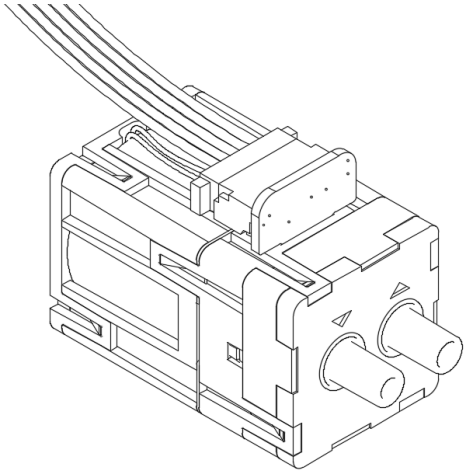
Electro Optical Components, Inc.
5464 Skylane Boulevard, Suite D, Santa Rosa, CA 95403
Toll Free: 855-EOC-6300
www.eoc-inc.com | info@eoc-inc.com

Click to go to Product Selector

Rev 0.2

Xavitech V100



Technical Data

	Min	Typical	Max
Supply Voltage (V100-12V)	-	12 V	-
Supply Voltage (V100-5V)	-	5 V	-
Flowrate	-	200 ml/min	-
Vacuum	-	300 mbar	-
I2C logic levels	-	2.8 V	3.3 V
I2C speed grade	-	100 kbit/s	400 kbit/s
VCC Capacitor (V100-12V)	100µF, 16V	-	-
VCC Capacitor (V100-5V)	47µF, 10V	-	-

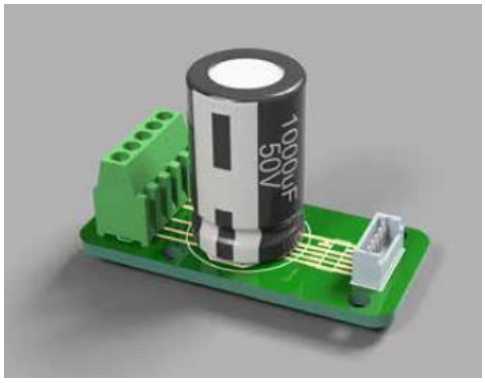
Electrical Interface

Red – VCC
Black – GND
White – IO
Green – SCL (I2C clock)
Blue – SDA (I2C data)

Pull-up resistors are needed for the I2C communication lines. Suitable values depend on the bus capacitance. Some master devices have built-in pull-up resistors. A Raspberry Pi for example has **1.8 kOhm** resistors installed.

Logic levels are 2.8 V (max 3.3 V).

A Capacitor parallel to the power supply is recommended to reduce power spikes generated by the electromagnet pump motor. An adapter board with capacitor is sold separately.



Adapter board

I2C Communication

I2C communication works with both standard mode: 100 kbit/s and full speed: 400 kbit/s.

The standard 7-bit address is **0x4A (74)**.

The I2C protocol is either sending or receiving 9 bytes of data plus a checksum. The first 9 bytes are reserved for values, usually 1, 2 or 4 bytes are used for setting or read a value. During a write operation a command byte is also sent, which gives a total of 11 bytes, please see example.

The checksum is calculated like this: **256 – mod(byte 1 + byte 2 + … + byte 9)** Another way of describing it is: All bytes including the checksum should add up to zero if you use an **unsigned 8-bit integer** and let it roll-over. This means that checking is easy, just add all the numbers and check if they add to zero.

Write Operation

Master: I2C write address + 11 bytes. The 11 bytes consist of Command Number (1 byte) and Data (10 bytes including checksum)

[I2C address + Write], [Command], [Data 1], [Data 2], […], [Data 10]

Arduino example:

```
// Set user frequency to maximum
uint8_t bytesToSend[] = {29,255,3,0,0,0,0,0,0,0,225};

Wire.begin();
Wire.setClock(100000);

Wire.beginTransmission(0x4A);          // Begin transmission (address: 0x4A)
Wire.write(bytesToSend,11);            // Write the bytes
Wire.endTransmission();                // End transmission
```

Raspberry Pi example using C++ and bcm2835.h:

```
// Set user frequency to maximum
uint8_t bytesToSend[] = {29,255,3,0,0,0,0,0,0,0,0,225};

bcm2835_i2c_begin();                   //Start I2C operations.
bcm2835_i2c_setSlaveAddress(0x4A);     //I2C address
bcm2835_i2c_set_baudrate(100000);      //baud rate

bcm2835_i2c_write((char*)bytesToSend,11);       // Write the bytes

bcm2835_i2c_end();                     // End transmission
bcm2835_close();
```

Read Operation

Master: I2C write address + 1 byte (Command Number).

[I2C address + Write], [Command]

Master: I2C read address

[I2C address + Read]

Slave: 10 bytes of data including checksum.

[Data 1], [Data 2], […], [Data 10]

Arduino example:

```
// Get user frequency
uint8_t bytesToReceive[10];
uint8_t command = 29;

Wire.begin();
Wire.setClock(100000);

Wire.beginTransmission(0x4A);          // Begin transmission (address: 0x4A)
Wire.write(&command,1);                // Write the command byte
Wire.endTransmission();                // End transmission

Wire.beginTransmission(0x4A);          // Begin transmission (address: 0x4A)
Wire.requestFrom(0x4A, 10);            // Request 10 bytes of data

uint8_t counter = 0;

while(Wire.available() || counter < 10)         // While bytes available, read bytes
{
    bytesToReceive[counter] = Wire.read();      // Read byte
    counter++;
}

Wire.endTransmission();                // End transmission
```

Raspberry Pi example using C++ and bcm2835.h:

```
// Get user frequency
uint8_t bytesToReceive[10];
uint8_t command[1] = {29};

bcm2835_i2c_begin();                   //Start I2C operations.
bcm2835_i2c_setSlaveAddress(0x4A);     //I2C address
bcm2835_i2c_set_baudrate(100000);      //baud rate

bcm2835_i2c_write((char*)command,1);   // Write the command byte
bcm2835_i2c_read(bytesToReceive,10);   // Read 10 bytes

bcm2835_i2c_end();                     // End transmission
bcm2835_close();
```

Commands

Commands can generally be temporary settings or stored. To store a setting add **64 (0x40)** to the command number while writing the command.

User Frequency
Command number: **29** (store: 93)

Value: 0-1023 (10 bits). 2 bytes are needed, LSB to MSB. Max value (1023) corresponds to calibrated value. 1 corresponds to lowest possible setting and 0 will turn the pump off.

Revision History

0.1 First revision
0.2 Fixed error in communication structure. Values are sent LSB first, not MSB first.