Instrument of Things

WIZnet Connect the Magic 2014 Design Challenge

Project Registration Number

WZ1295

Abstract

The Instrument of Things (IoT) shows how to extend your custom electrical instruments with industry standard capabilities for remote control via a TCP/IP interface. The WIZnet WIZ550io module is used to enable a basic web server, a port mapping service and a server for remote control of the instrument using the VXI-11 communications protocol by SCPI messages. Although development has not been completed yet, feasibility is shown using an instrument for accurate measurements of a water color test kit. The ultimate goal of the Instrument of Things is to easily add the VXI-11 communications protocol and LAN eXtensions for Instruments (LXI) technology to any electrical instrument project.

Contents

Introduction

Modern electronics lab instruments feature an Ethernet port for remote control and reading measurement data by a PC or other networked instrument. These instruments can be recognized by the LXI (LAN eXtensions for Instrumentation) logo, indicating compliance with standards defined by the LXI Consortium. One of the core features of an LXI-compliant instrument is a built-in web interface for remote configuration and possibly remote control of the instrument.

Another standard is the TCP/IP instrument protocol specification VXI-11, which is defined by the VXIbus Consortium. Since LXI-compliant instruments are based on Ethernet, the TCP/IP interface allows communication with the instrument using the VXI-11 interface. The VXI-11 protocol enables the transmission of ASCII commands known as Standard Commands for Programmable Instrumentation (SCPI). A common SCPI is the short string "*IDN?", which can be send to an instrument and should be replied to with a unique identifier (actually manufacturer name, model number, serial number and firmware version) of the instrument. Other SCPI commands allow the change of measurement settings, the triggering of measurements and the reading of measurement results.

The ultimate goal of the Instrument of Things (IoT), as submitted to the WIZnet Connect the Magic 2014 Design Challenge, is to deliver industry standard capabilities for remote control via a TCP/IP interface to any home-built electrical instrument. With the recent explosion in popularity of small and low cost microprocessor development boards, such as Arduino, Raspberry Pi, Beaglebone and others, every hobbyist can quickly implement a custom instrument for whatever type of measurement is needed to make the next step in a project. What was missing, up to now, was a LXI/VXI-11 package to easily integrate in the instrument and shorten the distance to the professional instruments.

The WIZnet WIZ550io module makes it very easy to add Ethernet capabilities to custom electronics instruments, especially when it is still under development on a breadboard. The module takes care of four layers of the OSI model for networking (see table 1), is preprogrammed with a unique MAC address and features automatic configuration of network settings. The remaining three layers of the OSI model have been implemented for this project in software running on the Atmel SAM3X8E ARM Cortex-M3 CPU of the Arduino Due microcontroller board.

Table 1, the layers of the OSI model for networking and the (partial) implementations in this project.

	OSI model layer	**implementation**
7	application layer	HTTP, VXI-11, port map
6	presentation layer	XDR
5	session layer	RPC
4	transport layer	TCP, UDP
3	network layer	IPv4
2	data link layer	MAC
1	physical layer	IEEE 802.3

The three networking-related applications are a basic web server (HTTP), a port map service and a server for remote control of the instrument using the VXI-11 communications protocol. The web server can be reached at the default port 80 on the instrument. The port map service operates at standard port 111 and processes RPC port mapper procedures using TCP or UDP. The VXI-11 server actually consists of up to three services: a VXI-11 Core listener, a VXI-11 Asynchronous listener and a VXI-11 Interrupt listener. The VXI-11 services can be set to listen at any available TCP port number, as long as the service is registered with the port map service for directing clients to the correct port. See table 2 for the unique program numbers of the VXI-11 listeners, which are registered with the port map service when available for clients.

Table 2, RPC program numbers of the VXI-11 listeners.

program number	
0x0607AF	DEVICE_CORE
0x0607B0	DEVICE_ASYNC
0x0607B1	DEVICE_INTR

The implementation of the XDR layer (eXtended Data Representation) consists of a library of functions, which are used to enable conversion to and from a standard data format for transmission. Incoming data on the physical interface of the WIZnet WIZ550io is first processed up to the transport layer of the OSI model. When the data is actually intended for one of the services running on the Arduino Due, the module will indicate that data is available in its buffers. This data is unpacked using the XDR library and processed further by the service running on the Arduino Due. The process of sending data from the instrument to the client follows the reverse route through the layers of the OSI model.

The parts of the instrument not related to networking are a 20x4 LCD character display, a micro SD card reader and the RGB color sensor. The ease of use of the Arduino platform and availability of an extensive set of add-on modules, makes it trivial to add other sensors or components for local operation by the user.

Figure 1 shows a block diagram of the Instrument of Things as presented in this document. The microSD card and the Wiznet WIZ550io ethernet module are connected to the SPI interface bus of the Arduino Due. Separate slave select signals (CS) are available for the modules on the SPI bus. One digital I/O pin of the Arduino Due is used to reset the WIZ550io module. The interrupt and ready status signals of the WIZ550io module are connected to two other digital I/O pins of the Arduino Due.

The 20x4 character LCD display and the RGB sensor are connected to separate I2C lines of the Arduino Due, although they could also share a single I2C line. The LCD module has been configured to use address 0x20 on the I2C bus, while the RGB sensor uses address 0x29. The USB ports on the Arduino Due are used for programming updated versions of the (compiled) source code and for debugging the software via a RS232 serial port terminal. The last connection of the Instrument of Things is the LAN ethernet connection to the Wiznet WIZ550io module.

Let's get to the point of the specific Instrument of Things presented here: a water color test kit meter. Water color test kits are chemical test sets, which can be used to measure the quality of water in e.g. an aquarium, swimming pool or rain water reservoir. Each set is specifically designed to measure a specific property or content, like acidity (pH-level) or concentration of nitrides. The result of a water color test is

typically a specific color of the water sample, which can be compared to a color chart to obtain an approximate value of the measured parameter. Unfortunately, no electronics are involved here at all.

The color chart that comes with a water color test kit typically has a scale of only 3 to 5 different colors. Interpolation between to colors can be challenging and will probably not result in reproducible results. Exposure to direct sunlight or prolonged storage in darkness will affect the quality of the color chart and thus the accuracy of the measurement. The visually impaired and people with color blindness will prefer an alternative method of reading the results of the water color test kit. Last but not least, some take performing measurements really serious. Instead of relying on the calibration of the color chart by the supplier of the test kit, calibration measurements become part of the sample measurement procedure. The color chart is disposed of and replaced by a more accurate color scale. Electronics are used to get accurate color readings, perform calculations on the measurement data and to store results.

The RGB sensor selected for the Instrument of Things features a white LED to light the sample. This LED is mounted on the same PCB as the RGB sensor, thus only reflected light will be measured by the sensor. Optionally, a second white LED is easily connected to spare pins on the Arduino Due to measure by transmission of light through the sample. A third option is to place a reflective surface, e.g. aluminum foil, on the opposite side of the test vial.

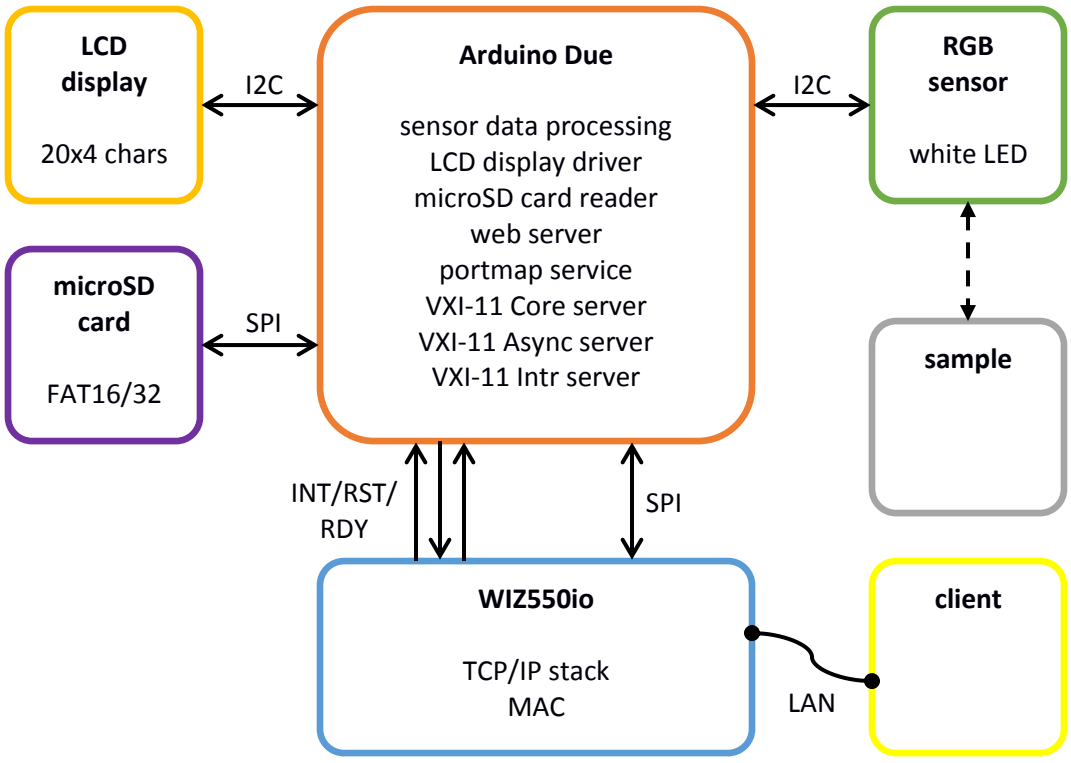


Figure 1, block diagram of the Instrument of Things.

Prototype 1

The first prototype of the Instrument of Things has been built on a breadboard. The Arduino Due development board is the center of the set-up and all peripherals are connected to the I/O-pins as defined below:

1. SD card module attached to the SPI bus of the Arduino Due

   MISO pin 74 (SPI MISO)
   MOSI pin 75 (SPI MOSI)
   CLK pin 76 (SPI SCK)
   CS pin 4 (SPI CS1)

2. WIZ550io module attached to the SPI bus of the Arduino Due

   MISO pin 74 (SPI MISO)
   MOSI pin 75 (SPI MOSI)
   CLK pin 76 (SPI SCK)
   CS pin 10 (SPI CS0)
   INT pin 46 (interrupt from module, active low)
   RST pin 48 (reset module, active low)
   RDY pin 50 (module ready)

3. A third module could be attached to the SPI bus:

   MISO pin 74 (SPI MISO)
   MOSI pin 75 (SPI MOSI)
   CLK pin 76 (SPI SCK)
   CS pin 52 (SPI CS2)

4. LCD module (note: 5V logic levels) attached to the I2C bus (address 0x20):

   SDA pin 20 (I2C TWI1 SDA)
   SCL pin 21 (I2C TWI1 SCL)

5. Adafruit TCS34725 breakout board attached to the I2C bus (address 0x29):

   SDA pin 20 (I2C TWI1 SDA)
   SCL pin 21 (I2C TWI1 SCL)

Figure 2 shows a photograph of the first prototype. After taking a sample in the test vial, reagents have to be added in a specific order and quantity. Two different water color test kits have been used: for nitrite (NO_2^-) and for nitrate (NO_3^-). The test vial with the sample and reagents for nitrite is shown on the left. The other test vial, with sample and reagents for nitrate, is placed in the sample holder with RGB sensor. The white LED on the sensor board is on and lights the orange liquid in the test vial.

The base plate shown on the right contains two breadboards, the Arduino Due, the LCD character display and the sample holder with test vial. The Wiznet WIZ550io module is placed on the top breadboard with the SD card module. The bottom breadboard contains BSS138 bi-directional level shifters, which enable control of the 5V LCD character display by the 3.3 V I2C bus of the Arduino Due.

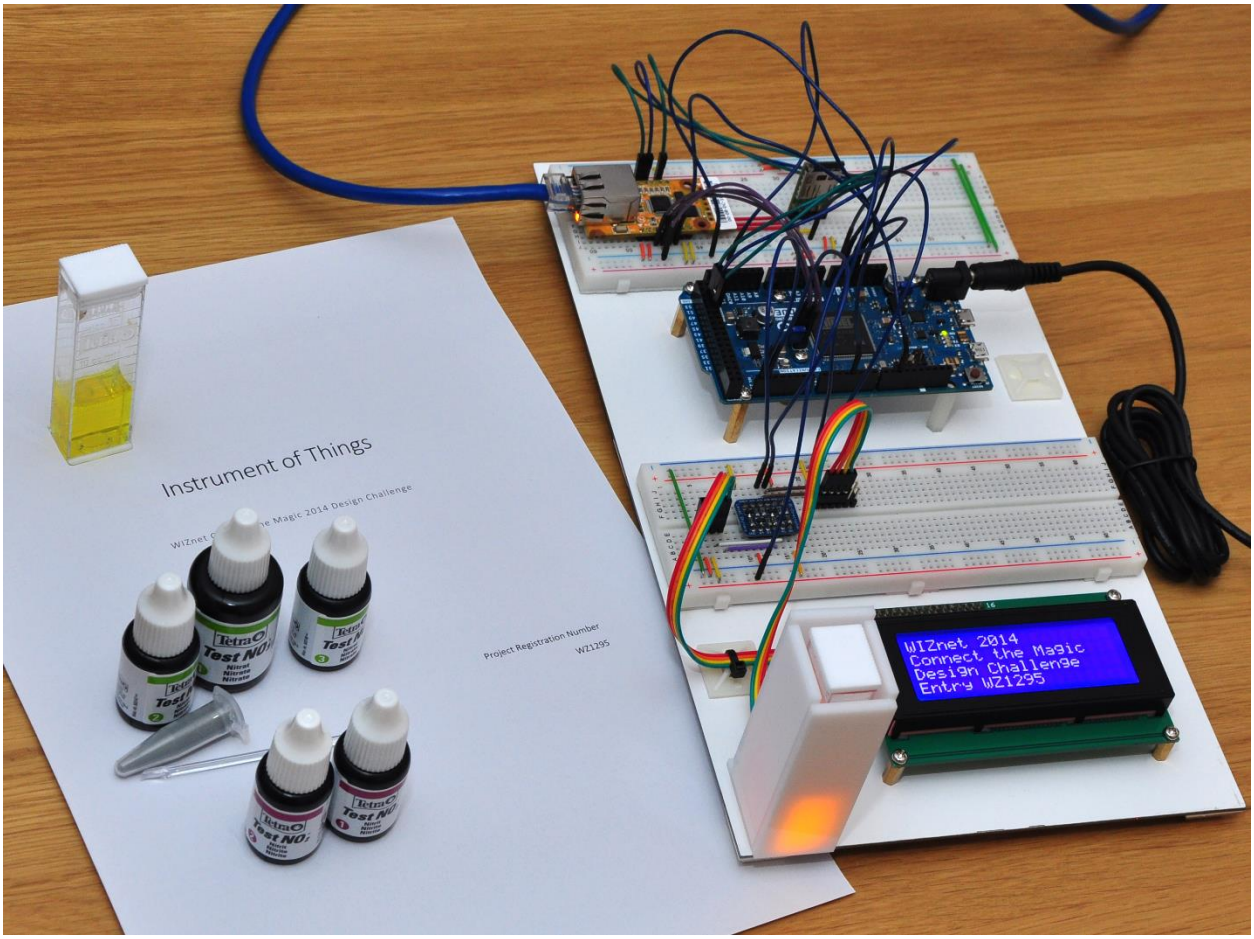


Figure 2, photograph of the first prototype of the Instrument of Things.

Prototype 2

For a second prototype build, a quick-and-dirty PCB has been designed in EagleCAD 6. The project files can be found in the zip-file WZ1295_EagleCAD6.zip. The PCB of prototype 2 has been manufactured using the Gerber files provided in zip-file WZ1295_Gerber.zip.

Table 3, modules used for prototype 2 of the Instrument of Things.

part ID	part description	supplier
M1	Arduino Due Atmel SAM3X8E ARM Cortex-M3 CPU	arduino.cc
M2	WIZ550io Ethernet controller v1.1	WIZnet
M3	MicroSD adapter 3.3V PMOD extension slot version	trioflex.ee
M4	Adafruit BSS138 4 bi-directional level shifters product ID: 757	adafru.it
M5	Adafruit TCS34725 RGB sensor product ID: 1334	adafru.it
M6	TC2004A-01 20x4 dot matrix LCD display	Tinsharp Industrial
M7	I2C serial interface module for 5V LCD LCM1602/2004, product ID: SKU077364	

Figure 3 shows a screenshot of the board layout editor of EagleCAD6 with the design of the prototype 2 dual layer PCB. The ground fill has been hidden to show the routing of signal and power lines.

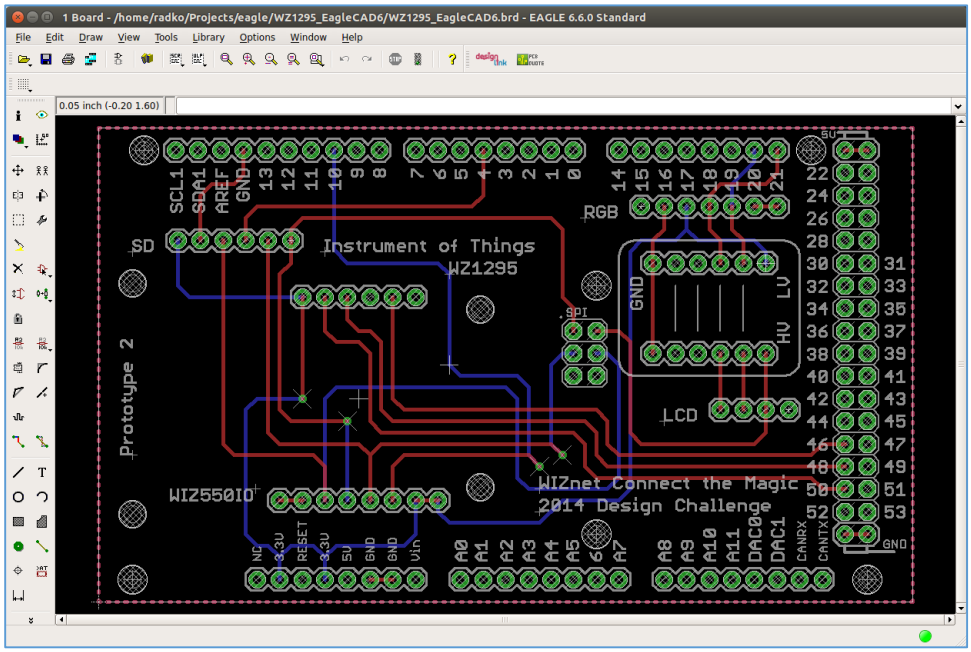


Figure 3, screenshot of the EagleCAD board editor with a front view of the prototype 2 PCB.

A photograph of the second prototype performing measurements on a test vial prepared for nitrates is shown in figure 4. The LCD display is continuously updated with the results of new measurements. Hexadecimal values are shown for the red, green, blue and clear sensors. The last two decimal values are the calculated color temperature in Kelvin and the calculated brightness in lux.

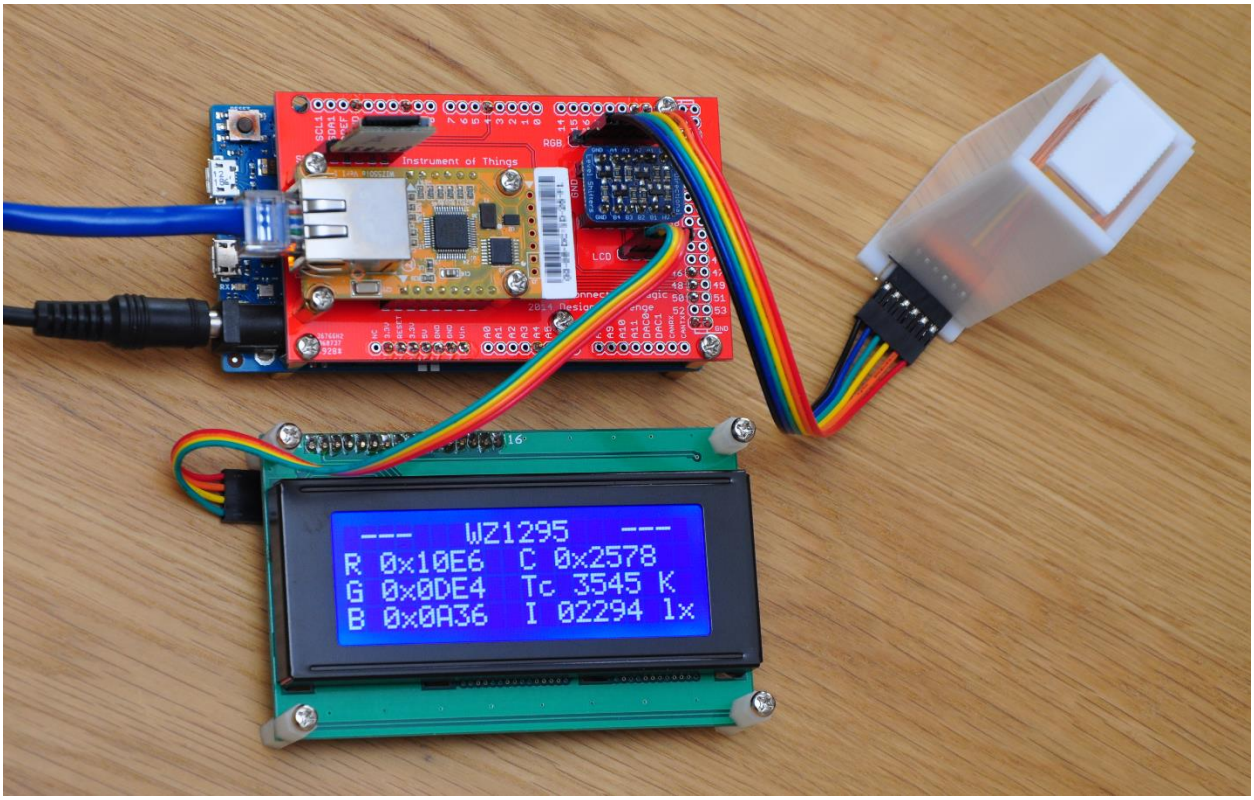


Figure 4, photograph of prototype 2.

Table 4 lists all project files provided in the WZ1295_Entry.zip archive as submitted for review.

Table 4, summary of all project files provided in the WZ1295_Entry.zip archive.

filename	description
WZ1295_BlockDiagram.docx	The block diagram of figure 1.
WZ1295_ColorChart_nitrate.png	Measured color chart for the nitrate test kit (figure 5a).
WZ1295_ColorChart_nitrite.png	Measured color chart for the nitrate test kit (figure 5b).
WZ1295_CompleteDocumentation.docx	This document in Microsoft Word format.
WZ1295_CompleteDocumentation.pdf	This document in PDF format.
WZ1295_EagleCAD6.zip	EagleCAD project of the prototype 2 PCB.
WZ1295_EagleCAD6_board_topview.png	Screenshot of the prototype 2 PCB of figure 2.
WZ1295_Files_sdcard.zip	Files to be placed on the SD card.
WZ1295_Gerber.zip	Gerber files for manufacturing of the PCB.
WZ1295_Prototype1_photo.png	Photograph of the working prototype 1 (figure 2).
WZ1295_Prototype2_photo.png	Photograph of the second prototype (figure 4).
WZ1295_Python_test_code.zip	Short Python test code with various SCPI commands.
WZ1295_Screenshot_webpage.png	Screenshot of the website served by the instrument.
WZ1295_Screenshot_serial_terminal.png	Screenshot of a serial terminal showing debugging info.
WZ1295_SourceCode-140802b.zip	Arduino archive with source code files.
WZ1295_Wireshark_portmap_call.png	Screenshot of a captured call to the portmap service.

Source code

Arduino Due

The source code for the Arduino Due can be found in the zip-file WZ1295_SourceCode-<timestamp>.zip, where <timestamp> consists of the archive date and archive version. The zip-file contains the source files listed in table 5. For each C++ source code file (.cpp file extension), a header file with corresponding file name (and .h file extension) is included.

The project has been written with version 1.5.6 of the Arduino IDE. In addition to this, the following libraries are required to build the project source code:

1. Wiznet Ethernet library for Arduino (https://github.com/Wiznet/WIZ_Ethernet_Library)
2. New LiquidCrystal library 1.2.1 (https://bitbucket.org/fmalpartida/new-liquidcrystal/wiki/Home)
3. Sdfatlib FAT16/FAT32 Arduino library for SD/SDHC cards 20131225 (https://code.google.com/p/sdfatlib)
4. Adafruit TCS34725 Color Sensor Driver for Arduino (https://github.com/adafruit/Adafruit_TCS34725)

Table 5, names and descriptions of the files in the source code package.

filename	description
colortestkitmeter/colortestkitmeter.ino	The main Arduino project file with the usual begin() and loop() functions.
colortestkitmeter/display.cpp	Functions to display information on the LCD character display.
colortestkitmeter/http.cpp	A (very) simple web server that reads files from the SD card and prints settings in HTML format.
colortestkitmeter/instr.cpp	The main instrument source code, which starts all parts of the Instrument of Things.
colortestkitmeter/keywords.txt	This file may contain special keywords for syntax highlighting in the Arduino IDE.
colortestkitmeter/misc.cpp	Scratchpad for miscellaneous functions.
colortestkitmeter/portmap.cpp	Source code of the port mapping server.
colortestkitmeter/rpc.cpp	The RPC servers for both TCP and UDP.
colortestkitmeter/sdcard.cpp	Source code to read settings from a configuration file and to serve files to the webserver.
colortestkitmeter/sensors.cpp	Functions to initialize the RGB sensor and perform measurements.
colortestkitmeter/vxi-11.cpp	VXI-11 server code including processing of SCPI messages.
colortestkitmeter/xdr.cpp	Functions for conversion of data to/from XDR format.

Unfortunately, some changes still have to be made to the libraries for them to work:

1. The WIZ550io module needs to be selected in the file w5100.h of the Wiznet Ethernet library for Arduino (libraries/Ethernet/utility/w5100.h):

```
#define W5500_ETHERNET_SHIELD
#define WIZ550io_WITH_MACADDRESS
```

2. Reduce the SPI clock frequency for the WIZ550io module in the file w5500.cpp (libraries/Ethernet/utility/w5500.cpp) to prevent communication errors:

```
SPI.setClockDivider(SPI_CS, 3); // 28 Mhz
//SPI.setClockDivider(SPI_CS, 2); // 42 Mhz
```

3. The liquid crystal library uses the function digitalPinToTimer() to check if the backlight pin is connected to a digital output or to a PWM output. This function is not working in the used version of the Arduino IDE when compiling for Arduino Due. Therefore disable the use of this function in the file LiquidCrystal.cpp (libraries/LiquidCrystal/LiquidCrystal.cpp):

```
if (true)
//if(digitalPinToTimer(_backlightPin) != NOT_ON_TIMER)
```

4. The liquid crystal library uses the function _BV(i) to obtain a byte with a 1 shifted left to position i. This function is defined for the AVR architecture (hardware/tools/avr/avr/include/avr/sfr_defs.h) and apparently not defined for SAM3X microprocessors. Add the definition of the function to file FastIO.h (libraries/LiquidCrystal/FastIO.h):

```
#define _BV(bit) (1 << (bit))
```

5. Change the configuration of the Sdfatlib library to use the Arduino SPI library in SdFatConfig.h (libraries/SdFat/SdFatConfig.h):

```
#define USE_ARDUINO_SPI_LIBRARY 1
```

6. Ensure the correct libraries are used instead of the libraries delivered with the Arduino IDE. Move the following libraries to a backup location:

   /opt/arduino-1.5.6/libraries/Ethernet

   /opt/arduino-1.5.6/libraries/SD

7. Disable the library delay.h in the Adafruit TCS34725 library (libraries/Adafruit_TCS34725/Adafruit_TCS34725.cpp):

```
//#include <util/delay.h>
```

Figure 5 shows the structure of classes in the Arduino source code for the instrument. The deviceServer class of type instrServer is the only object instantiated in the Arduino sketch (colortestkitmeter.ino). The function instrServer::begin() is called in the setup of the sketch and the instrServer::available() function is the only call needed in the main loop. The portmap server, VXI-11 servers, SD card file system, web server, LCD display driver and RGB sensor controller are all created by the deviceServer.

The namespace instrMemory forms the shared memory of the instrument. The global variables defined in this namespace can be used in all parts of the source code.

The WIZ550io module and/or the Arduino library does not appear to feature a virtual loopback interface. During initialization of the instrument, the VXI-11 listeners will have to register their service with the port mapper. However, the port mapper could not be reached by sending packets to the commonly used 127.0.0.1 IP address. The data of the registration packet is therefore send directly to the receiver function of the portmap server.

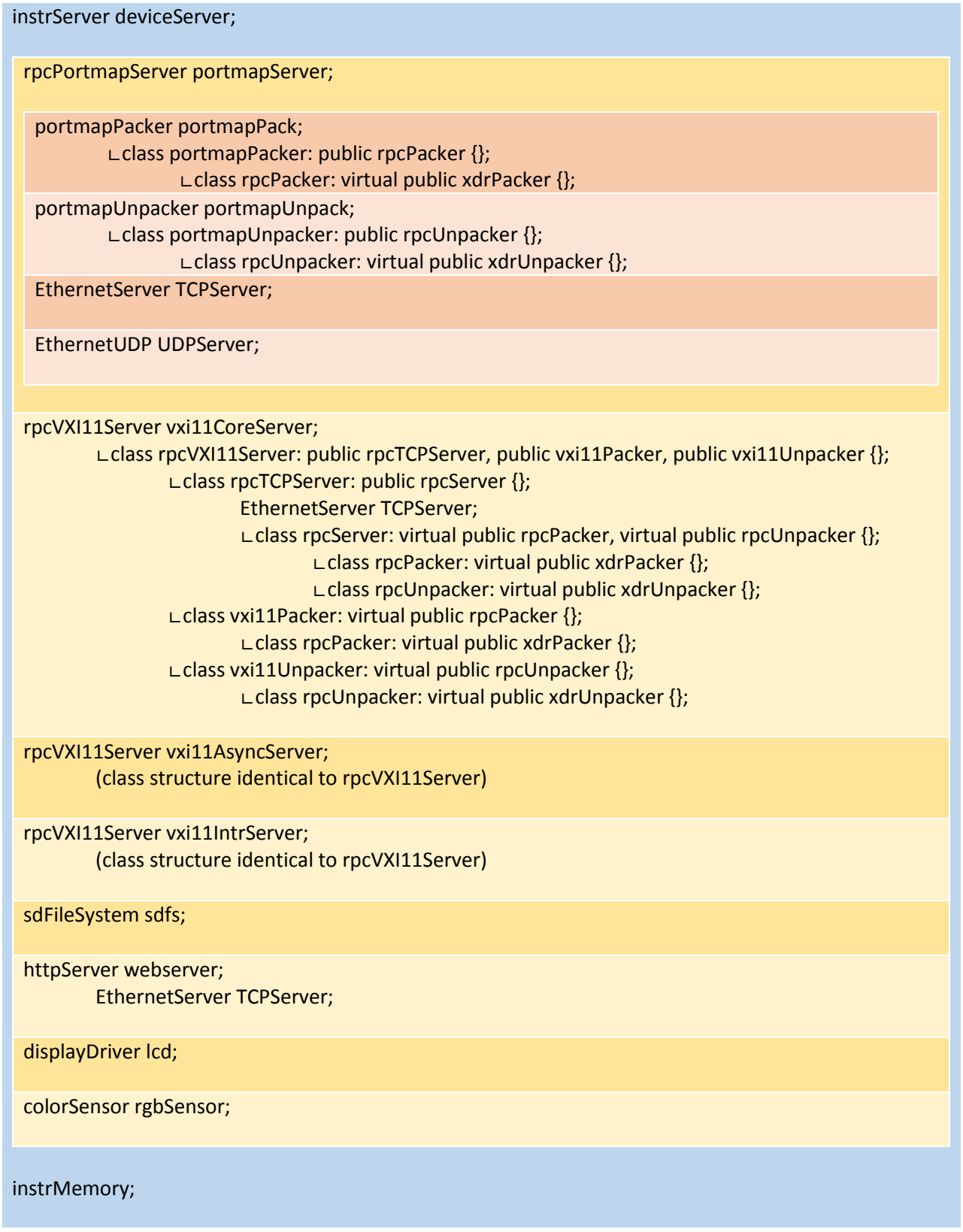


Figure 5, structure of classes in the Arduino source code for the instrument.

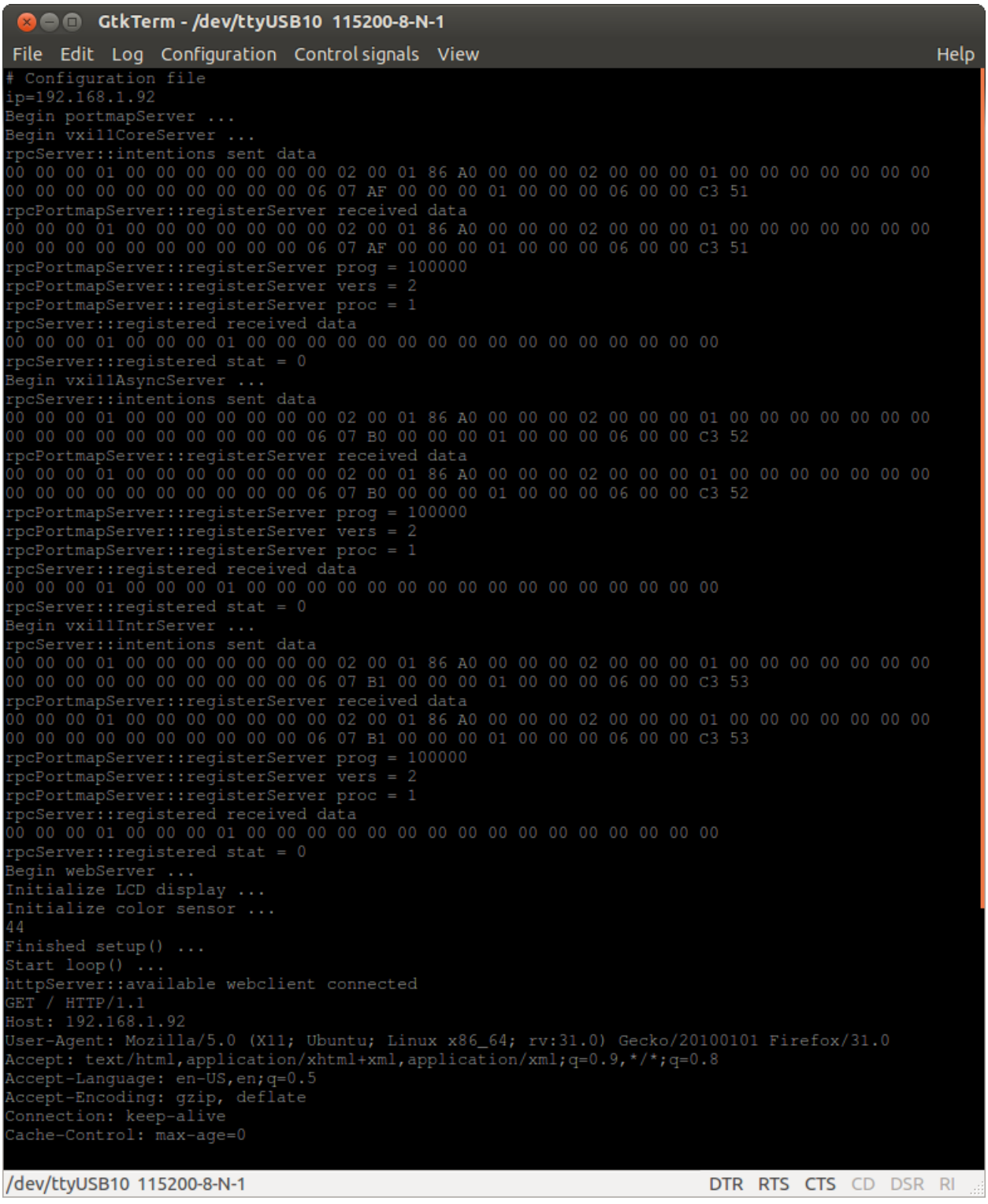


Figure 6, screenshot of a serial terminal with debugging information.

The default SPI configuration of the WIZ550io is set in the w5500.cpp file: CS pin 10, SCK at 42 MHz, SPI Mode 0. The bitorder is not set in the driver: MSB-first is set as default in the SPI library. The extended

SPI features of the Arduino Due appear to be supported by the driver. Since running the SPI line to the module at 42 MHz results in communication errors, the clock frequency is changed to 28 MHz.

SD card and webserver

The SD card contains a file for configuration of the instrument (default settings) and files for the webserver. Additional HTML files and images can be added to the SD card and made available via the webserver by changing the Arduino source code. The basic webserver has been set-up to serve only those files defined in the source code.

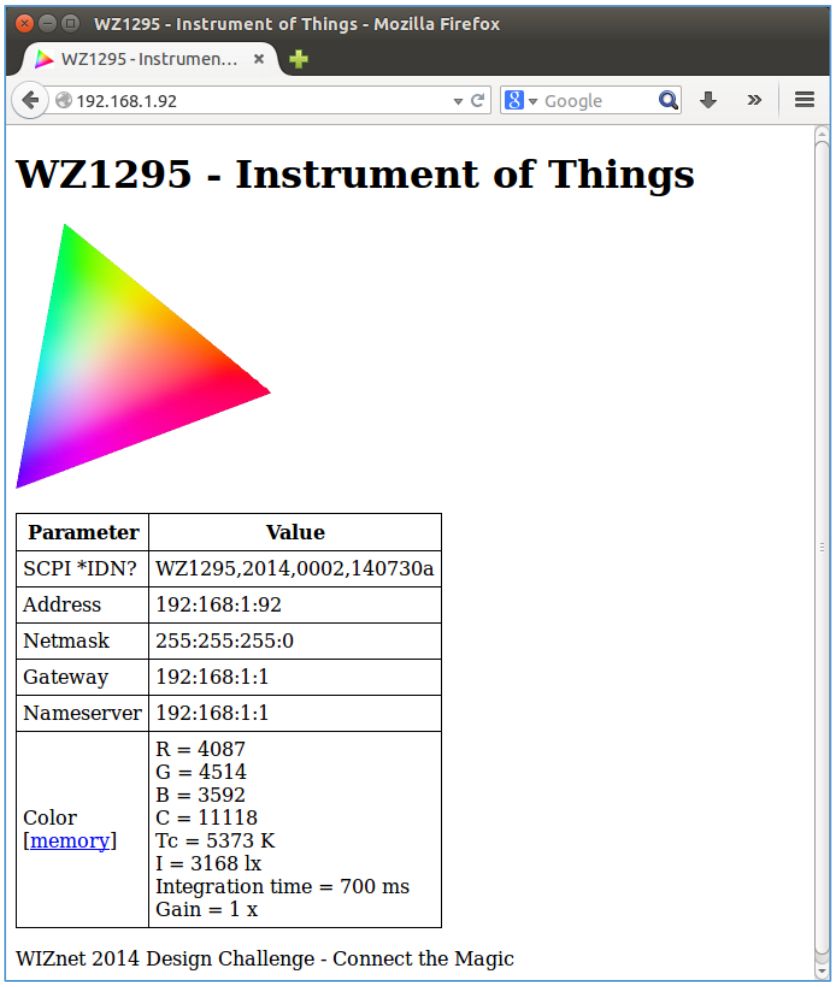


Figure 7, screenshot of the basic web page shown by the instrument.

Table 6, files in the archive WZ1295_Files_sdcard.zip

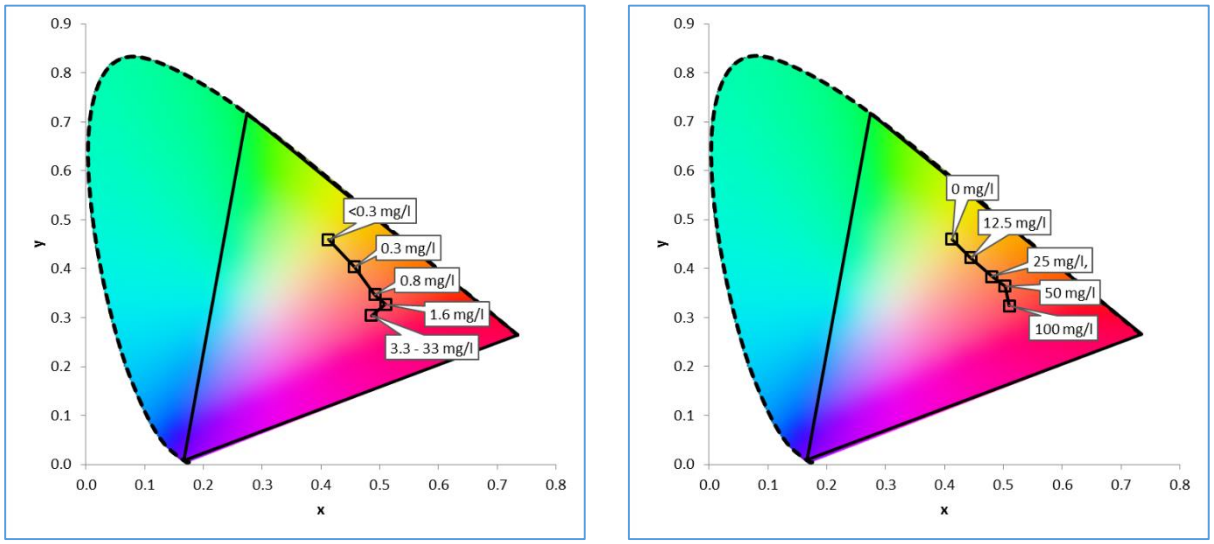
filename	description
color.jpg	Image to be displayed on the web page.
config.ini	Configuration file with default settings for the instrument.
index.htm	Main part of the HTML page shown in the web browser.

The main page shown in the web browser is partly defined by the index.htm file on the SD card. The remainder of the page is dynamically constructed in the Arduino code. The main page contains a link to another dynamically constructed HTML page, which prints the measurement data stored in internal memory in a comma-separated format to the web client.

Tests and instrument clients

Water color test kits

The used water color test kits for nitrite (NO2-) and for nitrate (NO3-) each contain a cardboard color chart for reference. The color charts have been digitized using a color scanner and the RGB color values were obtained by averaging pixels in photo editing program GIMP. The results are shown on the CIE 1931 chromaticity diagrams in figure 8. The triangle defines the borders of the RGB color space.



a) b)

Figure 8, measured colors on the color chart of the nitrite (a) and nitrate (b) test kits shown on the CIE 1931 chromaticity chart.

The chemistry of both water color test kits is based on the Griess test. See reference 8. The two nitrite reagents are most likely solutions of surfanilic acid and alpha-naphytylamine, but the supplier of the test kits failed to print this detail on the bottles, on the instructions or on their website. The small ampule with power (see figure 2) is probably zinc dust, which is used to reduce nitrate to nitrite.

Python VXI-11 client

A VXI-11 driver for controlling instruments over Ethernet from the Python programming language can be downloaded from the Github repositories. Once installed, the Python interpreter can be started to connect to the instrument. The following Python code connects to the instrument at IP address 192.168.1.91 and prints the reply to a request for identification of the instrument.

```
#!/usr/bin/python
import vxi11
instr =  vxi11.Instrument("192.168.1.91")

print(instr.ask("*IDN?"))

instr.close()
```

Figure 9, example Python code to connect to the instrument and read data.

The program Wireshark is ideal for recording and debugging the transmission of packets between the instrument and the client. The following string of bytes (in hexadecimal representation) is received when the Python test code connects to the portmap service running on the device:

```
80 00 00 38 00 00 00 01 00 00 00 00 00 00 00 02 00 01 86 A0 00 00 00 02 00 00 00 03 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 06 07 AF 00 00 00 01 00 00 00 06 00 00 00 00
```

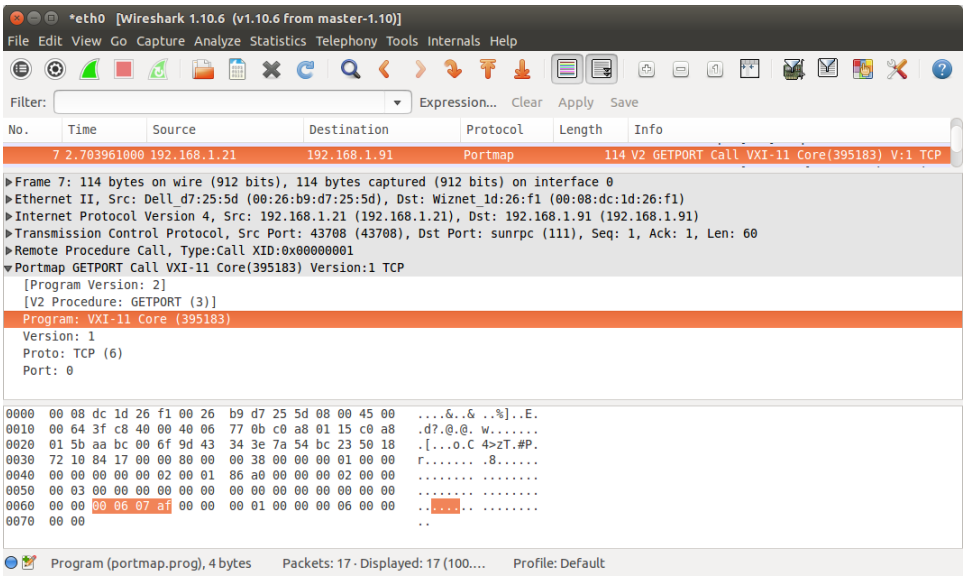


Figure 10, screenshot of an Ethernet packet captured by the Wireshark program.

A capture of the communication between the PC and the device, using the program Wireshark, shows the packet as “V2 GETPORT Call VXI-11 Core(395183) V:1 TCP”. The overhead of the packet has already been stripped. The payload of the packet breaks up into 15 parts of 4 bytes each, with the following meaning:

```
80 00 00 38 Fragment Length: 56
00 00 00 01 XID: 0x1 (1)
00 00 00 00 Message Type: Call (0)
00 00 00 02 RPC Version: 2
00 01 86 A0 Program: Portmap (100000)
00 00 00 02 Program Version: 2
```

```
00 00 00 03 Procedure: GETPORT (3)
00 00 00 00 Credentials Flavor: AUTH_NULL (0)
00 00 00 00 Credentials Length: 0
00 00 00 00 Verifier Flavor: AUTH_NULL (0)
00 00 00 00 Verifier Length: 0
00 06 07 AF Program: VXI-11 Core (395183)
00 00 00 01 Version: 1
00 00 00 06 Proto: TCP (6)
00 00 00 00 Port: 0
```

This packet needs to be replied to with the corresponding port of the VXI-11 Core program, according to the registration of this program with the portmap service.

The return string could be the following sequence of 8 parts of 4 bytes each:

```
80 00 00 1C Fragment header: Last fragment, 28 bytes
00 00 00 01 XID: 0x1 (1)
00 00 00 01 Message Type: Reply (1)
00 00 00 00 Reply State: accepted (0)
00 00 00 00 Verifier Flavor: AUTH_NULL (0)
00 00 00 00 Verifier Length: 0
00 00 00 00 Accept State: RPC executed successfully (0)
00 00 10 01 Port: 4097
```

Once the client is aware of the port of the VXI-11 Core service, it will be contacted with the following sequence (RPC = red, VXI-11 Core = blue):

```
80 00 00 40 Fragment header: Last fragment, 64 bytes
00 00 00 01 XID: 0x1 (1)
00 00 00 00 Message Type: Call (0)
00 00 00 02 RPC Version: 2
00 06 07 AF Program: VXI-11 Core (395183)
00 00 00 01 Version: 1
00 00 00 0A Procedure: CREATE_LINK (10)
00 00 00 00 Credentials Flavor: AUTH_NULL (0)
00 00 00 00 Credentials Length: 0
00 00 00 00 Verifier Flavor: AUTH_NULL (0)
00 00 00 00 Verifier Length: 0
1A C5 61 FA Client ID: 449143290
00 00 00 00 Lock Device: No
00 00 27 10 Lock Timeout: 10000
00 00 00 05 Device Name Length: 5
69 6E 73 74
30 00 00 00 Device Name: inst0
```

The reply should be (RPC = red, VXI-11 Core = blue):

```
80 00 00 28 Fragment header: Last fragment, 40 bytes
00 00 00 01 XID: 0x1 (1)
00 00 00 01 Message Type: Reply (1)
00 00 00 00 Reply State: accepted (0)
00 00 00 00 Verifier Flavor: AUTH_NULL (0)
00 00 00 00 Verifier Length: 0
00 00 00 00 Accept State: RPC executed successfully (0)
00 00 00 00 Error Code: No Error (0)
00 00 00 00 Link ID: 0
00 00 10 01 Abort Port: 4097
00 00 04 00 Maximum Receive Size: 1024
```

The link with the instrument is made and the first command can be send (RPC = red, VXI-11 Core = blue):

```
80 00 00 44 Fragment header: Last fragment, 68 bytes
00 00 00 02 XID: 0x2 (2)
00 00 00 00 Message Type: Call (0)
00 00 00 02 RPC Version: 2
00 06 07 AF Program: VXI-11 Core (395183)
00 00 00 01 Version: 1
00 00 00 0B Procedure: DEVICE_WRITE (11)
00 00 00 00 Credentials Flavor: AUTH_NULL (0)
00 00 00 00 Credentials Length: 0
00 00 00 00 Verifier Flavor: AUTH_NULL (0)
00 00 00 00 Verifier Length: 0
00 00 00 00 Link ID: 0
00 00 27 10 I/O Timeout: 10000
00 00 27 10 Lock Timeout: 10000
00 00 00 08 Flags:Wait Until Locked: False
                  Set EOI: True
                  Termination Character Set: False
00 00 00 05 Data Length: 5
2A 49 44 4E
3F 00 00 00 Data: *IDN?
```

SCPI syntax

The SCPI commands defined in table 7 have been implemented in the Arduino source code. With this set of instructions it is possible to obtain the unique identification of the instrument, read the configuration of the network interface, enable/disable the white LED, change the integration time of the RGB sensor and its gain, and finally read the result of the last measurement. Additional functionality of the instrument can be easily implemented and made available through additional SCPI commands.

Table 7, syntax and description of implemented SCPI commands.

SCPI syntax	Description
*IDN?	Query the current device information. The SCPI returns **WIZ1295,2014,0002,140802b**.
:CHAN1:DATA?	Query the most recent
:CHAN1:LED? :CHAN1:LED <state>	Query the state of the LED for channel 1. Set the state of the LED for channel 1 with <state> = {0|1|OFF|ON}. The SCPI is answered with either 0 or 1.
:CHAN1:ITIM? :CHAN1:ITIM <value>	Query the current integration time of the sensor. Set the integration time with <value> = {2.4|24|50|101|154|700}. The SCPI is answered with the numeric integration time in ms.
:CHAN1:GAIN? :CHAN1:GAIN <value>	Query the current gain of the sensor. Set the integration time with <value> = {1|2|16|60}. The SCPI is answered with the numeric gain factor.
:LAN:IPAD?	Query the IP address of the instrument.
:LAN:SMAS?	Query the subnet mask for the instrument.
:LAN:GAT?	Query the gateway address for the instrument.
:LAN:DNS?	Query the nameserver address for the instrument.

Common (?) pitfalls

The following list of tips and reminders may help to get through the first hurdles.

1. Verify the SPI clock frequency and mode using an oscilloscope or data logger.
2. Some data types on Arduino Due have a larger size, compared to other Arduino products. For example, a variable of type double provides 64-bit precision (8 bytes) on the Arduino Due and only 32-bit precision on most other Arduino development boards.
3. Use a 10x or larger oscilloscope probe to measure MHz signals like the SPI data bus.
4. Avoid the 5V output pins on the Arduino Due when using 3.3V modules. Cover the 5V output pins with a spare jumper to prevent mistakes and damaged parts.
5. Old libraries of previous versions of the Arduino IDE (in e.g. /usr/share/Arduino/libraries) may affect correct functioning of the sketch.
6. New build folders in /tmp: build*.tmp, console*.tmp and untitled*.tmp are created each time the Arduino IDE is started. The old folders should not affect building updated source code, but clean up to make sure.

Further development

The possibilities for further development for the Instrument of Things are close to endless. First of all is the challenge to accomplish full compliance with the VXI-11 and LXI industry standards. With the framework presented in this entry this can only be a matter of time and effort.

The basic Python VXI-11 client is a nice start and very user friendly for troubleshooting and development of additional SCPI functions. One of the next steps in development of the Instrument of Things is to deliver drivers for popular programs like Matlab and LabView. The latter provides a wizard for the creation of new instrument drivers, starting with a selection from standard templates. The availability of (generic) instrument drivers will strengthen the developed application and make its use in other electrical instruments in a lab environment more versatile.

The page shown by the webserver can be improved with additional information, the capability to change default settings of the instrument, features for remote control of the instrument and visualization of results. The available space on the SD card allows for extensive storage of measurement data.

Porting the source code of the Instrument of Things to a multi-threading microcontroller platform will enable parallel processing of measurements and interaction with the user. A real-time clock would be a nice additional feature, to timestamp measurement results and data files.

Further development of the color test kit measurement instrument is also needed. Adding functions that fit the various test kit color charts to the configuration of the instrument will enable the instrument to derive concentrations automatically from the measured color values. Measurement data can be averaged or filtered, both according to configuration settings of the instrument. These settings can then be read and set using SCPI commands or the internal web server.

The accuracy of the color measurements can be greatly improved by compensating for changes in intensity of the light. Since the RGB color space is not very suitable for manipulations, a conversion to the HSL (Hue Saturation Lightness) color space will be needed for normalization of the measurement and calibration results. In the HSL color space it will also be more trivial to include the data from the Clear light sensor in the normalization.

The current version of the Arduino source code occupies just 14% of the available program storage space (73,852 out of 524,288 bytes).

Closing remarks

The author of this entry would like to express his sincere thanks and appreciation to WIZnet for the free WIZ550io Ethernet Module used in the Instrument of Things and the opportunity to participate in the Connect the Magic 2014 Design Challenge.

References

1. Wiznet Ethernet Library for Arduino, https://github.com/Wiznet/WIZ_Ethernet_Library
2. LXI Consortium, http://www.lxistandard.org
3. VXIbus Consortium, http://www.vxibus.org
4. Arduino Ethernet Library for Wiznet chipsets, https://github.com/jbkim/W5200-Arduino-Ethernet-library
5. VXI-11 driver for Python, https://github.com/alexforencich/python-vxi11
6. VXI-11 Ethernet Protocol for Linux, http://optics.eee.nottingham.ac.uk/vxi11/

7. Arduino + Ethernet shield + XML-RPC + Python, http://omatic.net/projects/11/arduino-ethernet-shield-xlm-rpc-python
8. Griess chemical analysis test for detection of nitrite, http://en.wikipedia.org/wiki/Griess_test

Acronyms

CLK	Clock
CS	Chip Select
FAT	File Allocation Table
HSL	Hue Saturation Lightness color space
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
I2C	Inter-IC bus
INT	Interrupt
IoT	Instrument of Things
IP	Internet Protocol
LAN	Local Area Network
LCD	Liquid Crystal Display
LXI	LAN eXtensions for Instrumentation
MAC	Media Access Control
MISO	Master Input Slave Output
MOSI	Master Output Slave Input
PCB	Printed Circuit Board
RDY	Ready
RGB	Red Green Blue color space
RPC	Remote Procedure Call
RST	Reset
SCK	Serial ClocK
SCL	Serial Clock Line
SCPI	Standard Commands for Programmable Instruments
SDA	Serial DAta line
SPI	Serial Peripheral Interface
TCP	Transmission Control Protocol
TWI	Two Wire Interface
UDP	User Datagram Protocol
VXI-11	VME eXtensions for Instrumentation
XDR	eXtended Data Representation