RVM I2C

I2C Communication Protocol for P200-O and P201-O

Version 01.06

Advanced Microfluidics SA

BKi
15/06/2023

Document Purpose

Description for the I2C protocol used to communicate with the RVM as a slave.

Document Status and Revision History

Version	*Author*	*Release date*	*Modifications*	*Revised by*	*Approved by*
V01.00	BKi	29.03.2019	First release version (draft)	RRY	RRY
V01.01	BKi	28.06.2019	I2C registers	RRY	RRY
V01.02	ECo	23.08.2019	Removed confusion between decimal and hexadecimal numbers	RRY	RRY
V01.03	RRY	11.11.2019	Updated notes on fast mode, corrected minor errors, added information about bus design	BKi	BKi
V01.04	BKi	03.11.2020	I2C registers	RRY	RRY
V01.05	RRY	25.01.2021	P201 specific commands	BKi	BKi
V01.06	RRY	15.06.2023	Example code Arduino	MGE	MGE

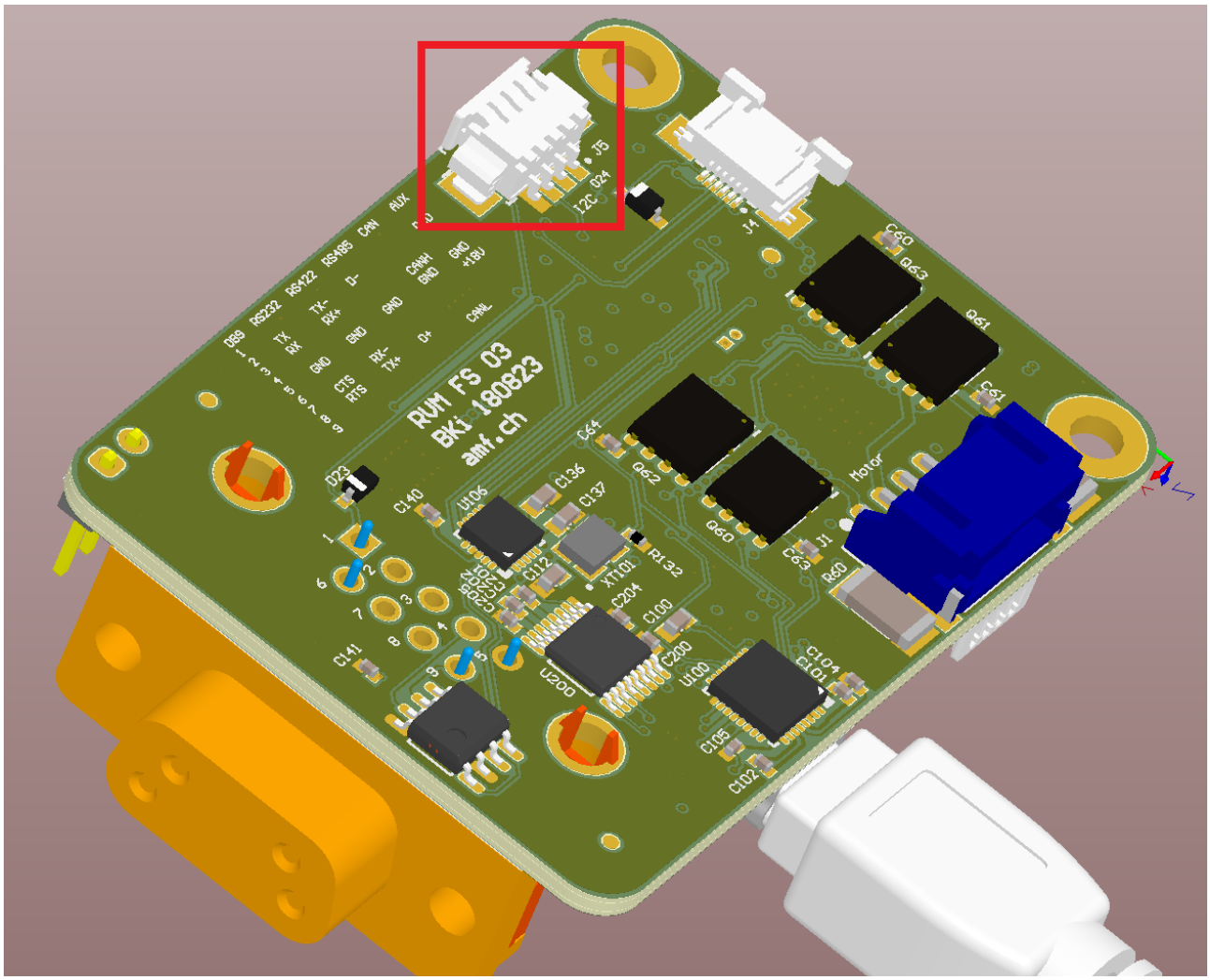
1. Contents

Document Purpose 2
Document Status and Revision History 2
2. Hardware 4
2.1. P201-O - I2C Connector J5 4
2.2. P201-O - J5 pinout 4
2.3. P200-O - I2C Connector J5 5
2.4. P200-O - J5 pinout 5
2.5. Recommended bus design 6
3. Software 7
3.1. Introduction 7
3.2. Frame format 7
3.2.1. Register write 7
3.2.2. Register read 7
3.3. Register Map 8
3.4. Register Details 9
3.4.1. 0x03 Interrupt clear 9
3.4.2. 0x04 Interrupt enable 9
3.4.3. 0x50 Valve status 9
3.4.4. 0x51 Valve command 10
3.4.5. 0x52 Valve current port 10
3.4.6. 0x55 Valve configuration 10
3.4.7. 0x56 Valve speed mode (P201-O only) 10
3.4.8. 0x60 – 0x62 Valve motion count 11
3.4.9. 0x63 Valve motion count reset 11
3.4.10. 0xB1 Secondary I2C Address 11
3.4.11. 0xB2 LED disable (P201-O only) 11
3.4.12. 0xB3 Interrupt configuration 11
3.4.13. 0xBA Reboot command register 12
3.4.14. 0xF8 uC Unique ID 12
3.4.15. 0xFF FW Version 12
4. Example code for Arduino 12

2. Hardware

2.1. P201-O - I2C Connector J5

The I2C connector J5 can be found on the bottom layer (the layer facing the motor) of the PCB.



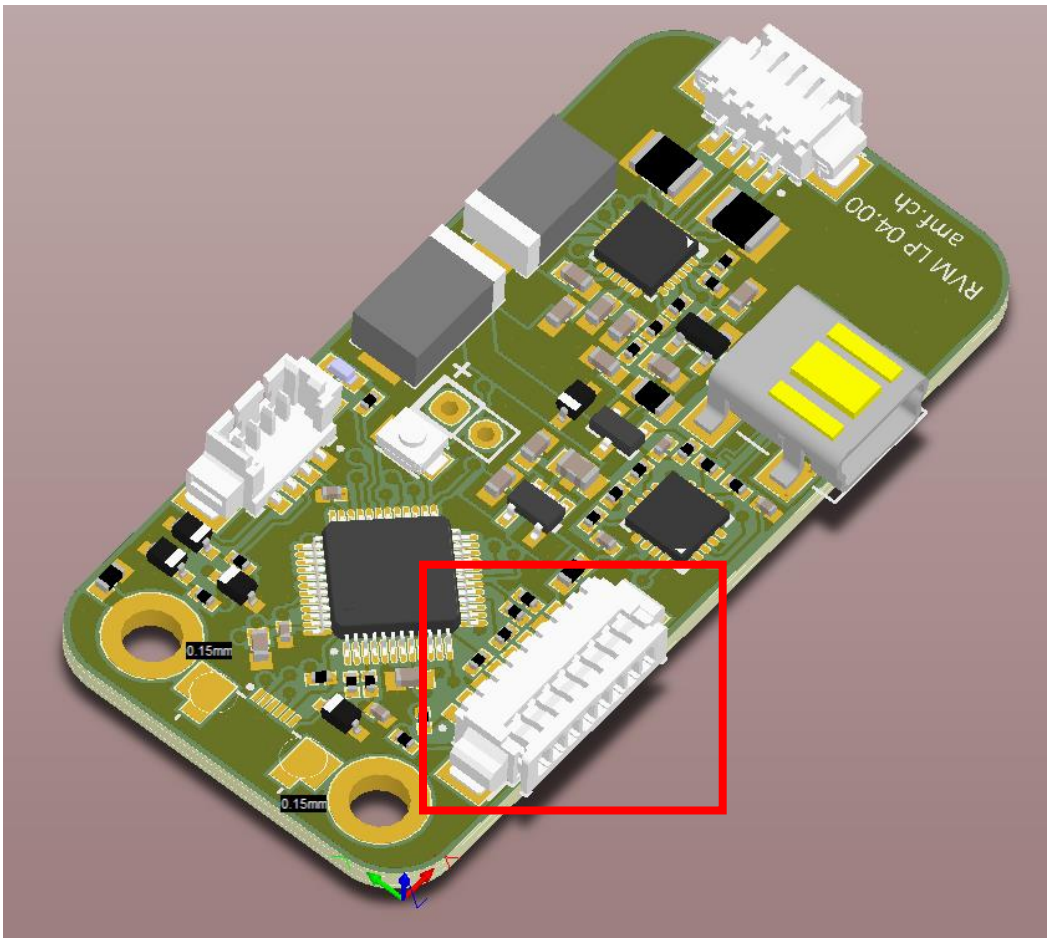
Reference for the connector: 4 pins Picoblade SMD R/A (Molex 0532610471)

2.2. P201-O - J5 pinout

J5 Pin N°	Name	Direction	Description
Pin 1 (marked with a dot)	SCL	I	Serial Clock (recommended 100kHz) 3.3V logic level Connected to VCC3V3 through a 20kΩ pullup
Pin 2	SDA	I/O	Serial Data 3.3V logic level Connected to VCC3V3 through a 20kΩ pullup
Pin 3	nATTN	O	Software driven Interrupt
Pin 4	GND		Ground

2.3. P200-O - I2C Connector J5

The I2C connector J5 can be found on the bottom layer (the layer facing the motor) of the PCB.



Reference for the connector: 8 pins Picoblade SMD R/A (Molex 0532610871)

2.4. P200-O - J5 pinout

J5 Pin N°	Name	Direction	Description
Pin 3	GND		Ground
Pin 4	SCL	I	Serial Clock (recommended 100kHz) 3.3V logic level Connected to VCC3V3 through a 10kΩ pullup
Pin 5	SDA	I/O	Serial Data 3.3V logic level Connected to VCC3V3 through a 10kΩ pullup
Pin 6	nATTN	O	Software driven Interrupt
Pin 7	GND		Ground
Pin 8	VSW		External power supply (5V, 500mA) when R50 is mounted.

Please be careful about how you power and communicate with the board of module P200-O. Here are the possibilities:

- Power via USB, communication via USB
- Power via picoblade, communication via picoblade
- Power via USB, communication via picoblade DO NOT do this if a power source is connected to the picoblade
- DO NOT power via picoblade and communicate via USB

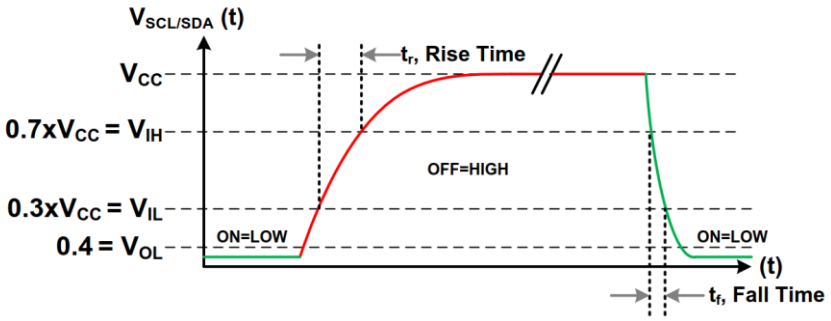
2.5. Recommended bus design

Note: the following paragraph is intended solely as general information.

The AMF modules are designed to work only with a 100 kHz clock. This speed corresponds to the "standard mode" of I2C and the following specifications apply:

Parameter		Standard Mode	Unit
$t_{r\ max}$	Rise time of both SDA and SCL signals (max)	1000	ns
$C_{b\ max}$	Capacitive load for each bus line (max)	400	pF

In practical applications, we advise the system designer to monitor with an oscilloscope the rise times of SDA and SCL lines to make sure that they remain below 1000 ns.



The rise time for the I2C bus can be written as:

$$t_r = 0.8473 * R_p * C_b$$

where R_p is the pull-up resistance

Since AMF modules are designed with an integrated pull-up resistance, we strongly advise against using another pull-up resistance on the I2C lines.

Hence the remaining parameter to modify the rise time is the capacitive load, C_b, of the bus lines. This capacitive load should remain as low as possible on both lines. Practically this is done by avoiding long wires and limiting the number of I2C slaves on the bus. If the application requires longer wires or additional slaves, we strongly advise to use an I2C buffer.

3. Software

3.1. Introduction

Communication with the controller is done using a standard I2C interface (with 3.3V logic level), where the controller (RVM) acts as a slave.

The I2C speed shall be set to **100kHz**.

The current 7-bit address of the controller is **0x64**.

Communication is done by reading/writing a set of 256 8-bit registers. The number of the first register to read/write is sent as the first byte of the I2C payload. The register number is automatically incremented, so several bytes can be read/written in one frame.

3.2. Frame format

White blocks are generated by the master, grey blocks by the controller (RVM).

3.2.1. Register write

START	Address + W	ACK	Register number	ACK	Register data	ACK	Register+1 data	ACK	…	Register+n data	ACK	STOP

3.2.2. Register read

START	Address +W	ACK	Register number	ACK	RESTART	Address +R	ACK	Register data	ACK	…	Register+n data	NAK (ACK)	STOP

The controller also supports an ACK on the last byte read instead of a NAK if it is easier for the master.

3.3. Register Map

Register number	Function	Mode
0x03	Interrupt clear	W
0x04	Interrupt enable	R/W
0x50	Valve status	R
0x51	Valve command	W
0x52	Valve current port	R
0x55	Valve configuration	R/W
0x56	Valve speed mode (P201-O only)	R/W
0x60 – 0x62	Valve motion count	R
0x63	Valve motion count reset	R/W
0xB1	Secondary I2C Address	R/W
0xB2	Led disable	W
0xBA	Reboot command register	W
0xF8	uC Unique ID	R
0xFF	FW Version	R

Writing a read only register has no effect.

Register numbers not specified are not used and should not be addressed.

For command registers, when a value is written in this register, this triggers the execution of the command in the controller. The register returns the written command number until the command starts to execute. At that time the Valve status register (0x50) changes to Busy and the Valve command register (0x51) turns back to 0. No new command must be written until the command register returns to 0 and the status changes to something different than BUSY (either Done or an error). It shall be noted that some commands can take several seconds to execute (movements for example).

Note on reading the Valve register status (0x50):

It is not recommended to poll the Valve status registers (0x50) to know if a command has ended as it may put too much load on the microcontroller (depending on the rate). The recommended way to do is to interface using the GPIO "nATTN pin" which is a software driven interrupt line that is asserted when the Valve status register has its value changed.

The best way to use it is first to enable this Interrupt functionality through the Interrupt enable register (0x04), then send the desired Valve command and finally poll the GPIO "nATTN". When it is asserted, read the Valve status register (0x50) to check if it is has executed correctly or ended with an error. Then do not forget to clear the interrupt through the Interrupt clear register (0x03).

3.4. Register Details

3.4.1. 0x03 Interrupt clear

This register is write only.

The valve interrupt is mapped to bit 2.

For example: writing 0x04 to this register will clear the valve interrupt.

3.4.2. 0x04 Interrupt enable

The valve interrupt is mapped to bit 2.

For example: writing 0x04 to this register will enable the valve interrupt on GPIO “nATTN” pin.

3.4.3. 0x50 Valve status

The unsigned 8-bit value of this register can have following values:

0x00 Done
The last command has been executed successfully.

0x80 Unknown Command
The last command is not a valid command.

0x88 I’m busy
A new command has been sent while another was still being executed.

0x89 Other system active
A command has been sent while another command was being executed by another internal module.

0x90 Not homed
A command was sent while the homing procedure has not been run before.

0xE0 Blocked
Something prevented the valve to move.

0xE1 Sensor error
Unable to read position sensor. This means probably that the cable is disconnected, or the sensor is broken.

0xE2 Missing main reference
Unable to find the valve’s main reference magnet during homing. This can mean that a reference magnet of the valve is bad/missing or that the motor is blocked during homing.

0xE3 Missing reference
Unable to find a valve’s reference magnet during homing. This can mean that a reference magnet of the valve is bad/missing or that the motor is blocked during homing.

0xE4 Bad reference polarity
One of the magnets of the reference valve has a bad polarity. This can mean that a reference magnet has been assembled in the wrong orientation in the valve.

0xFF Busy
The command is being executed. Will change either to Done or an error when the command ends.

3.4.4. 0x51 Valve command

The register goes back to 0 as soon as the command starts to execute. The register must not be written while another command is still being executed.

Following unsigned 8-bit values can be written:

0x10 Start homing procedure
This command must be run before any other command can be sent to the valve.

0x2X Move to port using shortest path
Change the active port of the valve using the shortest path where X is the port number in hexadecimal (1..9, A..C for a 12-port valve)

> Example
> 0x22 moves to port 2
> 0x2A moves to port 10

0x3X Move to port using clockwise rotation
Change the active port of the valve using clockwise rotation where X is the port number in hexadecimal

0x4X Move to port using counter-clockwise rotation
Change the active port of the valve using counter-clockwise rotation where X is the port number in hexadecimal

3.4.5. 0x52 Valve current port

Report the current valve active position.

A value of (0x00) means the valve has not been initialized, the homing procedure has not been run before.

3.4.6. 0x55 Valve configuration

Read or write the valve configuration (number of ports).

Possible values in decimal are: [4, 6, 8, 10 or 12] (default: 6) Writing an invalid configuration is ignored.
Once the register has been written, the new valve configuration is immediately active, and the homing procedure must be run. Reading it again will ensure of the actual configuration.

This setting is stored in memory and will be recalled on next power-ups.

Note on valve configuration:

Setting a value of ports not corresponding to the real number of ports should trigger the error 0xE3 Missing reference when trying to do the homing procedure.

For a 2-port on/off valve, please configure 4 ports as 2 of them are for position "on" and 2 are for position "off".

3.4.7. 0x56 Valve speed mode (P201-O only)

Read or write the valve speed mode (slow=0 or fast=1).

Possible values in decimal are: [0 or 1] (default: 0) Writing an invalid speed mode is ignored.
Once the register has been written, the new valve speed mode is immediately active. Reading it again will ensure of the actual speed mode.

This setting is stored in memory and will be recalled on next power-ups.

Note on speed mode for firmware version before 0.3.29.gba20 :

"Fast mode" can only be used in conjunction with a number of ports lower than 10. If possible, update to latest firmware to enable fast mode for all valves.

3.4.8. 0x60 – 0x62 Valve motion count

Report the current valve motion count as unsigned 24-bit (0x60: LSB, 0x62: MSB)

The three register must be read in the same I2C transaction in order to have a coherent value.

3.4.9. 0x63 Valve motion count reset

Reset the valve motion count.

The valve motion count is mapped to bit 2.

For example: writing 0x04 to this register will reset the valve motion count to 0.

3.4.10. 0xB1 Secondary I2C Address

Read or write the secondary I2C Address.

Possible values are: [8 to 119] (default: 100) Writing an invalid address is ignored.
Once the register has been written, the new address won't be active until next power-up. Reading it again before rebooting will ensure of the new address.

This setting is stored in memory and will be recalled on next power-ups.

Note on I2C addresses:

The main I2C address (0x64) is always active and should be considered as a "broadcasting address". To change the address of a RVM using the "broadcasting address", be careful that only one RVM is connected to the I2C bus, else every RVM on the bus will have their secondary I2C address changed.

3.4.11. 0xB2 LED disable (P201-O only)

Writing 0x00 enables the on-board led (default value).
Writing 0x01 to this register disables the on-board led.
This setting is stored in memory and will be recalled on next power-ups.

3.4.12. 0xB3 Interrupt configuration

Writing 0x00 configures the interrupt to be fired after EEPROM access (default value).

Writing 0x01 configures the interrupt to be fired before EEPROM access. The RVM will be busy for max 400ms after the interrupt is generated. **Communication during these 400 ms should be avoided. (P201-O only)**

This setting is stored in memory and will be recalled on next power-ups.

3.4.13. 0xBA Reboot command register

The reboot will only be triggered after 2 consecutive writes. The first write must be 0xDE and the second write must be 0x21. If the sequence is correct, the module will reboot.

3.4.14. 0xF8 uC Unique ID

The unique ID from the microcontroller (guaranteed unique by NXP), an unsigned 128-bit value, could be useful to identify each individual board.

3.4.15. 0xFF FW Version

The firmware version currently on the microcontroller, returned as a (max) 16 characters null terminated string.

4. Example code for Arduino

We present below a code example for I2C communication between an AMF module and an Arduino Board. Please note that the Arduino board should be correctly configured to ensure proper functionality. The following code is an only example and Advanced Microfluidics SA cannot be held responsible for issues arising from the complete setup. Please check the module functionality with USB communication to separate I2C issues from hardware issues.

```
#include <Wire.h>     // include Arduino Wire library

#define AMF_ADDRESS 0x64

void setup() {

  Wire.begin();
  Serial.begin(9600); // initialize serial communication

  byte valve_status;
  byte valve_pos;

  Serial.println("Valve initialization");

  Wire.beginTransmission(AMF_ADDRESS);  // start I2C communication with valve
  Wire.write(0x51);                     // valve command register
  Wire.write(0x10);                     // homing
  Wire.endTransmission(false);
  Wire.endTransmission();

  delay(5000);                          // wait until the valve is homed

  Wire.beginTransmission(AMF_ADDRESS);
  Wire.write(0x50); // valve status
  Wire.endTransmission(false);
  Wire.requestFrom(AMF_ADDRESS, 1, false);
  valve_status = Wire.read();
  Wire.endTransmission();
```

```
  if (valve_status)                    //if the value is different than 0, an error occurred
  {
    Serial.print("Valve error: ");
    Serial.println(valve_status);      // print the character
  }
  else
  {
    Serial.println("Valve homed");
    Serial.println("Valve move to port 2");

    Wire.beginTransmission(AMF_ADDRESS);
    Wire.write(0x51);                    // valve command
    Wire.write(0x22);                    // go to port 2
    Wire.endTransmission(false);
    Wire.endTransmission();

    delay(2000);

    Wire.beginTransmission(AMF_ADDRESS);
    Wire.write(0x52);                     // valve current port
    Wire.endTransmission(false);
    Wire.requestFrom(AMF_ADDRESS, 1, false);
    valve_pos = Wire.read();
    Wire.endTransmission();

    if(valve_pos == 2)
      Serial.println("Valve positioned on port 2");
    else{
      Serial.print("Valve error, port = ");
      Serial.println(valve_pos);
    }
  }
}


void loop() {

}
```