IP PUCK Implementation and Verification

Oriol Pallarés Valls

Polytechnic Institute of Catalunya

Mentor: Thomas C. O'Reilly

Summer 2013

Keywords: IP PUCK; Zeroconf; Sensor Modeling Language (SensorML); Sensor Interface Description (SID); Compliance tool

ABSTRACT

Sensor networks are the most common systems for studying and analyzing our environment. Each sensor in the network generates its own information, which is transferred to the processing systems and to other sensor networks through a communication network. The information of each sensor can be processed in order to give global information of an environment.

More than 90% of oceanographic instruments have serial interface, so they are not designed for IP network operation (K. & Edward, 2009). There are few standard communication or data exchange format for oceanographic instruments, beyond NMEA and recently, OGC PUCK

The necessity of a reliable and universal system and the lack of standards on this field make PUCK IP standard completely essential (Song & Lee, 2007).

PUCK implementation makes interoperability between marine instruments and services easy to achieve. With this standardization, each new instrument that is installed can be auto configured and made ready to work. Perhaps the most analogous standard to OGC PUCK is USB. When a USB device is plugged into a laptop, the laptop automatically downloads and installs the appropriate driver, based on identifying information read from the device, Similarly, OGC PUCK protocol provides this "plug and work" capability to any serial instrument.

This standard is been used by some manufacturers on its instruments with the RS232 functionality. In order to provide this "plug and work" functionality to any system inside a local network the IP PUCK standard has been created, which provides Sensor Web Enablement (SWE) capability to the serial instrument. For this reason it is important to provide a compliance tool capable of detecting if the manufactured instrument is completely compliant with IP PUCK protocol standard or RS232-PUCK protocol.

INTRODUCTION

Underwater sensor networks are formed by different nodes, capable to acquire real time data and to interoperate between them to create a whole instrumentation network. These sensors are powered and data connected through the main network to other networks or final users. So one of the main properties of a network should be the ease to scalability, making possible to a user plug a new instrument to it and use the data streams directly, without the necessity to configure anything. The instrument should automatically get an IP address and start to work: “plug and work”.

After the physical connection of a new sensor, a PC can be connected to the network with IP PUCK service discovery tool, based on Zeroconf technology (Williams, 2002), and automatically discover all the PUCK services connected to the network. This tool provides a human readable name for each instrument, from which can be extracted the IP and the IP PUCK port to start TCP/IP communication. Once the connection is performed, electronic datasheet and PUCK payload can be read, providing instrument capabilities and communication protocol in a standard way.

In this way all the necessary information for starting work automatically is provided, and it is not necessary for human intervention to do this configuration process, avoiding possible set-up errors.

These network utilities are useful for interconnecting different instruments in a network and allow them to exchange data, using the same standardized communication protocol. Because of the datasheet and PUCK payload of each instrument, different data types can be unified to the same data format. Moreover these standards provides the possibility of adding data servers capable of storage or publication on the internet of all the underwater sensor network information, in spite adding instruments that are completely unknown by the server, because Zeroconf tool will discover the instrument, and IP PUCK protocol will provide the communication capability between the sensor and the server, figure 1.

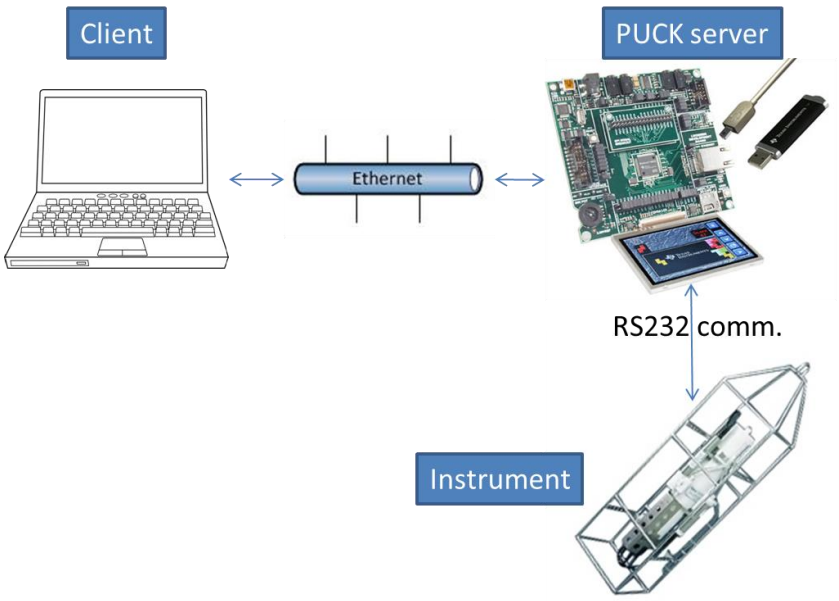


Figure 1 Workbench

All these situations that were presented illustrate the main capabilities of adding PUCK protocol to an instrument. PUCK protocol simplifies and automates the connection of a sensor to a network that formerly was done by an electrical engineer, adapting the data format and programming the communication protocol in order to connect all the nodes of the network, adding the possibility of some human errors, which are now eliminated.

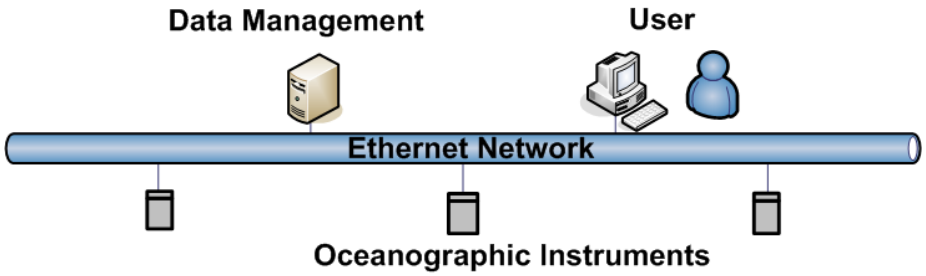
PUCK protocol is an OGC defined standard; each instrument with PUCK capabilities has to be tested in order to ensure its full compliance. Testing can be done manually with different tools in order to test if the system works properly, or running an automated compliance tool, which design is the goal of this internship. This compliance tool verifies IP PUCK compliance. This way, human error neither affects instrument installation nor instrument PUCK test compliance. It provides test reliability each time that a manufacturer needs to verify its new instrument PUCK capabilities.

This project has been built upon Dan Mihai's (2010 MBARI Intern) IP PUCK software (Mihai Toma, 2010). It was developed on an LM3S9B96 Luminary evaluation board in order to adapt serial instrument communication to IP PUCK protocol communication. During this internship this IP PUCK software has been

completed in order to make it fully compliant OGC IP PUCK specification (O´Reilly, 2009), and has been developed the IP PUCK v1.4 compliance tool based on MBARI engineer Bob Herlien's RS232 PUCK compliance tool, as will be presented on next chapters.

MATERIALS AND METHODS

Instrument Platform



This smart Ethernet instrument is implemented on the Stellaris Luminary LM3S9B96 microcontroller. This development kit was described on Daniel Mihai Toma "Interoperable Marine Monitoring System" MBARI Summer internship 2010 (Mihai Toma, 2010). The Cortex M-3 processor makes it fast and easy to program (Sadasivan, 2010), which in conjunction with its peripherals makes this development kit perfect for IP PUCK development (del Rio, et al., 2009).

Figure 2 Stellaris LM3S9B96 Microcontroller Development Kit (DK-LM3S9B96)

IP PUCK

Using the same platform and taking as background the software presented on (Mihai Toma, 2010), some new capabilities have been added to this 'C' code. (Mihai Toma, 2010) project was performed before the PUCK standard final release (O'Reilly, 2011), making some modifications on that previous IP PUCK software necessary to bring it into PUCK standard compliance.

In the previous version all PUCK commands were implemented, except the timeout specification and the unique client connection permission, which are parts of the IP PUCK specification (O'Reilly, 2011); these must be implemented in order to be in compliance of the IP PUCK specification.

As mentioned, all the necessary commands to deal with the PUCK information appropriately were implemented. So the automated configuration process, called "plug-and-work", was completely functional on the start of this internship, and a PC connected to the PUCK host board through Ethernet could retrieve instrument information using PUCK commands, figure 3.

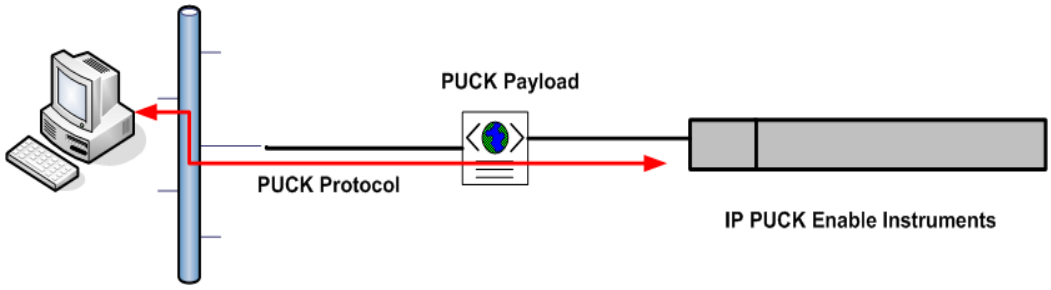


Figure 3 Application retrieve the IP PUCK payload

In table 1 are described all the commands implemented on the Luminary board that we will define as PUCK host. These commands permit the PUCK communication protocol, and the access to the PCUK memory allocated in the microSD slot available on the PUCK host, figure 4.

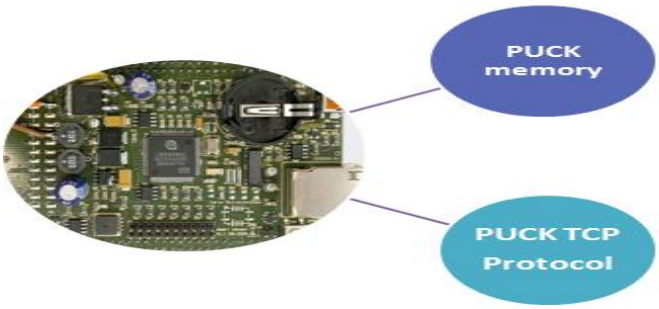


Figure 4 IP PUCK Implementation on Luminary microcontroller

Table 1 IP PUCK Command Summary

Command	Description
PUCKRM	Read from PUCK memory
PUCKWM	Write to PUCK memory
PUCKFM	End PUCK write session
PUCKEM	Erase PUCK memory
PUCKGA	Get address of PUCK internal memory pointer
PUCKSA	Set address of PUCK internal memory pointer
PUCKSZ	Get the size of the PUCK memory
PUCKTY	Query PUCK type
PUCKVR	Get PUCK protocol version string
PUCK	Null command

According to OGC PUCK Protocol Standard Version 1.4 (O'Reilly, 2011); besides to these commands, IP PUCK also defines other requirements, such as timeout and single client connection.

Only one client connection is allowed at any time on PUCK host port. This means that if a PC is connected to PUCK host, only it can interoperate with the PUCK host. After finishing the communication the PUCK client should free the PUCK port connection in order to allow other users to connect to it. This is a security requirement, because if more than one client is connected to the host then race conditions could occur. An example of race condition is when one client points to a memory direction in order to read its information, but before the reading the other client changes this pointer position, then the first client has done a bad reading but no error is detected.

In order to avoid this situation only one client can be connected on the same time, leaving incoming connections in a wait queue.

Then if the client does not close the connection or it is blocked, there is a security timeout to prevent PUCK host to be used only by one client in case of inactivity. This is realized by a two minutes timer. In case that the connected client does not send any PUCK command to the host, and there is not any function in process, then automatically the host sends a timeout message to the client and a connection reset petition. After this reset the host waits for the ACK of the client and closes the TCP connection. If the client is blocked and no ACK is received the connection is aborted.

After programming these two mechanisms, the entire IP PUCK stack is properly programmed on the PUCK host evaluation board. On the next figure is presented the timeout function execution and how is it integrated to (Mihai Toma, 2010) PUCK program. It is in application layer because we only want to reset timeout timer in case that a real PUCK command is received, so the PUCK application has to check it out. Then if any frame is detected, but PUCK frames, timer follow counting, due to it may be due a failure on the client.

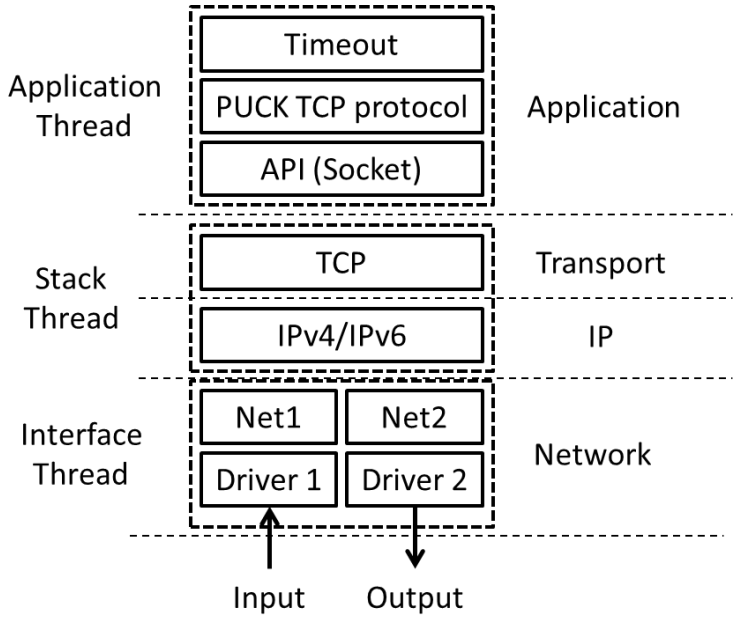


Figure 5 IP PUCK API on lwIP with PUCK timeout specification

Compliance tool

IP PUCK v.1.4 defined on (O'Reilly, 2011), is a defined communication standard. All the instruments IP PUCK compliant must accomplish all the statements defined on this article. To ensure global PUCK functionality some reliable system must be developed in order to check if the connected instrument performs all the IP PUCK functions as it is defined. Then if this test is passed, the manufacturer can assume that the instrument is fully PUCK v.1.4 compliant.

Before the start of this internship some compliance tools were implemented for RS232 PUCK, such as Bob Herlien's command line tool or Joaquin del Río LabVIEW compliance tool, but none of them could test IP PUCK completely as defined on the stack. So the main goal of this MBARI summer internship has been the design and development of this compliance tool.

For the design of the compliance tool, PUCK v.1.3 compliance tool has been used as background. There are some different functions between v.1.3 serial PUCK and v.1.4 IP PUCK, but the main flowchart is pretty similar. In the next figure is shown a block diagram of the RS-232 PUCK compliance tool, and then how it is modified in order to test all IP PUCK capabilities. In figure 6 , the main functions that are used in both PUCK tools, serial and IP are highlighted in blue. They are defined as general compliance tools in the PUCK specification. In orange we have RS-232 specific functions, which have been removed in order to port this tool to IP PUCK compliance. Finally in green are shown the IP PUCK specific functions that are replacing the orange ones.

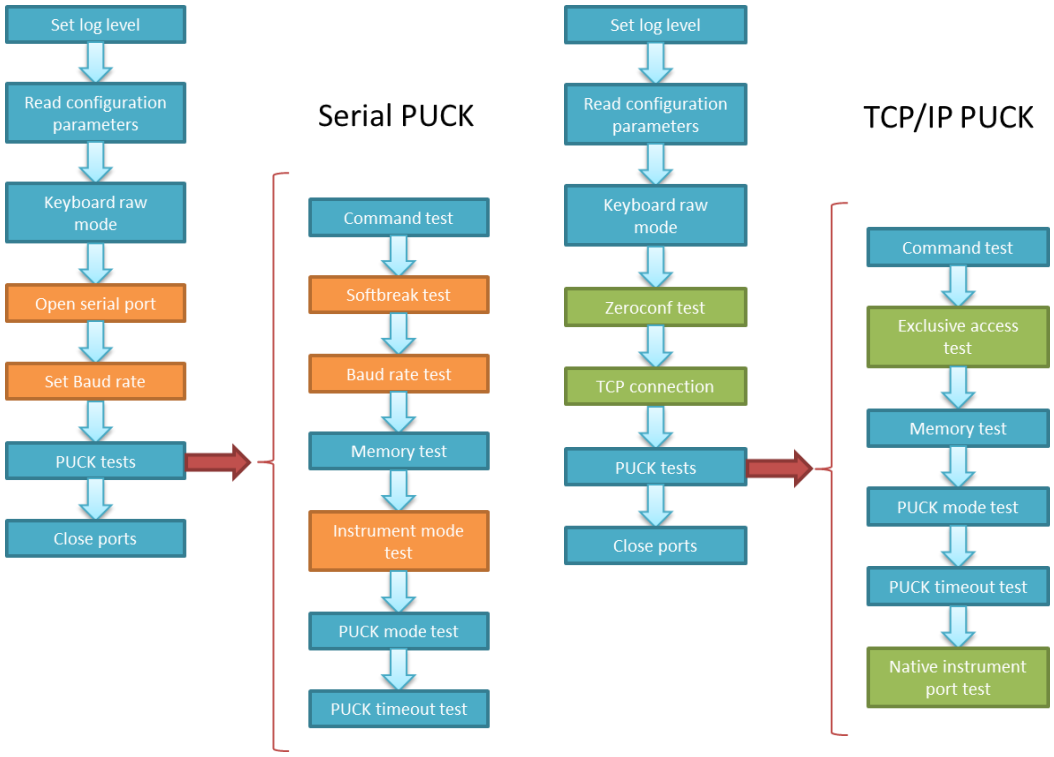


Figure 6 Serial PUCK compliance tool block diagram (left), IP PUCK compliance tool block diagram (right)

The compliance tool firmware has been developed in C programming language, due to its portability to other platforms and the facility and versatility of use. It includes a Make file so it can be compiled in any POSIX platform.

The project has been designed using the Eclipse environment, taking the same Make file that is used to compile over a POSIX platform to specify how to derive the target program and build it

Eclipse allows you to develop any kind of program with an easy and helpful user interface; in this case the kind of program is a C-language project using the specified make file on the project wizard.

The project contains all the source and header files in order to be compiled with the make files. It is not necessary to compile it with Eclipse environment; compilation can be done directly with the Make file, inside the project folder just calling ‘make’ function in a Linux environment.

Also included in the folder is a source file to write PUCK payload just in case of any failure and PUCK memory has to be recovered. This last program is not compiled in the Make file, so it should be compiled separately.

All source and header files are following the structure mentioned in picture 5. Now are described all of them in detail, explaining its functions and which ones have been modified for this IP PUCK v1.4 compliance tool.

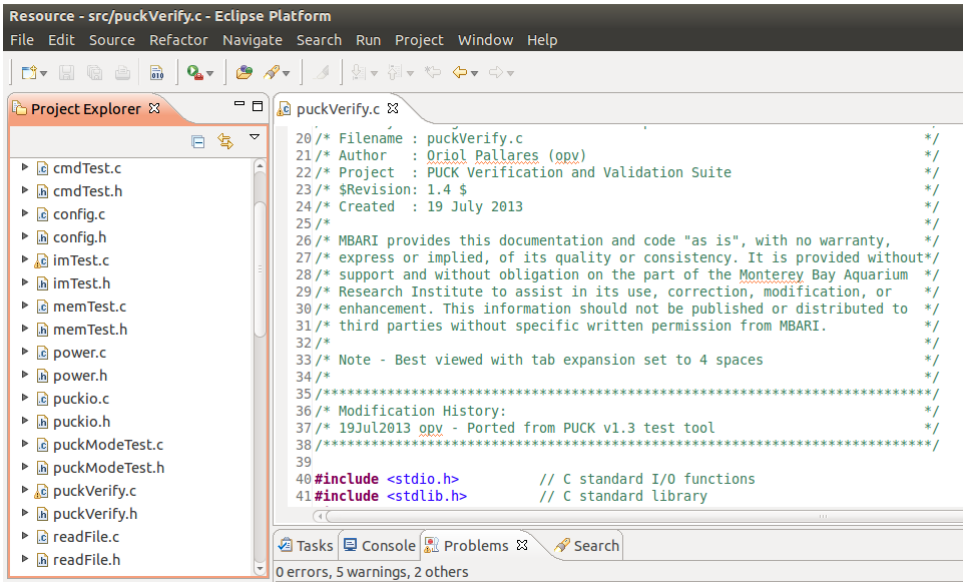


Figure 7 Project structure

In the "cmdTest" folder are defined all the necessary functions in order to test proper functionality of PUCK commands defined in Table 2. In order to do it, PUCK host is polled with all the functions not related with memory writings or readings, because it will be done in future tests, and is checked if the received response is what should be expected on a PUCK v1.4 instrument, if the test is performed with the DEBUG log option, then the responses are prompted to the command line, this way the user can check out if they are correct.

In the "config" file is parsed the configuration file in order to check all the possible input parameters for the compliance tool utility, such as instrument port, PUCK port, maximum softbreak tries, etc.

In the “imTest” instrument mode test are handled the functions that allows the user to communicate with the instrument port in order to verify the possibility of direct connection to the instrument and communicate with it with its own protocol and framing.

In the “memTest” source file is performed the memory test. This is composed by different writings and readings to the microSD data in order to verify it is not corrupted. This file has been modified from the previous v1.3 PUCK compliance tool. The main reason for modifying this test has been that in IP PUCK it is not possible to leave an invalid datasheet, due to the mDNS function use datasheet information to provide a human readable name that is used by Zeroconf utility to discover the server as was explained on (Mihai Toma, 2010). So if datasheet memory is modified by the memory test, as it was done on the previous version, the mDNS cannot assign a service name properly and it makes impossible to discover the service, what makes PUCK useless.

So the flowchart for the memory test is, first of all make a PUCK memory backup, in order to do this, memory size is polled to PUCK and after reading all the memory it is saved in a file. Then is performed a walking 1’s and 0’s test, what completely modify the whole memory values. Walking 0’s or 1’s test consist on writing this values to certain memory positions and then check if they have changed or not. After this memory checking test a functional test is performed, where are written different values to random memory positions and then is checked if the value is properly wrote and read.

Once this tests are finished all memory has been checked, so then it is checked a datasheet write and read, where the read datasheet has to meet the stored datasheet properties. If this test is passed too then the memory test is passed and the program proceeds to restore again the PUCK memory as it was before the memory test. It is done with the backup information stored in a file.

The “power” source file have not been modified from the previous version. This function checks if the memory is volatile or not. It’s done by resetting the microcontroller and checking if the memory is containing the same information that it had before the power off and power on.

The “puckio” file contains the functions to talk to PUCK. This file has been slightly modified, in order to change the destination framing socket, from serial socket to TCP socket.

The “PuckModeTest” contains the timeout function. This function checks if after two minutes of inactivity time the PUCK host sends a timeout frame and closes the TCP connection.

“puckVerify” is the main function of the compliance tool, where are handled all the mentioned functions and is implemented the block diagram of figure 6. This file has been modified in order to call the IP functions instead of the serial test functions.

The functions “readFile” and “writeFile” are in charge of memory backup mentioned on the memory test paragraph. They need the pointer to the name of the file to backup and the socket to get the information to be saved and restored.

The “tcp” function replaces the “udp” function. This code opens the socket to the TCP I/O port and implements the read/write functions to be called by all the other parts of the program. All the changes mentioned until now were due to the PUCK stack necessities. This last modification is needed by the program in order to be able to communicate through the TCP socket.

Finally the “ZeroconfBrowser.c” file contains the required functions to discover, resolve and identify the IP of a service. For realize these operations we are working with Bonjour browser, which is an apple application, able to discover all

the TCP/IP services that are running Zero configuration firmware in a Local Area Network, providing to the user the name or description to identify each instrument. Zeroconf functionality on PUCK host is described in (Mihai Toma, 2010).

For the compliance tool is needed a Zeroconf browser (Williams, 2002) in order to verify the proper functionality of the Zeroconf server on the host. The browser is handled by the "bonjour browser" daemon that is running in background since we installed this apple firmware. Then for making the API work with the rest of C code, it is performed a DNS service discovery operation, using apple's provided library, and then the results are received asynchronously.

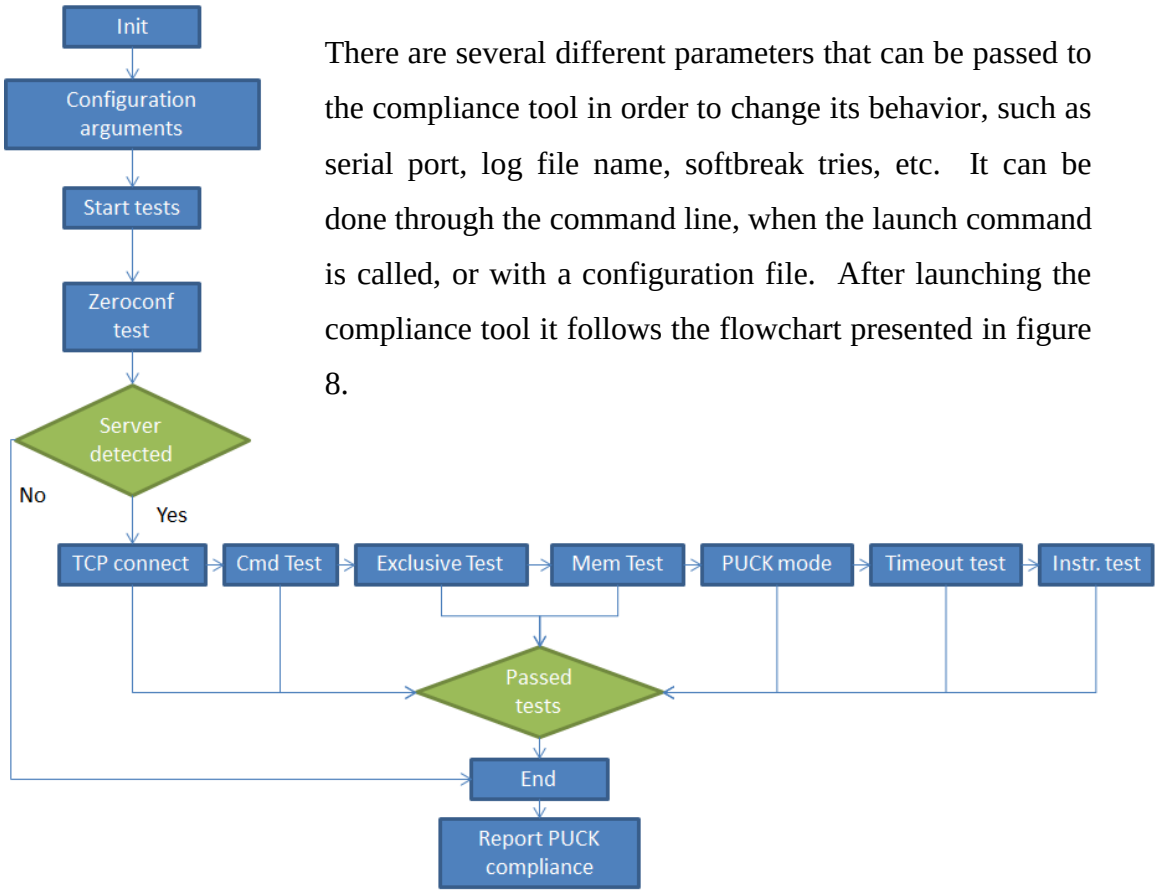
This means that you initiate a DNS-SD action such as browsing and provide the address of a callback function. When there is a response, the callback function is called and the appropriate information is passed to it. The same process is repeated for resolving the address and for getting its IP.

Here is presented an example of browsing on the command line for a Puck._tcp service:

C:\Users\xxxx>dns-sd -B _puck._tcp
Browsing for _puck._tcp

Timestamp	*A/R*	*Flags*	*if*	*Domain*	*Service Type*	*Instance Name*
10:26:40.493	*Add*	*2*	*11*	*local.*	*_puck._tcp.*	*MBARI_test_*

To demonstrate this project functionality, Luminary PUCK host was connected to the PC. Applying the mentioned modifications to (Mihai Toma, 2010) project, in order to make it fully compliant with PUCK v1.4. The serial port used on the compliance tool v1.3 is translated to a TCP connection as explained above. And then the PUCK TCP service is discovered by the software using Zeroconf firmware, which allows to start the test to the server chosen by the user.



There are several different parameters that can be passed to the compliance tool in order to change its behavior, such as serial port, log file name, softbreak tries, etc. It can be done through the command line, when the launch command is called, or with a configuration file. After launching the compliance tool it follows the flowchart presented in figure 8.

Figure 8 Compliance tool flowchart

It is a command line user interface so after running all the tests it will display how many tests have been successful and how many have failed, this way it is possible to detect if the connected instrument is PUCK compliant.

On the next chapter are shown the results of one test and what should be expected after running all the test mentioned on figure 8.

RESULTS

Automatic PUCK compliance test

Using the compliance tool and the IP PUCK protocol with Zeroconf for the Ethernet instrument implementation on the Luminary micro, the IP PUCK compliance verification tool was achieved. The Luminary development kit was used as IP instrument, and a laptop for running the compliance tool on a Linux OS

To test the implementation, the IP instrument was connected to the compliance tool, running on a Ubuntu OS virtual machine, through Ethernet connection, figure 9. This compliance tool was executed without any special parameter, such as port or IP. This way all the necessary connection constants were auto discovered by the Zeroconf firmware.

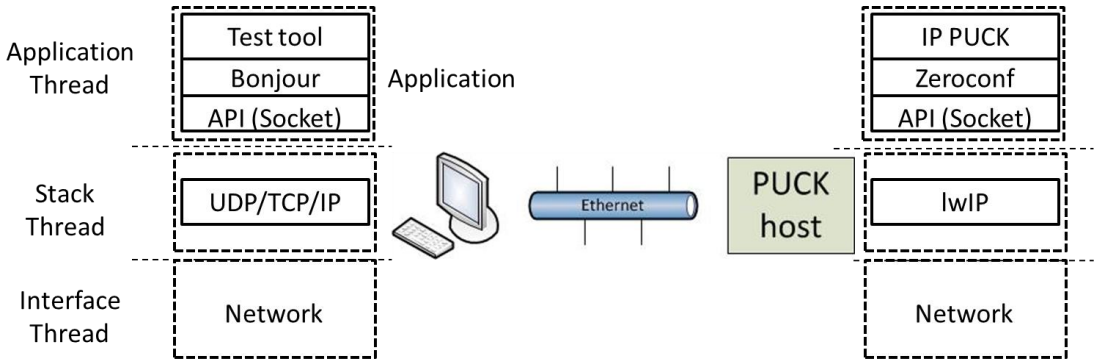


Figure 9 Compliance tool and IP PUCK connection

When the compliance tool starts, it prompts the user with the Zeroconf discovered services on the network. Then the user is asked to select one device on the list, this way the compliance tool can detect if the service is properly discovered on the network as it should be.

Selecting the desired service to test, "bonjour browser" discovers its port and IP to establish TCP/IP connection, if all this information is retrieved properly, the first test is considered passed. If some of these fields are wrong then the connection cannot be done and the program aborts with error message. Here is attached the

compliance tool received information on the command line, with the Zeroconf test details.

INFO - Using TCP communication xxx.xxx.xx.xx:xxxx
INFO - Check options complete OK
INFO - init Keyboard OK
INFO - Zeroconf Test
INFO - #server interfaceIndex, name, type, domain
INFO - ADD 0 2 MBARI_test_._puck._tcp.local.
INFO - Introduce the #server of the device to test:
INFO - 0 selected
INFO - MBARI_test_._puck._tcp.local.
INFO - RESOLVE: MBARI_test_._puck._tcp.local. is at MBARI_test_.local.:1541
INFO - PUCK IP 134.89.12.252
INFO - Zeroconf Test passed
INFO - Connection open to IP:134.89.12.252 and port:1541 OK

Once Zeroconf test is passed and the TCP connection done is performed the command test. If compliance tool is executed with all the log information displayed can be seen how the command test executes all the PUCK functions that do not affect memory writings or readings. The expected output should be like that:

INFO - Command Test
DEBUG - setPuckMode()
DEBUG - setPuckMode() - already in PUCK mode!
DEBUG - cmdTest(): now in PUCK mode
DEBUG - PUCKTY returned type 0
DEBUG - cmdTest: response to PUCKTY: 0002
DEBUG - cmdTest: response to PUCKVR: MBARI PUCK REV 1.4
DEBUG - Got expected response to bogus command
INFO - Command Test Passed

Then is performed exclusive access test, which has to result in a failed connection due to PUCK only allows one connection per time as the stack defines. The next text shows the results obtained by the compliance tool, and as it was expected the PUCK host only allows one connection, so this test is passed too:

INFO - Exclusive PUCK host access Test
It can take up to one minute...
DEBUG - Can't open more than 1 TCP connections to PUCK host (null).
INFO - Exclusive PUCK host access Passed

After that, following the program flowchart presented previously, compliance tool executes memory test. With the DEBUG log option can be seen how it is writing and reading the memory in order to checkout if its fully compliant with IP PUCK test specifications:

DEBUG - Backup memory done
INFO - Memory backup on file
INFO - Memory Test
DEBUG - Attempt up to 1 PUCK softbreaks
DEBUG - setPuckMode()
DEBUG - setPuckMode() - already in PUCK mode!
DEBUG - memTest - PUCKSZ returned 261
DEBUG - writeChunk(), got 20 from getPuckResponse()
INFO - Walking 1's memory test
DEBUG - testMem() startAddr=96, endAddr=165
DEBUG - Writing 21 bytes to address 96
DEBUG - Writing 31 bytes to address 117
DEBUG - Writing 17 bytes to address 148
INFO - Walking 1's memory test Passed
INFO - Walking 0's memory test
DEBUG - testMem() startAddr=96, endAddr=165
DEBUG - Writing 32 bytes to address 96
DEBUG - Writing 31 bytes to address 128
DEBUG - Writing 6 bytes to address 159
INFO - Walking 0's memory test Passed
INFO - Functional memory test
DEBUG - testMem() startAddr=96, endAddr=165
DEBUG - Writing 22 bytes to address 96
DEBUG - Writing 32 bytes to address 118
DEBUG - Writing 15 bytes to address 150
INFO - Functional memory test Passed
INFO - Memory Test Passed
DEBUG - Attempt up to 3 PUCK softbreaks
DEBUG - setPuckMode()
DEBUG - setPuckMode() - already in PUCK mode!
DEBUG - Erase PUCK memory...
DEBUG - Restore memory done
DEBUG - Attempt up to 3 PUCK softbreaks
DEBUG - setPuckMode()
DEBUG - setPuckMode() - already in PUCK mode!
INFO - Datasheet memory test Passed
DEBUG - Compare Datahseet data OK
DEBUG - Attempt up to 3 PUCK softbreaks
DEBUG - setPuckMode()
DEBUG - setPuckMode() - already in PUCK mode!

DEBUG - Erase PUCK memory...
DEBUG - Restore memory done
INFO - Data restored on PUCK memory

After testing all the functionalities specified for IP PUCK firmware, compliance tool proceeds to check if PUCK host is still reachable, by sending a PUCK null command and waiting PUCKRDY response. After that no more commands are sent during two minutes, waiting for the PUCK timeout functionality. This was explained on Materials and Methods chapter, and is one of the new functionalities that has been programmed during this internship to provide full IP PUCK compliance.

On the screen should be displayed a simple timer, decreasing each 5 seconds, this way the user does not think that the program is blocked. After the two minutes is displayed that the test is passed, which means that the PUCKTMO frame has been received:

INFO - PUCK timeout test
DEBUG - setPuckMode()
DEBUG - setPuckMode() - already in PUCK mode!
INFO - 120 seconds before PUCK mode timeout
INFO - 115 seconds before PUCK mode timeout
INFO - 110 seconds before PUCK mode timeout
INFO - 105 seconds before PUCK mode timeout
INFO - 100 seconds before PUCK mode timeout
.
.
.
INFO - 10 seconds before PUCK mode timeout
INFO - 5 seconds before PUCK mode timeout
INFO - PUCK timeout test Passed

Now the PUCK TCP connection is closed by the PUCK host, so is proceeded to test the native instrument port connection and then is displayed the information of the passed and failed tests. In this case, the IP PUCK instrument was fully IP PUCK v1.4 compliant, so no errors were detected as can be seen on the command line:

INFO - Instrument connection test
INFO - Connection open to IP:134.89.12.252 and port:8760 OK
INFO - Instrument connection test Passed
INFO - puckVerify: 8 tests passed, 0 tests failed
return value=0

Following this steps, an IP PUCK instrument can be verified if it is completely compliant with the defined stack. If an error occurs during the test, it is prompted with an ERROR log message, independently of the chosen log mode, and it causes a return value different than 0, which means that the connected instrument is not fully compliant, on the command line will be prompted the failed test and the reason of this failure

CONCLUSIONS/RECOMMENDATIONS

An IP PUCK v1.4 instrument has been developed on a Luminary evaluation kit board. It was built on a previous (Mihai Toma, 2010) project, and after some modifications have completed IP PUCK development on a low power instrument platform.

Also has been designed a compliance tool capable of verificaton of the connected instrument as IP PUCK v1.4 compliant. This provides a reliable compliance tool for the manufacturers who need to provide PUCK capabilities to its instruments.

For future work it would be a good approach to unify RS232 PUCK compliance tool with IP PUCK compliance tool. This way the user can select which kind of instrument is going to be tested and the compliance tool could modify its behavior to perform the desired verification.

ACKNOWLEDGEMENTS

I am grateful to Kent Headley, Duane Edgington, Daniel Mihai Toma, Joaquín del Río, Antoni Manuel, and especially Tom O´Reilly, whose encouragement, supervision and support for the duration of the program enabled me to develop the project.

I would also like to offer my thanks to my coordinators, George Matsumoto and Linda Kuhnz. Without them this research would not have been possible, and their tolerance, good humor, and insight added much.

I am grateful to the many interns who participated in the MBARI 2013 summer internship.

Lastly, I offer my regards to all of those who supported me in any respect during the completion of the project, and especially to David and Lucile Packard foundation, which makes this internship program possible.

References:

del Rio, J., Auffret, Y., Daniel, T. M., Shahram, S., André, X., Stéphane, B., et al. (2009). Smart sensor interface for sea bottom observatories. *Instrumentation Viewpoint*, 96-98.

K., W., & Edward, N. (2009). Coupling Wireless Sensor Networks and the Sensor Observation Service. *Bridging the Interoperability Gap*.

Mihai Toma, D. (2010). Interoperable Marine Monitoring System. *MBARI internship*.

O´Reilly, T. (2009). *MBARI*. Retrieved from http://www.mbari.org/pw/puck.htm

O'Reilly, T. (2011). OGC® PUCK Protocol Standard Version 1.4. *Open Geospatial Consortium*.

Sadasivan, S. (2010). *ARM* . Retrieved 2010, from http://www.arm.com/files/pdf/IntroToCortex-M3.pdf

Song, E., & Lee, K. (2007). Smart Transducer Web Services Based on the IEEE 1451.0 Standard. *Sensor Systems*.

Williams, A. (2002). *Zeroconf*. Retrieved from Zero Configuration Networking: http://files.zeroconf.org/draft-ietf-zeroconf-reqts-12.txt

Annex I

Work time

The implementation of the IP PUCK v1.4 compliant on the Luminary microcontroller has been done in approximately 2.5 weeks

The implementation of the compliance tool v1.4 has been done in approximately 4 weeks.

Annex II

Workbench setup

PUCK Host:

1. Plug Luminary board to the laptop through ICDI USB.
2. Code Sourcery project on “PUCKcode” folder. To open it on the Code Sourcery environment with the Luminary libraries, select this folder as workspace on the prompted window after launching the program.
3. This workspace will auto configure the desired board. In this case LM3S9B96.
4. The board can be programmed with the “Run” button on Code Sourcery, or directly writing the .bin file with the “LM flash programmer”, which is a free tool provided on www.ti.com/stellaris

PUCK verification tool:

1. On the folder “PUCK_v14/src” there is the “Make file project” in order to compile it under a POSIX platform
2. It can be launched with the command “./puckVerify –p [instrument port]”