MOD-IO2

----
firmware version 4.3
----

1. MOD-IO2 description

MOD-IO2 is an open-source hardware expansion board. It has two relays, analog inputs, analog outputs, two PWMs, a DAC, a number of GPIOs. MOD-IO2 is stackable and each board can be individually addressed – the user can connect multiple MOD-IO2 boards together to increase the number of inputs and outputs. MOD-IO2 can be attached to any development board over I2C (or TWI). It comes loaded with open-source custom software which makes it easier to use.

MOD-IO2 is easy to attach to other Olimex-made boards since it comes with UEXT connector (and the UEXT has SCL and SDA typically at pins #5 and #6). MOD-IO2 requires external power supply and 12V of voltage.

MOD-IO2 has PIC16F1503 microcontroller loaded with open-source firmware and the source code is available for modification. It is important to notice that PIC16F1503 is a 14-pin microcontroller. There is a lot of multiplexing. For example, if you are reading analog input 0 (AN0) or analog input 1 (AN1) the values might be not correct if you have ICSP programmer attached at the same time – the pins that have AN0 and AN1, also have ICSPDAT and ICSPCLK. Furthermore, not all analog inputs are available at the GPIO connector. Only AN0, AN1, AN2, AN3, and AN7 are available on the GPIO connector. AN7 is also used for both GPIO and PWM2 in the firmware.

Newer firmware might be uploaded to your MOD-IO2 via a PIC programmer or debugger. There is a prebuilt hex in the firmware archive for easier upgrading.

Important! MOD-IO2 is compatible with the i2c-tools package and the commands i2cdetect, i2cset, i2cget, i2cdump. Visit the following web page for more info:

https://i2c.wiki.kernel.org/index.php/I2C_Tools

Before proceeding, consider the following:

* This file describes operation with the newest version of the official firmware only. The firmware was made with MPLAB X and XC8 compiler.

* MOD-IO2 works only at 100kHz (or lower) I2C clock speed.

* Host must support clock stretching for proper work.

* When reading, after sending command, STOP-START condition must be send, not RESTART.

* 12V external power supply is required.

* SCL and SDA at UEXT_FEMALE connector are pulled up.

* Make sure PROG jumper is OPEN! If you close it accidentally, you might change the I2C address of the board.

* Analog inputs work up to 3.3V.

Make sure to check the schematic for locations and possible multiplexing of the analog inputs you have chosen for your tests. When in doubt always refer to the latest schematic of the design!

2. Default I2C address and command format:

The default I2C address of MOD-IO2 is **0x21**. The default I2C can be changed by closing the PROG jumper but you need to be careful. The change is detailed below.

The typical format of a command is:

[I2C_ADDRESS_OF_MOD-IO2] [I2C_COMMAND_CODE] <I2C_COMMAND_DATA>

Refer to chapter 3 for available commands and command codes; refer to chapter 4 for examples with i2c-tools.

3. I2C commands and respective codes:

#	Command code	(Set/Get) Description	Data range
1	0x01	Set GPIO direction	0x00 - 0x7f
2	0x02	Set GPIO output level	0x00 - 0x7f
3	0x03	Get GPIO input level	0x00 - 0x7f
4	0x04	Set GPIO pull-up	0x00 - 0x1f
5	0x10	Get analog input GPIO0	0xHHLL
6	0x11	Get analog input GPIO1	0xHHLL
7	0x12	Get analog input GPIO2	0xHHLL
8	0x13	Get analog input GPIO3	0xHHLL
9	0x15	Get analog input GPIO5	0xHHLL
10	0x20	Get board ID	0x23
11	0x21	Get firmware version	0x34
12	0x40	Set relay(s) on/off	0x00 - 0x03
13	0x41	Set relay(s) on	0x01 - 0x03
14	0x42	Set relay(s) off	0x01 - 0x03
15	0x43	Get current relay state	0x00 - 0x03
16	0x50	Set PWM off	0x01 or 0x02
17	0x51	Set PWM1 and duty cycle	0x00 - 0xff
18	0x52	Set PWM2 and duty cycle	0x00 - 0xff
19	0x60	Set GPIO2 as DAC	0x00 – 0x1f (0V–3.1V)

Get (reading) with i2c-tools requires first setting the command before getting the response (e.g. two commands at least – first “i2cset” and then “i2cget”).

In order to change default I2C address you need to close PROG jumper then issue new I2C address directly.

You might use decimal values for most of the commands, e.g. instead of “0xff”, you can use “255”.

More details and examples for each command can be found further down in the document.

3.1 SET GPIO DIRECTION – SET_TRIS(0x01):

Define port direction of the GPIO pins. Can be INPUT or OUTPUT. Note that GPIO3 can only be input with pullup always turned on. This is because the alternative function (e.g. output) is #MCLR (reset). To set GPIO as INPUT write 1 to the corresponding bitmask, or 0 for output.

Command format:

```
--------
START | ADDRESS | W | ACK | SET_TRIS | ACK | VALUE | ACK | STOP
```

where:

ADDRESS = 0x21 (this is the default address)
SET_TRIS = 0x01
VALUE   = 0b0ddddddd – the bitmask with the corresponding directions (“d” is “0” or “1”); GPIO0 – bit0, GPIO1 – bit1, etc… 0x7F – all input, 0x00 – all outputs)

3.2 SET GPIO OUTPUT LEVEL – SET_LAT(0x02):

Set the output level of a GPIO. In the bitmask 1 corresponds to logical “1", and 0 – logical "0". Again, GPIO3 can be only input, so the corresponding bit in the mask will be ignored.

Command format:

```
--------
START | ADDRESS | W | ACK | SET_LAT | ACK | VALUE | ACK | STOP
```

where:

ADDRESS = 0x21 (the default address)
SET_LAT = 0x02
VALUE   = 0b0dddxddd the binary bitmask, where bit0 is the value for GPIO0, bit1 – GPIO1, etc. bit3 is x – ignored.

3.3 GET GPIO INPUT LEVEL – GET_PORT(0x03):

Read current level of the GPIOs. The data will be valid if the GPIO is configured as INPUT.

Command format:

```
--------
START | ADDRESS | W | ACK | GET_PORT | ACK | STOP | START | ADDRESS | R | DATA | NACK | STOP
```

where:

ADDRESS = 0x21 (the default address)
GET_PORT = 0x03
DATA   = 0x0ddddddd – the bitmask containing the GPIOs status. bit0 – GPIO0, bit1 – GPIO1, etc.

3.4 SET GPIO PULLUP – SET_PU(0x04):

Turn on or off weak pullup. GPIOs must be configured as INPUTS. Note that only GPIO0 to GPIO4 have pullups, and GPIO3 is with pullup always enabled.

Command format:

```
--------
START | ADDRESS | W | ACK | SET_PU | ACK | VALUE | ACK | STOP
```

where:

ADDRESS = 0x21 (the default address)
SET_PU = 0x04
VALUE   = 0b000dxddd – bit0 corresponding to GPIO0, as "1" enables the pullup and "0" - disables it.

3.5 GET ANALOG INPUT -

GET_AN0(0x10) on GPIO0;
GET_AN1(0x11) on GPIO1;
GET_AN2(0x12) on GPIO2;
GET_AN6(0x13) on GPIO3;
GET_AN7(0x15) on GPIO5.

Read the voltage applied on any of the GPIOs mentioned above. If GPIO is configured as output, the execution of the command will set the GPIO as input. After the applied voltage is measured, the GPIO configuration will be as INPUT. After READ the master should read 2 bytes of data: HIGH and LOW byte.

Command format:

```
--------
START | ADDRESS | W | ACK | GET_ANx | ACK | STOP | START | ADDRESS | R | DATA_L | ACK | DATA_H | NACK | STOP
```

where:

ADDRESS = 0x21 (the default address)
GET_ANx = 0x10, 0x11, 0x12, 0x13, or 0x15
DATA_L = the byte that contains the low 8 bits of the ADC value
DATA_H = the byte that contains the high bits of the ADC value

3.6 GET BOARD ID – GET_ID(0x20):

Read the ID of the MOD-IO2. By default it must be 0x23.

Command format:

```
--------
START | ADDRESS | W | ACK | GET_ID | ACK | STOP | START | ADDRESS | R | ACK | DATA | NACK | STOP
```

where:

```
ADDRESS = 0x21 (the default address)
GET_ID  = 0x20
DATA    = should be 0x23
```

3.7 TURN ON OR OFF RELAY – SET_REL(0x40):

Turn on or off one or both the relays. The data is the state of the relays. Bit0 is the state of RELAY1, and bit1 – RELAY2. If 1 is written the relays is turned on, 0 is off. For example to turn on both relays you must write 0x03.

Command format:

```
--------
START | ADDRESS | W | ACK | SET_REL | ACK | DATA | ACK | STOP
```

where:

```
ADDRESS = 0x21 (the default address)
SET_REL = 0x40
DATA    = 0x00, 0x01, 0x02, or 0x03
```

THE BELOW COMMANDS WERE ADDED IN FIRMWARE VERSION 3.02:

3.8 GET FIRMWARE VERSION – GET_FV(0x21):

Read the firmware version of the MOD-IO2.

Command format:

```
START | ADDRESS | W | ACK | GET_FV | ACK | STOP | START | ADDRESS | R | ACK | DATA | NACK | STOP
```

where:

ADDRESS = 0x21 (the default address)
GET_FV = 0x21
DATA   = firmware version

3.9 TURN ON RELAY – SET_RELON(0x41):

Turns on the relays. This command is different from 0x40 since it can't turn off the relays, once turned on with 0x41 the relay can be turned off with 0x42 or 0x40. The data is the mask of affected relays. Bit0 is for RELAY1, and bit1 – RELAY2. If 1 is written the relays is affected, 0 – is not.

Command format:

```
START | ADDRESS | W | ACK | SET_RELON | ACK | DATA | ACK | STOP
```

where:

ADDRESS   = 0x21 (the default address)
SET_RELON = 0x41
DATA   = 0x01, 0x02, 0x03

3.10 TURN OFF RELAY – SET_RELOFF(0x42):

Turn off the relays. The data is the mask of affected relays. Bit0 is for RELAY1, and bit1 – RELAY2. If 1 is written the relays is affected, 0 – is not.

Command format:

```
START | ADDRESS | W | ACK | SET_RELOFF | ACK | DATA | ACK | STOP
```

where:

ADDRESS   = 0x21 (the default address)
SET_RELOFF = 0x42
DATA   = 0x01, 0x02, 0x03

3.11 GET CURRENT RELAY STATE – GET_RELAY(0x43):

Read state of relays of the MOD-IO2. The read data is the state of the relays. Bit0 is the state of RELAY1, and bit1 – RELAY2. If 1 is read the relay is turned on, 0 means that it is off.

```
Command format:
--------

START | ADDRESS | W | ACK | GET_RELAY | ACK | STOP | START | ADDRESS | R | ACK |
DATA | NACK | STOP
```

where:

```
ADDRESS   = 0x21 (the default address)
GET_ID    = 0x43
READ DATA = should be 0x01, 0x02, 0x03
```

3.12 SET PWM1 – SET_PWM1(0x51):

Open PWM1 and set duty cycle. PWM1 is GPIO6.

```
Command format:
--------

START | ADDRESS | W | ACK | OPEN_PWM1 | ACK | DATA | ACK | STOP
```

where:

```
ADDRESS   = 0x21 (the default address)
OPEN_PWM1 = 0x51
DATA      = 0x00..0xFF
```

3.13 SET PWM2 – SET_PWM2(0x52):

Open PWM2 and set duty cycle. PWM2 is GPIO5.

```
Command format:
--------

START | ADDRESS | W | ACK | OPEN_PWM2 | ACK | DATA | ACK | STOP
```

where:

```
ADDRESS   = 0x21 (the default address)
OPEN_PWM2 = 0x52
DATA      = 0x00..0xFF
```

3.14 CLOSE PWMx – SET_PWM(0x50):

Close PWMx and set GPIOx as input.

```
Command format:
--------

START | ADDRESS | W | ACK | CLOSE_PWM | ACK | DATA | ACK | STOP
```

where:

```
ADDRESS    = 0x21 (the default address)
CLOSE_PWM1 = 0x50
DATA       = 0x01 or 0x02
```

THE BELOW COMMANDS WERE ADDED IN FIRMWARE VERSION 4.3:

3.15 ENABLE AND CONFIGURE DAC – SET_DAC(0x60):

Set DACOUT1 and configure GPIO2 as analog output. Valid data range 0x00 to 0x1f, step is 0.1V (0V to ~3.1V).

```
Command format:
--------

START | ADDRESS | W | ACK | SET_DAC | ACK | DATA | ACK | STOP
```

where:

```
ADDRESS = 0x21 (the default address)
SET_DAC = 0x60
DATA    = 0..0x1F
```

4.Examples:

These examples were tested on a Linux board with i2c-tools – you can install i2c-tools with:

```
apt install i2c-tools
```

Make sure to detect the i2c port where the MOD-IO2 board is attached with i2cdetect command and possible i2c port number:

```
i2cdetect 1
i2cdetect 2
i2cdetect 3
```

and so on until you find it as shown below:

```
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- 21 -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

4.1 Turning on the relays:

Remember the relays require 12V external power supply.

4.1.1 Turning all relays on:

```
i2cset –y 2 0x21 0x40 0x03
```

where

i2cset – command for sending data over i2c;
-y – to skip the y/n confirmation prompt;
2 – I2C number (the one detected with i2cdetect);
0x21 – board address (0x21 should be used for writing);
0x40 – relay operations (as seen in the README.txt);
0x03 – should be interpreted as binary 011 – turns on both relays (0x02 would turn only second relay, 0x01 only the first, 0x00 would turn both off – 0x03 again would turn them off also);
Expected result: a specific sound would occur and relays' LED would turn on.

4.1.2 Turning on only the first relay:

```
i2set -y 2 0x21 0x41 0x01
```

where

i2cset – command for sending data over i2c;
-y – to skip the y/n confirmation prompt;
2 – I2C number (usually either 1 or 2);
0x21 – board address (0x21 should be used for writing);
0x41 – relay operations (as seen in the README.txt);
0x01 – should be interpreted as binary 011 – turns on both relays (0x02 would turn only second relay, 0x03 both);

Expected result: a specific sound would occur and first relay's LED would turn on. If there is no clicking sound then the power supply of the relay is insufficient.

4.1.3 Turning off only the first relay:

```
i2set -y 2 0x21 0x42 0x01
```

where

i2cset – command for sending data;
-y – to skip the y/n confirmation prompt;
2 – I2C number (usually either 1 or 2);
0x21 – board address (0x21 should be used for writing);
0x42 – relay operations (as seen in the README.txt);
0x01 – should be interpreted as binary 011 – turns off both relays (0x02 would turn off only second relay, 0x03 - both);

Expected result: a specific sound would occur and first relay's LED would turn off.

4.1.4 Reading the state of the relays:

```
i2cset –y 2 0x21 0x43
```

and then the read command

```
i2cget –y 2 0x21
```

where

i2cset – command for sending data;
i2cget – command for reading;
-y – to skip the y/n confirmation prompt;
2 – I2C number (usually either 1 or 2);
0x21 – board address (0x21 should be used for writing);
0x43 – read relay operations;

Expected result: 0x03 – should be interpreted as binary 011, e.g. both relays are on; the same as the command for turning the relays on.

4.2. Read analog input:

```
i2cset –y 2 0x21 0x10
```

and then the read command

```
i2cget –y 2 0x21 0x10
```

where

0x10 is the first analog IO;

The big thing here is that to read you actually have to write (“that you would read”). Read is a combination of i2cset and i2cget!

Expected results: on the terminal you would receive random and changing number or 0x00 or 0x08 or 0xFF whether you have the GPIO floating or set to 0V or set to 3.3V.

4.3. To read the ID of MOD-IO2

```
i2cset –y 2 0x21 0x20
```

```
i2cget –y 2 0x21
```

where

0x20 is the code to read ID according to chapter 2.

Expected result: on the terminal you would receive 0x23.

4.4. Set all analog IOs in high level

```
i2cset –y 2 0x21 0x01 0x01
```

where

0x01 according to chapter 2, SET_TRIS is used to define port directions;
0x01 is the high level (for low level use 0x00).

4.5. Read all analog IOs

```
i2cset –y 2 0x21 0x01
i2cget –y 2 0x21
```

4.6. Set PWM1 to half duty

Notice that PWM1 is GPIO6.

```
i2cset –y 2 0x21 0x51 0x7f
```

7f is half the duty of PWM1

4.7. Set PWM2 to full duty

Notice that PWM2 is GPIO5.

```
i2cset –y 2 0x21 0x52 0xff
```

ff is the max duty of PWM2

4.8. Free PWMx

```
i2cset –y 2 0x21 0x50 0x01
```

0x01 is PWM1, so this command frees GPIO6 and sets it as input.

4.9. Set DAC

```
i2cset –y 2 0x21 0x60 0x1f
```

1f is DAC max = 3.1V

4.10 Change I2C device address. Use with CAUTION!!!

In order to change the I2C device address first close the PROG jumper.

i2cset 2 0x21 0xHH where HH is new address in hexadecimal format

If you forget the number of the address you can use the modio2tool to find the address, the command and parameter would be "modio2tool -l". You can also reset the default address (0x21) with the command and parameter "modio2tool -X".

5. Release history:

21 JUNE 2019 – Released version 4.3 of the firmware – some bugs fixed, added analog out future on GPIO2;

26 MAY 2015 – Released version 3.02 of the firmware – added new features, document revised;

13 MAY 2015 – Released version 3.01 of the firmware – fixes a bug with the internal weak pull-ups;

24 JUL 2013 - Released version 3 of the firmware;

23 OCT 2013 - Revision 2 of the README.

6. Support: https://www.olimex.com/ and support@olimex.com