EEPROM

ASCII Code	Bytes	Data	Function
'1	5	*NU 0xAA 0x55*	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 J1 is tied to GND.

Reset Module

ASCII Code	Bytes	Data	Function
'@	5	*NU 0xAA 0x55*	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.

Enable SPI Mode PORT A

	ASCII Code	Bytes	Data	Function
Command Sent	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	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

Disable SPI Mode PORT A

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

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

Send SPI Data on PORT A

	ASCII Code	Bytes	Data	Function
Command Sent	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	SA	2 + #data bytes + data	*Return data*	

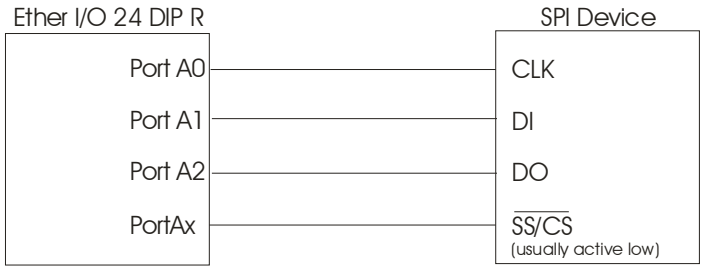
Operation: Sends out SPI data on PORTA. The command consists of SA which tells the firmware that SPI data will be sent on PORTA, then the number of data bytes to be sent out followed by the data bytes. Below are some examples of the commands

Overview of Ether I/O 24 DIP R SPI Mode

The following section is an overview of the SPI mode on the Ether I/O 24 DIP R and will cover what is required to interface to the port as well as setting up the SPI port for operation.

The SPI that is implemented on the Ether I/O 24 DIP R is a very basic SPI master device that allows 2 different modes depending on the initial clock pin state. The SPI mode only has the one clock speed which is mentioned below in the specifications. The SPI interface can be used to communicate with A/D, GPIO Expander, EEPROM chips. Shown in the diagram below is the typical connection of an SPI device to the Ether I/O 24 DIP R.

Typical Connection diagram between Ether I/O 24 DIP R and SPI device



Note: SPI mode is only available on the Port A.

Listed below are the SPI pin connections for PORT A used by the Ether I/O 24 DIP R.

- Port A0 is Serial Clock and must be set as an output for the SPI to function
- Port A1 is Serial Data Out and must be set as an output from the SPI to function
- Port A2 is Serial Data In and must be set as an input from the SPI to function
- All other Port A 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 CS when SPI is used. CS pins can be active high or active low depending on the setup 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.

2. Send command Enable SPI on PORTA
   This will set the direction register of bits 0-2 to the appropriate value for SPI. The Ether I/O 24 will echo the command to indicate that the SPI has been enabled

3. Once the SPI is enable then SPI data can be sent out and received.

Sending Bytes via SPI

4. To send out SPI data on Port A the command that needs to be sent out requires an "SA" prefix followed by the number of bytes you want to send followed by the data stream. To send out the following bytes 0xFF 0xAA 0x55 you would send the following
   UDP packet data contains the following "SA" + 0x03 + 0xFF + 0xAA + 0x55

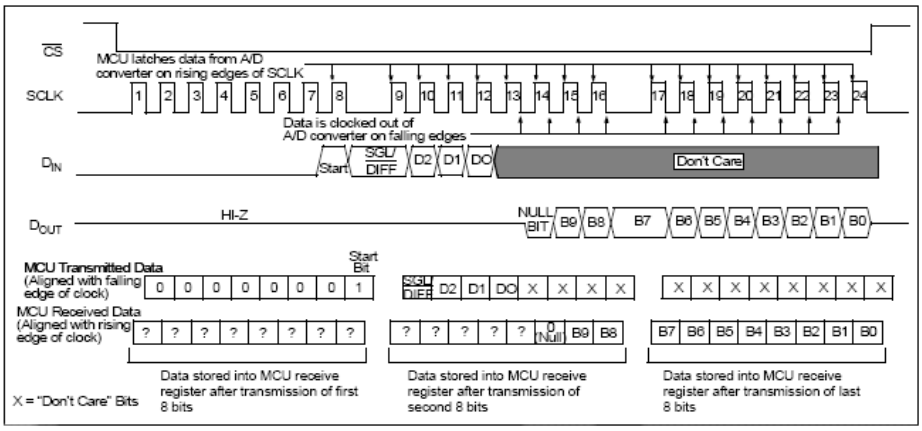
Receiving Bytes via SPI

5. Receiving bytes via SPI happens when the bytes are sent out, for every byte that is clocked out there is an incoming byte being received. The number of bytes received is determined by the number of bytes that has been sent out.

   In order to receive the number of bytes that is expected from the SPI device, 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.

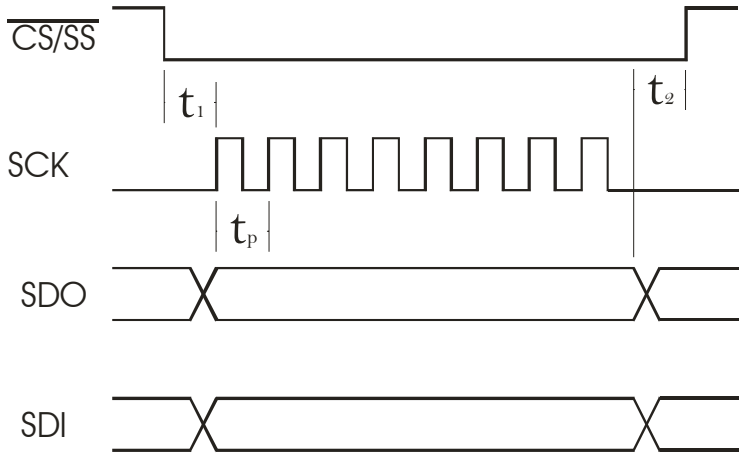


Example image for SPI communications with the MCP3004/3008 using 8 bit segments taken from the MCP3004./3008 datasheet. (**http://www.microchip.com**)

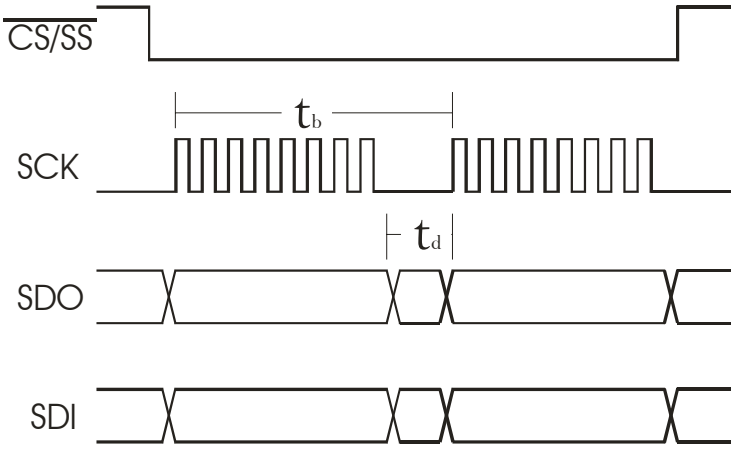
SPI Specifications

The table below outlines the specifications of the SPI mode implemented on the Ether I/O 24 DIP R.

SPI waveforms



SPI Byte timing



Symbol	Parameter	Min	Max	Units
t1	Time to First clock after CS has been lowered Max value is with CS in separate packet to SPI data	87us	2	mS
t2	Time to raise CS after last clock	42us	3.2	mS
tp	Clock Period	-	3	uS
tb	Time between start of bytes		45	uS
td	Time Between data bytes		22	uS

EEPROM Memory contents

The EEPROM on the Ether I/O 24 DIP R is used to store the board's MAC address / Serial number and other critical factory settings as well user settings for the module. There is even a spare area where user data can be stored on 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 5-29 are currently used to store the user settings like fixed IP address, port power-up settings and AutoScan mode settings. Words 29-47 are reserved for future use and words 48-63 are free for user 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 25 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.

EEPROM Control Bytes

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 0xFFFF. Because of this we use a 0 bit value to turn a function on. The bits currently used in Control Bits 1 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.

Specifications / 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	-0.6V to +5.6v 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

DC Characteristics (Temperature = 25°C, Power = 5VDC)

Parameter	Conditions	Min	Typ	Max	Units
DC Input		4.5		5.5	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

Programmers Reference Documentation

The following section will outline simple code examples on how to send and receive data from the Ether I/O 24 DIP. The project file for the following examples is available for download and can be found on our website at www.elexol.com under the Support downloads section.

UDP Interface example using Visual C# 2008 Express Edition

Sending UDP Packets

Broadcast IO24

Shown below is example code for a button (Ether_Scan) that broadcasts “IO24” on port 2424

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
//*****************************************************************
// Declare additional system-defined namespaces used by UDPClient
//*****************************************************************
using System.Net.Sockets;
using System.Net;

namespace EtherIO24example
{
    public partial class Form1 : Form
    {
        //*******************************************************
        //define variables that will be used throughout the code
        //*******************************************************
        public UdpClient udpClient = new UdpClient(2424);
        public byte[] sendBytes;

        public Form1()
        {
            InitializeComponent();
        }

        private void Ether_Scan_Click(object sender, EventArgs e)
        {
            //*******************************************************
            //Send out IO24 via broadcast(255.255.255.255) on port 2424
            //*******************************************************
            //*******************************************************
            //fill sendBytes buffer with "IO24"
            //*******************************************************
            sendBytes = Encoding.ASCII.GetBytes("IO24");

            //*******************************************************
            //Broadcast UDP packet with IO24 on port 2424
            //using the udpClient.send
            //*******************************************************
            udpClient.Send(sendBytes,           //buffer
                           sendBytes.Length,    //buffer length
                           "255.255.255.255",   //destination IP
                           2424);               //destination Port
            //*******************************************************
            //All units will respond with IO24
            //+ 6 byte MAC address + Version Number
            //*******************************************************
        }
    }
}
```

Example setting port direction registers and port values

The example code below shows the button code for setting the Port Direction register and the port values for PORT A on the Ether I/O 24 DIP R. The 'A' can be replaced with 'B' or 'C' depending in which port is being used.

Setting Port Direction Registers

```
private void Port_Direction_Click(object sender, EventArgs e)
{
   //******************************************************
   //Declare buffer variable
   //******************************************************
   byte[] buffer = new byte[5];

   //******************************************************
   //Assign buffer values for command '!' + 'A' + port value
   //******************************************************
   buffer[0] = Convert.ToByte('!'); //"!"
   buffer[1] = Convert.ToByte('A');//"A"
   buffer[2] = 0x64; //"0x48"

   //******************************************************
   //Send out Command (!A 0x64) to 10.10.10.10 on port 2424
   //******************************************************
   udpClient.Send(buffer,          //buffer
                  3,               //buffer length
                  "10.10.10.10",   //Destination IP
                  2424);           //Destination Port

}
```

Setting Port Values

```
private void Port_Value_Click(object sender, EventArgs e)
{
   //******************************************************
   //Declare buffer variable
   //******************************************************
   byte[] buffer = new byte[5];

   //******************************************************
   //Assign buffer values for command 'A' + port value
   //******************************************************
   buffer[0] = Convert.ToByte('A');//"A"
   buffer[1] = 0xFF; //"0xFF"

   //******************************************************
   //Send out Command (A 0xFF) to 10.10.10.10 on port 2424
   //******************************************************
   udpClient.Send(buffer,          //buffer
                  2,               //buffer length
                  "10.10.10.10",   //Destination IP
                  2424);           //Destination Port

}
```

Receiving UDP packets

UDP Listener

This example program will broadcast IO24 across the network on start up and list all Ether I/O 24's that respond to the command in a drop down combo box.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
//*************************************************************
//Declare additional system-defined namespaces used by UDPClient
//*************************************************************
using System.Net.Sockets;
using System.Net;

namespace UDPListener
{
 public partial class Form1 : Form
 {
  //*******************************************************
  //define variables that will be used throughout the code
  //*******************************************************
  public UdpClient udpClient = new UdpClient(2424);
  public byte[] sendBytes;
  public byte[] data = new byte[1024];
  string PCIPAddress;
  public string strHostName;
  string ReturnIPAddress;
  List<IPEndPoint> IPList = new List<IPEndPoint>();
  public int DeviceNo, list;
  public IPEndPoint EtherIP;
  //*******************************************************
  //the thread that will manage the data back from the board
  //*******************************************************
  private System.Threading.Thread thdUDPReciever;

  //*******************************************************
  //this Subroutine is to handle all the UDP return data
  //recieved from the thread UDPReciever
  //*******************************************************
  public delegate void ReturnUDPDataCallback(byte[] text, IPEndPoint IP);

  public Form1()
  {
    InitializeComponent();
  }

  private void Form1_Load(object sender, EventArgs e)
  {
    //*********************************************
    //Declare and start UDP recieve Thread
    //*********************************************
    thdUDPReciever = new System.Threading.Thread(new
System.Threading.ThreadStart(RecieveThread));
    thdUDPReciever.Start();
```

```
      //***********************************************
      //Find out PC IP address as UDP Recieve thread
      //recieves all UDP packets
      //***********************************************
      strHostName = System.Net.Dns.GetHostName();
      PCIPAddress =
System.Net.Dns.GetHostEntry(strHostName).AddressList[0].ToString();
   }

   public void RecieveThread()
   {
     while (true)
     {
       //***********************************************
       //Declare RemoteIPEndPoint
       //where RemotIPEndPoint is the any IP address
       //and Port Number of the incoming UDP packet
       //***********************************************
       System.Net.IPEndPoint RemoteIpEndPoint = new
System.Net.IPEndPoint(System.Net.IPAddress.Any, 0);
       //***********************************************
  //In remoteIPendpoint the IP address can be changed from any to a specific
       //IP address and the Port which is 0 in the above line can be changed
       //to a specific port number and it will only recieve data from that
      //port number.
      //***********************************************

      //***********************************************
      //Declare other variables used
      //***********************************************
      byte[] receiveBytes;
      string returnData;

      //***********************************************
      //Receive incoming UDP Packet
      //***********************************************
      receiveBytes = udpClient.Receive(ref RemoteIpEndPoint);
      returnData = System.Text.Encoding.ASCII.GetString(receiveBytes);

      //***********************************************
      // test to see if there is anything to recieve
      //***********************************************
      if (returnData.Length != 0)
      {
        //***********************************************
        //Filter data by IP address
        //if from PC IP address ignore
        //else accept incoming data
        //***********************************************
        if ((PCIPAddress == RemoteIpEndPoint.Address.ToString()))
        {
          //ignore any data sent by PC running application
        }
        else
        {
          if (ReturnIPAddress != RemoteIpEndPoint.Address.ToString())
          {
            //*****************************************
            // Setup list of IP address that have responded
            //*****************************************
            IPList.Insert(list, RemoteIpEndPoint);
            list = list + 1;
```

```
        }

          ReturnIPAddress = RemoteIpEndPoint.Address.ToString();

          //*********************************************
          //Return Data that was recieved from Ether I/O as well as IP
          //address
          //*********************************************
          ReturnUDPData(receiveBytes, RemoteIpEndPoint);

          }
        }
      }
    }

    public void ReturnUDPData(byte[] UDPData, IPEndPoint RemoteIP)
    {
      //*************************
      //Declare Variables
      //*************************
      string MacString;
      string VersionNumber;

      //*******************************************************
      //if 12 bytes were recieved in UDP Packet from the device
      //then we have recived IO24 + MAC +Version
      //*******************************************************

      if ((UDPData.Length == 12))
       {

        if (this.comboBox1.InvokeRequired)
        {
          //*************************************************
          //Need to invoke as the combobox and UDP reciever
          //is operating on different threads
          //*************************************************
          ReturnUDPDataCallback d = new
          ReturnUDPDataCallback(ReturnUDPData);
          this.comboBox1.Invoke(d, new object[] { UDPData, RemoteIP });
        }
        else
        {
          //*************************************************
          //Build up MAC string from the UDP data recieved
          //*************************************************
          MacString = "  MAC:" + Convert.ToString(UDPData[4], 16) + ":";
          MacString = MacString + Convert.ToString(UDPData[5], 16) + ":";
          MacString = MacString + Convert.ToString(UDPData[6], 16) + ":";
          MacString = MacString + Convert.ToString(UDPData[7], 16) + ":";
          MacString = MacString + Convert.ToString(UDPData[8], 16) + ":";
          MacString = MacString + Convert.ToString(UDPData[9], 16);
          MacString = MacString.ToUpper();

          //*************************************************
          //Build up Version Number string from the UDP data recieved
          //*************************************************
        VersionNumber = Convert.ToString(UDPData[10], 16) + ".";
        VersionNumber = VersionNumber + Convert.ToString(UDPData[11], 16);

          //*************************************************
          //Add Ether IP address & MAC Address to combo box
          //*************************************************
```

```
            this.comboBox1.Items.Add(RemoteIP.Address.ToString()
             + MacString + "   V" + VersionNumber);
             this.comboBox1.SelectedIndex = comboBox1.Items.Count - 1;

             DeviceNo = comboBox1.SelectedIndex;
             EtherIP = IPList[DeviceNo];

             }

            }
        }
        private void Ether_Scan_Click(object sender, EventArgs e)
        {
            //*****************************************************
            //Send out IO24 via broadcast(255.255.255.255) on port 2424
            //*****************************************************
            //*****************************************************
            //fill sendBytes buffer with "IO24"
            //*****************************************************
            sendBytes = Encoding.ASCII.GetBytes("IO24");

            //*****************************************************
            //Broadcast UDP packet with IO24 on port 2424
            //using the udpClient.send
            //*****************************************************
            udpClient.Send(sendBytes,           //buffer
                           sendBytes.Length,    //buffer length
                           "255.255.255.255",   //destination IP
                           2424);               //destination Port
            //*****************************************************
            //All units will respond with IO24
            //+ 6 byte MAC address + Version Number
            //*****************************************************

        }
    }
}
```

Read port direction registers and port values

Read Port A direction register

```
private void Read_DirA_Click(object sender, EventArgs e)
{
  //Command data
  data[0] = Convert.ToByte('!');//"!"
  data[1] = Convert.ToByte('a');//"a"
  //Sending out Command Data
  udpClient.Send(data, 2, EtherIP);
  //we expect a response of 3 bytes back from unit
  //"!" + "A" + direction value of port A
  //this data will be captured
 //and then dealt with in ReturnUDPData
}
```

Read Port A value register

```
private void Read_PORTA_Click(object sender, EventArgs e)
{
  //Command data
  data[0] = Convert.ToByte('a');//"a"
  //Sending out Command Data
  udpClient.Send(data, 1, EtherIP);
  //we expect a response of 2 bytes back from unit
  // "A" + port value of port A
  //this data will be captured
  //and then dealt with in ReturnUDPData
}
```

The return value from the read commands are dealt with by the ReturnUDPData function. The preceeding code is just a snippet for reading the port value and direction register for port A. These function could easily be modifed to read the EEPROM values from the device.

Further Reading and Examples

More information and examples for the ETHER I/O 24 DIP R can be found on our websites at www.elexol.com

Document Revision History

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

Ether I/O 24 DIP R User Manual Version 1.1 – Updated Hardware design. The design change connects the Ethernet signal pins from RTL8019AS directly to the DIP connections. The Pin out table has been updated to reflect these changes. Document updated 13th May 2009

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

Technical Support and Further Information

For any questions relating to the ETHER I/O 24 DIP R please contact us by Email:
support@elexol.com

Ph: +61 755 031202
Fax: +61 755 031206

Elexol Pty Ltd
Unit 1
8 Pirelli Street, Southport
Queensland 4215
Australia

Elexol Pty Ltd
PO Box 2742
Southport Business Centre
Queensland 4215
Australia

Product Use Limitations, Warranty and Quality Statement.

The ETHER I/O 24 DIP R should not be used in any situation where it's failure or failure of the PC or software controlling it could cause human injury or severe damage to equipment. This device is not designed for or intended to be used in any life critical application.

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

All ETHER I/O 24 DIP R 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 documentation.
This document provides information on our products 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/O 24 DIP R module is hereby limited, at ELEXOL's discretion, to replacement or repair.