IP I/O Manual

COLLABORATORS			
	TITLE : IP I/O Manual		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Göran Hasse and Robert Johansson	February 8, 2011	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The IP I/O module is an open-design device for distributed I/O over Ethernet. The new, second generation, IP I/O module (IP I/O II) has 4 digital input/outputs, 4 analog-to-digital, 4 digital-to-analog converters, a temperature sensor, a counter, a serial interface and an Ethernet interface. It has a DIN-rail mount, which makes it easy to install in industrial I/O installations.

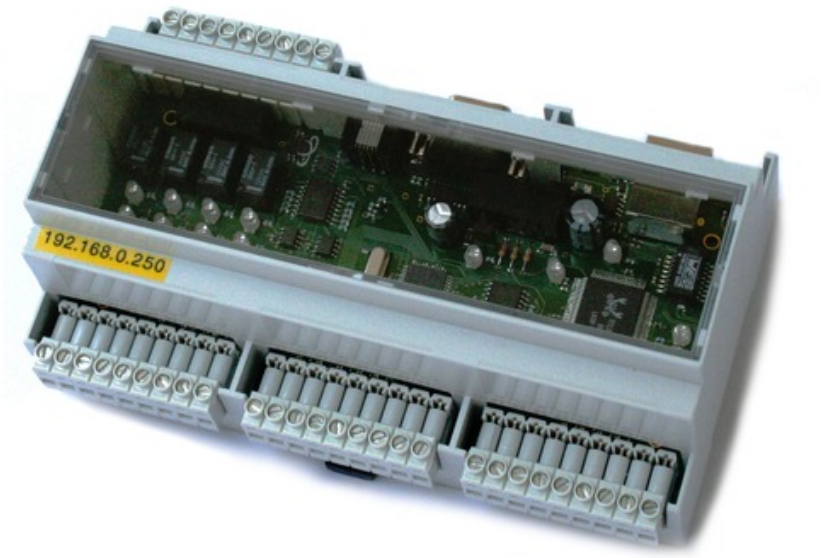

Figure 1.1: The IP I/O module (second generation).

1.1 About this document

This IP I/O manual is under active development. Please refer to the [IPIO-WWW] for the latest version of this document. The present version is incomplete and will be subject to updates and changes within the near future.

Chapter 2

Hardware

2.1 DIN rail mount

On the backside of the IP I/O module there is a DIN-rail mount. The DIN rail is a standardized (EN 50022, BS 5584, DIN 46277-3) 35 mm metal rail that is commonly used for mounting equipment in equipment racks.

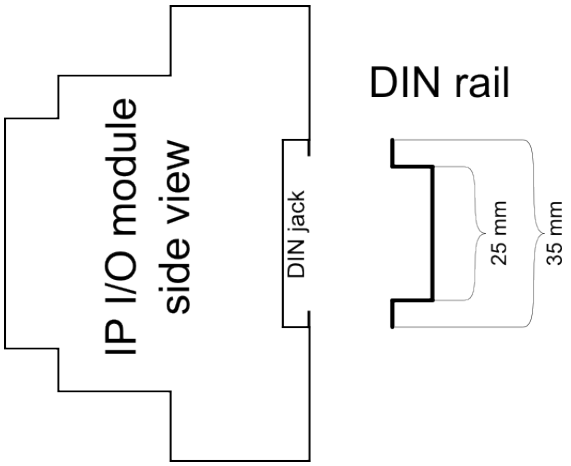

Figure 2.1: The DIN rail and the IP I/O module in profile.

2.2 Pin and diode layout

The pin and connector layout of the IP I/O module is shown in Figure 2.2. It also shows the placement of the diodes on the IP I/O circuit board, and an indication of their use.

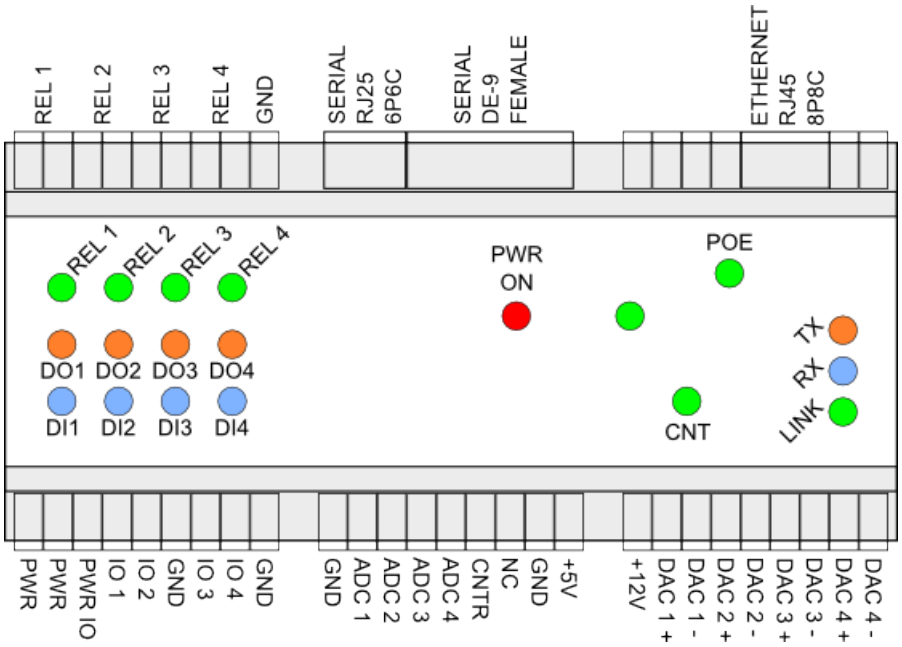

Figure 2.2: The pin and diode layout of the IP I/O Module.

2.3 Power supply

2.3.1 Power over Ethernet

The IP I/O module can be powered via POE (Power over Ethernet). This is the simplest and the recommended way of powering an IP I/O module.

2.3.2 External power source

The IP I/O module can also be powered by an external power supply, in which case a power source with voltage between 12 V and 48 V [verify] should be applied to the pins labelled PWR (see, e.g., Figure 2.2).

2.4 Electrical characteristics

2.4.1 Relays

2.4.2 Digital outputs and inputs

2.4.3 Digital-to-analog converters

2.4.4 Analog-to-digital converters

2.4.5 The counter

2.4.6 The thermometer

2.5 Application notes

In this section we give some examples of how electrical equipment can be connected to the input ports of the IP I/O module.

2.5.1 Digital inputs

To use the digital inputs on the IP I/O module, feed the PWR IO line with +12V. The digital input will then by default have potential equal to the feed voltage of the PWR IO line, and it will read state 0. When the input channel is pulled low, e.g. by short circuit the input line to ground, the state will change to 1. To limit the current from the digital input pin to ground it should be conencted through a ~1 kOhm resistor.

Chapter 3

Configuring IP address

3.1 Factory default

IP I/O modules have preconfigured IP and MAC addresses which are specified by a label on the module. The IP and the MAC addresses are stored in the IP I/O modules internal EEPROM memory, and they can both be reprogrammed easily.

IP Address	192.168.0.250
MAC Address	Label on the module (e.g., 02:00:00:00:00:FA)

Table 3.1: Factory default addresses.

3.1.1 Restoing the factory default settings

By pulling PORT B pin 5 LOW while the IP I/O is being restarted it is possible to restore the factory default IP address.

3.2 Finding the IP and MAC addresses

If the IP address has been reprogrammed and forgotten, it may be possible to identify its IP address by sending a broadcast ping (which the IP I/O module will reply to if it's on the same class-C network):

```
$ ping -b 192.168.0.255
PING 192.168.0.255 (192.168.0.255) 56(84) bytes of data.
64 bytes from 192.168.0.1: icmp_seq=1 ttl=255 time=1.06 ms
64 bytes from 192.168.0.4: icmp_seq=1 ttl=64 time=1.08 ms (DUP!)
64 bytes from 192.168.0.250: icmp_seq=1 ttl=64 time=5.96 ms (DUP!)
--- 192.168.0.255 ping statistics ---
2 packets transmitted, 2 received, +4 duplicates, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.586/2.482/5.968/2.341 ms
rob@rob-nozomi:~/Desktop/Raditex/FreeSCADA-3.0/freescada/trunk/doc/en/ipio-manual$
```

In this case the IP address 192.168.0.250 is assigned to an IP I/O module.

If the IP address is known, the MAC address can be extracted with the ipio_mac_get application:

```
$ ipio_mac_get 192.168.0.250
02:00:00:00:00:FA
$
```

3.3 Setting a new IP address

The IP address of an IP I/O module can be configured either with the ipio_setip command line tool, or the WinIPIO GUI application, as described below. In either way, the MAC address must be known to reset the IP address (see previous section).

3.3.1 The ipio_setip command-line tool

The IP I/O software package contains a command-line tool called ipio_setip, which takes three arguments: The interface name (e.g., "eth0"), the MAC address, and the IP address.

```
$ ipio_setip
usage: ipio_setip interface mac-addr ip-addr

  Assign a new IP address to a IP I/O module.

  MAC address format: aa:bb:cc:dd:ee:ff
  IP  address format: ddd.ddd.ddd.ddd

$ ipio_setip eth0
```

```
$ sudo ipio_setip eth0 02:00:00:00:00:FA 192.168.0.123
Sending request to unit with MAC address [on eth0]:
02:00:00:00:00:fa

New IP address:
192.168.0.123
$
```

Note that the ipio_setip application has to be run as root, because it sends a tweaked ARP packet to the IP I/O module in order to reset the IP address. The access to the low-level network needed for this requires root privileges.

3.3.2 The WinIPIO GUI application

The WinIPIO GUI application from the IP I/O software package can also be used for setting a new IP address (it uses the ipio_setip application behind the scenes). See Figure 4.2.

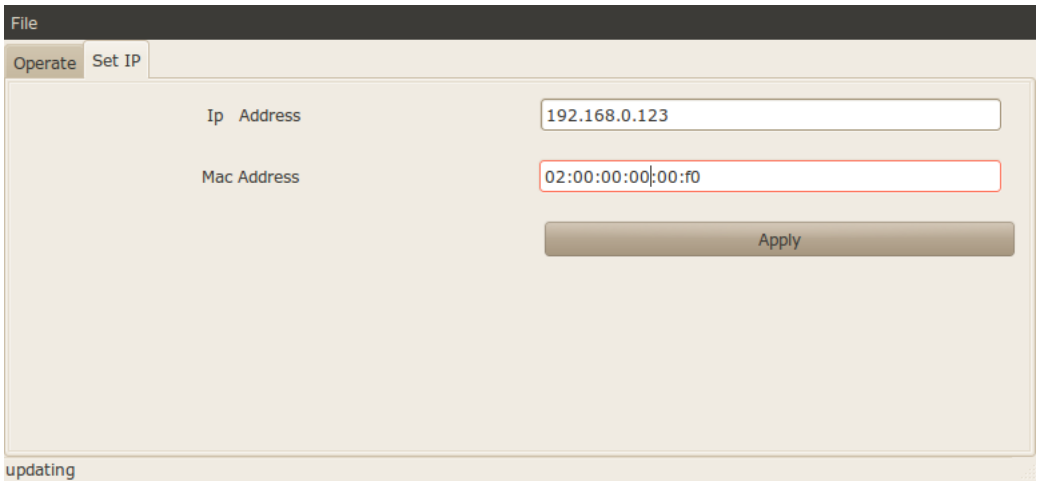

Figure 3.1: The "Set IP" tab in the WinIPIO applicaiton can be used to set a new IP address for a IP I/O module.

3.4 Advanced address configuration

The IP address is stored in the internal EEPROM of the IP I/O module, at address 64 to 67. The MAC address is stored at address 68 to 73. The EERPOM of can be read and written with the ipio_eeprom command-line application:

```
$ ipio_eeprom
usage: ipio_eeprom host addr command (range/value)
  addr    =  1,2,3,4
  command = read range | write value (value = 0 - 255)
$
```

The IP address and the MAC address can be read from the module as:

```
$ ipio_eeprom 192.168.0.250 64 read 4
EEPROM[64] = 192 = 0xC0
EEPROM[65] = 168 = 0xA8
EEPROM[66] = 0   = 0x00
EEPROM[67] = 250 = 0xFA
$
$ ipio_eeprom 192.168.0.250 68 read 6
EEPROM[68] = 2   = 0x02
EEPROM[69] = 0   = 0x00
EEPROM[70] = 0   = 0x00
EEPROM[71] = 0   = 0x00
EEPROM[72] = 0   = 0x00
EEPROM[73] = 250 = 0xFA
$
```

The IP and MAC can also be reprogrammed:

```
$ ipio_eeprom 192.168.0.250 64 write 192
$ ipio_eeprom 192.168.0.250 64 write 168
$ ipio_eeprom 192.168.0.250 65 write 25
$ ipio_eeprom 192.168.0.250 66 write 199
$ ipio_eeprom 192.168.0.250 67 read 4
EEPROM[64] = 192 = 0xC0
EEPROM[65] = 168 = 0xA8
EEPROM[66] = 25  = 0x19
EEPROM[67] = 199 = 0xC7
$
$ ipio_eeprom 192.168.0.250 71 write 10
$ ipio_eeprom 192.168.0.250 72 write 10
$ ipio_eeprom 192.168.0.250 68 read 6
EEPROM[68] = 2   = 0x02
EEPROM[69] = 0   = 0x00
EEPROM[70] = 0   = 0x00
EEPROM[71] = 10  = 0x0A
EEPROM[72] = 10  = 0x0A
EEPROM[73] = 250 = 0xFA
$
```

The new IP and MAC address is used from next hardware or software reset. A reset can be triggered with the ipio_reset application:

```
$ ipio_reset 192.168.0.250
$
```

Chapter 4

Software

The IP I/O software package includes demo GUI application, and a number of command-line applications that can be used for controlling IP I/O modules from the command line of from batch scripts. There is also a library for software development for the IP I/O module: the libipio-modbus.

4.1 WinIPIO

The WinIPIO application is a Graphical User Interface created with the GTK+ library. It is a demo and test application that can be used to control an IP I/O module, as well as for configuring IP addresses. Since it is distributed together with its source code, it is also a good starting point for custom application development. Internally, it uses the libipio-modbus library, see Section 4.3

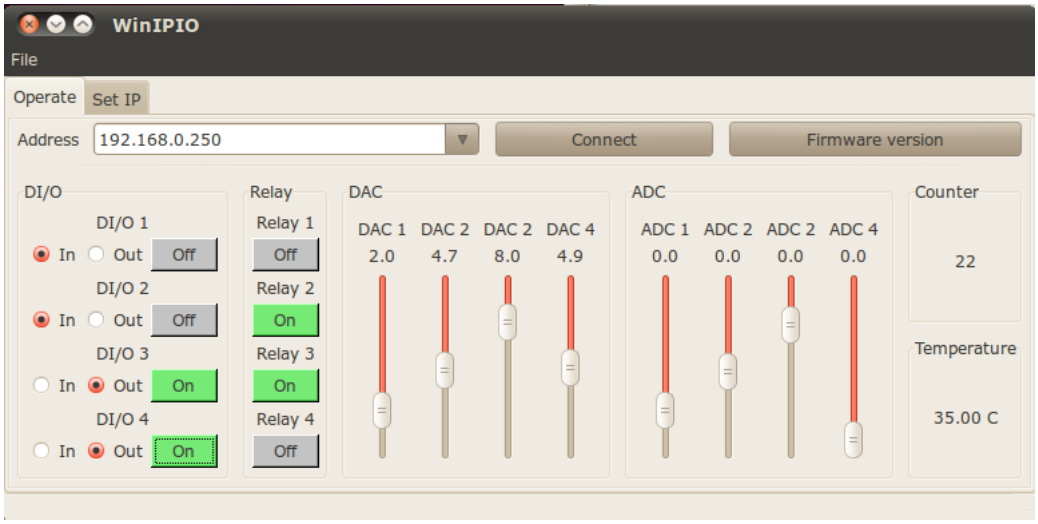

Figure 4.1: The main I/O control tab in the WinIPIO applicaiton can be used for setting/reading I/O, relay, and digital-to-analog converters, and for reading out the analog-to-digital converter, counter, and temperature values.

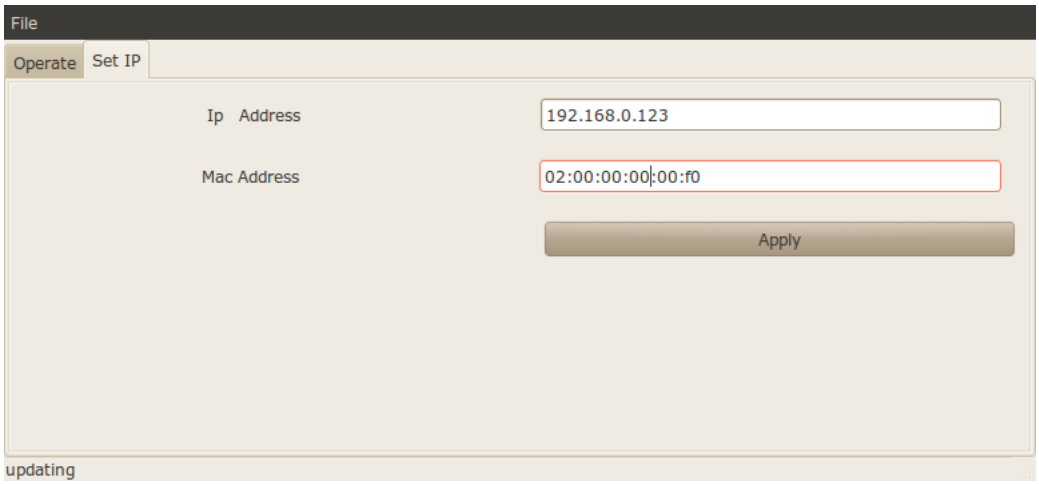

Figure 4.2: The "Set IP" tab in the WinIPIO applicaiton can be used to set a new IP address for a IP I/O module.

4.2 Command-line applications

Table 4.1 lists the command-line tools that are distributed with the IP I/O software package. Each command-line tool takes a different set of arguments, but all require a valid IP address to an IP I/O module as (first) argument. Run "ipio_xxx -h" at the command-line prompt for a description of the arguments for a specific command-line tool.

Command	Description
ipio_version	Get the firmware version of an IP I/O module.
ipio_reset	Trigger a software reset of an IP I/P module.
ipio_relay	Read/write a relay value.
ipio_digital_outputs	Set the value of a digital output channel.
ipio_digital_inputs	Read a digital input channel.
ipio_dac	Read/write the digital-to-analog converter value.
ipio_adc	Read a analog-to-digital converter value.
ipio_eeprom	Read/write to the internal EEPROM.
ipio_counter	Get the counter value.
ipio_temperature	Get the temperature reading.
ipio_setip	Set a new IP address.

Table 4.1: A summary of command-line tools for controlling IP I/O modules.

4.3 libipio-modbus: An Application Programmer's Interface for IP I/O communication

Although the IP I/O module supports the standard Modbus UDP communication protocol, see Chapter 5, we have created a wrapper library that encapsulate the Modbus specifics of the IP I/O communication, and provides a very simple and convenient Applications Programmer's Interface (API). The library is called libipio-modbus, and is written in the C programming language. Internally it communicates with the IP I/O modules using the Modbus protocol.

4.3.1 Building and linking

In order to use the libipio-modbus library, include the ipio-modbus.h file in your C source code, and link to the FreeSCADA "libmodbus" and "libipio-modbus" library (both included in the IP I/O software packages).

4.3.2 API reference

```
int ipio_relay_read(char *host, int addr);
int ipio_relay_write(char *host, int addr, int value);

int ipio_digital_output_read(char *host, int addr);
int ipio_digital_output_write(char *host, int addr, int value);

int ipio_digital_input_read(char *host, int addr);

int ipio_dac_read(char *host, int addr);
int ipio_dac_write(char *host, int addr, int value);

int ipio_adc_read(char *host, int addr);

int ipio_counter_read(char *host);

int ipio_version(char *host);

double ipio_temperature_read(char *host);
```

Chapter 5

Modbus UDP

The recommended way to communicate with the The IP I/O module is through Modbus UDP. Modbus is an industrial standard for communicating with control and data acquisition hardware such as the IP I/O module, and UDP is one of the transport protocols that can be used with Modbus. Devices and systems that support Modbus can be easily integrated with each other. The only information needed is which commands the device support, and what addresses its various resources are located at. This information, for the IP I/O module, is presented in this chapter.

We assume that the reader is familiar with the Modbus protocol and concepts. If not, please refer to the Modbus Protocol Specification [MODBUS-SPEC]. However, we start with a short summary of the Modbus protocol: see Table 5.1.

Field	Description	Length
Transaction Identifier	Arbitrary. Set by the client, echoed by the server in the response.	2 bytes
Protocol Identifier	Always 0x0000 for Modbus UDP.	2 bytes
Length	Number of remaining bytes in the packet (including the Unit and Function bytes below, so it is never smaller than 0x0002).	2 bytes
Unit Identifier	Address within the bus (not used in Modbus TCP/UDP, and should always be 0x00).	1 byte
Function code	Defines the function of request.	1 byte
Data	The format depends on the function code. See tables below.	X bytes

Table 5.1: A description of the fields in the Modbus protocol.

5.1 Modbus functions

The Modbus functions (commands) that are used by the IP I/O module, and their corresponding function codes, are summarized in Table 5.2.

Modbus function	Function code	Description
READ_COIL_STATUS	0x01	Used for reading on/off (1-bit) input resources.
READ_DISCRETE_INPUTS	0x02	Used for reading on/off (1-bit) input resources.
READ_HOLDING_REGISTERS	0x03	Used for reading register values (16-bit addressing).
FORCE_SINGLE_COIL	0x05	Used for writing on/off (1-bit) input resources.
PRESET_SINGLE_REGISTER	0x06	Used for writing a single 16-bit register value (16-bit addressing).

Table 5.2: Modbus functions and their corresponding function codes (selection, see [MODBUS-SPEC] for a complete list).

5.2 IP I/O resources and their Modbus function and address mapping

The following tables define which Modbus functions and addresses that should be used for read/write operations for the various IP I/O resources.

5.2.1 1-bit commands (I/O and relays)

Table 5.3, Table 5.4, and Table 5.5 describe the bit based IP I/O resources.

Hardware	Read command	Write command	Address
Digital Input 0 (di0)	READ_DISCRETE_INPUTS	-	0x0000
Digital Input 1 (di0)	READ_DISCRETE_INPUTS	-	0x0001
Digital Input 1 (di0)	READ_DISCRETE_INPUTS	-	0x0002
Digital Input 1 (di0)	READ_DISCRETE_INPUTS	-	0x0003

Table 5.3: Distrete input address mapping

Hardware	Read command	Write command	Address
Digital Output 0 (do0)	-	FORCE_SINGLE_COIL	0x0000
Digital Output 1 (do1)	-	FORCE_SINGLE_COIL	0x0001
Digital Output 2 (do2)	-	FORCE_SINGLE_COIL	0x0002
Digital Output 3 (do3)	-	FORCE_SINGLE_COIL	0x0003

Table 5.4: Distrete output address mapping

Hardware	Read command	Write Address	Write command	Write Address
Relay 0 (relay0)	READ_COIL_STATUS	0x0000	FORCE_SINGLE_COIL	0x0004
Relay 1 (relay1)	READ_COIL_STATUS	0x0001	FORCE_SINGLE_COIL	0x0005
Relay 2 (relay2)	READ_COIL_STATUS	0x0002	FORCE_SINGLE_COIL	0x0006
Relay 3 (relay3)	READ_COIL_STATUS	0x0003	FORCE_SINGLE_COIL	0x0007

Table 5.5: Relay address mapping

5.2.2 8-bit commands (register based)

Table 5.6, Table 5.7, and Table 5.8 describe the register based IP I/O resources.

5.3 Request and reply data formats

The format used in the data field for request and replies for the various IP I/O resources is defined in Table 5.9. Use the the tables in previous section to lookup which Modbus function the read and write requests for each resource should be paired with.

Hardware	Read command	Write command	Address
Digital-Analog Converter 0 (dac0)	READ_HOLDING_REGISTER	PRESET_SINGLE_REGISTER	0x0000
Digital-Analog Converter 1 (dac1)	READ_HOLDING_REGISTER	PRESET_SINGLE_REGISTER	0x0001
Digital-Analog Converter 2 (dac2)	READ_HOLDING_REGISTER	PRESET_SINGLE_REGISTER	0x0002
Digital-Analog Converter 3 (dac3)	READ_HOLDING_REGISTER	PRESET_SINGLE_REGISTER	0x0003

Table 5.6: Digital-to-Analog Converter (DAC) mapping

Hardware	Read command	Write command	Address
Analog-Digital Converter 0 (adc0)	READ_HOLDING_REGISTER	-	0x0004
Analog-Digital Converter 0 (adc1)	READ_HOLDING_REGISTER	-	0x0005
Analog-Digital Converter 0 (adc2)	READ_HOLDING_REGISTER	-	0x0006
Analog-Digital Converter 0 (adc3)	READ_HOLDING_REGISTER	-	0x0007

Table 5.7: Analog-to-Digital Converter (ADC) mapping

Resource	Read command	Write command	Address
Counter	READ_HOLDING_REGISTER	PRESET_SINGLE_REGISTER	0x0009
Temperature	READ_HOLDING_REGISTER	-	0x000A
Version	READ_HOLDING_REGISTER	-	0x000B
Reset	-	PRESET_SINGLE_REGISTER	0x000C
USART	READ_HOLDING_REGISTER	PRESET_SINGLE_REGISTER	0x000D
EEPROM	READ_HOLDING_REGISTER	PRESET_SINGLE_REGISTER	0x01XX

Table 5.8: Miscellanous hardware/software resources in the IP I/O module.

IP I/O resource	Request format	Reply format
Digital Input (read)	Start address (2 bytes). Number of values (2 bytes). Example: 0x00, 0x00, 0x00, 0x01	Length (1 byte). 1-byte bit mask encoding the 4 input states (least significant bits).
Digital Output (read)	Start address (2 bytes). Number of values (2 bytes).	Length (1 byte). 1-byte bit mask encoding the 4 output states (least significant bits).
Digital Output (write)	Address (2 bytes). Value (2 bytes): 0xFF00 or 0x0000.	echo of request
Relay (read)	Start address (2 bytes). Number of values (2 bytes). Example: 0x00, 0x00, 0x00, 0x01	Length (1 byte). 1-byte bit mask encoding the 4 input states (least significant bits).
Relay (write)	Address (2 bytes). Value (2 bytes): 0xFF00 or 0x0000.	echo of request
Digital to Analog (write)	Address (2 bytes). Value (2 bytes), only least significant byte is used. Example: 0x00, 0x0X, 0x00, 0xAA	echo of request
Analog to Digital (read)	Address (2 bytes). Number of values (2 bytes), always 0x0001. Example: 0x00, 0x0X, 0x00, 0x01	Length (1 byte), always 0x01. Value (1 byte), 8-bit integer encoding the ADC value (0 - 255).
Counter (read)	Start address (2 bytes). Number of values (2 bytes). Example: 0x00, 0x09, 0x00, 0x01	Length (1 byte), always 0x04. Value (4 byte), 32-bit integer encoding the counter value.
Tempeture (read)	Address (2 bytes). Number of values (2 bytes), always 0x0001. Example: 0x00, 0x0A, 0x00, 0x01	Length (1 byte), always 0x01. Value (1 byte), 8-bit signed integer encoding the temperature (-127 to 127).
Version (read)	Address (2 bytes). Number of values (2 bytes), always 0x0001. Example: 0x00, 0x0B, 0x00, 0x01	Length (1 byte), always 0x01. Value (1 byte), 8-bit integer encoding the firmware version number.
Reset (write)	Address (2 bytes). Value (2 bytes). Example: 0x00, 0x0C, 0x00, 0xFF	-
EEPROM (read)	Address (2 bytes). Number of values (2 bytes), always 0x0001. Example: 0x01, 0xAA, 0x00, 0x01	Length (1 byte), always 0x01. Value (1 byte), 1 byte read from EEPROM at addres 0xAA.
EEPROM (write)	Address (2 bytes). Value (2 bytes): 0xFF00 or 0x0000. Example: 0x01, 0xXX, 0x00, 0xYY	echo of request
USART (read)	Address (2 bytes). Number of values (2 bytes), always 0x0001. Example: 0x00, 0x0C, 0x00, 0x01	Length (1 byte), always 0x01. Value (1 byte), 1 byte read from USART.
USART (write)	Address (2 bytes). Value (2 bytes), only least significant byte used. Example: 0x00, 0x0C, 0x00, 0xYY. The value 0xYY is written to the USART.	echo of request

Table 5.9: Modbus request/reply data formats for IP I/O resources.

Chapter 6

Binary protocol

6.1 IP I/O custom protocol

In addition to Modbus UDP, there is a custom binary protocol that can be used to communicate with an IP I/O module. This binary protocol is succeeded by the Modbus UDP protocol (described in the previous chapter) and it should not be used in new application. The protocol is described here and remains in the IP I/O firmware for backward compatibility.

Command	Description
IPIOMOD_STATUS	Read out the entire status of the IP I/O module.
IPIOMOD_RELAY_WRITE	Write all relays (from relay mask)
IPIOMOD_RELAY_READ	Read all relays.
IPIOMOD_IO_WRITE	Write all digital outputs.
IPIOMOD_IO_READ	Read all digital inputs
IPIOMOD_DAC_WRITE	Set digital-to-analog converter.
IPIOMOD_DAC_READ	Read back the digital-to-analog converter value.
IPIOMOD_RESET	Reset the IP I/O module. Causes a watchdog timeout and reboot.
IPIOMOD_DAC_CALDATA	Write DAC calibration data.
IPIOMOD_COUNTER_READ	Read the counter value.
IPIOMOD_COUNTER_SAVE	Store the counter value.
IPIOMOD_ADC_READ	Read the analog-to-digital converter.
IPIOMOD_EEPROM_WRITE	Write to the EEPROM.
IPIOMOD_EEPROM_READ	Read from the EEPROM.
IPIOMOD_M24512_WRITE	Write to the external EEPROM.
IPIOMOD_M24512_READ	Read from the external EEPROM.
IPIOMOD_TEMP_READ	Read temperature value.

Table 6.1: IP I/O commands.

This section is currently under construction. For now, see the user manual for the first generation IP I/O module for more information about this protocol.

Chapter 7

Bibliography

[IPIO-WWW] *The IP I/O project's home page.*

[MODBUS-SPEC] *Modbus Application Protocol V1.1b*, 28 December 2006.