Introduction to
Aistin Virtual Machine Programming

Version 1.0
2014-05-08

iProtoXi

Table of Contents

1. Introduction

Aistin is the firmware provided with the iProtoXi Micro Controller Board. It is designed to support easy use of various add-on boards with sensors, leds, etc. Firmware uses a specific human readable protocol, the Aistin Protocol, to communicate with outer world. Aistin protocol makes it possible to access the networked sensors and actuators from the clients with a single, unified way. In current implementation, I2C types of devices are supported.

In this tutorial, a short description about the basic scripts for Aistin Virtual Machine programming is explained. This tutorial demonstrates how different sensors can be initialized for basic operation, how the sensory values can be read, and how the correct outputs can be derived from measurements. The purpose of this tutorial is to collect all the most commonly used functions in to one place so that they can easily be used in various setups or as a starting point to understand Aistin Protocol and Aistin Virtual Machine programming. The purpose of this tutorial is not to teach Assembly-like Aistin Virtual Machine programming but to give quick and easy guidelines how to get started. For more detailed and up-to-date information about Aistin Virtual Machine programming, please refer to iProtoXi Aistin Firmware Manual [1].

2. Aistin Protocol

The Aistin Protocol is a method which is used for iProtoXi Micro data transfers. The Aistin Protocol can also be used to transfer programs into Aistin Virtual Machine which resides in iProtoXi Micro firmware. The Aistin Virtual Machine can perform small programs which are written with Aistin bytecode. The Aistin Virtual Machine is a bytecode translator.

The Aistin Protocol is based on ASCII characters using any 7- or 8-bit ASCII bearer. It is designed to be readable both to humans and machines and it can be used with simple terminal programs (e.g. Windows Hyper Terminal).

The Aistin Protocol has a small memory footprint so instead of large set of different command words there are only few one-letter commands. The needed command divergence is achieved by using numerical codes called “addressess”. The protocol uses “node addressing” to address an individual iProtoXi Controller Board in a network. It's possible even though the board itself doesn't implement a TCP/IP stack.

2.1 Message Structure

The Aistin Message Structure is described in the table below. A single message may consist of all of the possible fields or only just a few, most important fields, such as the start flag and message ID (>X), location (@), data fields and the end flag (line feed or carriage return). Notice that the time field (%) may be configured to appear before or after the data. The message characters are detailed in section 2.2.

start flag	message ID	sender	receiver	location	data format	data	time	end flag
>	X	~	^	@	&	: $	%	LF/CR

2.2 Data Fields and Character Allocation

Symbols used in data messages are described in the table below.

>X	Beginning of a message with message type X, where X is: R(ead), W(rite), D(ata), S(can), A(cknowledge), V(ersion)
~senderID	Identifier string (nodeID) of the message sender, maximum length 12 characters
^receiverID	Identifier string (nodeID) of the message receiver, maximum length 12 characters
@DaRa	Target identifier (targetID): device address (Da), 8-bit register address (Ra)
@DaRaNb	Target identifier (targetID): device address (Da), 8-bit register address (Ra), number of byter (Nb)
@DaRaraNb	Target identifier (targetID): device address (Da), 16-bit register address (Rara), number of byter (Nb)
&Df	Data formatting code
:D0D1D2...	Data bytes (D1, D2, D3,...) in a hexadecimal format, no space between bytes
$string	Readable datastring, e.g. a textstring or a signed 16-bit integer
%time	Time value in decimal, most often in microseconds
LF	Line Feed (ASCII code 10), end of a message
CR	Carriage Return (ASCII code 13), end of a message
'	Comment line, ignored by Aistin system

Messages which don't start with characters > or a single-quote are handled as off-message data and are forwarded to a possible connected raw device, such as Bluetooth module.

3. Basic Messages

In this chapter some basic Aistin messages are introduced. For more detailed and up-to-date information about Aistin programming, please refer to iProtoXi Aistin Firmware Manual [1].

3.1 Scan

All the available device addresses can be scanned by reading the scan register from the device 80h. There's two ways to do this. One can either send the "read all info from the scanForDevices register from deviceAddress" message by typing the following:

```
>R@8038:FF
```

or use the alias message which does the same thing:

```
>S
```

After a scan command a sequence of hexadecimal numbers corresponding to the available device addresses is returned as a return message and it should look something similar to the following:

```
>D@803808:1D203480818D9091
```

3.2 Version

There are two easy ways to read the system information. One can either type a message for requesting version info by typing:

```
>V
```

or by simply giving an empty message by typing:

```
>
```

and hitting enter on terminal. Version information is then shown and will be something like the following:

`>D@80140C$iProtoXi_Micro`	*NodeID*
`>D@800805$0.8.0`	*Software version*
`>D@800C05$27EB142B0000`	*BoardUID*
`>D@802413$2014-01-01 00.00.00`	*Real time clock*
`>D@803806:5D80818D9092`	*Scan result: available devices*

3.3 Initializing Sequence for Virtual Machine Program

There is a specific initializing sequence that every Aistin Virtual Machine program must start with. This sequence is not needed when reading sensory values with basic Aistin commands, but if Virtual Machine is used, the following sequence must be at the beginning of the program:

```
>W@8D05:10       'command: re-program VM
>W@8E00:90040001 'VM v04 program ID
>W@8E04:91008000 'device address 91, version 00, reg. set size 0080
```

The sequence starts with a command to re-program the Virtual Machine. Then the Virtual Machine's version program ID is set. Third line sets up the program UID for Virtual Machine.

3.4 Start Virtual Machine Program

Following message will start the Virtual Machine program:

```
>W@8D05:04       '(re)start VM
```

If the program is needed to start automatically after uploading, the start command is added as a last line into the program file before uploading. However, whether the program starts again automatically after power-down, depends on a setting on the device 0x8D (Master Register Set). Next command will set the auto-start on:

```
>W@8D05:13        'set auto-start ON
```

3.5 Stop Virtual Machine Program

Following message will stop the Virtual Machine program:

```
>W@8D05:02        'stop VM
```

To turn off the auto-start, use command:

```
>W@8D05:12        'set auto-start OFF
```

3.6 Loops

Continuous reads can be done with loops. A single read message will get the sensor values only once but by setting jump commands before and after the reads it is possible to create polling reads. To start the loop, use command:

```
>W@8E08:1D        'loop begins
```

After the starting point of the loop is set, it is now possible to program whatever is needed to be processed in loops. In the end, to return to the beginning of the loop, use command:

```
>W@8E08:0DFF      'jump to loop beginning
```

3.7 Intervals

Sometimes it is necessary to have some delay between reads. A message read can be put to repeat itself by giving a time field with a value that sets the period between reads. Reading stops whenewer a new message is send or started. E.g.

```
>R@8024%1000000   'read from Da=80, Ra=24, on every 1000000 us
>R@8024%1000m     'read from Da=80, Ra=24, on every 1000 ms
>R@8024%1s        'read from Da=80, Ra=24, on every 1 s
```

will repeat reads once per second. On Virtual Machine program the similar reading interval can be set by typing:

```
>W@8E08:2CE803    'set interval 1 s
```

3.8 Virtual Machine Output Configuration

Output data can contain either plain data or data with real time clock values. Aistin Protocol for a Virtual Machine Program to configure output data in different forms:

```
>W@8D0D:00        'output plain data (no real time clock)
>W@8D0D:01        'output real time clock on VM start and stop
```

4. Operating Sensors with Virtual Machine

Sensory information can be read straight from the device registers as hexadecimal

values using the Aistin Protocol. Aistin Virtual Machine can be used to read sensor data e.g. in signed integer string format also. Basic example Aistin Protocol commands for programming the Aistin Virtual Machine to operate different iProtoXi sensors is described in the following sections. For more detailed and up-to-date explanation about Aistin Virtual Machine Program Instruction Set, please refer to iProtoXi Aistin Firmware Manual [1]. More information about the different registers of iProtoXi sensors can be found from sensor datasheets [2-6].

4.1 Accelerometer (0x18)

iProtoXi Motion Sensor Board consist of an accelerometer, a magnetometer and a gyroscope. Accelerometer data can be read straight from the sensor register using the Aistin Protocol commands as follows:

```
'initialize device
>W@1820:37              'write to Da=18, Ra=20, D0=37

'read values (x,y,z)
>R@18A806               'read from Da=18, Ra=A8, Nb=06
```

Another way to get sensory data is to use the Aistin Protocol to make an Aistin Virtual Machine Program to handle the readings. In such a program, the accelerometer device information must first be set to the Virtual Machines memory and then the VM has to be set to read and print the data. For example:

```
'initialize VM
>W@8E08:1E18            'set device address 18
>W@8E08:032037          'init & power on device

'set VM to read and print values
>W@8E08:0204A806        'read 6 bytes
>W@8E08:0B04060B        'print as data message
```

More information about the sensor registers can be found from the sensor datasheet [2].

4.2 Magnetometer (0x1E)

iProtoXi Motion Sensor Board consist of an accelerometer, a magnetometer and a gyroscope. Magnetometer data can be read straight from the sensor register using the Aistin Protocol as follows:

```
'init
>W@1E00:90              'write to Da=1E, Ra=00, D0=90
>W@1E02:90              'write to Da=1E, Ra=02, D0=00

'read
>R@1E8306               'read from Da=1E, Ra=83, Nb=06
```

Aistin Protocol for a Virtual Machine Program handling the magnetometer readings:

```
'init
>W@8E08:1E1E            'set device address 1E
>W@8E08:030090          'init & power on device
```

```
>W@8E08:030200          'init & power on device

'read
>W@8E08:02108306        'read 6 bytes
>W@8E08:0B10060B        'print as data message
```

More information about the sensor registers can be found from the sensor datasheet [2].

4.3 Gyroscope (0x6B)

iProtoXi Motion Sensor Board consist of an accelerometer, a magnetometer and a gyroscope. 3-dimensional gyroscope data can be read straight from the device register using the Aistin Protocol as follows:

```
'initialize
>W@6B20:CF              'write to Da=6B, Ra=20, D0=CF
>W@6B21:04              'write to Da=6B, Ra=21, D0=04
>W@6B23:30              'write to Da=6B, Ra=23, D0=30
>W@6B24:02              'write to Da=6B, Ra=24, D0=02

'read
>R@6BA806               'read from Da=6B, Ra=A8, Nb=06
```

Aistin Protocol for a Virtual Machine Program handling the gyroscope readings:

```
'initialize
>W@8E08:1E6B            'set device address 6B
>W@8E08:0320CF          'init & power on
>W@8E08:032104          'init & power on
>W@8E08:0323B0          'init & power on
>W@8E08:032402          'init & power on

'read
>W@8E08:020AA806        'read 6 bytes
>W@8E08:0B0A060B        'print as data message
>W@8D0D:00              'output plain data (no real time clock)
```

More information about the sensor registers can be found from the sensor datasheet [3].

4.4 Proximity Sensor (0x38)

iProtoXi Proximity Sensor and Ambient Light Sensor are placed on a single board. Proximity information data can be read straight from the device register using the Aistin Protocol as follows:

```
'init
>W@3881:02              'write to Da=38, Ra=81, D0=02 (PS forced)
>W@3882:D2              'write to Da=38, Ra=82, D0=D2 (init leds)

'proximity read
>W@3884:03              'write to Da=38, Ra=84, D0=03 (measure)
>R@388F01               'read from Da=38, Ra=8F, Nb=01 (led-1)
```

Aistin Protocol for a Virtual Machine Program handling the PS measurement:

```
'init
>W@8E08:1E38            'set device address 38
>W@8E08:038102          'init & power on PS
>W@8E08:0382D2          'init & power on leds

'proximity read
>W@8E08:038403          'start proximity measurement
>W@8E08:01168F          'read single byte from current device
>W@8E08:0B160100        'print as data message
```

Proximity sensor output data (PS) can be converted to irradiance (micro watts per square centimeter) with the following function:

$$I(\mu W/cm^2) = 10^{(PS \cdot 0,0197)}$$

More information about the sensor registers can be found from the sensor datasheet [4].

4.5 Ambient Light Sensor (0x38)

iProtoXi Proximity Sensor and Ambient Light Sensor are found on one board. Ambient light information data can be read straight from the device register using the Aistin Protocol as follows:

```
'init
>W@3880:02              'write to Da=38, Ra=80, D0=02 (ALS forced)

'als read
>W@3884:03              'write to Da=38, Ra=84, D0=03 (measure)
>R@384C01               'read from Da=38, Ra=4C, Nb=01 (ALS-lo)
>R@384D01               'read from Da=38, Ra=4D, Nb=01 (ALS-hi)
```

Aistin Protocol for a Virtual Machine Program handling the ALS readings:

```
'init
>W@8E08:1E38            'set device address 38
>W@8E08:038002          'init device

'als read
>W@8E08:038403          'start measurement
>W@8E08:011A4C          'read ALS-lo
>W@8E08:011B4D          'read ALS-hi
>W@8E08:0B1A020A        'print as 16 bit unsigned decimal data
```

More information about the sensor registers can be found from the sensor datasheet [4].

4.6 Barometer (0x5C)

iProtoXi Air Pressure Sensor and Temperature Sensor are found on one board. Air pressure information data can be read straight from the device register using the Aistin Protocol as follows:

```
'init
```

```
>W@5C20:E0             'write to Da=5C, Ra=20, D0=E0

'read
>R@5C2801             'read from Da=5C, Ra=28, Nb=01 (pressure XL)
>R@5C2901             'read from Da=5C, Ra=29, Nb=01 (pressure L)
>R@5C2A01             'read from Da=5C, Ra=2A, Nb=01 (pressure H)
```

Aistin Protocol for a Virtual Machine Program handling the barometer measurement:

```
'init
>W@8E08:1E5C          'set device address 5C
>W@8E08:0320E0        'init & power on device

'read
>W@8E08:011E28        'read pressure XL
>W@8E08:011F29        'read pressure L
>W@8E08:01202A        'read pressure H
>W@8E08:0521F4        'set exponent
>W@8E08:0B1E0412      'print as data message
```

More information about the sensor registers can be found from the sensor datasheet [5].

4.7 Temperature Sensor (0x5C)

iProtoXi Air Pressure Sensor and Temperature Sensor are found on one board. Temperature information data can be read straight from the device register using the Aistin Protocol as follows:

```
'init
>W@5C20:E0             'write to Da=5C, Ra=20, D0=E0

'read (2 complement)
>R@5CAB02              'read from Da=5C, Ra=AB, Nb=02 (temp L,H)
```

Aistin Protocol for a Virtual Machine Program handling the temperature measurement:

```
'init
>W@8E08:1E5C          'set device address 5c
>W@8E08:0320E0        'init & power on device

'read
>W@8E08:052280        'copy byte to rd
>W@8E08:2123AB        'read word
>W@8E08:B8230E00      'calculate temp
>W@8E08:A7235005      'calculate temp
>W@8E08:0525F3        'set exponent
>W@8E08:0B220412      'print as decimal
```

Temperature sensor output data (TS) can be converted to temperature as Celcius degrees with the following function:

$$T = 42{,}5 + TS/480$$

More information about the sensor registers can be found from the sensor datasheet [5].

4.8 Programmable RGB LED Controller (0x34)

iProtoXi Programmable RGB LED Controller can drive 9 leds. Each LED can be controlled directly and independentyly, or LED drivers can be grouped together for preprogrammed flashing patterns. The LED Controller has three independent program execution engines, so it's possible to form three idependently programmable LED banks. Each bank can contain 1 to 9 LED driver outputs. Instructions are stored in a separate program memory. The total amount of the program memory is 96 instructions, and the user can allocate the memory as required by the engines. LED luminance is controlled with a pulse width modulation scheme with a resolution of 12 bits.

Aistin Protocol for a Virtual Machine Program to turn on/off LEDs and to simply adjust luminance:

```
'initializations
>w@8e08:1e34            'set device address 0x34
>w@8e08:030040          'init direct..
>w@8e08:03365b          '..PWM

'set red leds off
>w@8e08:031c00          'LED0-R (D7) write 00 to reg 1c
>w@8e08:031d00          'LED1-R (D8) write 00 to reg 1d
>w@8e08:031e00          'LED2-R (D9) write 00 to reg 1e

'set green leds off
>w@8e08:031600          'LED0-G (D1) write 00 to reg 16
>w@8e08:031800          'LED1-G (D3) write 00 to reg 18
>w@8e08:031a00          'LED2-G (D5) write 00 to reg 1a

'set blue leds off
>w@8e08:031700          'LED0-B (D2) write 00 to reg 17
>w@8e08:031900          'LED1-B (D4) write 00 to reg 19
>w@8e08:031b00          'LED2-B (D6) write 00 to reg 1b

'set all leds on, dimmest luminocity
>w@8e08:031c01          'LED0-R (D7) write 01 to reg 1c
>w@8e08:031601          'LED0-G (D1) write 01 to reg 16
>w@8e08:031701          'LED0-B (D2) write 01 to reg 17
>w@8e08:031d01          'LED1-R (D8) write 01 to reg 1d
>w@8e08:031801          'LED1-G (D3) write 01 to reg 18
>w@8e08:031901          'LED1-B (D4) write 01 to reg 19
>w@8e08:031e01          'LED2-R (D9) write 01 to reg 1e
>w@8e08:031a01          'LED2-G (D5) write 01 to reg 1a
>w@8e08:031b01          'LED2-B (D6) write 01 to reg 1b

'set all leds on, brightest luminocity
>w@8e08:031cff          'LED0-R (D7) write ff to reg 1c
>w@8e08:0316ff          'LED0-G (D1) write ff to reg 16
>w@8e08:0317ff          'LED0-B (D2) write ff to reg 17
>w@8e08:031dff          'LED1-R (D8) write ff to reg 1d
>w@8e08:0318ff          'LED1-G (D3) write ff to reg 18
>w@8e08:0319ff          'LED1-B (D4) write ff to reg 19
>w@8e08:031eff          'LED2-R (D9) write ff to reg 1e
>w@8e08:031aff          'LED2-G (D5) write ff to reg 1a
>w@8e08:031bff          'LED2-B (D6) write ff to reg 1b
```

More information about the sensor registers can be found from the sensor datasheet [6].

5. Examples

5.1 Temperature LEDs

```
'TEMPLEDS - Aistin program for temperature and leds
'(c) iProtoXi Oy
'Created 2013-11-11 by Jni

>w@8d05:10              'command: re-program VM
>w@8e00:90040001        'VM-ID
>w@8e04:91008000        'App-ID: devAddr-regVer-regSize

'initializations
>w@8e08:1e5c            'select barometer
>w@8e08:0320f0          'init

'init leds
>w@8e08:1e34            'select LEDs device
>w@8e08:030040          'init direct..
>w@8e08:03365b          '..PWM
>w@8e08:23063030        'set logarithmic output
>w@8e08:23083030        'set logarithmic output
>w@8e08:230a3030        'set logarithmic output
>w@8e08:230c3030        'set logarithmic output
>w@8e08:030e30          'set logarithmic output
>w@8e08:031c00          'write LEDx-R 0
>w@8e08:031600          'write LEDx-G 0
>w@8e08:031d00          'write LEDx-R 0
>w@8e08:031900          'write LEDx-B 0
>w@8e08:031a00          'write LEDx-G 0
>w@8e08:031b00          'write LEDx-B 0
>w@8e08:1d              'loop tag
>w@8e08:1e5c            'select barometer
>w@8e08:2c8000          'sync
>w@8e08:2104ab          'read temp
'>w@8e08:0b040300       'send value as a message
>w@8e08:1e34            'select LEDs device
>w@8e08:0e05F1          'compare
>w@8e08:6f0b            'skip if >=
>w@8e08:031780          'LED0-B
>w@8e08:031800          'LED1-G
>w@8e08:031e00          'LED2-R
>w@8e08:0dff            'jump to loop beginning
>w@8e08:0e05F2          'compare
>w@8e08:6f0b            'skip if >=
>w@8e08:031740          'LED0-B
>w@8e08:031840          'LED1-G
>w@8e08:031e00          'LED2-R
>w@8e08:0dff            'jump to loop beginning
>w@8e08:0e05F3          'compare
>w@8e08:6f0b            'skip if >=
```

```
>w@8e08:031700          'LED0-B
>w@8e08:031880          'LED1-G
>w@8e08:031e00          'LED2-R
>w@8e08:0dff            'jump to loop beginning
>w@8e08:0e05F4          'compare
>w@8e08:6f0b            'skip if >=
>w@8e08:031700          'LED0-B
>w@8e08:031840          'LED1-G
>w@8e08:031e40          'LED2-R
>w@8e08:0dff            'jump to loop beginning
>w@8e08:031700          'LED0-B
>w@8e08:031800          'LED1-G
>w@8e08:031e80          'LED2-R
>w@8e08:0dff            'jump to loop beginning
>w@8d05:04              'start VM
>w@8d2a:00              'clear ownerNode (to broadcast)
```

5.2 Blue Blink

```
'blue blink - Aistin VM Program
'(c) iProtoXi Oy
'Created 2014-04-16 Hpi
'Tested to work with aistin-firmware 0.9.8+

>w@8d05:10              'command: re-program VM
>w@8e00:90040001        'VM type ID V03 with 0x0100 bytes of room
>w@8e04:a4000600        'Application UID: devAddr-regVer-regSize

'Code begins
>w@8e08:1e34            'select LEDs device
>w@8e08:030040          'init direct..
>w@8e08:03365b          '..PWM

'all leds of first
>w@8e08:031c00          'LED0-R
>w@8e08:031600          'LED0-G
>w@8e08:031700          'LED0-B
>w@8e08:031d00          'LED1-R
>w@8e08:031800          'LED1-G
>w@8e08:031900          'LED1-B
>w@8e08:031e00          'LED2-R
>w@8e08:031a00          'LED2-G
>w@8e08:031b00          'LED2-B
>w@8e08:1d              'loop begins

'blue leds on
>w@8e08:031701          'LED0-B
>w@8e08:031901          'LED1-B
>w@8e08:031b01          'LED2-B
>w@8e08:2c6400          'sync at 0x64 (100) ms

'blue leds off
>w@8e08:031700          'LED0-B
>w@8e08:031900          'LED1-B
>w@8e08:031b00          'LED2-B
>w@8e08:2c6400          'sync at 0x64 (100) ms
```

```
>w@8e08:0dff            'loop to beginning
>w@8D05:13              'set auto-start ON
>w@8d05:04              'start VM
```

5.3 Bars in Decimal

```
'Aistin virtual machine program for barometer only
'(c) iProtoXi Oy
'Works on Aistin-firmware 0.8.5+
'(To read the provided float value in decimal:
'>r@9c0404&12)
'Changed: just start VM to get the value: >w@8d05:04
'Created 2013-09-26 by JNi

>w@8d05:10              'command: re-program VM
>w@8e00:90040001        'VM type ID
>w@8e04:9c010c00        'App ID: devAddr-regVer-regSize

'initializations
>w@8e08:1e5c            'select barometer
>w@8e08:0320e0          'init
>w@8e08:1d              'loop tag
>w@8e08:2c6400          'sync in ms
>w@8e08:010828          'read bar LX-byte
>w@8e08:010929          'read bar LO-byte
>w@8e08:010a2A          'read bar HI-byte
>w@8e08:050bf4          'set exp=-12

'Note: need to have a copy, otherwise it is
'possible to get intermediate result!

>w@8e08:06040804        'copy bars
>w@8e08:0b040412        'send values
'>w@8e08:0dff           'jump to loop beginning
>w@8d05:04              'start VM
```

6. References

[1] iProtoXi Aistin Firmware Manual. IprotoXi. URL: http://iprotoxi.fi/images/Downloads/iProtoXi-Aistin-Firmware-098.pdf

[2] 3-axis Accelerometer and 3-axis Magnetometer LSM303DLM Datasheet. STMicroelectronics. URL: http://www.st.com/st-web-ui/static/active/en/resource/technical/document/datasheet/DM00026454.pdf

[3] MEMS Motion Mensor: Three-axis Digital Output Gyroscope L3GD20 Datasheet. STMicroelectronics. URL: http://www.st.com/st-web-ui/static/active/en/resource/technical/document/datasheet/DM00036465.pdf

[4] 1chip Optical Proximity + Ambient Light Sensor IC BH1772GLC Datasheet. Rohm Semiconductor. URL: http://rohmfs.rohm.com/en/products/databook/datasheet/ic/sensor/proximity/bh1772glc_2-e.pdf

[5] MEMS Pressure Sensor LPS331AP Datasheet. STMicroelectronics. URL: http://www.st.com/st-web-ui/static/active/en/resource/technical/document/datasheet/DM00036196.pdf

[6] Programmable RGB LED LP5523 Datasheet. Texas Instruments. URL: http://www.ti.com/lit/ds/symlink/lp5523.pdf

Appendix A. Master Register Set (0x80)

Master Register Set is used to control basic features of the iProtoXi Aistin firmware. Detailed descriptions are listed in the table below.

Register name	Register description	Access	Size	Reset value	Register address
deviceAddress	Identifies this device, “iProtoXi Master”	R	byte	*0x80*	0x00h
regsVersion	Identifies this register set version	R	byte	*0x81*	0x01h
regsSize	Total size of this register set	R	word	*0x0046*	0x02h
deviceState	State of this device: *0x00 = stopped* *0x01 = running*	R	byte		0x04h
deviceControl	Write to control this device: *0x01 = reset to defaults* *0x02 = stop* *0x03 = run*	RW	byte		0x05h
debugLevel	Debug messages: *0x00 = no messages* *0x01 = show all*	RW	byte		0x06h
(reserved)	Reserved for future use	-	-		0x07h
software-Version	Version of iProtoXi Aistin firmware in hexadecimal e.g. 0080h equals 0.8.0	R	word		0x08h
arduino-Version	Version of the used Arduino build environment in hexadecimal e.g. 0104h equals 1.0.4	R	word		0x0Ah
hardware-Version	Version of iProtoXi Micro board we are running on in hexadecimal e.g. 0120h equals 1.2.0	R	byte		0x0Ch
boardUID	Six-byte unique board identifier	RW	6 bytes		0x0Eh
nodeID	Null-terminated board (“node”) identifier string	RW	13 bytes		0x14h
(reserved)	Reserved for future use	-	-		0x21h
i2cBase-Address	Offset for I2C addresses (not implemented)	RW	byte	*0x00*	0x22h

Register name	Register description	Access	Size	Reset value	Register address
virtualI2c-BaseAddress	Offset for virtual I2C addresses (not impl.)	RW	byte	*0x80*	0x23h
datetime	Real time clock, value is in seconds when read as hexadecimal. For text-format date and time, read zero bytes instead. For setting, send a string as $YYYY-MM-DD HH.MM.SS	RW	long		0x24h
(reserved)	Reserved for future use	-	-		0x28h
usCounter	Free running microseconds counter	RW	long		0x2Ah
message-BeginChar	Message begin character: > (not implemented)	RW	byte		0x2Eh
positiveAck	Protocol control: *0x00 = don't send positive acknowledgement* *0x01 = always send acknowledgement*	RW	byte		0x2Fh
message-Fields	Protocol control: *0x00 = include nothing* *0x01 = include location (@)* *0x02 = include senderID (~)* *0x03 = include both*	RW	byte		0x30h
include-Timestamp	Protocol control: *0x00 = no timestamp* *0x01 = include before data* *0x02 = include after data*	RW	byte		0x31h
dataEncoding	Protocol control: encode data as hex, no spaces (other values not currently supported)	RW	byte	*0x00*	0x32h
comm-Channels	Serial channels used for messaging: *0x00 = no messaging* *0x02 = ch 0 (USB)* *0x04 = ch 1 (e.g. Bluetooth)* *0x06 = both serial channels*	RW	byte		0x33h

Register name	Register description	Access	Size	Reset value	Register address
output-Channels	Where to output data messages, see “comm-Channels” for accepted values.	RW	byte		0x34h
forward-Channels	Where to forward messages, see “comm-Channels” for accepted values. This can be used to monitor message traffic from another channel.	RW	byte		0x35h
nonipx-Channels	Where to forward off-message characters, see “commChannels” for accepted values. This can be used to communicate with e.g. a Bluetooth module.	RW	byte		0x36h
debug-Channels	Where to send debug printing – currently not supported.	RW	byte		0x37h
scan-ForDevices	Read to get list of available device addresses, both real I2C devices and software devices will be listed.	Rff	byte		0x38h
(reserved)	Reserved for future use	-	-		0x39h-0x3Eh
power-SaveMode	Operating mode: *0x00 = no power save* *0x01 = basic power saving* *0x02 = advanced power saving (preliminary)*	RW	byte		0x42h
(reserved)	Reserved for future use	-	-		0x43h
latest-ErrorCode	Latest error code	R	byte		0x44h
detected-Connections	Detected connections: *0x00 = no connections* *0x02 = USB connection*	R	byte		0x45h

Appendix B. Virtual Machine Register Set (0x8D)

Aistin Virtual Machine is a special device that can be used to run user's own programs. This register set is used to control the Virtual Processor.

Register Name	Register Description	Access	Size	Reset value	Register Address
deviceAddress	Identifies this device, “Virtual Machine”	R	byte	*0x8D*	0x00h
regsVersion	Identifies register set version	R	byte	*0x82*	0x01h
regsSize	Total size of this register set	R	word	*0x0031*	0x02h
deviceState	State of the VM: *0x00 = stopped* *0x01 = running*	R	byte		0x04h
deviceControl	Write to control VM: *0x01 = reset* *0x02 = stop* *0x03 = continue* *0x04 = (re)start program* *0x10 = start programming* *0x12 = set autostart off* *0x13 = set autostart on*	RW	byte		0x05h
debugLevel	State of debug message: *0x00 = no messages* *0x01 = show messages*	RW	byte		0x06h
vmControl	VM program autostart: *0x00 = do not autostart* *0x01 = auto-start after reset*	R	byte		0x07h
program-MemSize	Size of available program memory for a VM program	R	word		0x08h
ramSize	Size of available “RAM” memory for a VM program	R	word		0x0Ah
output-Channels	Channels where VM sends messages, see “Master Register Set” for more info	RW	byte		0x0Ch
output-Options	*0x00 = output plain data* *0x01 = output real time clock on VM start and stop*	RW	byte		0x0Dh

Register Name	Register Description	Access	Size	Reset value	Register Address
condition	VM condition code register	R	byte		0x0Eh
(reserved)	Reserved for future use	-	-		0x0Fh
pc	Program Counter, address of the instruction in progress	R	word		0x10h
firstTag	Address of first TAG in the program	R	word		0x12h
prevTag	Address of previous TAG in the program	R	word		0x14h
vmStart-Datetime	Real time when VM started to run the program	R	long		0x16h
vmStartTime	Time in microseconds when VM started to run program	R	long		0x1Ah
(reserved)	Reserved for internal or future use	-	-		0x1Eh - 0x29h
owner-NodeID	ID of Node that currently controls VM (null-terminating string). This is set automatically, when writing to deviceControl byte.	RW	13 bytes		0x2Ah-0x37h

Appendix C. Virtual Machine Program Register Set (0x8E, 0x90)

This program register set is used to store Virtual Machine's application program. Programming is controlled using Virtual Machine Register Set (see device 0x8D). Address 0x08h can be used for each instruction line to send a program to Virtual Machine. The system automatically keeps track where the next program instruction must be located.

Register Name	Register Description	Access	Size	Reset Value	Register Address
deviceAddress	Identifies this device as Virtual Machine Program. *0x8E = when programming* *0x90 = when programmed*	R	byte		0x00h
regsVersion	Identifies this register set version. Is alo used to check program's compatibility with VM.	R	byte	*0x04*	0x01h
regsSize	Total size of this register set. This is the maximum size of VM application, including this header.	R	word	*0x0100*	0x02h
program-UID[0]	The device number that the VM application is implementing; will be copied to VM's RAM. *0xA0 - 0xEF*	RW	byte		0x04h
program-UID[1]	The register set version that the VM application is implementing; will be copied to VM's RAM.	RW	byte		0x05h
program-UID[2-3]	The register set size that the VM application is implementing; will be copied to VM's RAM.	RW	word		0x06h
program	The actual VM application's program code.	RW			0x08h-0xFFh

Appendix D. Virtual Machine RAM Register Set (0x8F, 0x91-0xEF)

Virtual Machine RAM Register Set is used to store your own application's data. It's possible to store static data, such as contants and strings, into this area and they will resist over power-down, since data is stored to EEPROM an re-loaded on boot-up. Note, that the first four bytes shaould always be laid out as described below for they will be directly copied from the VM Program Register Set from the bytes 4-7 of the programUID register.

Register Name	Register Description	Access	Size	Reset Value	Register Address
deviceAddress	Identifies your device that the VM application is implementing. At the beginning of programming address is set to *0x8F.* When program is started, default value *0x91* (“anonymous application”) is used if the program doesn't specify any other value. *0x8F = unprogrammed* *0x91 = anon. application* *0x92-0x9F = iProtoXi app.* *0xA0-0xEF = user app.*	R	byte		0x00h
regsVersion	*0x00 - 0xFF* Identifies your register set version.	R	byte		0x01h
regsSize	*0x00 - 0xFF* Total size of your register set; the bytes provided as an interface to be read and written by the Aistin Protocol.	R	word		0x02h
ram	RAM and static data area for free use by self-made VM applications.	R	byte		0x04h-0x7Fh

Version History

Ver.	Created	Author	Notes
1.0	2014-05-08	Heli Pihlajamaa	Chapters 1-6, Appendices A-D