



# **EZ-Red**

Power I/O module for PC

See <http://www.xonelectronics.it> for other manuals

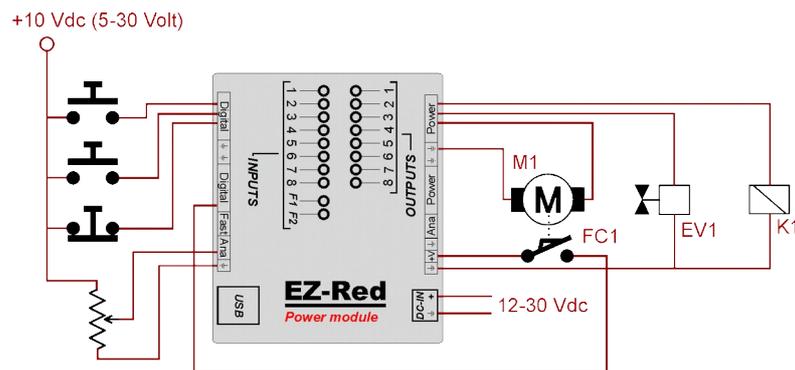
## **Index**

Introduction.....	2
Power supply.....	2
Digital inputs (24 volts).....	3
Fast, opto-coupled digital inputs.....	3
Analog inputs.....	3
24V power outputs.....	4
Analog outputs 0-10V.....	4
Auxiliary output power supply @12 volt.....	4
Process control and data acquisition with a PC.....	5
Using the supplied DLL or the terminal emulation mode (text commands).....	5
Internal PLC.....	5
Internal watch-dog (integrity control).....	5
Configuration variables.....	6
USB interface.....	6
Driver installation.....	6
"ezreddll.dll" library usage.....	7
External function declaration.....	7
C#.....	7
Delphi (pascal).....	7
C++.....	7
Visual basic.....	7
List of functions exported from the library.....	8
Functions details.....	9
ezrd_open(comport: integer).....	9
ezrd_close.....	9
ezrd_model: PChar (or char* in "C").....	9
ezrd_int_ios(var ings, outs, da1, da2, an1, an2: byte).....	9
ezrd_set_allouts(parm: cardinal).....	11
ezrd_set_io(which1_8: cardinal; onoff: cardinal).....	11
ezrd_up1..ezrd_up8.....	11
ezrd_dn1..ezrd_dn8.....	11
ezrd_set_ana1(parm: cardinal).....	11
ezrd_set_ana2(parm: cardinal).....	11
ezrd_read_ain1(var parm: cardinal).....	11
ezrd_read_ain2(var parm: cardinal).....	11
ezrd_read_ios(var onoff: cardinal).....	11
ezrd_isup_io(which1_8: cardinal; var onoff: cardinal).....	11
ezrd_isup1(var onoff: cardinal)..ezrd_isup8(var onoff: cardinal).....	12
ezrd_readencoder(var flags, encoder: cardinal).....	12
ezrd_SetWatchdog(tmout, ioconf, aout1, aout2: cardinal).....	12

## Introduction

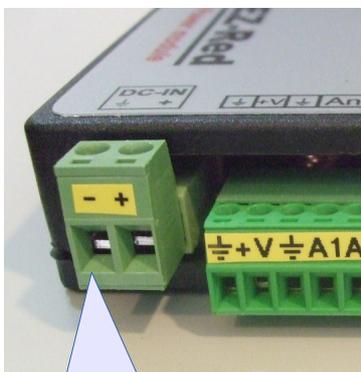
EZ-Red is a power interface module which connects through the USB interface to a computer in order to drive external electric devices at 24 volts. The module features:

- 8 digital inputs, which accept voltage from 5 to 30 volts
- 8 digital power outputs at 24 volts, 2A, which can drive inductive loads
  - output voltage is the same as power supply: from 12 to 30 volts
  - maximum output current of all the power outputs together is 4A
- 2 fast (10 KHz) opto-coupled inputs, usable as is or suitable for a quadrature encoder
- 2 analog inputs 0-10V
- 2 analog outputs 0-10V
- 1 auxiliary, stabilized output power supply at 12 volts (13.5 volts), 200 mA
- Continuous monitoring on power outputs for overload and open circuit
- Continuous monitoring on power supply voltages
- Internal, configurable watch-dog, to protect from computer hang-ups
- Internal PLC which stores and executes autonomous cycles
- Flash, non volatile memory to store PLC program and configuration data



*This example shows a few possible connections: switches, variable resistors, electro valves, relays and small DC motors.*

### Power supply

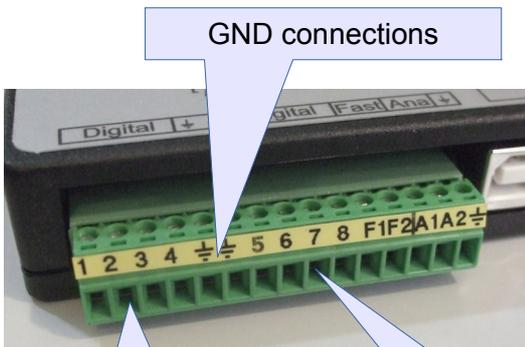


Power supply input

The device must be powered by DC voltage from 12 to 30 volts. The current consumption for the EZ-Red alone is 200 mA; to this consumption must be added the ones drawn by the external devices connected to the power outputs.

An auxiliary power supply (+V) is generated internally. Both external and internal-generated power supplies are monitored by EZ-Red. The device can be powered with voltages lower than 15 volts (but anyway higher than 10 volts); in this case the analog outputs will not reach their full 10 volt peak, and will lose linearity.

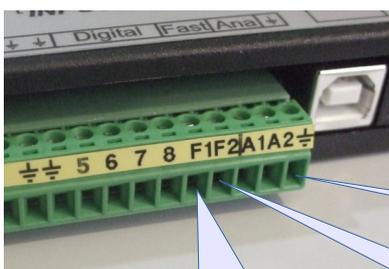
### **Digital inputs (24 volts)**



The eight digital inputs X1..X8 have an input impedance of 10 Kohm and a protecting circuitry that limits the maximum frequency of the input signals to about 1 Khz. Two of these inputs, X1 e X2, have an associated hardware counter which can count the rising edges of the input signal without CPU waste.

Digital inputs 1, 2, 3, 4      Digital inputs 5, 6, 7, 8

### **Fast, opto-coupled digital inputs**



The two inputs FX1 and FX2 (F1 and F2 on the connector) are opto-coupled and have a hardware counter and an interface for channels A and B of a quadrature encoder, with frequencies up to 10 Khz. The three GND (ground) pins are internally connected.

Input FX1      Input FX2      Additional GND (ground)

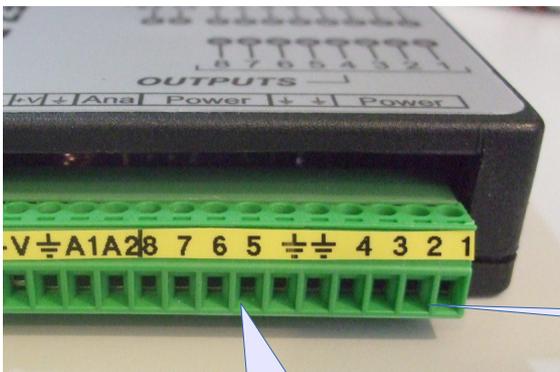
### **Analog inputs**



The two analog inputs AIN1 and AIN2 (A1 and A2 on the connector) accept analog voltages from 0 to 10 volts; there is a protection circuit which limits voltage to 10 volts.

Ingresso analogico AIN1      Ingresso analogico AIN2

### 24V power outputs



The eight power outputs Y1..Y8 can source, each, up to 2 amperes, but no more than 4A in total (all the currents summed up). They output the same voltage supplied as main power supply, and are protected and monitored.

Their status (overload and open/broken circuit) can be read by the computer or a PLC program.

Outputs 1, 2, 3, 4

Outputs 5, 6, 7, 8

### Analog outputs 0-10V

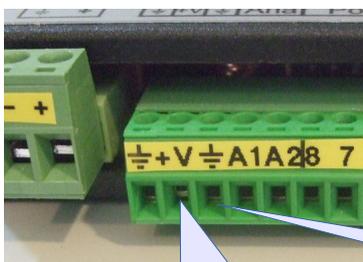


The analog outputs AOUT1 and AOUT2 (A1 and A2 on the connector) can source about 20 mA at maximum, and feature a protection circuit. The output voltage ranges from 0 to 10 volts.

Analog output AOUT2

Analog output AOUT1

### Auxiliary output power supply @12 volt



On the output side connector there is an auxiliary power supply generated internally by the EZ-Red. The voltage is stabilized at 13,4 volts, with maximum output current of 200 mA, and is protected by an auto-resetting fuse.

All the GND are connected together

Power supply output 12V 200mA

## **Process control and data acquisition with a PC**

Using its USB interface, it is possible to dialogate with EZ-Red in order to read its inputs and set its output. There are two possible ways to interact; using the supplied windows library "ezreddll.dll" or using a simple text-based protocol in a terminal emulation mode, like a modem. The DLL is suitable for high-level compiled languages, including LabView®, and frees the application to deal with low level communication details; the text commands mode (console) is more simple but still effective, even if the application must implement a normal bi-directional serial communication.

### **Using the supplied DLL or the terminal emulation mode (text commands)**



```

COM4 38400 bps, 8N1, no handshake
OK
>ylampmask1=255
OK
?y1
Y1=0
?y1

```

This manual describes the library EZRedDLL and its usage. For info about simple text commands please refer to the manual "**Terminal emulation (text commands)**".

The terminal emulation mode is also suitable for using EZ-Red under Linux.

## **Internal PLC**

The EZ-Red module contains an internal PLC, which can aid the computer in carrying out strict-time duties, or use the EZ-Red in stand-alone mode, without a computer. In order to utilize the PLC, a program (PLC cycle) must be written and stored in the non-volatile memory of the module.

To learn more about the programming language and how to write a PLC cycle, please refer to the manual "**Programming manual**". The program must then be compiled and transferred to the EZ-Red using the supplied utility, TSMon.exe, which is a true programming environment. See the "**TSMon user manual**".

The stored PLC program can be protected with a password (by the way, it is not possible to read the internal PLC memory anyway) and, additionally, a PLC program can set a bit which inhibits the USB communication. This bit can be reset by the PC if the correct password is known.

## **Internal watch-dog (integrity control)**

When the EZ-Red is employed, together with a PC, in a process control, it can be desirable that, if the computer stops to work, the process is brought in a secure state, a state where the outputs are set in a known and safe configuration.

The watch-dog, if enabled, monitors the communication with the computer to ensure that the computer (and the communication with it) is working: if for some time ("too much time") the communication ceases, the system is assumed to be unsafe because not responsive; the watch-dog fires and sets the outputs to a determined pattern. The time-out (acceptable delay in communication) is configurable, as is the time-out. Another effect the watch-dog can be programmed to have, is to stop the running PLC cycle.

The watch-dog can also be instructed to fire if an error condition arises on the power outputs, either overload or open circuit, or a failure in the power supply. At power-on the watch-dog is disabled; it can be configured and enabled by the PC or by the PLC cycle (if present) executed automatically at start-up.

## Configuration variables

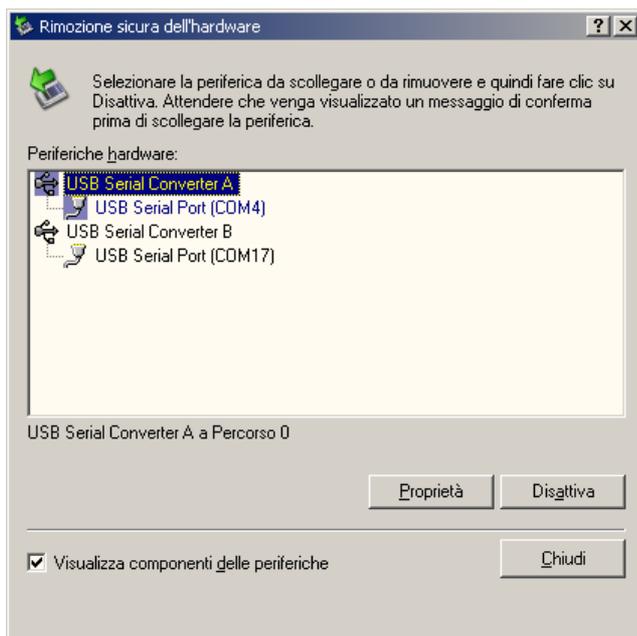
In the EZ-Red flash memory it is possible to store not only a PLC cycle and a password, but also a set of configuration data. This way a single cycle can be adapted to different cases, and the customer can be supplied with an ad-hoc program to modify the configuration data without touching the PLC program. The PLC cycle can also modify this configuration variables, so the value will be retained for future sessions.

## USB interface



The EZ-Red module connects to the computer through an USB interface, with a standard connector USB “Type B” shown aside.

When the module is powered, the USB cable connected and o.s. drivers installed, two virtual serial ports will be available; the first one (*USB Serial converter A*) is the communication channel, while the second (*USB Serial converter B*) is not used.



This picture shows that the first channel, “A”, is associated the virtual port COM4. This is the port to refer to when initiating a communication to the EZ-Red.

The name of the port can be different than this one (COM4): it depends on the operating system. Using the Control Panel of the o.s. it is possible, partly, to change the assigned name.

## Driver installation

In order to use the USB interface, the correct FTDI drivers must be installed. They can be downloaded from XON Electronics web site, or directly from the FTDI web site ([www.ftdichip.com](http://www.ftdichip.com)). If these driver are not already present in the computer, they will be asked for as soon as the EZ-Red device is detected from the operating system. Whatever is the case, to install the drivers (VCP type, Virtual COM Port) a file containing them must be downloaded and extracted to any directory. Then, it is possible to right-click on the file “FTDIPIPORT.INF” and select “Install” from the pop-up menu, or turn the EZ-Red on and connect it to the computer to make the o.s. ask for the drivers: then direct the o.s. to the directory where the files where extracted.

## "ezreddll.dll" library usage

---

Place a copy of the library in the same directory where the application resides, or into a common path of the operating system (for example "[C:\WINDOWS](#)" - but this is o.s. dependent). Then declare, inside your application, the exported functions of ezreddll explained next in this document. Only the needed functions have to be declared; it is possible to use the EZ-Red by declaring only two external functions.

All the exported functions have a **lowercase name**: they are written in uppercase in this manual for greater evidence, but in the real library their name is lowercase.

Most of the function results are 32 bits signed integers; EZRD\_CLOSE does not return a result and EZRD\_MODEL returns a "C" string (a string terminated by a NUL character, which is normal in Windows®). The arguments of the functions vary; all the functions use the *stdcall* model, which is the standard for Windows®. The "Cardinal" type denotes an unsigned 32 bit integer.

An application must invoke EZRD\_OPEN prior to any other library invocation; pass to this function EZRD\_OPEN the port number as seen before. The function returns 0 (zero) in case of success or a non-zero number otherwise, for example if the specified port does not exists or is already is use.

### External function declaration

Some declaration example, for different languages, is shown here for reference.

#### **C#**

```
public class ExternalEZRD {
    [DllImport("ezreddll.dll")]
    public static extern int ezrd_open(int comport);
    [DllImport("ezreddll.dll")]
    public static extern void ezrd_close();
    [DllImport("ezreddll.dll")]
    public static extern int ezrd_int_ios(ref byte ings, ref byte outs, ref byte da1,
        ref byte da2, ref byte an1, ref byte an2);
}
```

#### **Delphi (pascal)**

```
mydll = 'ezreddll.dll';
function ezrd_open(comport: integer): integer; stdcall; external mydll;
procedure ezrd_close; stdcall; external mydll;
function ezrd_int_ios(var ings, outs, da1, da2, an1, an2: byte): integer; stdcall;
    external mydll;
```

#### **C++**

```
extern "C" __declspec(dllimport) __stdcall {
    int ezrd_open(int comport);
    void ezrd_close();
    int ezrd_int_ios(unsigned char* ings, outs, da1, da2, an1, an2);
}
```

#### **Visual basic**

```
Declare Function ezrd_open Lib "ezreddll.dll" (comport as integer) as integer
Declare Sub ezrd_close Lib "ezreddll.dll"
Declare Sub ezrd_int_ios Lib "ezreddll.dll" (ByRef ings as byte, ByRef outs as Byte, _
    ByRef da1 as Byte, ByRef da2 as Byte, ByRef an1 as Byte, ByRef an2 as Byte) as
integer
```



## List of functions exported from the library

Function name	Input parameters	Result	Description
EZRD_OPEN	COM number to use, like "3" for "COM3"	Integer; 0 means OK, otherwise an error code.	Call this before any other function.
EZRD_CLOSE	None	None	Optional; close communication with EZ-Red.
EZRD_MODEL	None	NUL-terminated string (C-style string)	Optional, to know model and firmware release of the device.
EZRD_INT_IOS	See details	0=OK, or error code<>0	Most general function to read all inputs and set all outputs.
EZRD_SET_ALLOUTS	Outputs pattern	0=OK, or error code<>0	Sets the 8 power outputs to a value.
EZRD_SET_IO	Output 1..8, and status 0 or 1	0=OK, or error code<>0	Sets a single output.
EZRD_UP1 ...to... EZRD_UP8	None	0=OK, or error code<>0	Turns on (UP) the specific power output.
EZRD_DN1 ...to... EZRD_DN8	None	0=OK, or error code<>0	Turns off (DN, down) the output.
EZRD_SET_ANA1 ...and... EZRD_SET_ANA2	A value from 0 to 255	0=OK, or error code<>0	Sets a voltage for the analog output proportional to the argument: 0=0 volts, 255=10 volts
EZRD_READ_AIN1 ...and... EZRD_READ_AIN2	Pointer (address) to the variable to fill	0=OK, or error code<>0	Reads the analog input, and writes a number from 0 to 255 in the specified variable.
EZRD_READ_IOS	Pointer (address) to the variable to fill	0=OK, or error code<>0	Reads all the digital inputs and writes them to the specified variable.
EZRD_ISUP_IO	Input number 1..8 and pointer to the variable to fill	0=OK, or error code<>0	Reads a single digital input and fills the indicated variable.
EZRD_ISUP1 ...to... EZRD_ISUP8	Pointer (address) to the variable to fill	0=OK, or error code<>0	Reads the input and fills the indicated variable.
EZRD_READENCODER	Pointer for flags and pointer for variable to fill with encoder position	0=OK, or error code<>0	Reads encoder position and flags.
EZRD_PRESETENCODER	New value to set the encoder position to	0=OK, or error code<>0	Change the current position (PRESET) of the encoder.
EZRD_SETWATCHDOG	Time-out (ms), outputs pattern, values for the two analog outputs	0=OK, or error code<>0	Configures, enables or disables the internal EZ-Red watch-dog.

## Functions details

---

The following paragraphs explain in detail the single functions. The used syntax is Pascal (or Delphi). Every function which returns a result (those having a closed parenthesis followed by a colon), return 0 (zero) for success and non-zero for failure.

Most of the parameters are unsigned 32-bit integers. When the parameters are be passed by reference and not by value, is because the function modifies them. Passing by reference is denoted with the word VAR in pascal, with the symbol "&" (ampersand) in C, and with the word BYREF in basic. The C syntax is sometimes called "passing a pointer".

A pascal example like this:

```
if EZRD_READ_IOS(inputs)<>0 then writeln('Communication error.');
```

is written like this in C:

```
if ( ezrd_read_ios(&inputs) ) printf("Communication error.");
```

and like this in basic/VB:

```
If ezrd_read_ios(inputs)<>0 Then Print "Communication error."
```

### **ezrd\_open(comport: integer)**

It is used to initialize the library and the communication with EZ-Red. The USB interface creates two virtual COM ports: the one to use is labelled "A" - see which number is assigned to port "A".

This function must be called once, before access is allowed to the others. The library opens the indicated port and queries the device. If all the handshake phases succeed, the result is zero. If it is impossible to open the port (because busy, non-existent...) a Windows® error code is returned. If the port can be opened, but no EZ-Red is found, the error code 8 is returned.

### **ezrd\_close**

Closes the communication, freeing the port. There is no returned result, and it is not an error to close the communication even if none had been opened before. After the call to this function, all the other functions will return an error, because the communication is no more valid.

### **ezrd\_model: PChar (or char\* in "C")**

This function does not return an integer but a pointer (a C nul-terminated string). The string itself contains, in text mode, name and release of the device and its firmware like "EZ-RED 1.2".

The returned pointer must be considered read-only; it can be NULL and it can point to a NUL byte; in both cases this means that an error has happened.

### **ezrd\_int\_ios(var ings, outs, da1, da2, an1, an2: byte)**

This function is the core to read and write all inputs and outputs of the device. This function is rather complicated, there are other more simple than this.

All the arguments to the function are 8-bit unsigned integer, and must be passed BY REFERENCE (not

by value). Prior to execute the invocation the arguments must be set up in the following way:

- ings**            bit pattern (mask) to indicate the power outputs to operate on:  
                  (bit0=output 1 ... bit7=output 8)  
                  Bits that are 1 let the output be changed, bits that are 0 leave it as is.
- outs**            bit pattern to indicate the new state of the output, if the corresponding  
                  bit of “ings” is 1. A bit of “outs” is ignored if the corresponding bit  
                  of “ings” is 0.

For example, to modify only the power output 1, set `ings=1`; then set `outs=1` (to turn on) or `outs=0` (to turn the output off). The only bit set to 1 in `INGS` says that the only output that must be changed is the first, corresponding to bit0. The other bits in `OUTS` will be ignored so the corresponding outputs will remain untouched. To operate on all output together, use `INGS=255`, and the desired pattern, one bit for every output, in `OUTS`: to turn all outputs on, `OUTS=255`, to turn all outputs off, `OUTS=0`. To operate on the analog outputs 1 & 2, follow the same principle; the argument `AN2` must contain a bit set to 1 for an analog output to be updated; only the first two bits count, because there are only two analog outputs. The arguments `DA1` and `DA2` contain the intended value (from 0 to 255), used only if enabled by the bits in `AN2`.

For example, in C, to read inputs without modifying any outputs:

```
ings=0; an2=0;     // no outputs will be affected
res=ezrd_int_ios(&ings, &outs, &da1, &da2, &an1, &an2);
```

The following updates only the analog output 1:

```
da1=128;            // about 5 volts
ings=0; an2=1;     // only bit0 of an2 set to 1, to update only analog output 1
res=ezrd_int_ios(&ings, &outs, &da1, &da2, &an1, &an2);
```

The following turns off all the power outputs and bring the analog outputs to 0 volts:

```
ings=255;           // all the bits set to 1: affect all the power outputs
outs=0;             // all the bits set to 0: all the output off
an2=3;              // b0 & b1 set to 1; update both analog outputs
da1=0; da2=0;       // values for the two analog outputs
res=ezrd_int_ios(&ings, &outs, &da1, &da2, &an1, &an2);
```

After the function returns, the variables are update to reflect the new states of input and outputs:

- ings**    records the state of digital inputs, bit by bit: b0=input 1, up to b7=input 8
- outs**    records, similarly, the state of the digital (power) outputs
- da1**    contains the value, from 0 to 255, of the DAC of the analog output 1
- da2**    contains the value, from 0 to 255, of the DAC of the analog output 2
- an1**    contains the value, from 0 to 255, proportional to the voltage at analog input 1
- an2**    contains the value, from 0 to 255, proportional to the voltage at analog input 2

The functions explained below are simpler, but require more communication. Given the latency time and speed of the (virtual) serial communication, the computer can send about 50 commands per second to the EZ-Red. Hence, by using two separate calls to read inputs and write outputs, the speed is halved.

### **ezrd\_set\_allouts(parm: cardinal)**

This function sets all the power outputs together, one bit for every output: bit0 corresponds to output 1 and so on. Please note that by using this function, you modify all the outputs.

A “cardinal” is an unsigned 32 bit integer.

### **ezrd\_set\_io(which1\_8: cardinal; onoff: cardinal)**

This function modifies a single output, specified by WHICH1\_8 (use a number from 1 to 8); the new state is ON if ONOFF is 1, and OFF if ONOFF is zero.

### **ezrd\_up1..ezrd\_up8**

These calls set a single power output to ON (the output affected is the one indicated by the name of the function).

### **ezrd\_dn1..ezrd\_dn8**

These functions turn off the corresponding output.

### **ezrd\_set\_ana1(parm: cardinal)**

Sets, with a value from 0 to 255, the output voltage of the analog output 1. The voltage is proportional to the indicated value: 0 means 0 volts and 255 means 10 volts; so, 128 means 5 volts.

### **ezrd\_set\_ana2(parm: cardinal)**

The same as before, for the analog output 2.

### **ezrd\_read\_ain1(var parm: cardinal)**

Reads the voltage at analog input 1, writing in PARM a value from 0 to 255 proportional to the voltage 0..10 volts.

### **ezrd\_read\_ain2(var parm: cardinal)**

The same as before, for analog input 2.

### **ezrd\_read\_ios(var onoff: cardinal)**

Reads all the digital inputs, modifying the argument ONOFF with a bit set for every input that is ON (has voltage on it).

### **ezrd\_isup\_io(which1\_8: cardinal; var onoff: cardinal)**

This function reads a single digital input. Differently from the previous one, avoids to operate with bit masks. The argument WHICH1\_8 must contain a number from 1 to 8 that specifies which input to read; the argument ONOFF will contain 0 or 1 reflecting the state of the input. A “cardinal” is an unsigned 32 bit integer.

**[ezrd\\_isup1\(var onoff: cardinal\)..ezrd\\_isup8\(var onoff: cardinal\)](#)**

These functions read the indicated input (by the name of the function itself) and fill the passed argument, which is a 32 bit integer.

**[ezrd\\_readencoder\(var flags, encoder: cardinal\)](#)**

Reads the encoder position (if an encoder is connected to the fast inputs FX1 and FX2) and a set of flags related in general to the device. Both the arguments are passed by reference, for the function to fill them. The returned flags are packed in a pattern; the following table explains them:

Bit position and name	Bit mask	Description
1 WDTFIRED	2	ON when the internal watch-dog is fired (in effect)
3 CYCLERUN	4	Indicates whether the internal PLC is executing a cycle (ON=run)
8 FX1	256	ON/OFF state of the fast input FX1
9 FX2	512	ON/OFF state of the fast input FX2
13 FLASHFAIL	8192	When the internal flash memory fails, this bit goes high

**[ezrd\\_SetWatchdog\(tmout, ioconf, aout1, aout2: cardinal\)](#)**

Configures and enables/disables the internal watch-dog for the communication time-out. The argument TMOUT must contain a value from 0 to 65535 which indicates the number of milliseconds to wait between a USB packet (command) and the next one. If this number is 0, there is no time-out and the watch-dog will not fire for a communication loss (but it can fire for other reasons). When TMOUT is non-zero, no USB packets can arrive more than TMOUT milliseconds after the previous one. If TMOUT is 2000, for example, the computer must send at least a command every 2 seconds, otherwise the communication is considered broken. When the watch-dog fires, it sets the outputs in the way IOCONF, AOUT1 and AOUT2 specify.

IOCONF must contain a bit pattern for the power outputs, a bit for every output (bit0=power output 1, and so on). The power-on status of this pattern is 192 (0xC0 in hexadecimal), which corresponds to outputs 1..6 off and output 7 and 8 on.

AOUT1 and AOUT2 specify, respectively, the analog voltages to set when the watch-dog fires; the power-on value is 0 for both analog outputs. Acceptable values range from 0 to 255, corresponding to voltages from 0 to 10 volts.



XON ELECTRONICS SRL  
 WWW.XONELECTRONICS.IT  
 INFO@XONELECTRONICS.IT

*Internet product page is <http://www.xonelectronics.it/prodotti/industriali/EZ-Red>*

*Please report any error or imprecision to [web@xonelectronics.it](mailto:web@xonelectronics.it)*