UM Unified Microsystems

IOX-16 User's Manual

Version 1.00

April 2013

Overview

The IOX-16 Arduino compatible shield is an easy way to add 16 additional digital Input/Output (I/O) lines to your Arduino system. It uses I2C communications for control. I2C is a bidirectional serial protocol that only requires two control lines. The I2C protocol allows interfacing with a wide variety of devices, and multiple units of the same device. For example, up to 8 IOX-16 boards may be stacked providing a total of 128 additional I/O lines. Note that I2C is also referred to as a Two Wire Interface (TWI).

The heart of the IOX-16 is a Microchip Technology MCP23017 I/O expander IC. The MCP23017 has two 8 bit ports, A & B. Each port has its own set of registers that must be configured in your program (sketch) for the desired operation. Each bit in the ports can be individually configured as an input or output. The pins configured as outputs can be set high or low and the state of the input pins can be read by the Arduino.

The MCP23017 is a complex IC. The examples in Appendix 1 use the simplest modes to be easier to understand and make changes. If you need to use the more complex features of the IOX-16 it is suggested you study the MCP23017 data sheet (available on the Microchip website – www.microchip.com) and write your program to match your particular application.

The IOX-16 has several hardware features to make it as flexible as possible to meet the needs of your application. This requires the IOX-16 to be configured before use.

IMPORTANT! The IOX-16 voltage jumper must be set before plugging it into your Arduino or Arduino compatible CPU modules. Severe damage to the IOX-16 and/or Arduino can result from improper configuration.

Specifications

Models:

IOX-16	Expander with 8 pin header for use with Duemilanove, Uno, etc.
IOX-16-R3	Expander with 10 pin connector for use with Uno R3 and later Arduino models

Size: 2.75" X 2.1"

Maximum external DC supply voltage: 15V

Internal Supply voltage: 3.3V or 5.V (selectable)

Output sink/source current maximum: Individual pin: 25ma IC total: 125ma

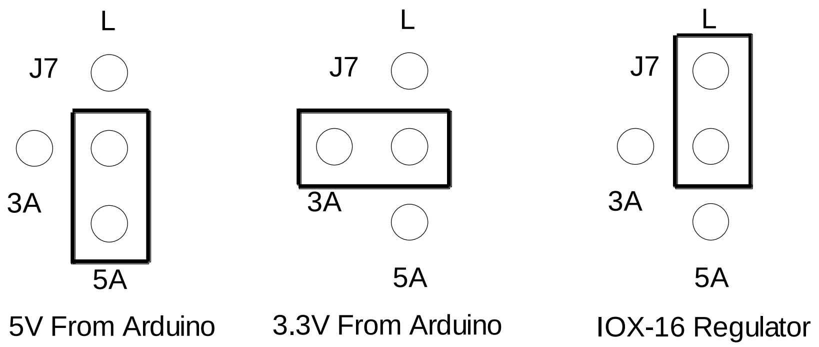
Configuration

This section covers how to set the jumpers on the IOX-16 prior to use. ***The jumpers must be configured before installing the IOX-16 on your Arduino.***

Voltage Selection

Jumper J7 is used to set the operating voltage. It is critical that the voltage is set to match the voltage of the Arduino or Arduino compatible CPU. Mixed voltages may damage or destroy the Arduino, IOX-16 or other shields in the stack. You can use the 5V or 3.3V supplied by the Arduino by setting the jumper J7 to 5A (5V from Arduino) or 3A (3.3V from Arduino).

The Arduino on board regulator may not be able to supply enough current if your application draws too much power. Also, USB supplied power is limited to 500ma. If your application requires more current, you can supply power through the DC in connector on the Arduino. Then you can use the IOX-16 5V on-board regulator by setting jumper J7 to the L position (Local Regulator).



Address Selection

The Arduino communicates to the IOX-16 using the I2C protocol. The I2C protocol includes a two part address. The first part of the address identifies the type of IC on the bus. This allows using ICs with different functions on the I2C bus. You could for example have an I2C memory device or a real time clock device in addition to the IOX-16 expander. The base address for the IOX-16 is hex 0x20. The second part is the unit address. It represents the unit number. You can have up to 8 IOX-16 boards stacked to get up to 128 additional I/O lines.

Each IOX-16 must have a different address with a value of 0-7. The first IOX-16 will normally have the address 0, the second will be address 1, etc. The factory default is address 0. The jumpers S0, S1 and S2 set the board address. Add or remove the shorting blocks per the table below to set each IOX-16 board addresses.

Unit	ADDRESS	S2	S1	S0
0	0x20	IN	IN	IN
1	0x21	IN	IN	OUT
2	0x22	IN	OUT	IN
3	0x23	IN	OUT	OUT

Unit	ADDRESS	S2	S1	S0
4	0x24	OUT	IN	IN
5	0x25	OUT	IN	OUT
6	0x26	OUT	OUT	IN
7	0x27	OUT	OUT	OUT

I2C Control Signal Source

The IOX-16 uses two I2C control lines, SCL (clock) and SDA (data). These signals are shared with the A4 and A5 signals on the Arduino Uno and Duemilanove. Newer version Arduino processor boards may expand the connector with I/O pins D8-D13 to 10 pins and put the SDA and SCL signals on the new pins. The Uno R3 has SDA and SCL on both the A4 and A5 lines as well as these expansion pins. The IOX-16 PCB has support for future use of the alternate source.

Jumpers J16 and J17 select the source for SDA and SCL. These jumpers should be left in the factory default state with the shorting blocks to the sides with the J16 and J17 labels when using Arduino Uno R3 and earlier boards that provide the I2C signals on pins A4 and A5.

I2C Control Signal Pull Ups

The SDA and SCL I2C control lines are open collector types. This means they need to have pull up resistors to the source voltage. Only one set of pull up resistors should be on the I2C bus. Jumpers G1 and G2 are used to connect the 2.2K ohm pull up resistors to the supply voltage. Remove G1 and G2 if you have another shield using I2C and it is using its own pull ups. If you have multiple IOX-16 boards in your project, leave G1 and G2 installed on one board and remove them on the others.

I/O Headers

The 16 expansion I/O lines are available on 10 pin right angle connectors on the edge of the board. The bits are segregated into two 8 bit ports, Port A and Port B. The right angle headers are designed to be used with standard .1" ribbon connectors. The pin out of the connectors is shown in the following table.

Pin	Signal
1	AX/BX Spare - connect as needed
2	Ground
3	Port bit 7
4	Port bit 6
5	Port bit 5

Pin	Signal
6	Port bit 4
7	Port bit 3
8	Port bit 2
9	Port bit 1
10	Port bit 0

Prototyping Support

The IOX-16 has areas to put your own custom circuitry needed for your project in the form of pads in a .1" grid. The IOX-16 has additional features to make wiring your custom circuits easier. There are two areas of the prototype area with white outline boxes near the edges of the circuit board. These indicate areas where the Arduino CPU board has connectors that might interfere with your circuit wiring. Special care must be used above the USB connector where the metal case may short out your circuitry. Use care when placing components in these zones.

Ground Pads

There are two sets of 3 pads in a white box labeled GND. You can get access to ground by wiring to any of these pads.

VDD Pads

There are 3 pads in a white box labeled VDD near the center of your board. You can access your supply voltage from any of these pads. The voltage at these pads will be the same as the voltage selected with J7.

Arduino Signal Pads

The signals from the Arduino are brought out to pads next to the stacking connectors. These pads are labeled for your convenience.

I/O Expansion Signal Pads

The 16 I/O lines are routed to the 10 pin right angle connectors. Each signal also has a solder pad if your custom on-board circuitry needs access. These are labeled with the letter A or B and the bit number. For example A0 is bit 0 of Port A. B3 is Port B, bit 3, etc.

I/O Connector Spare

The right angle I/O headers leave one pin available for connection to one of the Arduino signals, power or a signal on your custom circuit. These go to pin 1 of the right angle headers. Pad AX is for Port A and BX is on the Port B connector.

Interrupt lines

The MCP23107 IC has a number of options that allow the IC to generate an interrupt to the host CPU under certain circumstances such as an input pin changing state. The interrupt signal lines for Port A and Port B are available at the pads marked IA and IB respectively. If you want to use interrupts you will need to wire up one or both of these pins to one of the Arduino pins supporting interrupts. Refer to the MCP23017 data sheet for information on configuring the internal registers for generating interrupts.

Programming Information

Since the IOX-16 uses the I2C interface, the easiest way to communicate with it is with the standard Arduino Wire library functions. If you are not familiar with I2C or the Arduino Wire library more information can be found at http://arduino.cc/en/Reference/Wire If the Wire library is not already in your library directory, down load it from the Arduino site and install it.

The Microchip MCP23017 is a complex IC. To fully utilize its capability refer to the data sheet available from www.microchip.com. However, for simple applications the examples in Appendix 1 will help get you going quickly. You can down load these examples from the Support and Download page at **www.unifiedmicro.com**.

Warranty Information

The IOX-16 is warranted for 90 days from purchase date. Unified Microsystems will, at its option, repair or replace defective units returned during the warranty period. **Warranty does not cover damage due to improper setting of the voltage select jumpers or damage caused when adding circuits to the prototype area.** Unified Microsystems reserves the right to change specifications of it products at any time without notice. Schematics, sample programs, FAQs, and additional information can be found at **www.unifiedmicro.com/support**. Support is provided by email only: **w9xt@unifiedmicro.com** Include IOX-16 in the subject line.
www.unifiedmicro.com

APPENDIX 1

Example Program – Output Demo
This program can be entered by hand or can downloaded from the support and down load page at www.unifiedmicro.com

```
//#########################################################################################################################
// IOX-16 Arduino Output Demo program - writes test pattern to LEDs on both ports
// This program is a simple example to show how to set up the IOX-16 or other Arduino shield that uses a Microchip MCP23017 I/O expander IC.
// It will turn the LEDs on and off in a set of patterns.  The LEDs can be connected to either port.
//
// The IOX-16 must be set up as follows:
// * The address is set up to 0
// * The output pins are connected to LEDs with appropriate current limit resistors
//
//   Port pin ----/\/\/\----------------->|-----  Ground
//            330 ohm resistor      LED
//
//   Duplicate the above circuit for each Port pin (0-7)
//
// Revision 1.00  4/19/2013 Gary C. Sutcliffe and Unified Microsystems  www.unifiedmicro.com
// This program is released to the public domain
//*************************************************************************************************************************
#include <Wire.h>  //The I2C communications uses the Wire library
```

```
const byte  IOX_BASE_ADR =0x20;      // Base Address of MCP23017 Chip with address lines all set to zero (grounded)
                                     // Address for second MCP23017 would be 0x21, the next 0x22, etc.

//MCP23017 internal registers - Not all registers included. See the Microchip MCP23017 datasheet for full list

const byte  IODIRA = 0x00;          // Port A direction register. Write a 0 to make a pin an output, a 1 to make it an input
const byte  IODIRB = 0x01;          // Port B direction register
const byte  GPIOA = 0x12;           // Register Address of Port A
const byte  GPIOB = 0x13;           // Register Address of Port B

#define MAXPAT 15  //Number of LED test patterns

// the testPat array holds the patterns to output to turn the LEDs on and off. A 1 will turn the LED on.
const byte testPat[] = {0xff,0x00,0xaa,0x55,0x00,0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80,0x00,0x0f};

void setup()
{
    Wire.begin();             // join i2c bus

  //The port pins can be either inputs or outputs. Set both ports to all outputs
  Wire.beginTransmission(IOX_BASE_ADR);  //all I2C commands begin with this statement and the address of the chip.
  Wire.write((byte)IODIRA); // select a write to the IODIRA register
  Wire.write((byte)0x00); // set all of bank A to outputs. 0 = output, 1 = input
  Wire.write((byte)0x00); // set all of bank B to outputs. If we do multiple writes inside a single I2C command, the port address increments
  Wire.endTransmission();  // all I2C commands must end with this statement
  } /*** end of setup() ***/

void loop()
{
//Each pass through the loop will incremend idx which is used to index the pattern in the testPat[] array
for(byte idx = 0;idx < MAXPAT; idx++)
   {
   Wire.beginTransmission(IOX_BASE_ADR);   //All accesses to the expander start with this statement to the chip address
   Wire.write(GPIOA);      // address bank A
   Wire.write((byte)testPat[idx]);  // value to send
   Wire.endTransmission();

//after sending the pattern to Port A, repeat to Port B
   Wire.beginTransmission(IOX_BASE_ADR);
   Wire.write(GPIOB);      // address bank B
   Wire.write((byte)testPat[idx]);  // value to send
   Wire.endTransmission();
    delay(300);    // wait 300 miliseconds before changing to the next pattern
   }
} /*** endof loop() ***/
```

Example Program – Input Demo

This program can be entered by hand or can downloaded from the support and down load page at www.unifiedmicro.com

```
//#############################################################################################################################################
// IOX-16 Arduino Input Demo program - Read the values of the inputs and send the results to the Arduino Serial Monitor
// This program is a simple example to show how to set up the IOX-16 or other Arduino shield that uses a Microchip MCP23017 I/O expander IC.
// It reads Port A and prints the the results in the Serial Monitor.
//
// The IOX-16 must be set up as follows:
// * The address is set up to 0
// * For a simple test Port A data pins can be connected to ground through switches
//
//   Port A pin ----- \---------------  Ground
//              Switch
//
```

```
//   Duplicate the above circuit for each Port A pin (0-7)
//
// Enter this program and Upload to the Arduino. Once upload is completed, open the Serial Monitor in the Tools tab. If a switch is closed
// the corresponding pin will show up as a 1 on the Serial Monitor display
// Revision 1.00  4/20/2013 Gary C. Sutcliffe and Unified Microsystems  www.unifiedmicro.com
// This program is released to the public domain
//***************************************************************************************************************************************
#include <Wire.h>  //The I2C communications uses the Wire library. You might need to download from Arduino.cc and put into your library directory

const byte  IOX_BASE_ADR =0x20;      // Base Address of MCP23017 Chip with address lines all set to zero (grounded)
                                // Address for second MCP23017 would be 0x21, the next 0x22, etc.

//MCP23017 internal registers - Not all registers included. See the Microchip MCP23017 datasheet for full list
const byte  IODIRA = 0x00;          // Port A direction register. Write a 0 to make a pin an output, a 1 to make it an input
const byte  GPIOA = 0x12;           // Register Address of Port A - read data from or write output data to this port
const byte  GPPUA =  0x0c;          // Register to enable the internal pull up resistors on Port A. 1 = pull up enabled

void setup()
{
Serial.begin(9600);  //configure the serial port so we can use the Serial Monitor
Wire.begin();       // join i2c bus

 //The port pins can be either inputs or outputs. Set Port A pins as all inputs
Wire.beginTransmission(IOX_BASE_ADR);  //all I2C commands begin with this statement and the address of the chip.
Wire.write((byte)IODIRA); // select a write to the IODIRA register
Wire.write((byte)0xff); // set all of bank A to inputs. 0 = output, 1 = input
Wire.endTransmission();  // all I2C commands must end with this statement

 // This sets the pull ups on the port.  Don't do this if port is actively driven or circuit has external pull ups.
Wire.beginTransmission(IOX_BASE_ADR);
Wire.write((byte)GPPUA); // Select port A pull up register
Wire.write((byte)0xff);  //turn all the pull ups on. 1=ON, 0=OFF
Wire.endTransmission();
} /*** end of setup() ***/

void loop()
{
//Port A will be read each pass through the loop and the values on Port A will be displayed in the Serial Monitor
byte readData;  // location to store data from Port A

Wire.beginTransmission(IOX_BASE_ADR);  //need to set up the proper register before reading it
Wire.write(GPIOA);              // Even though we are doing a read, we first have to write the address of the register the read will be from
Wire.endTransmission();

Wire.requestFrom(IOX_BASE_ADR,(byte)1); //Now we start the actual read
while (Wire.available()==0);      //wait unit the data gets back
readData = ~Wire.read();          // get the data, note bitwise inversion to make it easier to tell what switch is closed
Wire.endTransmission();
Serial.println(readData,BIN); //Print out results on Serial Monitor.

delay(500);    // wait 500 miliseconds before checking again
} /*** end of loop() ***/
```