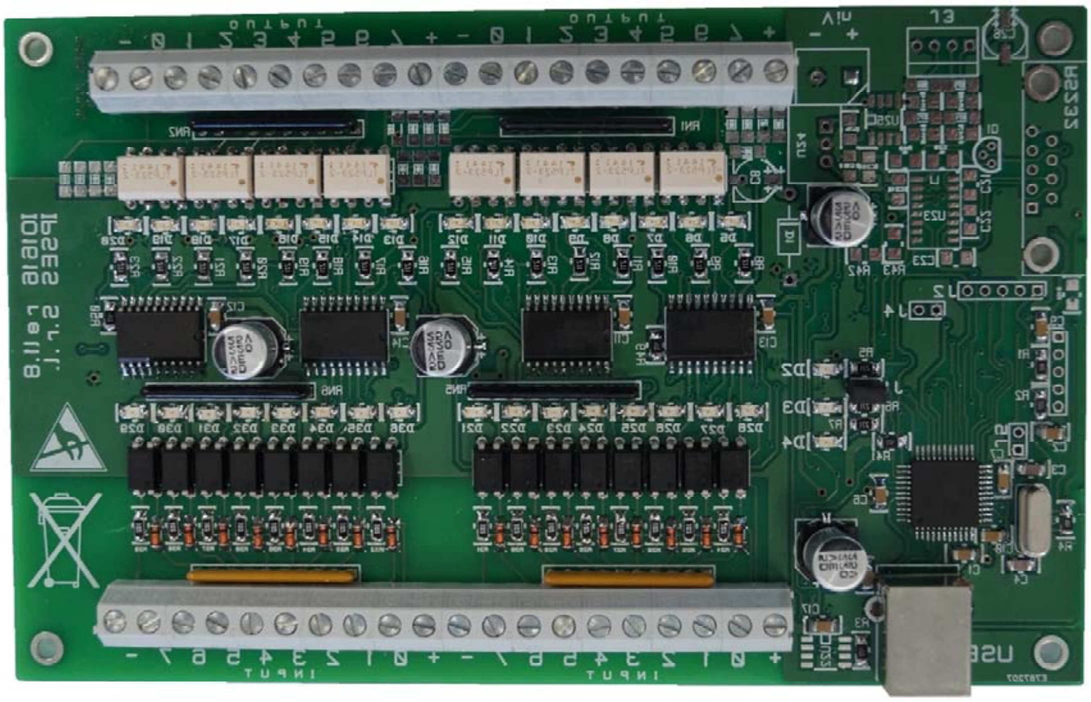
IO1616 Control Unit
USER MANUAL

Rel. 01.06.0001

(Hardware code: IO1616)



CONCEIVING
PLANNING
DEVELOPMENT
IN SCIENTIFIC
ELECTRONICS

www.ipses.com

GUARANTEE

IPSES warrants to the end-user in accordance with the following provisions that its branded hardware products, purchased by the end-user from IPSES company or an authorized IPSES distributor will be free from defects in materials, workmanship and design affecting normal use, for a period of one year as of the original purchase date. Products for which proper claims are made will, at IPSES's option, be repaired or replaced at IPSES's expense[1].

Exclusions

This Guarantee does not apply to defects resulting from: improper or inadequate installation, use or maintenance; actions or modifications by unauthorized third parties or the end-user; accidental or wilful damage or normal wear and tear.

Making a claim

Claims must be made by contacting IPSES office within the guarantee period.
Please, contact:

IPSES S.r.l. - Via Suor Lazzarotto, 10 - 20020 Cesate (MI) Italy
Tel. (+39) 02 39449519 – (+39) 02 320629547
Fax (+39) 02 700403170
http://www.ipses.com - e-mail: support@ipses.com

Limitation and Statutory Rights

IPSES makes no other warranty, guarantee or like statement other than as explicitly stated above and this Guarantee is given in place of all other guarantees whatsoever, to the fullest extent permitted by law. In the absence of applicable legislation, this Guarantee will be the end-user's sole and exclusive remedy against IPSES.

General Provisions

IPSES makes no express warranties or conditions beyond those stated in this warranty statement. IPSES disclaims all other warranties and conditions, express or implied, including without limitation implied warranties and conditions of merchantability and fitness for a particular purpose.
IPSES's responsibility for malfunctions and defects in hardware is limited to repair and replacement as set forth in this warranty statement.
IPSES does not accept liability beyond the remedies set forth in this warranty statement or liability for incidental or consequential damages, including without limitation any liability for products not being available for use or for lost data or software.

[1] With the exclusion of shipping costs for and from IPSES's development office.

WARNING!
ELECTRICAL DEVICES COULD DAMAGE EQUIPMENT OR PROPERTY OR CAUSE PERSONAL INJURY

This guide contains instructions and technical features of the IO1616 Control Unit.
Read with attention before attempting to install.
It is the responsibility of the technician to undertake all the safety rules provided by the law during the installation and the use of this device.
For any information which is not contained in this guide, please contact:

IPSES S.r.l. - Via Suor Lazzarotto, 10 - 20020 Cesate (MI) Italy
Tel. (+39) 02 39449519 – (+39) 02 320629547
Fax (+39) 02 700403170
http://www.ipses.com - e-mail: support@ipses.com

TABLE OF CONTENTS

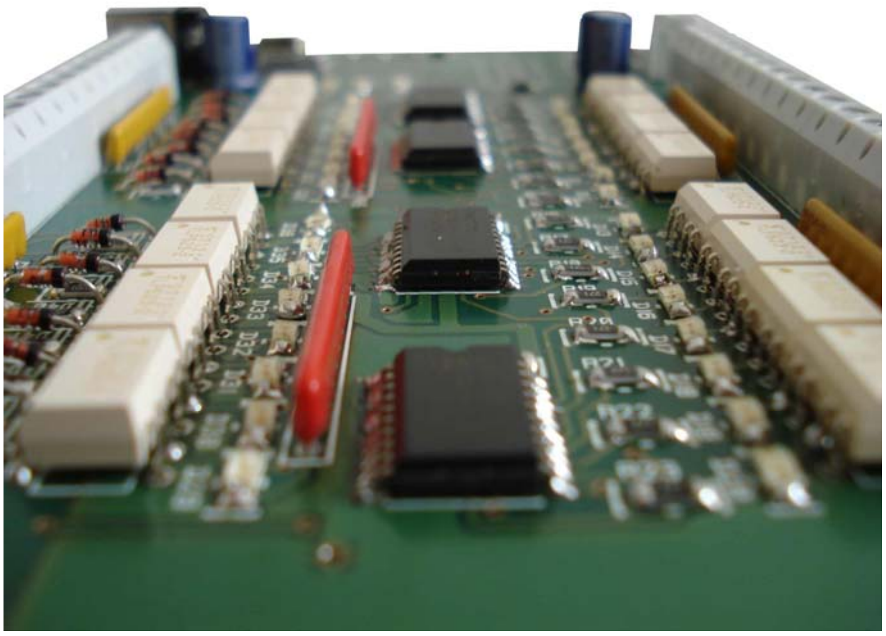
REVISION HISTORY

Manual revision history

Revision/ Date	Change description	Author
01.00.0000 August, 2008	First version Released	Mancuso C.
01.01.0000 January, 2009	Modified and integrated the Demo software and the Low Level Library functions description following the software upgrade with the S/N selection field. Other minor upgrade.	Rivolta A. Mancuso C.
01.02.0000 May, 2009	Modified input description in Picture 1. Added “ReadInputPort_All” in Low Level Library after the firmware upgrade. Other minor upgrade.	Rivolta A.
01.03.0000 June, 2009	Modified and integrated the Demo software and the Low Level Library functions description following the software upgrade with filters and mask for output managing, USB TimeOut and Bootloader mode.	Zancanato A.
01.04.0000 July, 2009	Added schematics examples with external load.	Zancanato A.
01.05.0000 October, 2009	Modified the Demo Software description due to the log file generation added. Modified the definitions and descriptions of the DLL functions due to a firmware restyling.	Rivolta A.
01.05.0001 June, 2015	Update document layout	Bottaccioli M.
01.06.0000 March 2016	Inputs electrical specifications updated Compatibility updated Picture updated	Bottaccioli M.
01.06.0001 August, 2016	Added ISO 9001:20015 logo	Bottaccioli M.

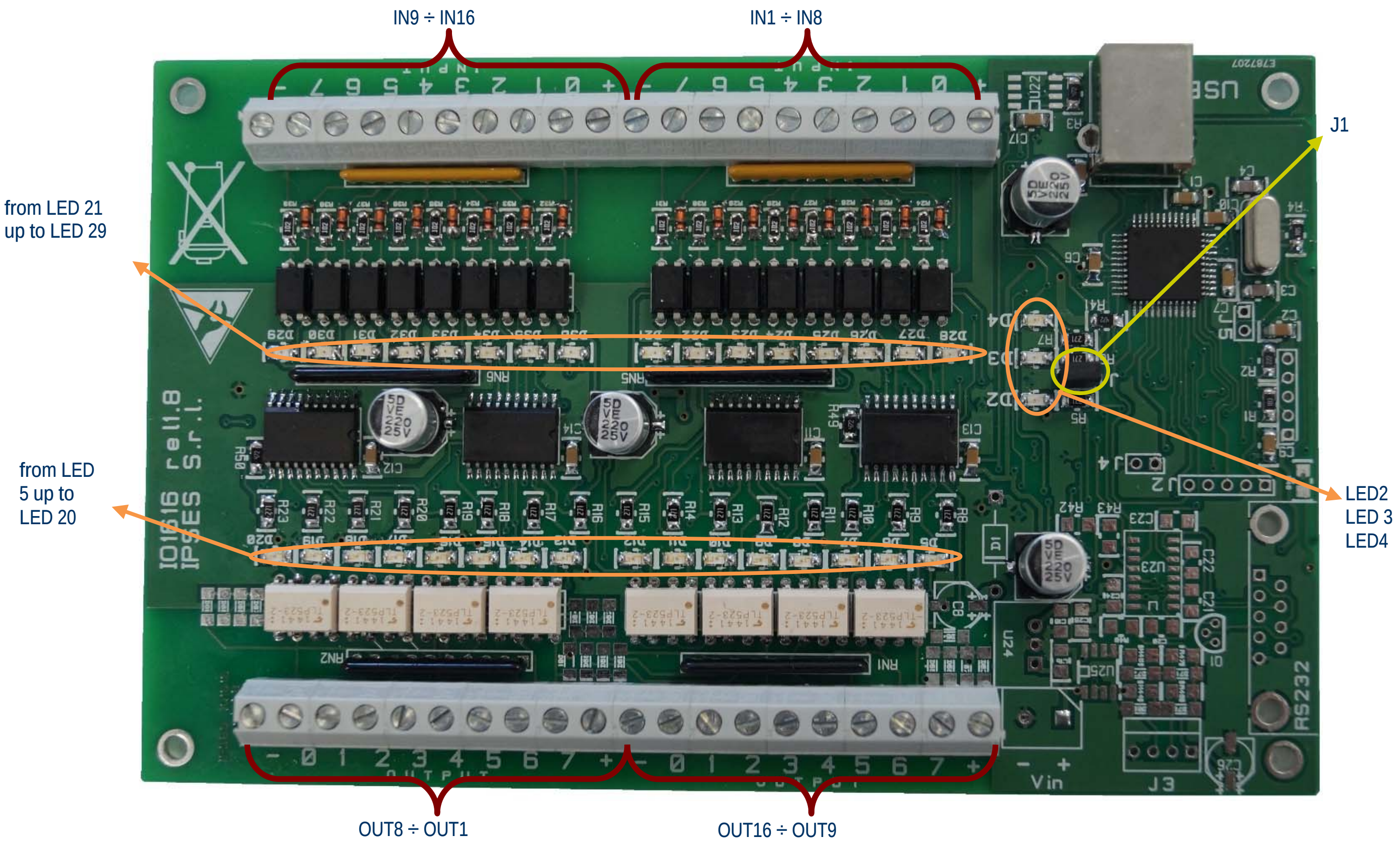
GENERAL FEATURES

IO1616 is a control unit integrated on *European Card Format* (160 x 100 mm – 6,30 x 3,94 inches) equipped with USB interface.
The card is self powered through USB: a connection to a PC allows the device to work.
IO1616 can check sixteen optocoupled inputs and driving sixteen optocoupled outputs. Both are reciprocally isolated in two groups of eight.
The control and the configuration of the device are achieved through USB interface, easy to use thanks to the driver and the *DLL* provided with, or using the simple demo software.
The board is equipped with a non volatile memory where it is possible to store the power-on status of every output.



CARD DESCRIPTION

IO1616 card is shown in the picture below: in the upper part of the card the sixteen outputs are divided in two groups of eight (numbered from 0 to 7 on the card serigraphy), and, similarly, in the lower part of the card there are the sixteen inputs (numbered and divided in the same way).



Picture 1: IO1616 card

The LEDs are:

D2	Green LED: System Power-on
D3	Green LED: Status LED (see relating paragraph)
D4	Red LED: Status LED (see relating paragraph)
D5	Red LED: OUT 16 activated
D6	Red LED: OUT 15 activated
D7	Red LED: OUT 14 activated
D8	Red LED: OUT 13 activated
D9	Red LED: OUT 12 activated
D10	Red LED: OUT 11 activated
D11	Red LED: OUT 10 activated

D12	Red LED: OUT 9 activated
D13	Red LED: OUT 8 activated
D14	Red LED: OUT 7 activated
D15	Red LED: OUT 6 activated
D16	Red LED: OUT 5 activated
D17	Red LED: OUT 4 activated
D18	Red LED: OUT 3 activated
D19	Red LED: OUT 2 activated
D20	Red LED: OUT 1 activated
D21	Green LED: V_{high} applied at IN 8
D22	Green LED: V_{high} applied at IN 7
D23	Green LED: V_{high} applied at IN 6
D24	Green LED: V_{high} applied at IN 5
D25	Green LED: V_{high} applied at IN 4
D26	Green LED: V_{high} applied at IN 3
D27	Green LED: V_{high} applied at IN 2
D28	Green LED: V_{high} applied at IN 1
D29	Green LED: V_{high} applied at IN 16
D30	Green LED: V_{high} applied at IN 15
D31	Green LED: V_{high} applied at IN 14
D32	Green LED: V_{high} applied at IN 13
D33	Green LED: V_{high} applied at IN 12
D34	Green LED: V_{high} applied at IN 11
D35	Green LED: V_{high} applied at IN 10
D36	Green LED: V_{high} applied at IN 9

Jumper description:

J1	Bootloader Mode

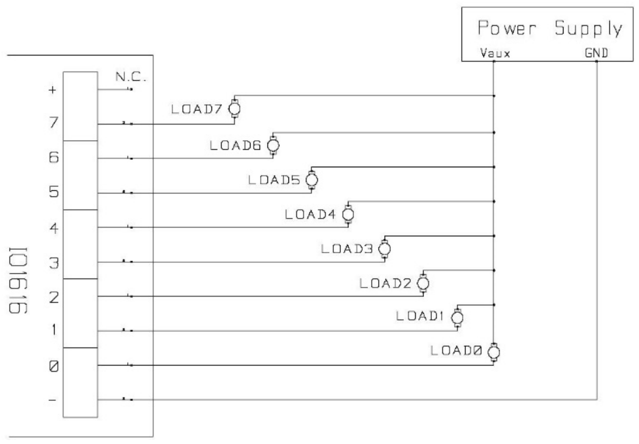
LED STATUS

The LEDs D3 and D4 indicate the status of the system:

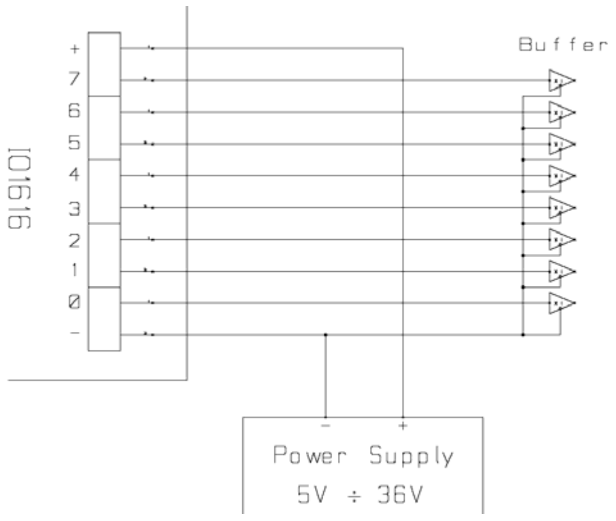
LED D3	LED D4	Status Description
Off	Blinking	Pending Windows driver installation
Blinking	Blinking	Device detected by Windows and driver correctly installed
Off	On	Bootloader Mode

OUTPUT

The sixteen outputs are completely isolated, both between them in two groups of eight and with other signals on the device. Here below there are the diagrams of two typical connections of external device to IO1616 card: in the first case (Picture 2a), the card will manage directly some loads (with maximal current of 150mA). In the second case (Picture 2b), the card is connected to a high impedance device (i. e. the inputs of a PLC).



Picture 2a: diagram of the output connections.



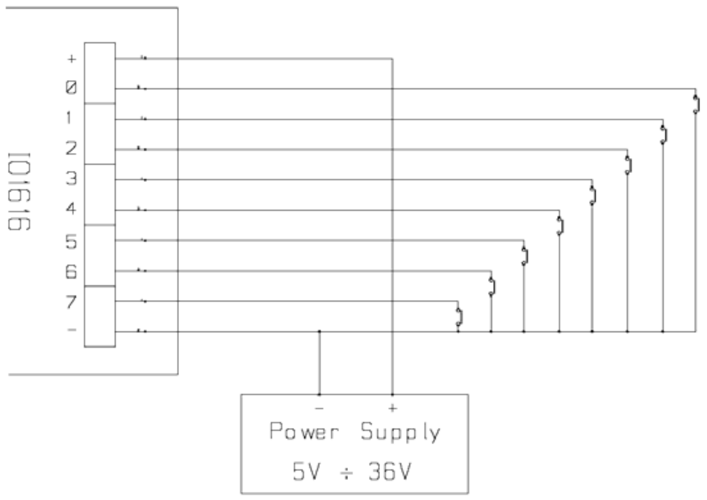
Picture 2b: diagram of the output connections.

Output status is displayed by LED placed near every connector (LED from D5 to D20, showed in Picture 1).

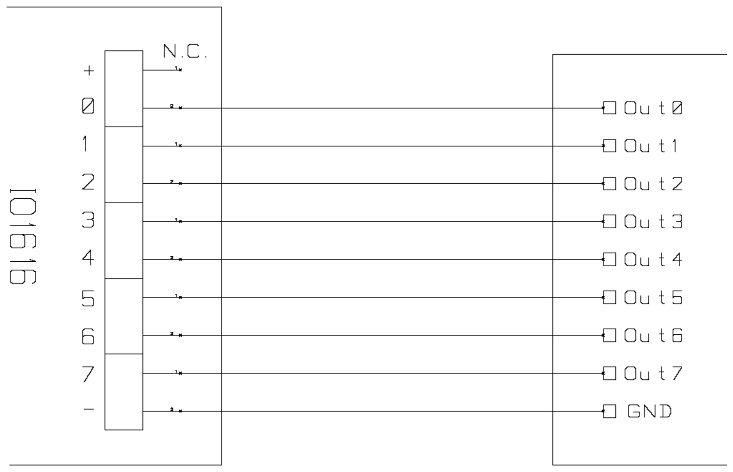
INPUT

The sixteen inputs are completely isolated, both between them in two groups of eight and with other signals on the device.

We suggest to connect inputs following one of the diagrams displayed below:
-Picture 3a: use this way in case inputs have to detect the pression of a switch or an open collector output.
-Picture 3b: use this way in case inputs are directly controlled by a voltage.



Picture 3a: diagram of input implementation.



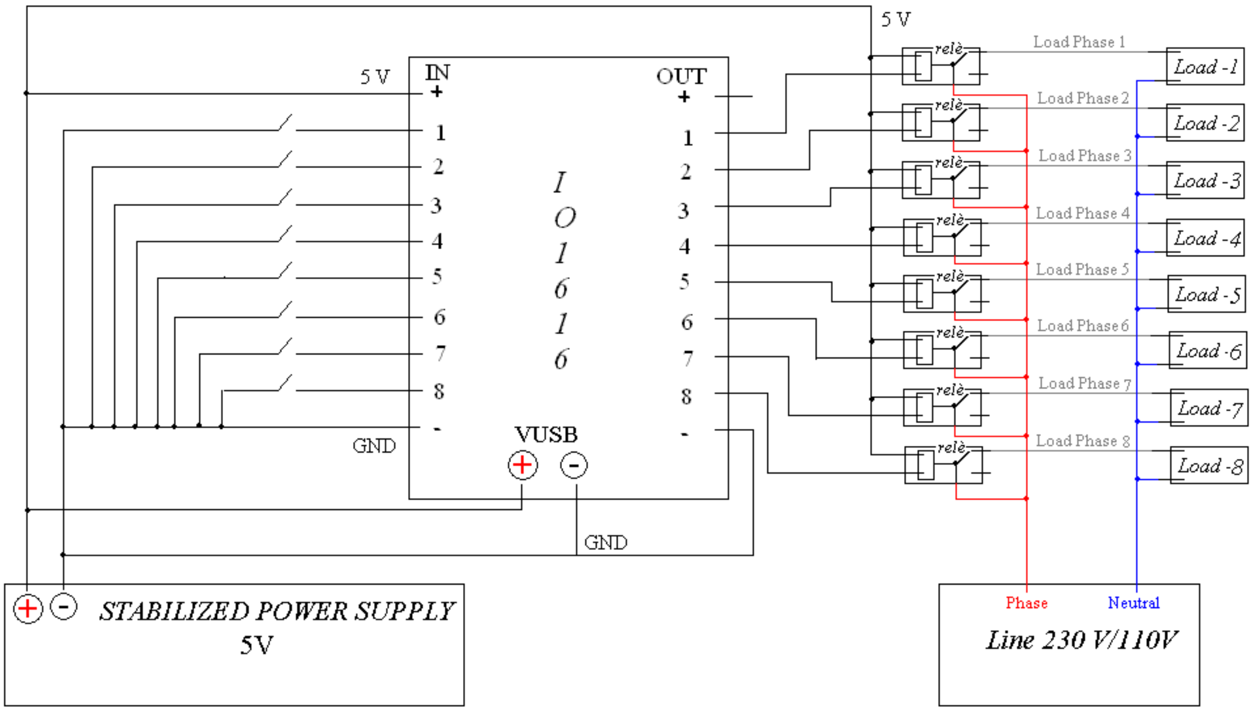
Picture 3b: diagram of input implementation.

Input status is displayed by LED placed near every connector (LED form D21 to D36, showed in Picture 1).

EXAMPLE OF USAGE

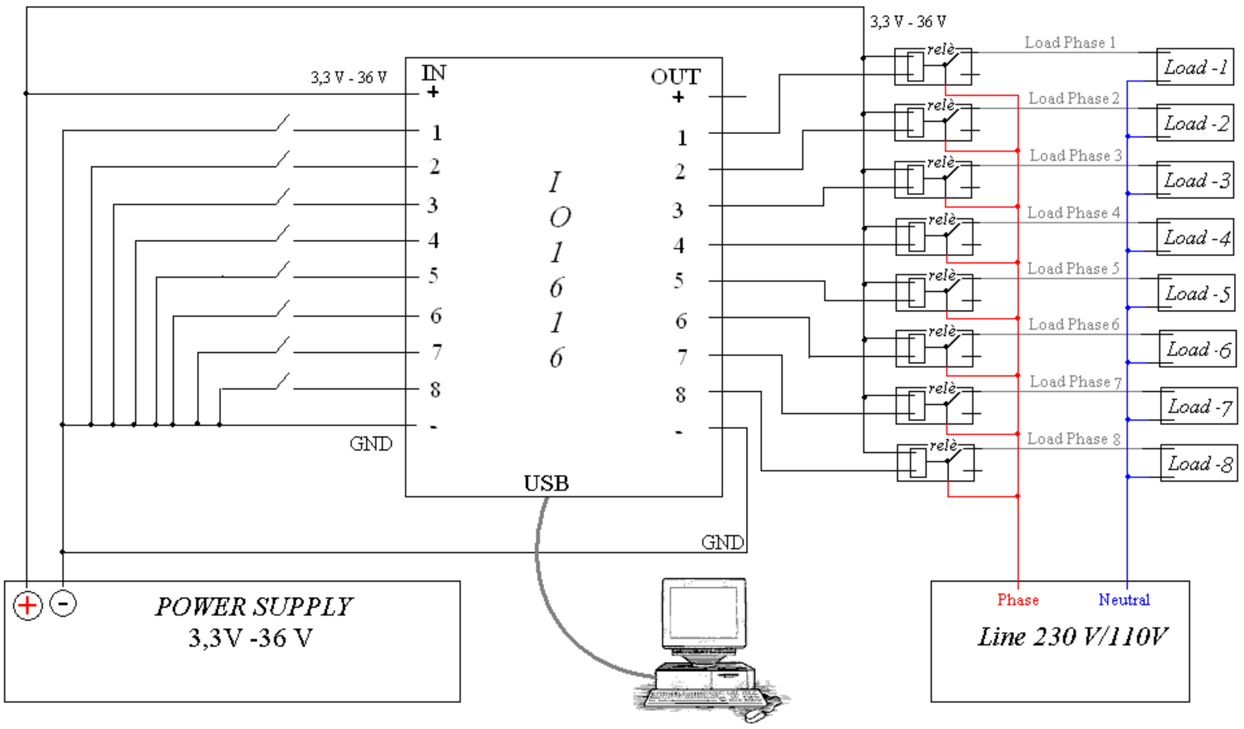
The follow examples show how you can connect IO1616 for manage external load with line supply.

Picture 4a show how connect IO1616 in *stand-alone* mode (without PC connections). Before using the device in this mode you have to configure IO1616 using *demo software.*



Picture 4a: Stand alone mode connections (without PC).

Picture 4b show how can use IO1616 with external supply load and PC control.



Picture 4b: PC control connections.

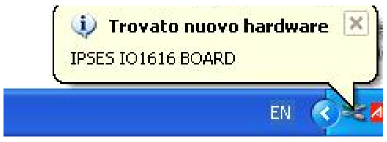
USB DRIVER FOR PC

To make IO1616 operative, the proper provided driver has to be installed on your *Windows* PC. The ***driver*** installs on your PC a ***DLL*** library to manage the communication directly toward the USB interface.
The driver works with *Windows XP*, *Windows Vista* and *Windows 7* (both 32 and 64 bits).

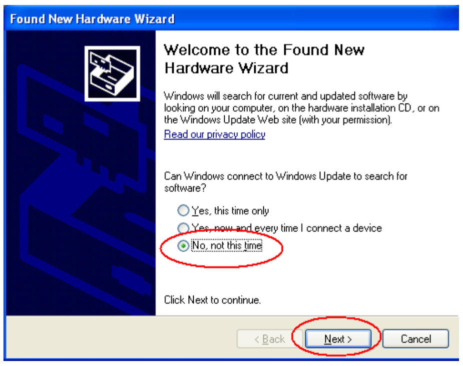
DRIVER INSTALLATION

To install the driver provided with, you need to have the administrator privilege level, so follow the instructions listed below.

1) Connect IO1616 to the PC using a USB cable. *Windows* operative system will detect a new device, showing the displayed message:


Trovato nuovo hardware
IPSES IO1616 BOARD

2) In the following window "Found New Hardware Wizard" choose "No, not this time" and then "Next".


Found New Hardware Wizard

Welcome to the Found New Hardware Wizard

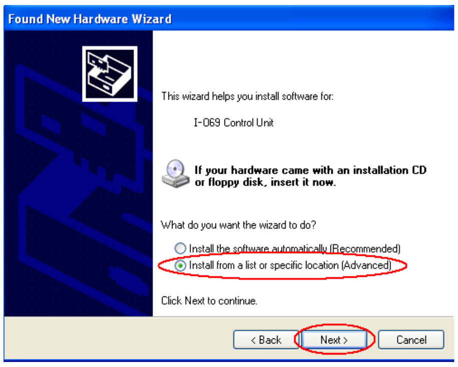
Windows will search for current and updated software by looking on your computer, on the hardware installation CD, or on the Windows Update Web site (with your permission).

Read our privacy policy

Can Windows connect to Windows Update to search for software?

Yes, this time only [✓ No, not this time]
Yes, now and every time I connect a device

Click Next to continue.


Found New Hardware Wizard

This wizard helps you install software for:

I-069 Control Unit

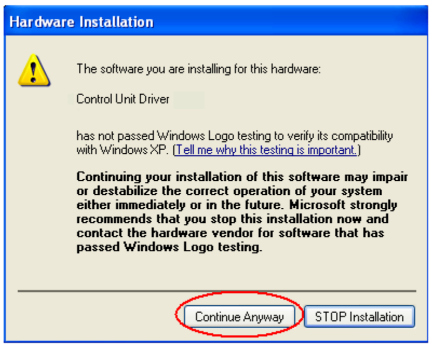
If your hardware came with an installation CD or floppy disk, insert it now.

What do you want the wizard to do?

Install the software automatically (Recommended)
[✓ Install from a list or specific location (Advanced)]

Click Next to continue.

3) Then choose "Install from a list or specific location (Advanced)" and "Next". Follow instructions displayed and set USB driver location.



Hardware Installation

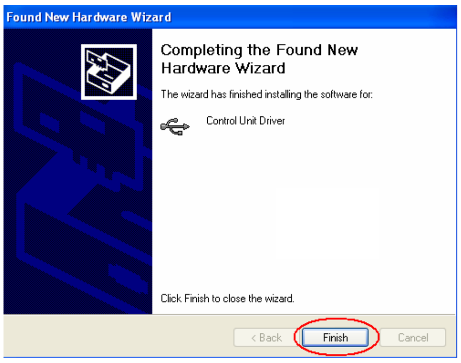
The software you are installing for this hardware:

Control Unit Driver

has not passed Windows Logo testing to verify its compatibility with Windows XP. (Tell me why this testing is important.)

Continuing your installation of this software may impair or destabilize the correct operation of your system either immediately or in the future. Microsoft strongly recommends that you stop this installation now and contact the hardware vendor for software that has passed Windows Logo testing.

4) During installation, operative system should give hardware installation warning. To proceed, select “Continue Anyway”: provided driver is *Windows* compatible.



Found New Hardware Wizard

Completing the Found New Hardware Wizard

The wizard has finished installing the software for:

Control Unit Driver

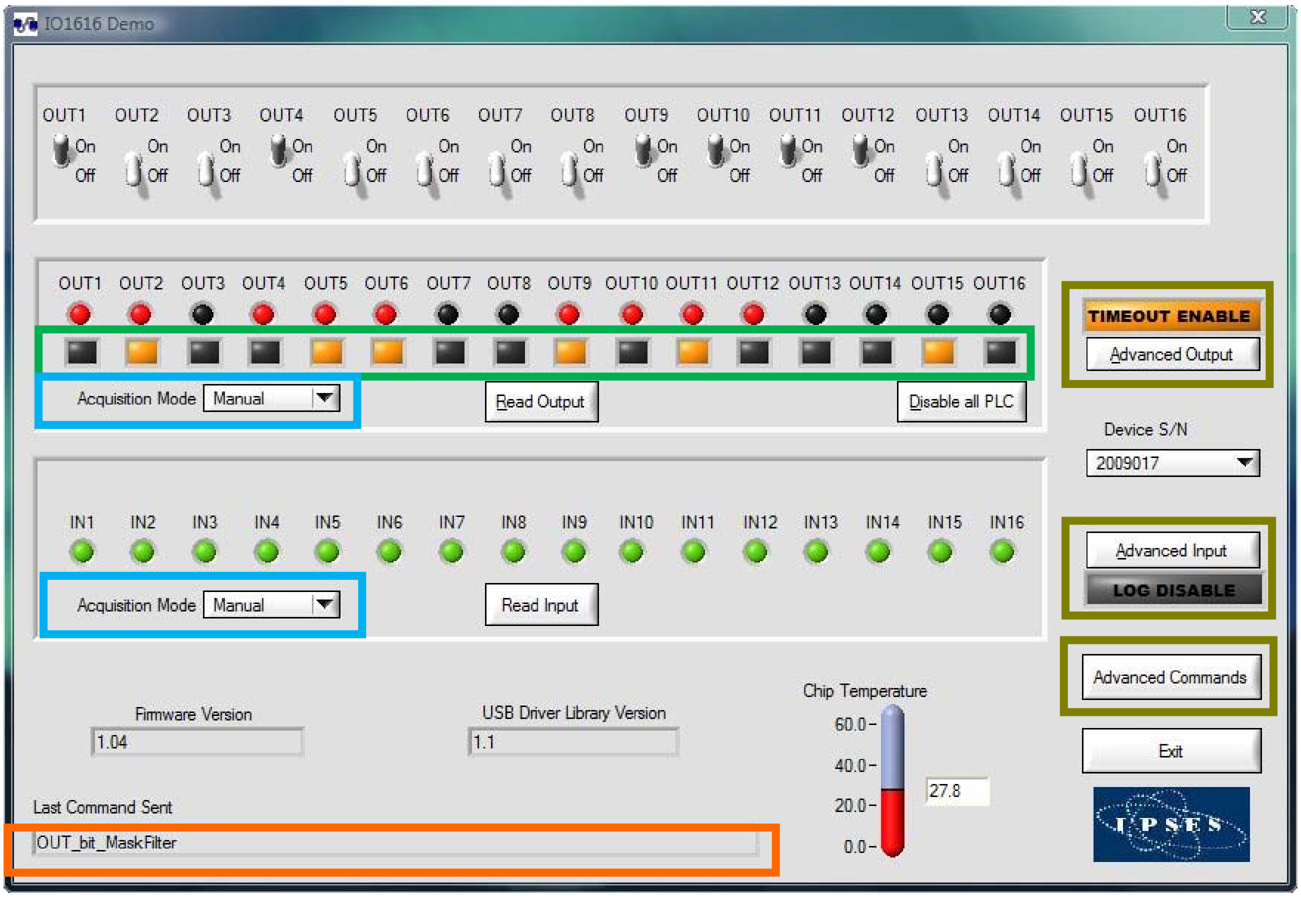
Click Finish to close the wizard.

5) Installation is completed when “Found New Hardware Wizard” is displayed. To exit, choose “Finish”.

DEMO SOFTWARE

A CD with a demo software is provided with the card. This demo software allows to manage the main functions of IO1616.

In the next picture there is a *snapshot* of the main window of the software. With this software it is possible to manage up to 128 IO1616 devices contemporarily connected to a PC. At the software start-up, the program detects the number of connected devices and draws up the list of corresponding Serial Numbers in the ring menu *Device S/N*. To enable the control of a desired board, select its Serial Number item from that menu. If a board is connected during the software running, it will be not possible to control it: so close the software before adding any other card. For every selected board the *Firmware Version* and *USB Driver Library Version* fields are updated according the value read from the device. If a temperature sensor is mounted on the card, its value is periodically updated during the use of the device.



Acquisition Mode: Manual

Acquisition Mode: Manual

Device S/N: 2009017

Firmware Version: 1.04

USB Driver Library Version: 1.1

Chip Temperature: 27.8

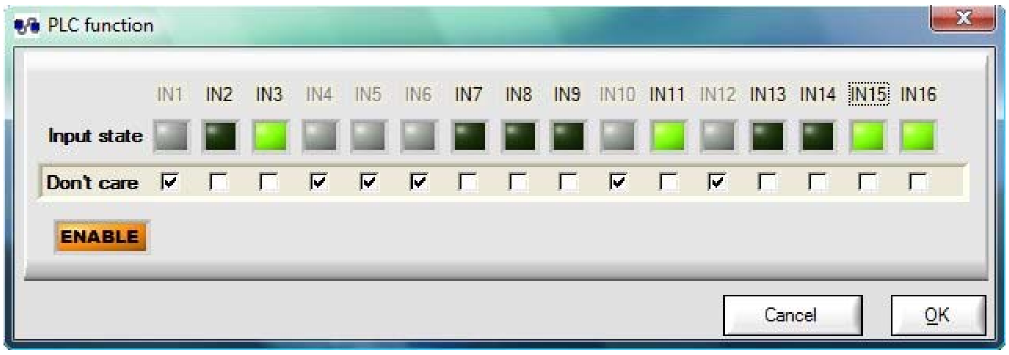
Last Command Sent: OUT_bit_MaskFilter

Picture 5a: Main windows of the demo software.

The field *Last Command Sent* (surrounded in orange in Picture 5a) shows the last operation executed. The selectors in the upper part directly manage the sixteen outputs in real time, while the *Read Output* button update the software output LEDs exactly as in the card (the software colour, red, is the same of the card).

The square LED, place under every output indicator (surrounded in green in Picture 5a), show if are define masks and filters for an output.
To disable all filters and masks for every output push *Disable All PLC* button.

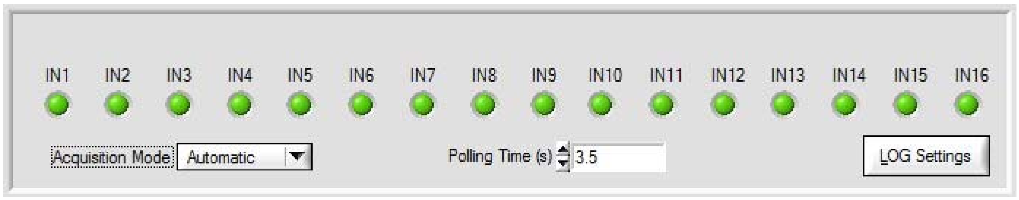
To enable/disable masks and filters for a particular output push the corresponding square LED and configure the *PLC panel* (Picture 5b).


PLC function
IN1 IN2 IN3 IN4 IN5 IN6 IN7 IN8 IN9 IN10 IN11 IN12 IN13 IN14 IN15 IN16
Input state
Don't care
ENABLE
Cancel
OK

Picture 5b: PLC panel.

In *PLC panel* you can enable or disable the PLC mode for the selected output using the *LED-Button ENABLE/DISABLE*. To create a custom mask you can use the *check boxes* while to configure the corresponding input state you can use the square LEDs of this window.

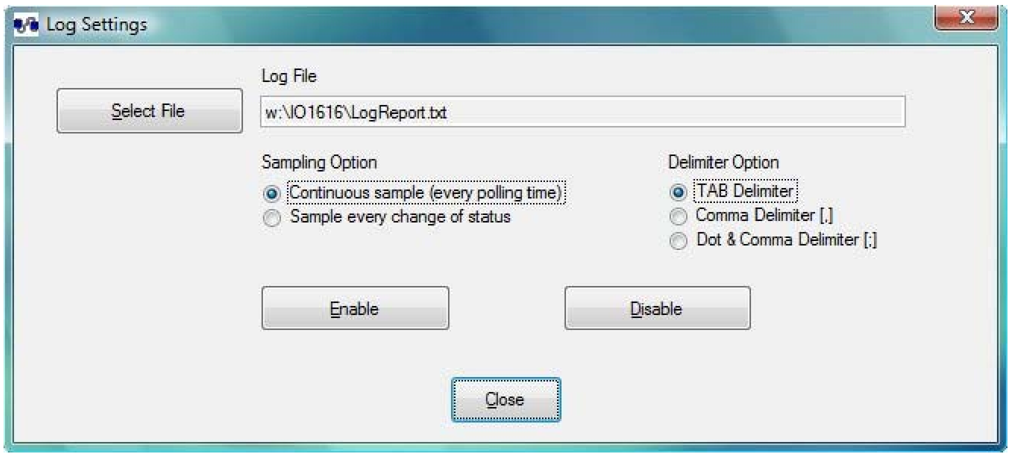
In main window you can read the logic inputs/outputs status in two ways, thanks to the selector ring menu (surrounded in blue in Picture 5a). The manual mode performs an asynchronous sampling when you push the *Read Input/Read Output* button, while the automatic mode performs a continuous polling of the input/output status. The polling rate is customable by the *Polling Time* control (showed in Picture 5c): this field accepts values between 0.5s and 10s. In both cases, at every sample, the software green/red LEDs referred to the inputs/outputs status are updated.


IN1 IN2 IN3 IN4 IN5 IN6 IN7 IN8 IN9 IN10 IN11 IN12 IN13 IN14 IN15 IN16
LOG Settings

Acquisition Mode: Automatic
Polling Time (s): 3.5

Picture 5c: Automatic acquisition mode of the inputs.

During the automatic acquisition mode of the inputs, it is possible to generate a log file: the *LOG Settings* button allows to open the configuration window of the *log'* parameters (showed in Picture 5d).



Log File
w:\IO1616\LogReport.txt

Sampling Option
[✓ Continuous sample (every polling time)]
Sample every change of status

Delimiter Option
[✓ TAB Delimiter]
Comma Delimiter [,]
Dot & Comma Delimiter [;]

Picture 5d: Log settings window.

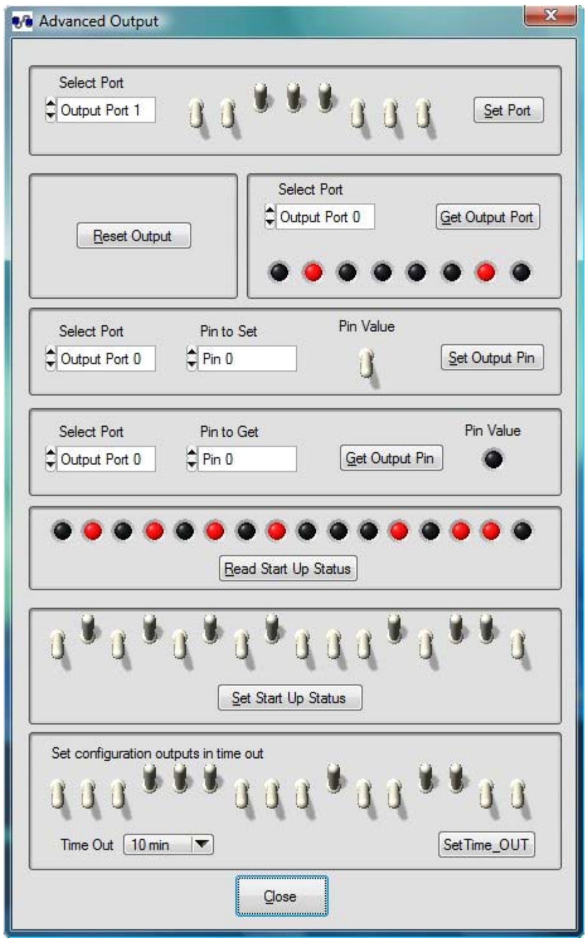
The Select File button allows to browse the path and the file name of the log file (.txt). If the file does not exist, it will be created at the first writing.
The Sampling Option selector allows to choose the data logging mode: every polling time, or every polling time when the actual status is different from the previous one. The default is the *Continuous sample* mode.
The *Delimiter Option* selector allows to select the delimiter of the fields in the log file: the default is the TAB.
If the input acquisition mode is set to *Manual* while the logging is enable, this will be automatically suspended until the acquisition mode will not set to *Automatic*.
The structure of the generated log file is:

```
S/N        Date(dd/mm/yyyy)   Time(hh:mm:ss)     Inputs value(hex)    Inputs value(bin)
```

The value of the inputs, both in hexadecimal and binary format, are coded as Big Endian bitwise, where the less significant bit (LSB) is referred to input 1 and the most significant bit (MSB) is referred to input 16.

Three buttons on the right (surrounded in green in Picture 5a) allow to open three windows: one for the outputs (*Advanced Output)*, one for the inputs (*Advanced Input)* and the last one for the read and write commands in the non volatile memory (*Advanced Commands*).

The *Advanced Output* window (see Picture 5e) allows the typical operations you can performed on the outputs of the device.



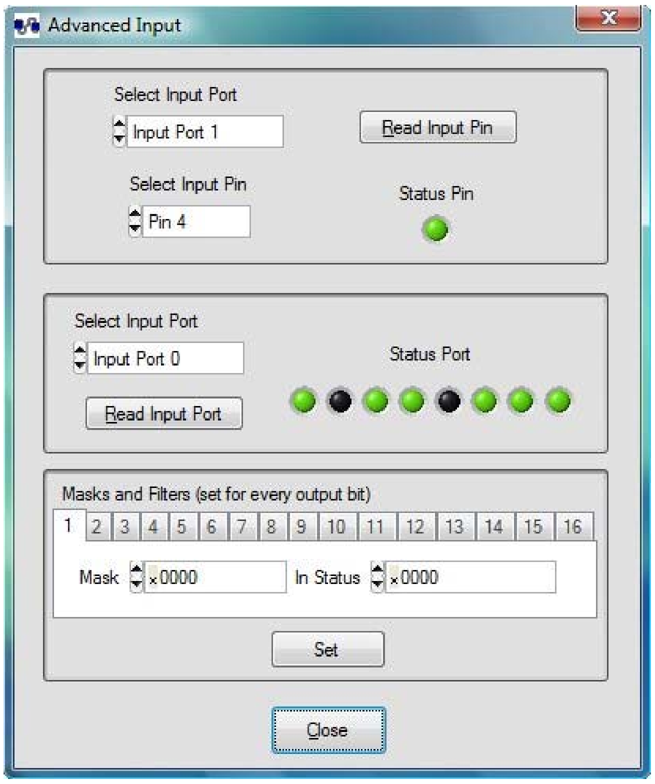
Picture 5e: Advanced Output window.

The output commands are listed below:

- Activation of each output for both ports
- Reset of the outputs (all outputs are switched off)
- Reading of the status of each output for each port
- Command and reading of a single output
- Reading of the output status at the power on
- Setting the output status at the power on
- Manage the customizable USB *timeout* with associate outputs configuration

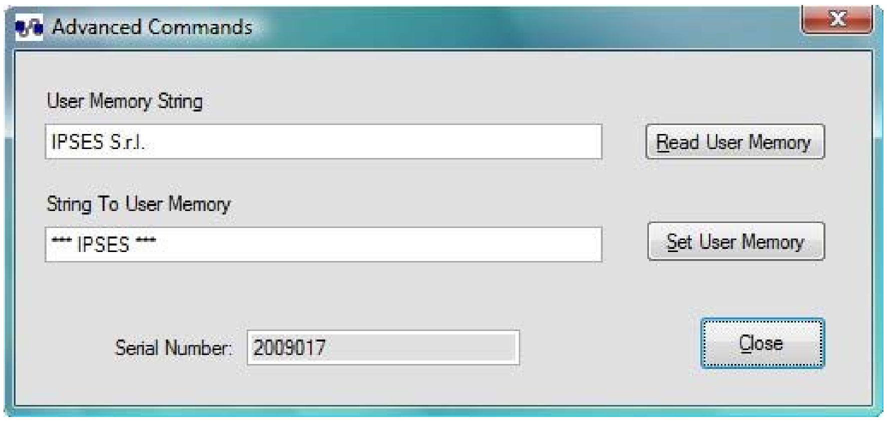
Advanced Input window (see Picture 5f) allows the typical operations you can performed on the inputs of the device.
The input commands are listed below:

- Reading of the status of each input for each port
- Reading of the status of a single input
- Set masks and filters for command outputs (See also *PLC panel*)



Picture 5f: Advanced Input window.

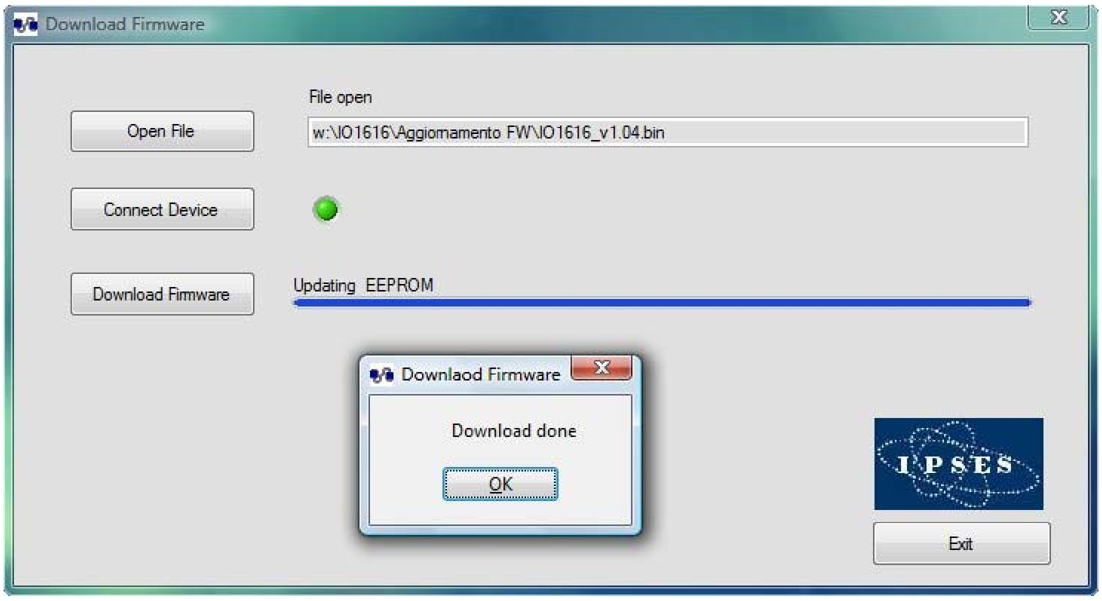
The *Advanced Commands* window (see Picture 5g) allows to enter in a non volatile memory area which is available for the user on the device. Through this windows the user can write strings of data to keep stored inside the device addressed by the current Serial Number.



Picture 5g: Advanced Command window.

FIRMWARE UPGRADE FUNCTIONALITY

IO1616 is provided with a Boot Loading for firmware update by USB. To set the unit in firmware upgrade mode, select jumper J1, connect the device using the USB port and check led D4 is ON, then execute the Demo Software. The Software automatically will load the proper interface as shown in Picture 6.



File open
w:\IO1616\Aggiornamento FW\IO1616_v1.04.bin

Updating EEPROM

Downlaod Firmware
Download done

Picture 6: Firmware upgrade software start-up.

To download a new firmware open the new firmware file pushing *Open File* button, then activate connection choosing *Connect Device* button (if the connection is disabled the LED stays off), then push *Download Firmware* and wait for the pop-up message (fail or pass).

IO1616LowLevel LIBRARY

IO1616 card is equipped with a library implementing several functions which are described below. To install correctly the DLL **IO1616_LowLevel** Library it is necessary to run the relevant setup program and to confirm the installation path: "*C:\IPSES_Lib\IO1616*". It is possible to use the *DLL* to develop a customized software. All functions return a 32 bit integer variable. If there are no problems during the execution, its value is '0', otherwise see the table at the end of the paragraph for code errors details.

The typedefs of the parameters passed to the function are defined in the header file (.h) provided with the DLL. The used definitions are shown following for convenience:

BYTE:	unsigned char
CHAR:	char
UINT:	unsigned int
PORTx:	enum (0, 1)
BITx:	enum (0, 7)
STATUSx:	enum (0, 1)

The device_index parameter, that address the current device, is zero-indexed based.

DLL Load in the memory

int LoadDLL (BYTE *DeviceConnected);

To allows communication toward the device(s), the first operation is to load the DLL in the memory. The function assigns to the pointer the number of connected devices (that can be a maximum of 128).

Read of the Firmware and USB DLL versions

int ReadVersion (BYTE device_index, CHAR *FWVersion, CHAR *DriverDLLVersion, CHAR *LowLevelDLLVersion);

This function assigns to two pointers the two versions of *Firmware* and USB Driver *DLL, respectively,* of the current device selected by *device_index*. The third pointer receives the IO1616LowLevel DLL Library version. These ones are ASCII strings at two fields separated by a dot("."): the first field is the *major*, the second one is the *minor*.

Read of the Serial Number

int ReadSerialNumber (BYTE device_index, CHAR *SerialNumber);

This function assigns to the pointer the *Serial Number* string of the current device addressed by *device_index*.

Reset of the output ports

int ResetOutputPorts (BYTE device_index);

This function switches off all the outputs of the device addressed by the parameter.

Set the status of one output port

int SetOutputPort (BYTE device_index, PORTx iNumVL, BYTE iState);

This function sets the outputs of one of the two ports; the first parameter is the index of the current device, the second is the port 0 or 1 (the value must be either 0 or 1, otherwise there is an error), while the third parameter is a binary byte which corresponds to a single output (the 8 bits are codified as Big Endian bitwise).

Set the status of the output ports

int SetOutputPort_All (BYTE device_index, UINT iState);

This function sets the status of both output ports; the first parameter is the index of the current device while the second parameter is a binary word at 32bit which corresponds to both output (only the 16 LSB bits are relevant and they are codified as Big Endian bitwise).

Read the status of one output port

int GetOutputPort (BYTE device_index, PORTx iNumVL, BYTE *out);

This function gives back the status of one of the two output ports; the first parameter is the index of the current device, the second is the port 0 or 1 (the value must be either 0 or 1, otherwise there is an error), while the third parameter is a binary byte which corresponds to the single outputs (the 8 bits are codified as Big Endian bitwise).

Set a pin of the output ports

int SetOutputPin (BYTE device_index, PORTx iNumVL, BITx pinNum, STATUSx iState);

This function sets the output of one of the two ports; the first parameter is the index of the current device, the second is the port 0 or 1 (the value must be either 0 or 1, otherwise there is an error), the third parameter is the pin to be set (the value has to be between 0 and 7), while the last parameter can be 0 or 1 (1 for the pin activated, 0 for the pin deactivated).

Read the status of the output ports

int GetOutputPorts (BYTE device_index, BYTE *out0, BYTE *out1);

This function gives back the status of both output ports; the first parameter is the index of the current device, the second parameter is the port 0, while the last is the port 1 (the bytes codified as Big Endian bitwise).

Read the status of a pin of the output ports

int GetOutputPin (BYTE device_index, PORTx iNumVL, BITx pinNum, BYTE *out);

This function gives back the status of the selected output of one of the two ports; the first parameter is the index of the current device, the second parameter is the port 0 or 1 (the value must be either 0 or 1, otherwise there is an error), the third parameter is the pin to be read (the value has to be between 0 and 7), while the last one can be 0 or 1 (1 for the pin activated, 0 for the pin deactivated).

Read the status of both input ports

int ReadInputPort_All (BYTE device_index, UINT *in);

This function gives back the status of both input ports; the first parameter is the index of the current device, while the second parameter is a binary word at 32bit which corresponds to a single input (only the 16 LSB bits are relevant and they are codified as Big Endian bitwise).

Read the status of one input port

int ReadInputPort (BYTE device_index, PORTx iNumVL, BYTE *in);

This function gives back the status of one of the two input ports; the first parameter is the index of the current device, the second parameter is the port 0 or 1 (the value must be either 0 or 1, otherwise there is an error), while the third parameter is a binary byte which corresponds to a single input (the 8 bits are codified as Big Endian bitwise).

Read the status of a pin of the input ports

int ReadInputPin (BYTE device_index, PORTx iNumVL, BITx pinNum, BYTE *in);

This function gives back the status of the selected input of one of the two ports; the first parameter is the index of the current device, the second parameter is the port 0 or 1 (the value must be either 0 or 1, otherwise there is an error), the third parameter is the pin to be read (the value has to be between 0 and 7), while the last one can be 0 or 1 (1 for the pin activated, 0 for the pin deactivated).

Read the temperature value

int ReadTemperature (BYTE device_index, double *temperature);

This function, available only when the temperature sensor is mounted on the device, gives back the value of the temperature of the card (in floating point 4 byte IEEE format) indexed by *device_index*.

int ReadTemperatureInt (BYTE device_index, int *temperature);

This function, available only when the temperature sensor is mounted on the device, gives back the temperature value on the card (in integer format with resolution of 0.0625°C) indexed by *device_index*.

To extract the real value of temperature from the parameter sets by the function, it is necessary to multiply this one by 0.0625. In example, a value of 400 is 25°C.

Write the user memory

int WriteUserMemory (BYTE device_index, BYTE *dataToWrite);

This function writes data in a non volatile memory through a byte pointer (max length of the vector to be written is 62 bytes) in the device indexed by the first parameter.

Read the user memory

int ReadUserMemory (BYTE device_index, BYTE *readData);

This function gives back the data previous stored in the user memory from the device indexed by the first parameter.

Set the output pin status at the power on

int SetPowerOnOut (BYTE device_index, BYTE out0, BYTE out1);

This function sets the status of the outputs of the two ports at the power on. The first parameter is the index of the current device, the second parameter is the port 0 or 1 (the value must be either 0 or 1, otherwise there is an error), while the third parameter is a binary byte which corresponds to a single output (the 8 bits are codified as Big Endian bitwise).

Read the status of the output pin at the power on

int GetPowerOnOut (BYTE device_index, BYTE *out0, BYTE *out1);

This function gives back the status of the outputs of the two ports at the power on. The first parameter is the index of the current device, the second parameter is the port 0 or 1 (the value must be either 0 or 1, otherwise there is an error), while the third parameter is a binary byte which corresponds to a single output (the 8 bits are codified as Big Endian bitwise).

Set USB timeout and associate outputs state

int TimeOut_Set (BYTE device_index, BYTE TimeOut, UINT SafeOut);

This function sets the value o USB timeout and the associate outputs state. The first parameter is the index of the current device, the second set the timeout value and the permitted values are:

0	TimeOut Disabled
1	3 sec
2	5 sec
3	10 sec
4	30 sec
5	1 min
6	5 min
7	10 min
8	30 min
9	1 h

while the third parameter is a binary word at 32bit which corresponds to both output (only the 16 LSB bits are relevant and they are codified as Big Endian bit wise).

Request USB timeout value and associate outputs state

int TimeOut_Request   (BYTE device_index, BYTE *TimeOut, UINT *SafeOut);

This function request the value of USB timeout and the associate outputs state. The first parameter is the index of the current device, the second get the timeout value and the permitted values are:

0	TimeOut Disabled
1	3 sec
2	5 sec
3	10 sec
4	30 sec
5	1 min
6	5 min
7	10 min
8	30 min
9	1 h

while the third parameter is a pointer to a binary word at 32bit which corresponds to both output (only the 16 LSB bits are relevant and they are codified as Big Endian bit wise).

Define mask and filter for an Output

int OUT_bit_MaskFilter   (BYTE device_index, BYTE out, UINT mask, UINT filter);

This function define input mask and filter command an output. The first parameter is the index of the current device, the second indicate the number of output bit select while the third parameter is the mask value finally the forth parameter is the filter value.

Note: masks work with *and* logic. So if you are interested in logic status of first and third bit you will set 0x0005 value for mask.

Request mask and filter for an Output

int OUT_bit_MaskFilterRead (BYTE device_index, BYTE out, UINT *mask, UINT *filter);

This function return input mask and filter defined for an output. The first parameter is the index of the current device, the second indicate the number of output bit select while the third parameter return the mask value finally the forth parameter return the filter value.

Error codes definition

All functions described above give back a 32 bit integer number. If this number is not ‘0’, there is an error; in the following table there are the possible errors.

Code	Description
-1	USB protocol error (CRC)
-2	Timeout on USB reading
-3	Timeout on USB writing
-4	USB Driver Library not loaded
-5	USB Driver Library not found
-15	Range Error on *Line* writing
-16	Range Error on *State* writing (0 or 1)
-23	Writing error on port or pin number
-24	DLL IO1616_LowLevel not loaded
-25	Internal error
-26	Temperature sensor not mounted
-27	Error in the USB End Point closing

Example using the DLL

The following example allows to open the communication (to the device indexed by device_index=0), to read firmware, IO1616 DLL and USB Driver DLL version, to read the card temperature, to set a pattern on the sixteen outputs, to read the sixteen inputs, to set a timeout and to operate with filter and mask associated to an output:

On the provided CD there are some programming examples in Visual Basic and C++.

```
// Variables definition
BYTE numberofdevice, iState1, iState2, out1, out2;
int dummy_int;
CHAR FWVersion, DriverDLLVersion, LowLevelDLLVersion;
double temperature;
UINT outState, mask, filter;

// Open Device Communication
// Load DLL
if (LoadDLL(&numberof device) != 0)
{
        MessagePopup ("ERROR",  “No device connected or program not running as Administrator”);
        goto Error;
}
// Check if error
if (numberofdevice == 0)
{
        MessagePopup ("ERROR", “No device connected or program not running as Administrator”);
        goto Error;
}
// Check if more than 128 device
if (numberofdevice > 128)
{
        MessagePopup ("ERROR",  “There are more than 128 devices connected”);
        goto Error;
}

// Read Firmware, USB DLL and IO1616LowLevel DLL Version
dummy_int = ReadVersion (0, FWVersion, DriverDLLVersion, LowLevelDLLVersion);
// Check if error
if (dummy_int)
{
        MessagePopup ("ERROR", “Connection error during read version function”);
        goto Error;
}

// Read Temperature by Microcontroller
dummy_int = ReadTemperature (0, &temperature);
// Check if error
if (dummy_int)
{
        MessagePopup ("ERROR",  “Temperature chip is not mounted”);
        goto Error;
}

// Set pattern on all 16 output using two commands
iState1 = 0x55  // Port 1 value to be set
iState2 = 0xAA  // Port 2 value to be set
// Set Output Port 1
dummy_int = SetOutputPort(0, 0, iState1);
// Check if error
if (dummy_int)
{
            MessagePopup ("ERROR", “Error during output port set”);
            goto Error;
}
// Set Output Port 2
dummy_int = SetOutputPort(0, 1, iState2);
// Check if error
if (dummy_int)
{
            MessagePopup ("ERROR", “Error during output port set”);
            goto Error;
}

// Set pattern on all 16 output using a single command commands
dummy_int = SetOutputPort_All(device_index,0xAA55);
// Check if error
if (dummy_int)
{
            MessagePopup ("ERROR", “Error during output port set”);
            goto Error;
}

// Read First Port
dummy_int = ReadInputPort (0, 0, &out1);
// Check if error
if (dummy_int)
{
            MessagePopup ("ERROR", “Error during output port read”);
            goto readOutError;
}

// Read Second Port
dummy_int = ReadInputPort (0, 1, &out2);
// Check if error
if (dummy_int)
{
            MessagePopup ("ERROR", “Error during output port read”);
            goto readOutError;
}

// Set USB TimeOut  (3 sec) and TimeOut Outputs configuration
```

```
outState = 0x0F0F;
dummy_int = TimeOut_Set (0, 1,OutState);    //1 for 3sec TimeOut
// Check if error
if (dummy_int)
{
            MessagePopup ("ERROR", "Error during Time out Setting");
            goto readOutError;
}

// Set filter and mask for output bit 3
mask = 0x0F0F;
filter  = 0x0003;
dummy_int = OUT_bit_MaskFilter (0, 3, mask, filter);
// Check if error
if (dummy_int)
{
            MessagePopup ("ERROR", "Error during setting mask and filter");
            goto readOutError;
}

// Disable filter and mask for output bit 3
mask = 0x0000;
dummy_int = OUT_bit_MaskFilter (0, 3, mask, filter);    //Don't  care filter value
// Check if error
if (dummy_int)
{
            MessagePopup ("ERROR", "Error during setting mask and filter");
            goto readOutError;
}

// Request  filter and mask for output bit 3
dummy_int = OUT_bit_MaskFilterRead(0,3,&mask,&filter)
// Check if error
if (dummy_int)
{
            MessagePopup ("ERROR", "Error during request mask and filter");
            goto readOutError;
}
```

PRODUCT CODE

Code	Description
IO1616	IO1616 control card (without temperature sensor)
IO1616T	IO1616 control card with temperature sensor
IO1616-DIN	IO1616 control card mounted on a universal support for DIN rail (without temperature sensor)
IO1616T-DIN	IO1616 control card mounted on a universal support for DIN rail (wit temperature sensor)
IO1616Library	LabVIEW 7.1 (and following versions) Library for IO1616 cards
USB-A-B	USB cable to connect USB card
USB-A-B-ill	USB cable with light end to connect USB card

TECHNICAL FEATURES

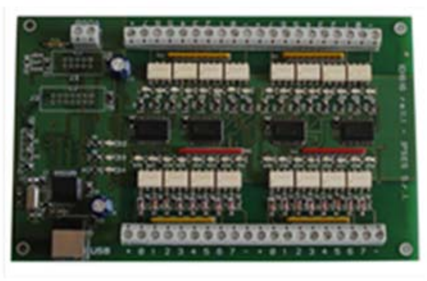
Power supply:	Self-powered through USB
Working temperature	:From 0°C up to +60°C
Storage temperature:	From -40°C up to +85°C
Interface toward:	USB
Card dimensions:	160 x 100 mm (6.30 x 3.94 inches) Thickness (with components): 15 mm (0.59 inches)
Inputs:	Sixteen optocoupled inputs, reciprocally isolated in two groups of eight Maximum applicable voltage: 24V Input Impedance: ≈ 1kohm Logical LOW level: < 1V Logical HIGH level: > 3,3 PULL UP logical HIGH level: >5V Ports read average time execution: 45ms
Outputs:	Sixteen optocoupled outputs, reciprocally isolated in two groups of eight, in an open-collector configuration Maximum output voltage: 36V Maximum output current: 150mA Ports write average time execution: 15ms
Protection:	Optocouplers with 2.500V_{RMS} maximum operative isolation voltage
Temperature sensor:	Resolution: 0.0625°C Accuracy: ±1°C (max.) from +25°C to +65°C ±2°C (max.) from -40°C to +25°C and from +65°C to +85°C ±3°C (max.) from -55°C to -40°C and from +85°C to +125°C

IPSES I/O CARD AVAILABLE MODELS

IO-69: Input/output Card with 6 inputs and 9 relay outputs and USB interface

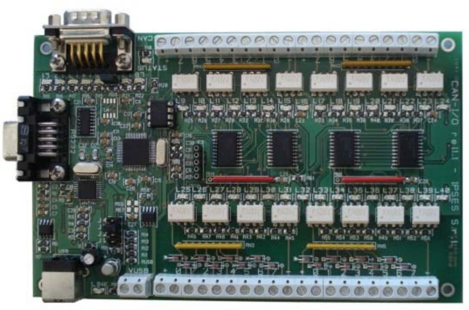
IO-69-USB is a self-powered card to manage six optocoupled inputs and nine relay outputs with USB interface.
A timeout control allows to protect the connecting devices, turning off all the outputs if it does not receive commands from the host within a time configurable through software.
Furthermore, there is the possibility to program all the outputs so that each one will activate only when inputs reach assigned conditions: in this case, IO- 69 acts like a programmable logic controller (PLC).
The card is produced in two versions: with single pole double throw relay (SPDT) and with single pole single throw relay (SPST).

IO-1616: Input/output Card with 16 inputs and 16 outputs and USB or RS232 interface



IO1616 is a self-powered card to manage sixteen optoisolated inputs and sixteen optoisolated outputs with USB interface. The model is available also with RS232 interface, in this case the card needs external power supply.
IO1616 can be directly connected to PLC, to input devices from operator and to other I/O systems. the status of each input
On request, an integrated temperature sensor allows to know in real time the temperature of the system IO1616 is placed in.

CAN-I/O: Input/output Card with 16 inputs and 16 outputs with CAN, USB and RS232 interface



CAN-IO is a control unit equipped with CAN, USB and RS232 interfaces to manage sixteen optocoupled inputs and outputs. The card can work as standalone device on CAN BUS. Its configuration is achieved either through USB (in this case the board is self-powered) or through RS232 interface. Easy to use and to configure, thanks to the provided software, CAN-IO is the right answer to the need to acquire and to drive digital signals through already existing CAN bus.
CAN-IO can be directly connected to PLC, to input devices by operator and to other I/O systems.
Each input and output status can be read by a field bus at any moment. Besides, thanks to LEDs fixed on, the status is shown directly on the board. An integrated temperature sensor allows to know in real time the temperature of the system CAN-IO is placed in.

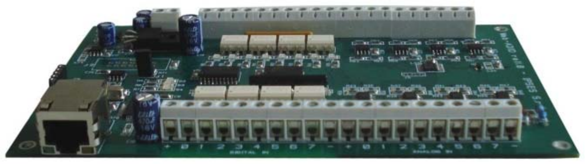
WEB-IO: Input/output Card with 16 inputs and 16 outputs, Ethernet interface, integrated web, telnet and SNMP servers and SMTP client.

WEB-IO is a card to manage sixteen optocoupled inputs and sixteen optocoupled outputs with Ethernet interface, equipped with a web, a telnet and an SNMP servers, and an SMTP client. The web server allows to connect and to manage the card using any web browser (i. e. Internet Explorer or Firefox), with no needs to install a software on your PC. Besides, the card can be connected directly to a switch or to a router with no need to use a PC. It is also possible to develop a customized software managed by telnet service or SNMP client. The SMTP client allows to send alert email based on inputs status change events.
WEB-IO can be directly connected to PLC, to input devices from operator and to other I/O systems. Each input and output status can be read by a web browser or a telnet client at any moment, besides it is shown directly on the board thanks to LEDs fixed on. On request, the card can be equipped with an integrated temperature sensor which allows to monitor in real time the temperature around the regulator voltage module. Through expansion connectors the card can be interfaced to a RTCLOG (Real Time Clock and Logger) optional module: by this way, it can perform a log of the I/O states on a dedicated memory.
WEB-IO is available also in box version, it is provided with a demo software for Windows environment, based on telnet service.

WEB-IO-WiFi: Input/output Card with 16 inputs and 16 outputs, Ethernet and WiFi interfaces, integrated web, telnet and SNMP servers

WEB-IO-WiFi is a card to manage sixteen optocoupled inputs and sixteen optocoupled outputs with Ethernet and WiFi interfaces, equipped with a web, a telnet and an SNMP servers. The web server allows to connect and to manage the card using any web browser (i. e. Internet Explorer or Firefox), with no needs to install a software on your PC. Besides, the card can be connected directly to a switch or to a router, by this way it can be accessed by any PC connected to Internet. It is also possible to develop a customized software managed by telnet service or SNMP protocol. The board is available with built-in antenna or with ultra-miniature coaxial (U.FL) connector for external antenna connection.
WEB-IO-WiFi can be directly connected to PLC, to input devices from operator and to other I/O systems. Each input and output status can be read by a web browser or a telnet client at any moment, besides it is shown directly on the board thanks to LEDs fixed on. On request, the card can be equipped with an integrated temperature sensor which allows to monitor in real time the temperature around the regulator voltage module.

WEB-ADIO: Input/output Card with 8 analogical inputs, 8 digital inputs, 8 analogical outputs and 8 digital outputs, Ethernet interface, integrated web, telnet and SNMP servers



WEB-ADIO is a card to manage 8 optocoupled digital inputs, 8 analogical inputs, 8 optocoupled digital outputs and 8 analogical outputs with Ethernet interface, equipped with a web, a telnet and an SNMP servers. The WEB server allows to connect and to manage the card using any web browser (i. e. Internet Explorer and Firefox), with no needs to install a software on your PC Beside, the card can be connected directly to a switch or to a router with no need to use a PC.

It is also possible to develop a customized software managed by telnet service.

WEB-ADIO can be directly connected to PLC or to analogical transducer, to input devices from operator and to other I/O systems. The analogical inputs and outputs have an operative voltage from 0V to 10V, with a resolution of 10mV and are calibrated one by one. Each input and output status can be read by a web browser or a telnet client at any moment, besides, the status of digital inputs and outputs it is shown directly on the board thanks to LEDs fixed on.

WEB-ADIO-WiFi: Input/output Card with 8 analogical inputs, 8 digital inputs, 8 analogical outputs and 8 digital outputs, Ethernet and WiFi interfaces, integrated web, telnet and SNMP servers

WEB-ADIO-WiFi is a card to manage 8 optocoupled digital inputs, 8 analogical inputs, 8 optocoupled digital outputs and 8 analogical outputs with Ethernet and WiFi interfaces, equipped with a web, a telnet and an SNMP servers. The web server allows to connect and to manage the card using any web browser (i. e. Internet Explorer and Firefox), with no needs to install a software on your PC Beside, the card can be connected directly to a switch or to a router with no need to use a PC. The board is available with built-in antenna or with ultra-miniature coaxial (U.FL) connector for external antenna connection.

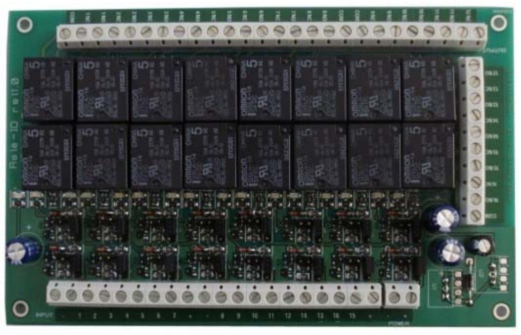
It is also possible to develop a customized software managed by telnet service.

The analogical inputs and outputs have an operative voltage from 0V to 10V, with a resolution of 10mV and are calibrated one by one.

WEB-ADIO-WiFi can be directly connected to PLC or to analogical transducer, to input devices from operator and to other I/O systems. Each input and output status can be read by a WEB browser or a telnet client at any moment, besides, the status of digital inputs and outputs it is shown directly on the board thanks to LEDs fixed on.

.

RELE' I/O: Interface module with16 digital inputs that can control 16 SPDT relay outputs 5A



RELAY I/O(-SEL) is an expansion module with16 digital inputs that can control 16 SPDT relay outputs 5A @ 250VAC or 5A @ 24VDC each

These modules can be used as an expansion for ant I/O card, transforming the TTL or contact freedmen open-collector type outputs (up to a maximum of 16 ones) in 16 relay outputs with NO and/or NC contact.

IPSES provides two board models, based on different relay output tipology:

- RELÈ-IO board: the sixteen outputs are divided in two groups of eight with common COM contact and both NC and NO contacts available on output connectors.
- RELÈ-IO-SEL board: each output is independent and each relay provides COM contact and one contact selectable between NC and NO according dedicated selector configuration.

To operate the cards require an external power supply. Two version are available: RELE'-IO(-SEL)-5 which requires an external power supply of $5V_{dc}$ or RELE'-IO(-SEL)-24 which requires an external power supply from $7V_{DC}$ up to $24V_{DC}$.

The card is in standard Eurocad format (100 x 160 mm - 3,94 x6,30 inches) and can be supplied mounted on opened DIN rail.

N8-USB: *Input Card with 8 inputs and USB interface*

IN8 is a low size auto powered control unit equipped with USB interface. IN8 can check eight galvanic isolated inputs: on each input it is possible to apply voltages regardless form the USB ground, with a maximum voltage of 30V.

Easy to use, thanks to the driver provided with and to the LabVIEW library available on demand, IN8 also reduce installation costs.

The board low size to be easily integrated in several systems. Besides, IN8 has its inputs galvanically isolated to protect from electromagnetic disturbances and ground loops, improving its reliability and quality.

IN8 is the right answer to the need to acquire digital signals from a PC in an industrial environment.

LabVIEW Library for I/O cards:

A complete library for LabVIEW, giving the feasibility of I/O devices remote control, is available on request. The Library allows to implement a LabVIEW application without knowing the details of the communication protocol, making the development quick and easy. Each library is provided with a help file which explains deeper each function included with.

CONTACTS

IPSES S.r.l. conceives, projects and markets electronic and scientific instruments. The customized planning of our devices allows us to answer specific necessities for customers asking for embedded systems. IPSES clients enjoy access to a dedicated project engineering team, available as needed.
Our pool consists of highly competent professionals whose experience in this field is extremely strong. Thanks to constant updating and technical development, IPSES is a leading company, combining the dynamism of a young group into the competence and reliability of a qualified staff.

IPSES S.r.l.

Research and development office:
Via Suor Lazzarotto, 10
20020 Cesate (MI)
Italy

tel. (+39) 02 39449519 - (+39) 02 320629547
fax (+39) 02 700403170
e-mail: info@ipses.com
http://www.ipses.com

SISTEMA DI GESTIONE CERTIFICATO
ITALCERT
UNI EN ISO 9001

SUPPORT INFORMATION

The customer is at liberty to contact the relevant engineer at IPSES S.r.l. directly.

Telephone	:	(+39) 02 39449519
		(+39) 02 320629547
Fax	:	(+39) 02 700403170
Email	:	support@ipses.com

PROBLEM REPORT

The next page is a standard template used for reporting system problems. It can be copied and send as a fax. Alternative bugs may be reported by emails, in this case please insure that the mail contains similar information listed in the *Engineering Problem Report* form.

ENGINEERING PROBLEM REPORT

Problem describer

Name			**IPSES s.r.l.** **Via Suor Lazzarotto, 10** **Cesate (MI)** **Italy** **Fax (+39) 02 700403170** **e-mail *support@ipses.com***
Company			
Date	Tel.	Fax	

Product

Name	Version	Serial No.

Report Type (bug, change request or technical problem)

Major bug	☐	Urgency:	
Minor bug	☐	High	☐
Change request	☐	Medium	☐
Technical problem	☐	Low	☐

Problem Description

Reproduction of Problem

IPSES s.r.l. Action notes

Received by	Date	Report No.	Action

(Product code IO1616 Rel. 01.06.0001)

IPSES S.r.l.
Via Suor Lazzarotto, 10
20020 Cesate (MI) - ITALY
Tel. (+39) 02 39449519 – (+39) 02 320629547
Fax (+39) 02 700403170
e-mail: info@ipses.com
support@ipses.com