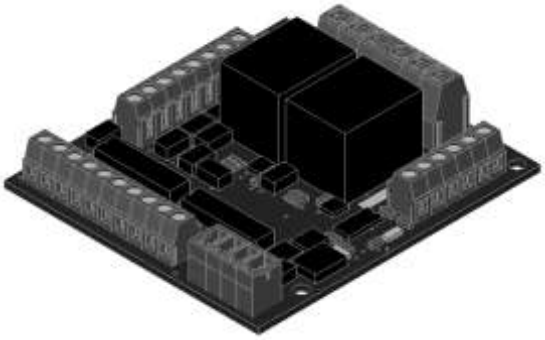
Input/Output expansion board DV0 bus compatible

Features

- 3 opto-coupled digital input, one of them with frequency measurement and counting capability.
- 2 Analog inputs, from 0 to 3.3V, 1mV resolution
- Both analog inputs compatible with 4-20mA sensors
- 4 open collector outputs, up to 7.5A, 30V three of them with PWM capability (motors and RC servos)
- 2 SPDT relays outputs, 5A at 250Vac
- Link led indicator
- 6 micro switches to select board address
- Power supply: 5V from DV0 connector



Technical specifications

- Opto-coupled isolation: 75V
- Opto-coupled input voltage range: 4V to 50V
- Opto-coupled maximum frequency measurement: 15 KHz
- Minimum counting period: 40ms
- PWM resolution: 1/10000
- PWM frequency range: from 31Hz to 2000Hz
- Relay electric contact endurance: 100000 operations.
- Relay maximum switching frequency: 1800 operations/hr
- Analog inputs source impedance: maximum of 2.5KΩ, recommended 500Ω
- Board consumption (no relays activated): 6mA
- Board consumption (both relays activated): 134mA
- Command latency from user application to IO_A outputs: 15ms

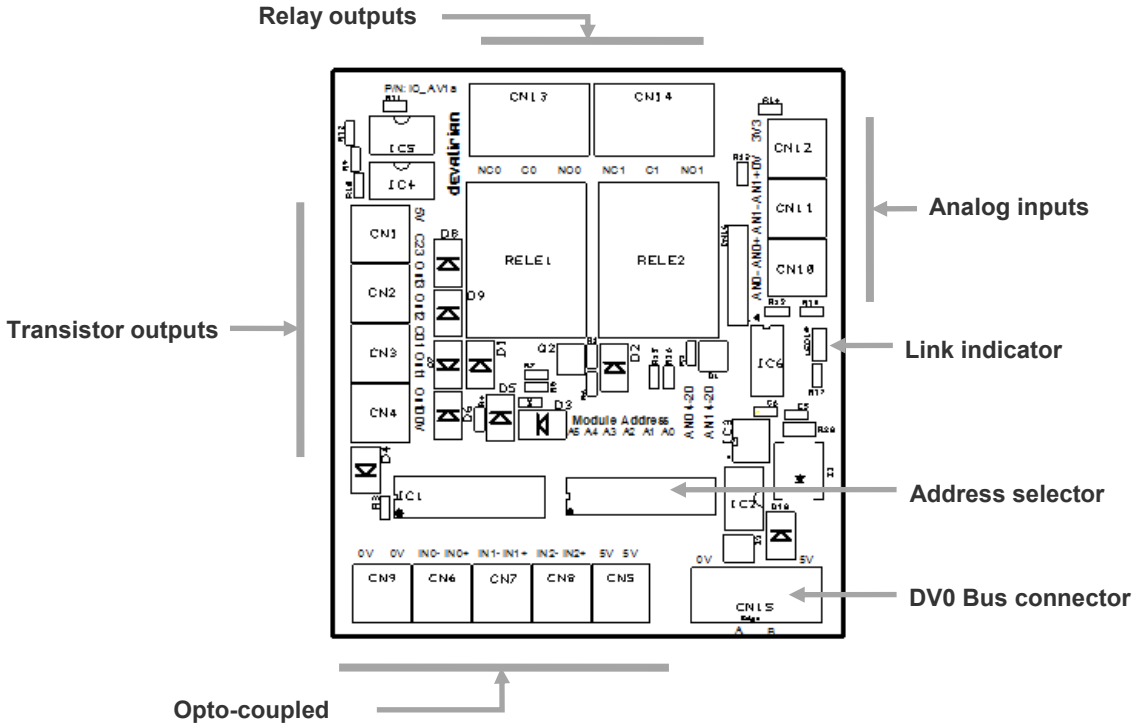


Fig. 1 Simplified connectors diagram

Table of contents

Revision history:

V1.0 Jan-2015

1. FUNCTIONAL OVERWIEW

The IO_A boards continuously listen to the DV0 bus serial lines for incoming commands from the Header. This commands can be either a setting for an output, or an enquiring for the actual value of an input or a request for the IO_A manifest. Commands from the header starts with the address of the target and every board in the bus compares the command address with the actual value of the address selector.

CAUTION:

Care must be taken to ensure that there are no duplicate address board in the bus, because the response collision will hung the boards involved and an unpredictable behavior of the rest of the boards is very likely.

At power up, the IO_A board blinks the Link Indicator, reporting that it is powered but not yet addressed by the Header. When the first command that mach its address is received, the Blink Indicator remains continuously lighted.

1.1 Address selection

Address is selected by setting the individual switches of the Address Selector, at it is shown in the Figure 2.

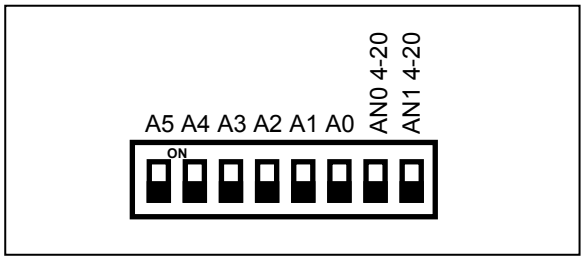


Fig. 2 Address Selector

The switches involved in the address selection are those labeled as A0 to A5, coded as a binary number where A0 is the least significant bit and A5 the most significant bit. The switch meaning is “1” if it is “ON” and “0” otherwise. Factory values are all “OFF”, that means “Normal Analog Input” and “Address = 0”. Keep in mind that zero is not a valid address value and will be ignored.

The binary switches codification is illustrated in the Figure 3.

1.2 DV0 Bus connection

This receptacle connector contains receives power and data from the Header. An extracting vertical socket with screws is supplied with the IO_A board to facilitate bus cabling (AWG 16..24, cross section 1.5 to 0.2 mm)

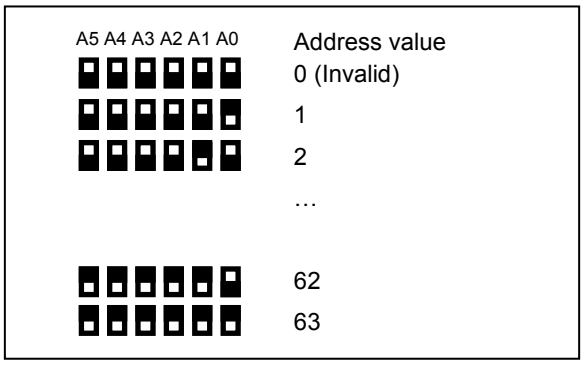


Fig. 3 Address coding example

Vertical socket technical data:

- Socket reference: 20020008-D041B01LF from FCI.
- Solid/Stranded wire: AWG 16 to AWG 26
- Wire cross section: 1.5mm to 0.2mm
- Data signal A and B ESD, EFT and Surge protection: IEC 61000-4-2 (ESD), IEC 61000-4-4 (EFT) and IEC 61000-4-5 (Surge)

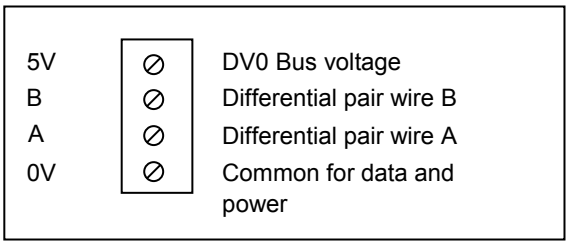


Fig. 4 DVO Bus connector pin out top view

Connect differential pair A to all A inputs of connected IO boards, and pair B to all B inputs as well. Also, connect the common data pin (0V) to all IO boards. The output marked as 5V is the DV0 bus voltage output.

1.3 Manifest

The manifest of IO_A, as it is seen by the API function *GetManifest* is:

```
Board name    : IO_A
PWM Outputs   : 3
Pulse Inputs  : 3
Analog Inputs : 2
Digital output group number 0 has 2 outputs
Digital output group number 1 has 4 outputs
Digital input group number 0 has 3 inputs
```

2. ELECTRICAL CHARACTERISTICS

This chapter explains the electrical characteristics of each IO_A input and output.

2.1 Absolute maximum ratings

Absolute maximum ratings for the IO_A board are listed below. Exposure to these maximum rating conditions for extended periods may affect device reliability. Functional operation of the device at these, or any other conditions above the parameters indicated in the operation listings of this specification, is not assured.

Operating ambient temperature……....-0°C to +70°C
Storage ambient temperature .....-55°C to +125°C
Storage ambient humidity……...35% to 85%
Operating ambient humidity……...35% to 85%
Voltage at 5V DV0 connector……6V
Opto-coupled inputs voltage……..……55V
Opto-coupled inputs reverse voltage……..…55V
Voltage between opto-coupled inputs……..…..55V
Voltage between opto-coupled inputs and 0V……...55V
Analog inputs direct voltage……3.3V
Analog inputs reverse voltage……..….-0.1V
Transistor output voltage……..….30V
Transistor output continuous current……....8A
Transistor output continuous reverse current……...2.8A
Voltage between relay contact and 0V……...1000VDC
Voltage between relay contacts……...500VDC
Relay output current……....6A

2.2 Opto-coupled inputs

Figure 5 shows the simplified connection diagrams for the three opto-coupled inputs block. There are five 2-way headers assembled together, three of them for each input (IN0, IN1 and IN2), one for 0V and another one supplying 5V, for easy cabling.

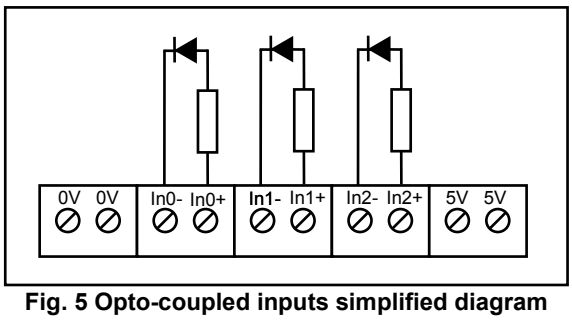


Fig. 5 Opto-coupled inputs simplified diagram

To activate an input, the user must supply current to the input marked as "+" and sink current from the input marked as "-". This current can be provided by the 5V of the connector block, as it is shown in the figure 6, or can be supplied by an independent power, as it is discussed in the section 3.3.

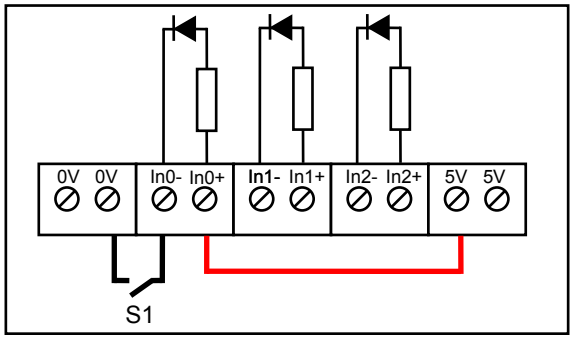


Fig. 6 Opto-coupled input connection example

When the switch S1 of Figure 6 is closed, the input *In0* is "activated", which means that when the user application calls the function *GetIndividualInput* obtains a "1". The same applies for In1 and In2. All inputs allow the counting pulses capability.

The input *In0* has a special feature since it is capable to measure the signal frequency if it is greater than 100Hz and lower than 15000Hz. This feature is specially interesting in industrial applications to count actions or to measure the speed of moving ribbons or wheels.

2.2.1 Operating conditions

- Voltage range to activate *In0*, *In1* and *In2*: from 4V to 50V
- De-bounce time for counting function: 15ms
- Maximum frequency for counting function: 25Hz
- Counting range: from 0 to 65535. User application is responsible for maintaining the carry on overrun
- Frequency range for frequency measure in *In0*: from 100Hz to 15000Hz for an accuracy better than 5%
- On period of signal for frequency measure: 6µs minimum
- Off period of signal for frequency measure: 32µs minimum

2.2.2 Enumeration

From the point of view of API function, the opto-coupled inputs are seen as follows:

Input name	Group number	Input number
In0	0	0
In1	0	1
In2	0	2

For *GetFrequeny* function the parameter *FrecuencyInput* must be 0 .

2.2.3 API functions

The following table summarizes the API functions related to the opto-coupled inputs. See 4.1.3 for more detailed information.

Function name	Description
`GetIndividualInput`	Returns the current state for an individual input, where "1" means "activated". The parameter *Group* must be zero, and the parameter *Input* must be 0, 1 or 2 for *In0*, *In1* and *In2*.
`GetGroupInput`	Returns the current state for each individual input, joined in an integer, where "1" means "activated", *In0* is the least significant bit and *In2* the most significant bit. The parameter *Group* must be zero
`GetPulseCounting`	Returns the accumulated counted pulses (0 to 65535) on *In0, In1 or In2*
`GetFrequency`	Returns the last frequency measured on *In0* in Hertz (0 to 65535 Hz). The parameter *FrequencyInput* must be 0

In2	In1	In0	Value
Off	Off	Off	0
Off	Off	On	1
Off	On	Off	2
Off	On	On	3
On	Off	Off	4
On	Off	On	5
On	On	Off	6
On	On	On	7

2.3 Analog inputs

Figure 6 shows the simplified connection diagrams for the two analog inputs block. There are three 2-way headers assembled together, two of them for each input (AN0 and AN1) and one for 0V and 3.3V, for easy cabling.

The series resistance of 100Ω protect the analog to digital converter of the microprocessor against transient over voltages. Since this impedance is low compared to the ADC internal impedance, it has an almost negligible influence to the converting process.

If an industrial 4-20mA sensor is used, the current supplied by the sensor can be converted to voltage setting "ON" the switches 1 or 2, whichever is connected to the sensor.

Since the output given by the function *GetAnalogValue* is expressed in mV, to obtain the sensor current (in mA), only a division per 100 is required. So, when SW1 or SW2 are on and an industrial 4-20mA is connected to AN0 or AN1, then the input current is:

I_{sensor} = *GetAnalogValue result / 100* [mA]

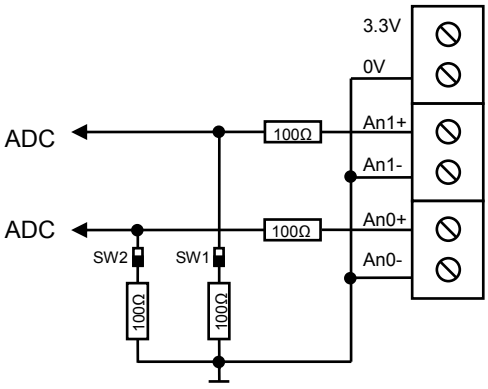


Fig. 6 Analog inputs simplified diagram

Both *An0+* and *An1+* are single ended referenced to 0V. Connect the sensor output to the input marked as "+" and connect the sensor ground to the input marked to "-". Although it is not strictly required to connect the sensor ground to the "-" inputs, it is preferable to avoid connecting the sensor ground whatever 0V exists: this can cause undesirable ground noise. So connect the sensor ground to "-" input as well that 0V.

2.3.1 Operating conditions

- Minimum input voltage: 0V
- Maximum input voltage: 3.3V
- Sampling time: 5ms
- API result expressed in mV as an integer: from 0 (0V) to 3300 (3.3V)
- API result is the average of the last 4 samples so the result has a great rejection to 50Hz noise
- Maximum source output impedance: 2.5KΩ
- Maximum recommended source output impedance: 500Ω
- Converter accuracy: 8mV

2.3.2 Enumeration

Input name	Input number
An0	0
An1	1

2.3.3 API functions

The following table summarizes the API functions related to the opto-coupled inputs. See 4.1.3 for more detailed information.

Function name	Description
`GetAnalogValue`	Returns the average of the last 4 samples, from 0 (0V) to 3300 (3.3V). The parameter *Input* must be 0 for *An0* and 1 for *An1*.

2.4 Relay outputs

Figure 7 shows the simplified connection diagrams for the two relays outputs.

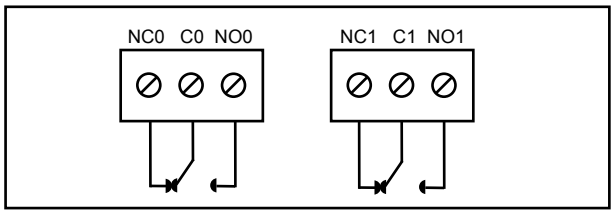


Fig. 7 Relay outputs simplified diagram

When the relays are not activated *C0* is connected to NC0 (Normally Close) and *C1* is connected to NC1 . These are the initial condition. Once a "1" is set as value for *SetIndividualOutput* or *SetGroupOutput,* the relays are energized and *C0* is connected to *NO0* (Normally Open) as well as *C1* is connected to *NO1*.

2.4.1 Operating conditions

- Maximum contact current: 5A
- Rated ac voltage: 250Vac
- Electric contact endurance at 5A/250Vac: 100000 operations
- Maximum switching frequency: 1800 operations/hr
- Isolation between contacts and 0V: 1000V

2.4.2 Enumeration

Contact	Group number	Output number	Value
NC0-C0	0	0	0
NO0-C0	0	0	1
NC1-C1	0	1	0
NO1-C1	0	1	1

2.4.3 API functions

The following table summarizes the API functions related to the relay outputs. See 4.1.3 for more detailed information.

Function name	Description
SetIndividualOutput	If the parameter *Value* is 1, his functions energizes the Relay 0 if the parameter *Output* is zero or the *Relay* 1 if this parameter is 1. The parameter *Group* must be always zero
SetGroupOutput	This function allows to manage the two relays in one beat. The bit 0 of the parameter *Value* stands for *Relay* 0 and the bit 1 of *Value* stands for *Relay* 1. The parameter *Group* must be always zero

2.5 Transistor outputs

Figure 9 shows the simplified connection diagrams for the four transistor driven outputs. There are four 2-way headers assembled together, which contains the four outputs, plus the common rail for outputs 0-1 and 2-3, the 0V ground reference and the 5V power.

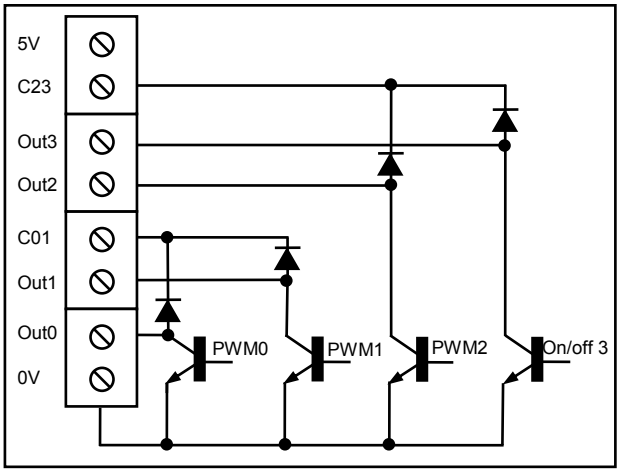


Fig. 9 Transistor outputs simplified diagram

Outputs 0 to 2 can be set/reset or PWM driven while Ouput3 can only be set or reset. In those outputs, "Set" or "1" means that the transistor is "activated", so a current can flow inside the "Output". Note that when the transistor is not "activated", no current is drawn from the output. That means that loads connected to an Output must be powered by some external source.

The diodes connected to the common rails allows the use of inductive loads on that outputs. Those diodes are known as "freewheeling" diode and they are mandatory when dealing with inductive loads. Otherwise, the transistor outputs will crash.

CAUTION:

> Never, never, never let an inductive load connected to a transistor output without the freewheeling diode protection. Sooner or later the transistor will crash.

As the 5V present on those connectors is the same that comes from the DV0 bus, its available current can't be determined, as it depends of the number and type of boards connected to the bus. However, it is possible to connect light loads, like LEDs, a pull-up resistor to control RC servomotors (see example 3.2) or a small relays. When connecting relays to 5V power, don't forget to connect the common rail to gain the protection of the internal diode. Figure 10 shows a

small relay powered by 5V and managed by the *Output2*. Note that C23 is connected to 5V.

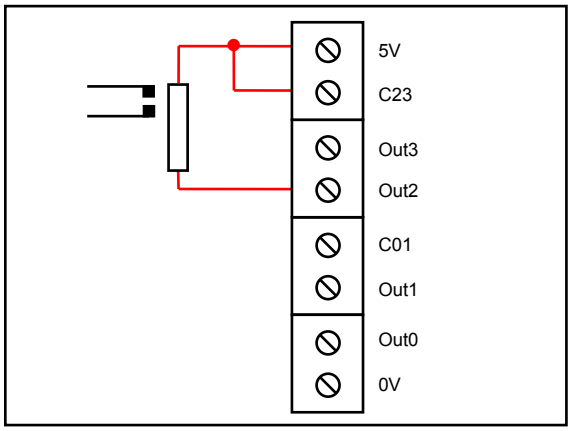


Fig. 10 Small relay connected to 5V example

In this example, to activate the relay, the user application must activate the output 2, calling the function *SetIndividualOutput* with parameter *Group* set to "1", parameter *Output* set to 2 and parameter *Value* set to "1" .

Figure 11 shows how to connect a 12V DC motor to the *Output0* powered by an external supply. Note that C01 is connected to the positive rail of the 12V power supply and that the negative rail of that supply is connected to the 0V common ground.

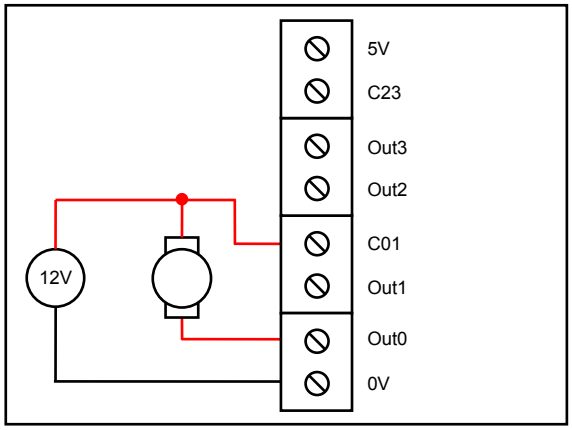


Fig. 11 DC Motor externally powered

By calling *SetIndividualOutput*, the user application can run the motor to its maximum power or to stop it. For example, writing in C++ language, and assuming that the address selection for this board is 16, then:

```
SetIndividualOutput(16,1,0,1); // Motor starts
SetIndividualOutput(16,1,0,0); // Motor stops
```

As *Group* is "1" for transistor outputs and *Output0* is just "0".

Also, it is possible to control the power delivered to the motor by using a *Pulse With Modulation* (PWM) as a current form, instead of maintaining the transistor full activated. Figure 12 shows how *Output0* can be driven using PWM.

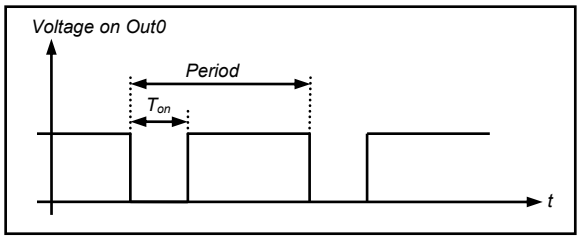


Fig. 12 PWM form

In the above illustration, when the voltage at *Output0* is zero, the transistor is activated and current flows through the motor. When this voltage is high, no current flows and the motor loses energy through the freewheeling diode. If *Period* is small compared to the physics of the motor (10ms, for example), this continuous turning on and off is not appreciated. Controlling the ratio between the total period and the time that the transistor is on (T_{on}) is an easy and efficient way to control the power of the motor.

Let's define the *Duty Cycle (D)* as

$$D = Ton / Period \quad \text{[Adimensional]}$$

Sometimes expressed as a percent

$$D = 100 \times T_{on} / Period \quad \text{[per cent]}$$

Then, the greater the value of *D*, the greater the power supplied to the motor. Both the *Period* and the *D* value can be set for the *Outputs* 0 to 2 using he DV0 API functions *SetPWMLimits* and *SetPWMValue*. See the application note 3.1 and 3.2 for more explanations on how to manage PWM outputs

2.5.1 Operating conditions

- Maximum voltage: 30V
- Maximum continuous current: 7.5A
- Maximum period time: 32ms
- Minimum period time: 500µs

2.5.2 API functions

The following table summarizes the API functions related to the transistor outputs. See 4.1.3 for more detailed information.

Function name	Description
`SetIndividualOutput`	If the parameter *Value* is 1, his functions activates the

	transistor of the Output that indicates the parameter *Output.* The parameter *Group* must be always 1
SetGroupOutput	This function allows to manage the four *Outputs* altogether. The bit 0 of the parameter *Value* stands for *Output0*, the bit 1 of *Value* stands for *Output1* and so forth. The parameter *Group* must be always 1
SetPWMValue	Set the output pulse with in steps of 0.01 percent. The parameter *Value* can be in the range of 1 (minimum pulse with) to 10000 (100%, maximum pulse with) for normal output ON voltage (0V) and from -1 to -10000 for inverted output ON voltage A value of 0 means that no pulse is generated
SetPWMLimits	Parameter *Period*, *TimeForZero* and *TimeFor100* range in microseconds (from 0 to 32000 µs) *TimeForZero* sets the minimum pulse with while *TimeFor100* sets the maximum

3. APPLICATION NOTES

This section contains some useful examples of IO_A applications

3.1 Managing DC motors

The speed that a DC motor can reach depends on the load that it has to move and on the average power delivered to it. The average power is proportional to the voltage and the current that flows through the motor. So, to manage the motor power, the designer can choice between changing the voltage or changing the average current

The most efficiently choice is to manage the average current by using a transistor turning on and off, quickly. As the voltage drop on the transistor is low, the power losses of this technique are low as well.

The IO_A board can manage up to three DC motors using the PWM outputs 0 , 1 and 2 of the transistor output block.

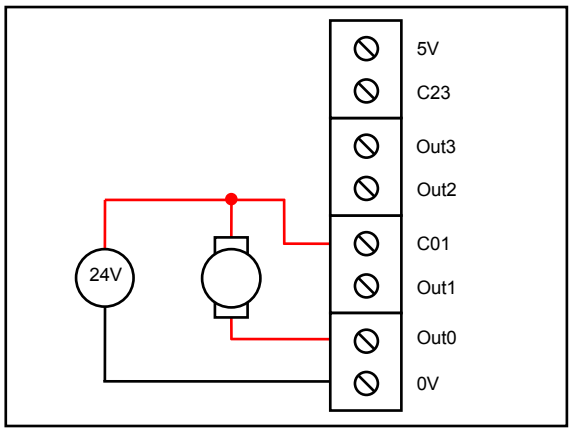


Fig. 13 24V DC Motor PWM managed

Figure 13 shows a 24V DC motor connected to a 24V power supply and managed by the *Out0*.

The Motor can be full stopped or full powered by turning on and off the *Out0* using `SetIndividualOutput` or `SetGroupOutput` (this is a digital management) or by using `SetPWMValue`, which is a proportional management. Ensure that the motor can withstand a 24V power permanently.

Lets decide to generate a 500Hz PWM (2ms) and a full duty span, that is, at 0% the *Out0* will be disconnect and at 100%, *Out0* will be permanently activated.

Here is an example written in C language (assuming that the board address has been set as 3)

```
DV0 dv0; // Create instance
int Board = 3;
// Open
int r = dv0.Open(NULL, 0, NULL, NULL)
if (r == 0) {
    // Open success, set PWM
    dv0.SetPWMLimits(Board,0,2000,0,2000);
    // Set to 50%
    dv0.SetPWMValue(Board,0,5000);
    // Set to 10%
    dv0.SetPWMValue(Board,0,1000);
    // Set to 73.21%
    dv0.SetPWMValue(Board,0,7321);
    // Set to 100%
    dv0.SetPWMValue(Board,0,10000);
    // Off
    dv0.SetPWMValue(Board,0,0);
}
```

3.2 Managing a servomotor for RC

Most RC Servomotors can be powered from 4.8V to 6V so it is possible to connect straightforward to the transistor block. Although it is not mandatory, we recommend to put an electrolytic capacitor in parallel with the 5V output to improve the RC servo behavior. The output control to the servo is easily obtained just connecting a 1KΩ pull-up, as it is shown in the figure 14. In this case, for example, *Out2* is used and a 1000uF electrolytic capacitor has been populated.

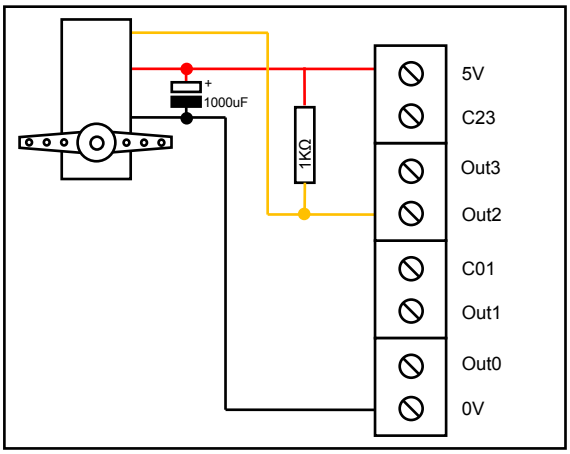


Fig. 14 Connecting a RC servomotor

The cable colors code mostly used is that red means positive supply, black is the negative one and pulse control can be brown, orange or white, depending on manufacturer.

The pulse itself (*Out2*) most often ranges from 1 ms to 2 ms, with ~1.5 ms being the neutral position. For servos with larger ranges, you might see pulses between 0.5 and 2.5 ms (neutral will still be around 1.5 ms).

Following is an example written in C language that sets the *Out2* as an RC Servomotor PWM control. IMPORTANT: to full fit the expected control waveform, the sign of the Duty cycle must be NEGATIVE.

```
DV0 dv0; // Create instance
int Board = 3;
// Open
int r = dv0.Open(NULL, 0, NULL, NULL)
if (r == 0) {
    // Open successfully, set PWM to 20ms period
    // Minimum value: 1ms, maximum value: 2ms
    dv0.SetPWMLimits(Board,2,20000,1000,2000);
    // Set to neutral
    dv0.SetPWMValue(Board,2,-5000);
    // Set to one extreme
    dv0.SetPWMValue(Board,2,-1);
    // Set to the opposite extreme
    dv0.SetPWMValue(Board,2,-10000);
    // Expand the range from 0.6ms to 2.5ms
    dv0.SetPWMLimits(Board,2,20000,600,2500);
    // Set to neutral
    dv0.SetPWMValue(Board,2,-5000);
}
```

3.3 Connecting NPN or PNP industrial sensors

There are a lot of digital sensors in the industrial field powered to 12, 24, 36 or even 48V, intended for detecting objects, measuring inclination, light barrier, etc. These sensors can be connected to the opto-coupled inputs of the IO_A board easily. This section explains how to connect those sensors depending on the their kind of output.

Digital sensor output can be NPN or, more often, PNP. The different is where this output is referenced: to ground or to positive rail. Figure 15 shows a PNP sensor connected to the *Input0* while Figure 16 shows a NPN sensor at the same input

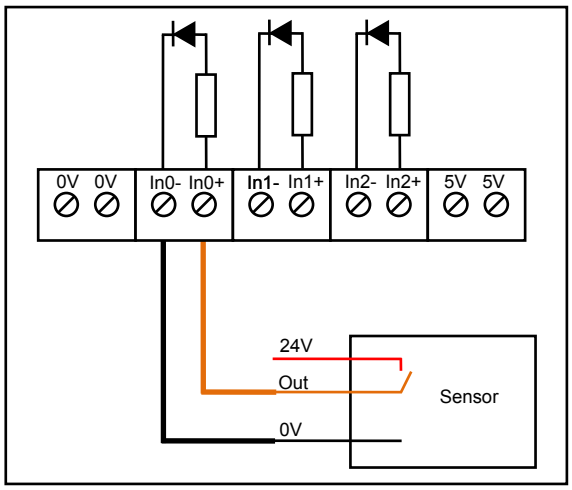


Fig. 15 PNP Sensor connection

In both figures, the rail 0V is not necessarily connected to the 0V of IO_A, they can be isolated.

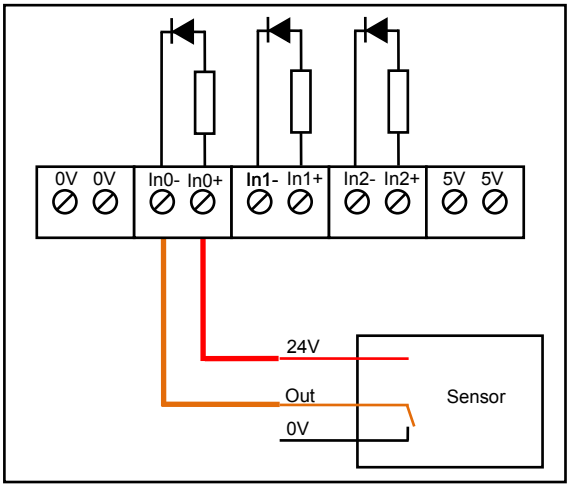


Fig. 16 NPN Sensor connection

Also, whatever the kind of sensor used, from the point of view of user application, functions like *GetIndividualInput*, *GetPulseCounting* or *GetFrequency* don't see any difference.

3.4 Connecting analog sensors

When connecting analog sensor to the analog inputs, the designer must take into account two points:

- The output impedance of the analog source should be less than 2.5KΩ. As a rule, the lower, the better.
- Noise from the power supply can be propagated and amplified by the electronic inside the sensor itself.

For example, a resistive pressure sensor exhibits a resistance from 1MΩ if no force is applied over it and close to 1KΩ when 1Kg is applied. In this case, it is mandatory to insert an operational amplifier in a buffer configuration, as Figure 17 illustrates.

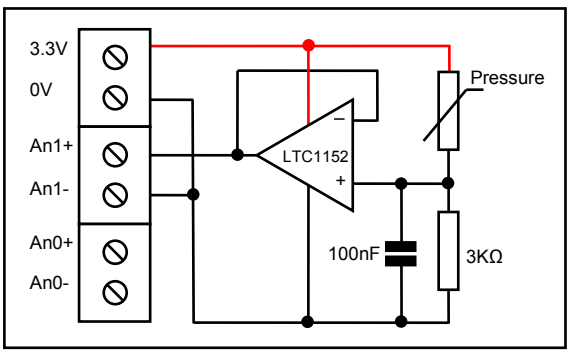


Fig. 17 NPN Sensor connection

Care must be taken when selecting the operational amplifier. Designers must check the following features:

- Low voltage. Ensure that the operational amplifier can work with 3.3V.
- Rail to rail input and output. Ensure that the operational amplifier is capable of swinging the output from 0V to 3.3V and of accepting input voltage from 0V to 3.3V

Finally, the 100nF capacitor smoothes the noise present at 3.3V and the noise induced by near equipments if wires are long.

3.5 Measuring temperature with TMP36/37

The TMP36 and the TMP37 are a 3-wires temperature sensor that yields a voltage output proportional to the ambient temperature. Both are suitable to be connected to the IO_A analog inputs because:

- They operate from 2.7V to 5V, so 3.3V is correct
- They have a very low output impedance

The TMP36 output is 10mV per centigrade degree, with an offset of 500mV at zero degrees. This sensor is interesting if a measuring under 0ºC is desired, but it is worth to note that IO_A analog inputs have an accuracy of 8mV, that means an error of close to 1ºC just measuring.

The TMP37 output is 20mV per centigrade degree which offers a better accuracy than the TMP36, but it can not measure temperatures below zero degrees.

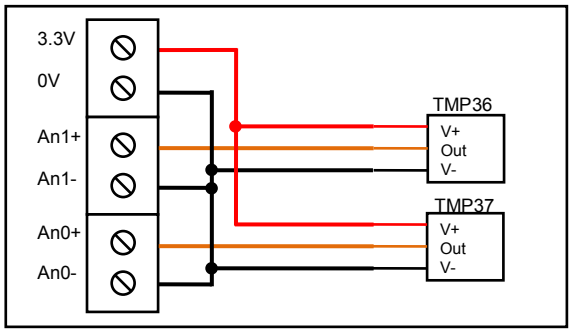


Fig. 18 Connecting TMP36/37 example

Because of their low output impedance and a large power supply rejection (less than 0.1ºC/V), a large wire is acceptable even without filtering capacitors.

Following there is an example written in C language that reads both sensors and computes the temperature measured.

```
DV0 dv0; // Create instance
int Board = 3; int Value;
double T0, T1;
// Open
int r = dv0.Open(NULL, 0, NULL, NULL)
if (r == 0) {
    // Open successfully
    // read Analog1,(in mV)
    dv0.GetAnalogValue(Board,1,&Value);
    // And compute temperature from TMP36
    // who has an offset of 0.5V at 0ºC
    T1 = (double)(Value-500)/10.0;
    // Now, get Analog0,
    dv0.GetAnalogValue(Board,0,&Value);
    // And compute temperature from TMP37
    // who has no offset voltage
    T0 = (double)(Value)/20.0;
}
```

3.6 Controlling motorized blinds

Motorized blinds have two-phases motors that allows to raise and to lower the blinds. These motors are powered by the AC mains, which has a voltage of 125Vac up to 240Vac that can only be managed by the contacts of the relay outputs.

CAUTION:

These voltage levels can be **lethal**. Never manipulate this cables without previously assuring that there is no voltage on them and that these lines are conveniently protected by a short circuit breaker and a residual-current circuit breaker

Figure 19 shows an IO_A board that manages one two-phases blinds motor and two push buttons to raise and lower the blind. As an example, there is a fragment of C code to manage the push buttons and de blinds motor.

```
DV0 dv0; // Create instance
int Board = 3; int Value;
double T0, T1;
// Open
int r = dv0.Open(NULL, 0, NULL, NULL)
if (r == 0) {
   // Get In0 push button
   dv0.GetIndividualInput(Board,0,0,&Value);
   if (Value == 1) {
      // Push button 0 pressed
      // Activate Up phase
      dv0.SetIndividualOutput(Board,0,0,1);
      Wait(20); // Wait 20 seconds
      // Deactivate Up phase
      dv0.SetIndividualOutput(Board,0,0,0);
   }
   // Get In1 push button
   dv0.GetIndividualInput(Board,0,1,&Value);
   if (Value == 1) {
      // Push button 1 pressed
      // Activate Down phase
      dv0.SetIndividualOutput(Board,0,1,1);
      Wait(20); // Wait 20 seconds
      // Deactivate Down phase
      dv0.SetIndividualOutput(Board,0,1,0);
   }
}
```

CAUTION:

Programmers must ensure that never will the phase up and down be energized simultaneously. Otherwise, although the motor is thermally protected, there are some mechanical stresses that can damage the motor

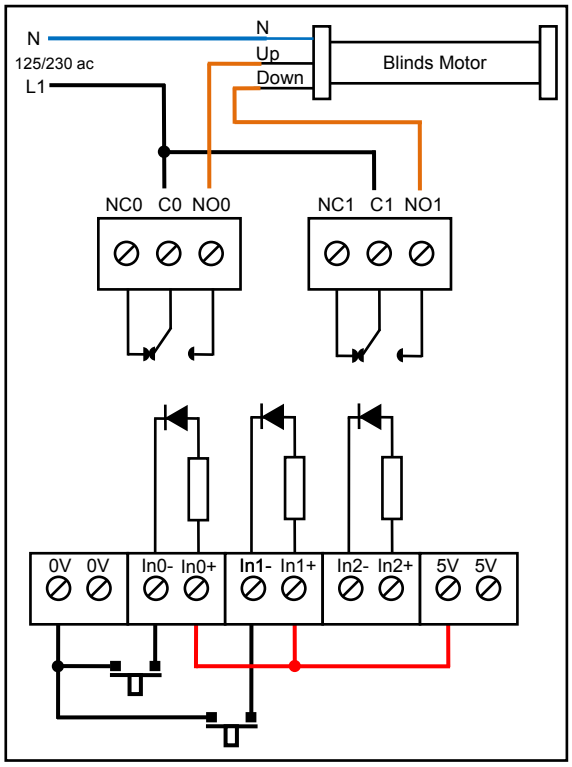


Fig. 19 Raising and lowering blinds

4. SOFTWARE

It is recommended to read the Software section of Header1 prior reading this section.

4.1 *DV0* API functions related to IO_A

The DV0 API contains a set of functions intended to manage the IO_A board that can be found for Java, C++ and Python languages.

Some examples of those functions have been illustrated in the sections 2 and 3. This section is intended as a formal declaration of parameters and return values

WARNING:

This data sheet summarizes the features of DV0 API functions related to IO_A board. It is not intended to be a comprehensive reference source. For more information, refer to the DV0 API Reference Guide for C++, Java and Python at www.devalirian.com, in the Technical Information section

The steps for managing the IO_A control board programmatically are:

- Create an instance of the class DV0
- Call the function Open
- If successful, call IO_A functions related.

In the following sections, an example of each language will be given for creating instance, for calling Open function and for calling related functions. Also, more complete examples can be found at the end of this chapter.

Table 5 summarize the functions related to IO_A. Note that only the name of the function is listed in the table below. This is because the parameters and return values are slightly different among the three implementation (Java, C++ and Python).

Table 1. IO_A related functions

Name	Description
Open	Initial and mandatory function to connect with dv0_manager
SetIndividualOutput	Set/reset an individual output of a group
SetGroupOutput	Set/reset all outputs belonged to the group
SetPWMValue	Set the duty cycle of a PWM output
SetPWMLimits	Set the PWM period, as well as minimum an maximum pulse value
GetIndividualInput	Returns the state of a particular input of a group
GetGroupInput	Returns the state of all inputs belonging to a group
GetAnalogValue	Returns the value (in mV) of an analog input
GetPulseCounting	Returns the accumulated counted pulses on *In0*
GetFrequency	Returns the frequency measured on *In0*

4.1.1 Creating an instance of DV0 class

Java, C++ and Python are all object oriented languages and it is useful to encapsulate all the accessing to the API throughout one object. All subsequent managing functions will pass through this variable. Let's assume that this variable will be called *dv0*. For C++ or Java, the creation of this variable is thoroughly:

```
DV0 dv0; // C++ or Java
```

And so is for Python, but slightly different (actually, It is not a class)

```
import dv0
```

4.1.2 Connecting to *dv0_manager*

The connection to *dv0_manager* is made through the function Open.

Function Open generic description

```
int Open(IpAddress, Port, Login, Password);
```

Open connection with the dv0_manager. Mandatory for all other members. Blocking.

Parameters:

IpAddress is the IP where dv0_manager listens to incoming connections. By default is "localhost" that suits perfectly is this application runs in the same machine that dv0_manager does. If a remote connection is desired, call dv0_manager with the argument -ip_address followed by the IP address of the internet interface of your board (use ifconfig to know it) and pass this value to the current IpAddress parameter.
Port is the UDP port where dv0_manager listens to incoming connections. By default is 6900 but DV0_manager can listen to whatever port selected by command line argument -port. In this case, give the same value to current Port parameter. Useful only if another application had catch this port.
Login and **Password** are required if the dv0_manager daemon is called with -login <username> argument. Current Login parameter must match with <username> and a valid Password must be entered

Return values

Returns 0	if connection is successful
DV_NOT_CONNECTION	if dv0_manager does not respond. Check IpAddress and Port
DV_INVALID_LOGIN	if dv0_manager refuses login or password

C++ Syntax

```
int Open(char *IpAddress = NULL, int Port = 0, char *Login = NULL, char *Password = NULL);
```

Java Syntax

```
int Open(String IpAddress, int Port, String Login, String Password);
```

Python Syntax

```
def Open(ipAddress, port, login, password):
```

Example: Connect to a *dv0_manager* running in the same Raspberry that the current program, using de default port and with no login required. The *dv0_manager* call can be:

```
/home/pi/dv0/dv0_manager -force_date
```

From a C++ user program, then:

```
DV0 dv0;
…
int r = dv0.Open(NULL, 0, NULL, NULL)
if (r == 0) {
…
} else DisplayError(r);
```

From a Java user program, then:

```
DV0 dv0;
…
int r = dv0.Open(“”, 0, “”, “”)
if (r == 0) {
…
} else DisplayError(r);
```

From a Python program, there are no so much differences:

```
import dv0
res = dv0.Open('',0,'','')
if (res == 0):
    print ("Connected to dv0_manager")
```

```
else:
    DisplayError(res)
```

Example: Connect to a *dv0_manager* running in a Raspberry connected to internet through the interface 10.0.0.2, and the current program running somewhere, login name 'pi' required (with default password 'raspberry') , using de default port. The *dv0_manager* call can be:

```
/home/pi/dv0/dv0_manager -ip_address 10.0.0.2 -force_date -login pi
```

From a C++ and Java user program, then

```
DV0 dv0;
…
int r = dv0.Open("10.0.0.2", 0, "pi", "raspberry")
if (r == 0) {
…
} else DisplayError(r);
```

From a Python program, there are no so much differences:

```
import dv0
res = dv0.Open('10.0.0.2',0,'pi','raspberry')
if (res == 0):
    print ("Connected to dv0_manager")
else:
    DisplayError(res)
```

4.1.3 Summary of API functions related to IO_A

Function *SetIndividualOutput* generic description
`int SetIndividualOutput(int Board, int Group, int Output, int Value);`
`Activates the "Output" belonging to "Group" if Value is 1 or deactivate otherwise`
Parameters:

`Pre-conditions : "Board" matches with address micro switches in board`
`Group can be 0 or 1.`
`Output can be 0 or 1 if Group is 0 (relay) or can be from 0 to 3 if group is 1 (transistor outputs)`
`Default value is 0 for all outputs (deactivated)`

Return codes

`Returns 0 if successful or`
`DV_NOT_CONNECTED  if previous Open call had failed or connection has been canceled`
`DV_INVALID_BOARD  if that Board doesn't exist or it is not responding`
`DV_NOT_SUPPORTED  if there is not such group or output in that board`

C++ Syntax
`int SetIndividualOutput(int Board, int Group, int Output, int Value);`
Java Syntax
`int SetIndividualOutput(int Board, int Group, int Output, int Value);`
Python Syntax
`def SetIndividualOutput(Board, Group, Output, Value):`

Function *SetGroupOutput* generic description

```
int SetGroupOutput(int Board, int Group, int Value);
```

Set all outputs of that "Group" to 0 or 1 (deactivated or activated) depending on bits of "Value" where bit 0 acts over the less significant Out of that group. Non existent outputs should set to 0

Parameters:

Pre-conditions : "Board" matches with address micro switches in board
Group from 0 to 1, Value from 0 to 3 if Group is 0 or from 0 to 7 if Group is 1
Default values are all zero (deactivated)

Return codes

Returns 0 if successful or
DV_NOT_CONNECTED if previous Open call had failed or connection has been canceled
DV_INVALID_BOARD if that Board doesn't exist or it is not responding
DV_NOT_SUPPORTED if there is not such group in that board

C++ Syntax

```
int SetGroupOutput(int Board, int Group, int Value);
```

Java Syntax

```
int SetGroupOutput(int Board, int Group, int Value);
```

Python Syntax

```
def SetGroupOutput(Board, Group, Value):
```

Function *SetPWMValue* generic description

```
int SetPWMValue(int Board, int Output, int Value);
```

Set the output pulse with in steps of 0.01 percent. The output pulse is zero volt during the ON period if Value is positive or is left open if Value is negative

Parameters:

Pre-conditions : "Board" matches with address micro switches in board
Output from 0 to 3
Value from 1 (minimum pulse with) to 10000 (100%, maximum pulse with) for normal output ON voltage and from -1 to -10000 for inverted output ON voltage. A value of 0 means that no pulse is generated
Default value is zero

Return codes

Returns 0 if successful or
DV_NOT_CONNECTED if previous Open call had failed or connection has been canceled
DV_INVALID_BOARD if that Board doesn't exist or it is not responding
DV_NOT_SUPPORTED if there is not such output in that board

C++ Syntax

```
int SetPWMValue(int Board, int Output, int Value);
```

Java Syntax

```
int SetPWMValue(int Board, int Output, int Value);
```

Python Syntax

```
def SetPWMValue(Board, Output, Value):
```

Function *SetPWMLimits* description

```
int SetPWMLimits(int Board, int Output, int Period, int TimeForZero, int TimeFor100);
```

```
Define the limits and behavior of PWM.
For example, typical RC Model servo expects a 20ms period pulse with a minimum value of 1ms
(servo to left) and a maximum of 2ms (servo to right) which means:
    SetPWMValue(board, output, 20000, 1000, 2000)
and to center the servo, just order a 50% setting:
    SetPWMValue(board, output, 5000) or whatever other value in between
Default values are Period = 20000, TimeForZero = 1000 and TimeFor100 = 2000
```

Parameters:

```
Pre-conditions : "Board" matches with address micro switches in board
Output from 0 to 3
TimeForZero and TimeFor100 in micro seconds (from 0 to 32000 µs)
Period in micro seconds (from 500 to 32000  µs)
```

Return codes

```
Returns 0 if successful or
DV_NOT_CONNECTED      if previous Open call had failed or connection has been canceled
DV_INVALID_BOARD      if that Board doesn't exist or it is not responding
DV_NOT_SUPPORTED      if there is not such output in that board
```

C++ Syntax

```
int SetPWMLimits(int Board, int Output, int Period, int TimeForZero, int TimeFor100);
```

Java Syntax

```
int SetPWMLimits(int Board, int Output, int Period, int TimeForZero, int TimeFor100);
```

Python Syntax

```
def SetPWMLimits(Board, Output, Period, TimeForZero, TimeFor100):
```

Function *GetIndividualInput* description

```
int GetIndividualInput(int Board, int Group, int Input, int *Value);
```

```
Fills Value with the current opto-coupled input value
```

Parameters:

```
Pre-conditions : "Board" matches with address micro switches in board
Group: always 0
Input: from 0 to 2
```

Return codes

```
Returns 0 if successful or
DV_NOT_CONNECTED      if previous Open call had failed or connection has been canceled
DV_INVALID_BOARD      if that Board doesn't exist or it is not responding
DV_NOT_SUPPORTED      if there is not such input or group in that board
```

C++ Syntax

```
int GetIndividualInput(int Board, int Group, int Input, int *Value);
```

Java Syntax

```
int GetIndividualInput(int Board, int Group, int Input, int Value[]);
```

Python Syntax

```
def GetIndividualInput(Board, Group, Input):
# Returns a 2-elements tuple containing the current value of this input and the return code
```

Function *GetGroupInput* description

```
int GetGroupInput(int Board, int Group, int *Value);
```

Fills *Value with all current opto-coupled input values. In0 is represented on Bit0, In1 on Bit1 and In2 on Bit2.

Parameters:

Pre-conditions : "Board" matches with address micro switches in board
Group: always 0

Return code

Returns 0 if successful or
DV_NOT_CONNECTED if previous Open call had failed or connection has been canceled
DV_INVALID_BOARD if that Board doesn't exist or it is not responding
DV_NOT_SUPPORTED if there is not such group in that board

C++ Syntax

```
int GetGroupInput(int Board, int Group, int *Value);
```

Java Syntax

```
int GetGroupInput(int Board, int Group, int Value[]);
```

Python Syntax

```
def GetGroupInput (Board, Group, Input):
# Returns a 2-elements tuple containing the current value of all opto-coupled inputs and
# the return code
```

Function *GetAnalogValue* description

```
int GetAnalogValue(int Board, int Input, int *Value);
```

Fills Value[0] with current input voltage in mV, from 0 (0V) to 3300 (3.3V)

Parameters:

Pre-conditions : "Board" matches with address micro switches in board
Input: from 0 to 1

Return code

Returns 0 if successful or
DV_NOT_CONNECTED if previous Open call had failed or connection has been canceled
DV_INVALID_BOARD if that Board doesn't exist or it is not responding
DV_NOT_SUPPORTED if there is not such input in that board

C++ Syntax

```
int GetAnalogValue(int Board, int Input, int *Value);
```

Java Syntax

```
int GetAnalogValue(int Board, int Input, int Value[]);
```

Python Syntax

```
def GetAnalogValue (Board, Input):
# Returns a 2-elements tuple containing the voltage detected on such Input
# and a return code
```

Function *GetPulseCounting* description

```
int GetPulseCounting(int Board, int CountInput, int *Value);
```

Fills Value[0] with the accumulated counted pulses(0 to 65535) on the opto-coupled input 0

Parameters:

Pre-conditions : "Board" matches with address micro switches in board
CountInput: from 0 to 2

Return code

Returns 0 if successful or
DV_NOT_CONNECTED if previous Open call had failed or connection has been canceled
DV_INVALID_BOARD if that Board doesn't exist or it is not responding
DV_NOT_SUPPORTED if CountInput is not 0, 1 or 2

C++ Syntax

```
int GetPulseCounting(int Board, int CountInput, int *Value);
```

Java Syntax

```
int GetPulseCounting(int Board, int CountInput, int Value[]);
```

Python Syntax

```
def GetPulseCounting (Board, CountInput):
# Returns a 2-elements tuple containing the count input value for In0
# and a return code
```

Function *GetFrequency* description

```
int GetFrequency(int Board, int FrequencyInput, int *Value);
```

Fills Value[0] with the current frequency in Hertz (100 to 15000 Hz) measured on the opto-coupled input 0

Parameters:

Pre-conditions : "Board" matches with address micro switches in board
FrequencyInput: from 0 to 2, BUT ONLY input 0 measures frequency. Inputs 1 and 2 always return 0

Return code

Returns 0 if successful or
DV_NOT_CONNECTED if previous Open call had failed or connection has been canceled
DV_INVALID_BOARD if that Board doesn't exist or it is not responding
DV_NOT_SUPPORTED if FrequencyInput is not 0, 1 or 2

C++ Syntax

```
int GetFrequency(int Board, int FrequencyInput, int *Value);
```

Java Syntax

```
int GetFrequency(int Board, int FrequencyInput, int Value[]);
```

Python Syntax

```
def GetFrequency (Board, FrequencyInput):
# Returns a 2-elements tuple containing the frequency measured at In0
# and a return code
```

4.1.4 API examples

In this section, a complete example of how to manage Header1 related functions are described both in C++ and Python languages. Java example is omitted because it is so close to C++ that only adds confusion. However, there is an example of how to program an Android APP with the DV0 API, that is not included in this section for the benefit of simplicity but it can be found at www.devalirian.com, inside the Technical Information section.

4.1.4.1. C++ ExampleIO_A.cpp

```
#include <iostream>
#include <stdio.h>
using namespace std;
#include "DV0.h"

char *ServAddress       = NULL;
char *LoginName = NULL;
char *Password          = NULL;
int  Port = 0;
DV0  dv0;

void DisplayError(int r);
void DoTest(void);
int  input (char *text);

int main() {
        int r;
        // Trying to connect
        r = dv0.Open(ServAddress, Port, LoginName, Password);
        if (r == 0) {
                cout << "Connection to dv0_manager successful " << endl;
                DoTest();
        } else DisplayError(r);
        return r;
}

void DoTest(void) {
        int board, group, output, value, res, option, period, timeForZero, timeFor100;
        int Input;
        unsigned int Uvalue;
        // Print menu
        cout << "1-SetIndividualOutput 2-SetGroupOutput 3-SetPWMOutput" << endl;
        cout << "4-SetPWMLimits 5-GetIndividualInput 6-GetGroupInput" << endl;
        cout << "7-GetAnalogInput 8-GetPulseCounting 9-GetFrequency" << endl;
        cout << "10-Get Manifest 0-Exit" << endl;
        // Get user option
        cin >> option;
        // And execute
        switch (option) {
                case 1:
                        //----------------------------------------Individual output example
                        board = input ("Enter board address:");
                        group = input ("Enter group :");
                        output= input ("Enter output:");
                        value = input ("Enter value (1/0):");
                        res = dv0.SetIndividualOutput(board, group, output, value);
                        DisplayError(res);
                        break;
                case 2:
                        // ---------------------------------------- Group output example
                        board = input ("Enter board address:");
                        group = input ("Enter group :");
                        value = input ("Enter value :");
                        res = dv0.SetGroupOutput(board, group, value);
                        DisplayError(res);
                        break;
                case 3:
                        // ---------------------------------------- Set PWM value example
                        board = input ("Enter board address:");
                        output= input ("Enter output:");
                        value = input ("Enter value (0 to 10000) :");
```

```
        res = dv0.SetPWMValue(board, output, value);
        DisplayError(res);
        break;
case 4:
        // ---------------------------------------- Set PWM limits example
        board = input ("Enter board address:");
        output= input ("Enter output:");
        period= input ("Enter period(us):");
        timeForZero= input ("Enter time for zero(us):");
        timeFor100 = input ("Enter time for 100%(us):");
        res = dv0.SetPWMLimits(board, output, period, timeForZero, timeFor100);
        DisplayError(res);
        break;
case 5:
        // ---------------------------------------- Get Individual Input example
        board = input ("Enter board address:");
        group = input ("Enter group :");
        Input = input ("Enter input:");
        res = dv0.GetIndividualInput(board, group, Input, &value);
        if (res == DV0_OK)
                cout << "Value: " << value << endl;
        else
                DisplayError(res);
        break;
case 6:
        // ---------------------------------------- Get Group Input example
        board = input ("Enter board address:");
        group = input ("Enter group :");
        res = dv0.GetGroupInput(board, group, &value);
        if (res == DV0_OK)
                cout << "Value: " << value << endl;
        else
                DisplayError(res);
        break;
case 7:
        // ---------------------------------------- Get Analog Input example
        board = input ("Enter board address:");
        Input = input ("Enter input:");
        res = dv0.GetAnalogValue(board, Input, &value);
        if (res == DV0_OK)
                printf("Value: %3.3f V\n", (double)value/1000.0);
        else
                DisplayError(res);
        break;
case 8:
        // ---------------------------------------- Get Pulse counting example
        board = input ("Enter board address:");
        Input = input ("Enter input:");
        res = dv0.GetPulseCounting(board, Input, &Uvalue);
        if (res == DV0_OK)
                cout << "Value: " << Uvalue << endl;
        else
                DisplayError(res);
        break;
case 9:
        // ---------------------------------------- Get Frequency example
        board = input ("Enter board address:");
        Input = input ("Enter input:");
        res = dv0.GetFrequency(board, Input, &value);
        if (res == DV0_OK)
                cout << "Value: " << value << endl;
        else
                DisplayError(res);
        break;
case 10:
        // ---------------------------------------- Get Manifest example
        board = input ("Enter board address:");
        char description [756];
        res = dv0.GetBoardManifest(board, description);
        if (res == DV0_OK)
                cout << "Value: " << description << endl;
        else
                DisplayError(res);
        break;
```

```
        }
}

int input (char *text) { int option;
        cout << text;
        cin >> option;
        return option;
}

void DisplayError(int error) {
        switch (error) {
                case DV_NOT_CONNECTION:
                        cout << "dv0_manager does not respond. Check IpAddress and Port" << endl; break;
                case DV_INVALID_LOGIN:  cout << "dv0_manager refuses login or password" << endl; break;
                case DV_NOT_CONNECTED:  cout << "DVO header board is not connected" << endl; break;
                case DV_INVALID_BOARD:  cout << "Invalid board address" << endl; break;
                case DV_NOT_SUPPORTED:  cout << "Invalid peripheral" << endl; break;
                case DV0_OK: break;
                default: cout << "Unexpected error code" << endl; break;
        }
}
```

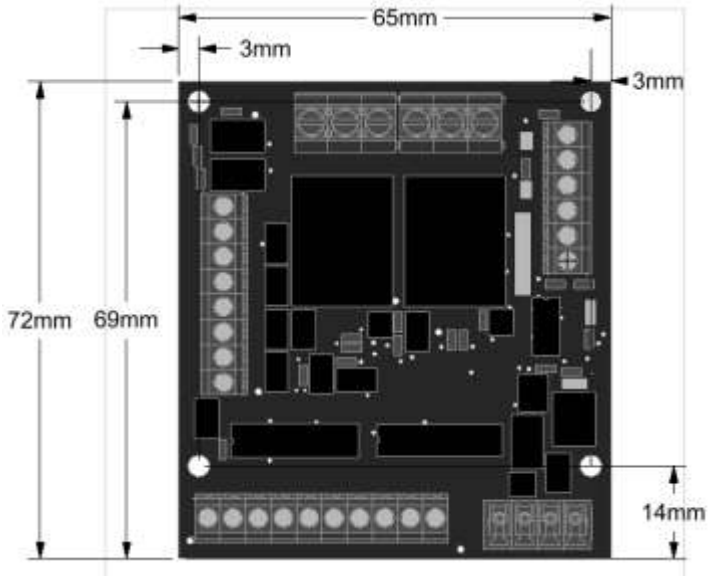
4.1.4.2. Python ExampleIO.py

```
import sys
import dv0

def DisplayError(error):
    if error == 0:
        return
    if error == dv0.DV_NOT_CONNECTION:
        print ("dv0_manager does not respond. Check IpAddress and Port")
    elif error == dv0.DV_INVALID_LOGIN:
        print ("dv0_manager refuses login or password")
    elif error == dv0.DV_NOT_CONNECTED:
        print ("DVO header board is not connected")
    elif error == dv0.DV_INVALID_BOARD:
        print ("Invalid board address")
    elif error == dv0.DV_NOT_SUPPORTED:
        print ("Invalid peripheral")
    else:
        print ("Unexpected error code: ", error)

# Assuming that the Raspberry is not this computer and connected as 10.0.0.2
# Also, dv0_manager is called like "dv0_manager -ip_address 10.0.0.2 -login pi"
res = dv0.Open('10.0.0.2', 6900, 'pi', 'raspberry')
# Assuming that the Raspberry is this computer
# and dv0_manager is called just like "dv0_manager"
# res = dv0.Open('',0,'','');

looping = True
if (res == 0):
    print ("Connected to dv0_manager")
else:
    DisplayError(res)
    looping = False
while looping:
    print ("1-SetIndividualOutput 2-SetGroupOutput 3-SetPWMOutput")
    print ("4-SetPWMLimits 5-GetIndividualInput 6-GetGroupInput")
    print ("7-GetAnalogInput 8-GetPulseCounting 9-GetFrequency")
    print ("10-Get Manifest 0-Exit")
    option = int(input("Enter option:"))
    if option == 1:
        # ----------------------------------------- Individual output example
        board = int(input ("Enter board address:"))
        group = int(input ("Enter group :"))
        output= int(input ("Enter output:"))
        value = int(input ("Enter value (1/0):"))
        res = dv0.SetIndividualOutput(board, group, output, value)
        DisplayError(res)
    elif option == 2:
        # ----------------------------------------- Group output example
        board = int(input ("Enter board address:"))
        group = int(input ("Enter group :"))
        value = int(input ("Enter value :"))
        res = dv0.SetGroupOutput(board, group, value)
        DisplayError(res)
    elif option == 3:
        # ----------------------------------------- Set PWM value example
        board = int(input ("Enter board address:"))
```

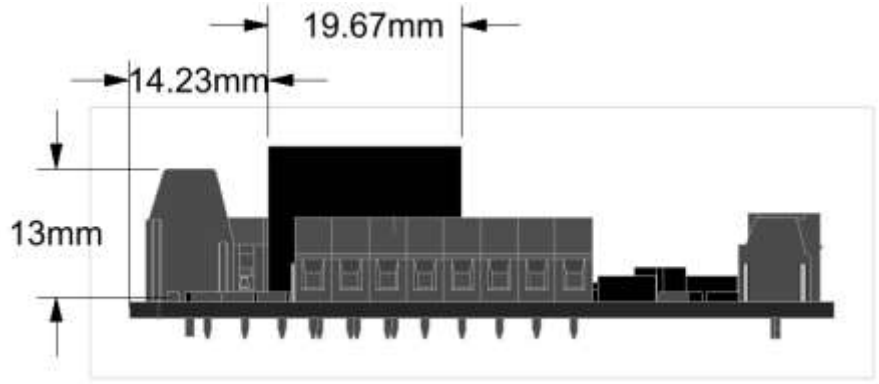
```
        output= int(input ("Enter output:"))
        value = int(input ("Enter value (0 to 10000) :"))
        res = dv0.SetPWMValue(board, output, value)
        DisplayError(res)
    elif option == 4:
        # ---------------------------------------- Set PWM limits example
        board = int(input ("Enter board address:"))
        output= int(input ("Enter output:"))
        period= int(input ("Enter period(us):"))
        timeForZero= int(input ("Enter time for zero(us):"))
        timeFor100 = int(input ("Enter time for 100%(us):"))
        res = dv0.SetPWMLimits(board, output, period, timeForZero, timeFor100)
        DisplayError(res)
    elif option == 5:
        # ---------------------------------------- Get Individual Input example
        board = int(input ("Enter board address:"))
        group = int(input ("Enter group :"))
        Input = int(input ("Enter input:"))
        value, res = dv0.GetIndividualInput(board, group, Input)
        if (res == dv0.DV0_OK):
            print ("Value: ", value)
        else:
            DisplayError(res)
    elif option == 6:
        # ---------------------------------------- Get Group Input example
        board = int(input ("Enter board address:"))
        group = int(input ("Enter group :"))
        value, res = dv0.GetGroupInput(board, group)
        if (res == dv0.DV0_OK):
            print ("Value: %x " % value)
        else:
            DisplayError(res)
    elif option == 7:
        # ---------------------------------------- Get Analog Input example
        board = int(input ("Enter board address:"))
        Input = int(input ("Enter input :"))
        value, res = dv0.GetAnalogValue(board, Input)
        if (res == dv0.DV0_OK):
            print ("Value: %3.3f V" % (value/1000))
        else:
            DisplayError(res)
    elif option == 8:
        # ---------------------------------------- Get Pulse Counting example
        board = int(input ("Enter board address:"))
        Input = int(input ("Enter input :"))
        value, res = dv0.GetPulseCounting(board, Input)
        if (res == dv0.DV0_OK):
            print ("Value: ", value)
        else:
            DisplayError(res)
    elif option == 9:
        # ---------------------------------------- GetFrequency  example
        board = int(input ("Enter board address:"))
        Input = int(input ("Enter input :"))
        value, res = dv0.GetFrequency(board, Input)
        if (res == dv0.DV0_OK):
            print ("Value: ", value)
```

```
        else:
            DisplayError(res)
    elif option == 10:
        # ---------------------------------------- Get Manifest example
        board = int(input ("Enter board address:"))
        description, res = dv0.GetBoardManifest(board);
        if (res == dv0.DV0_OK):
            print (description)
        else:
            DisplayError(res)
    elif option == 0:
        looping = False
```

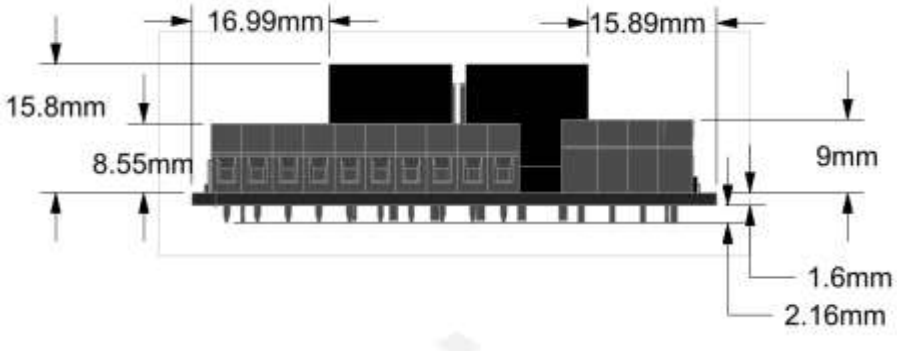
5. MECHANICAL DRAWINGS



Board dimensions and hole positions (top view)



Board dimensions (left view)



Board dimensions (front view)

6. ACCESSORIES

The IO_AI board comes with the following accessories:

- One DV0 socket. Model 20020008-D041B01LF

Also, a DIN RAIL adapter is available at www.devalirian.com

IMPORTANT NOTICE

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications.

deValirian MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE**.**

deValirian disclaims all liability arising from this information and its use. Use of ***deValirian*** devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless ***deValirian*** from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any ***deValirian*** intellectual property rights.

Header1 board is not designed to be radiation tolerant

Please be sure to implement in your equipment using the safety measures to guard against the possibility of physical injury, file or any other damaged cause in event of the failure of IO_A board. deValirian shall bear no responsibility whatsoever for you're your use of IO_A board outside the prescribed scope or not in accordance with this manual.

Reproduction of significant portions of ***deValirian*** information in ***deValirian*** data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. ***deValirian*** is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Web Site: www.devalirian.com

Mail info: info@devalirian.com