Development and Design of Firmware Programming Tools for the openHPSDR Hardware

Dave Larsen, KV0S
openHPSDR Development Group
Columbia, Missouri, USA
openhpsdr.org
kv0s.dave@gmail.com

Abstract

Over the past 10 years the High Performance Software Defined Radio project has moved from the early attempt of producing an audio interface and a USB interface to a sophisticated single board single FPGA (Field Programmable Gate Array). In this process the communication protocol and the software tools to manage that process have changed several times. Some of the reason things are organized in certain ways are an artifact of the history of the changes. This paper will chronicaly these changes and some of the resulting software that allow the loading and replacement of the firmware in the radio boards.

Introduction

This paper presents the design rational and motivation behind the firmware programming tools used by the openHPSDR Development Group (openhpsdr.org). To understand the need for the programming tools, you need to know some history of the project.

I started to participate in the HPSDR group in 2007. At that time the group had already developed the Atlas, which is a passive backplane board, Ozymandias an FPGA based interface controller card that provides the input and output connections to the real world, and Janus a dual, full duplex, A/D and D/A converter board, which is a board that could be used as a sound card interface to the FlexRadio SDR1000. At this point in the project, custom firmware programming tools were not need as the only board with an FPGA (Field Programmable Gate Array) was the Ozymandias board (Altera Cyclone II FPGA), which would read and reload its firmware on every startup from a file on the client computer.

In 2008 at the DCC in Chicago, Scotty, WA2DFI asked me to manage the HPSDR web site. Dale, AE5K had started the web site in 2006 and had done a nice job setting up the structure. Since our group is not formally organized other than we are interested in the development of Software Designed Radio (SDR) for the Amateur Radio community. I decided to maintain most of the original structure and keep it as a documentation of the development of the projects that the group worked on over the years. Many Hams find this makes the site confusing as they are use to an organization that is trying to sell them a new product, older parts disappear from the site. This site endeavors to keep all the ideas posted and even the deadend ones remain as someone might go and pick up one of these ideas again for further development. Also we have a Wiki system that was maintained, not by me, but the original author and many people started to develop interesting projects but then lose interest in the continued development before the project completion. My preference is to document past history of the project developments.

In 2007-2008 the group focused on a transmit board called Penelope, which is a digital up converter (DUC) a 1/2-watt transmitter/exciter board. The developers were all in a learning curve and most of us were at different points in a learning curve. At this point, FPGAs were very reasonably priced and

added tremendous flexibility to the development process. The Penelope used an Altera Cyclone II like the Ozymandias board. This started the need to develop firmware for the individual component board that was not directly connected to a computer interface. We used the Altera software to create and load the firmware into the component boards directly to pin connections on each individual component board.

Then from 2008-2010 we developed the Mercury board, a 0-65MHz Direct Sampling Receiver with a Altera Cyclone III FPGA. This board started the slow progression of changing hardware in the FPGA components. We would use the Altera software with a USB Blaster connected to a pin connector on the component board. The boards generally need a change of jumper pins to accept the firmware and a different pin setting to run the board. While the developer group was quite adept at programming boards, for the occasional user it was a daunting task. I have met many openHPSDR board owners that have not changed the original firmware shipped in the boards.

So now we had a working transceiver and we found out the the boards were much more capable than we expected and the USB2 technology was one of the limiting factors. At this point, USB3 had been invented but was years from being widely available. Additionally, the USB technology had several quirks and the driver coding was quite complicated to write from the open source point of view.

At this point in time, Ethernet 10/100 baseT technology was in most computers and 1000 base T was available. It was a stable and well known technology. The 100k bandwidth could provide comparable bandwidth to the USB2 we were using. This led to the development of the Metis board, which provides either a 100 baseT or Gigabit (1000 baseT) interface to the host PC. It was designed so that the boards and the computer could function with either the Ethernet interface board or the USB interface board. Additionally the interface could function as the radio was directly wired to the computer with a standard Ethernet cable or run through a router or switch.

With the advent of the Ethernet board Metis, and comtemplating the differences between the USB2 protocol and the Ethernet protocol, several interesting features were included with the Metis board.

First the FPGA on the Penelope TX board and the Mercury RX board had the capability of being programmed with the JTAG protocol. Up to this point, we used the Altera software to connect to a JTAG connector or to the flash memory interface. We realized that the FPGA could be programmed to function as a JTAG programmer. This still required the setting of jumper pins but no additional hardware was necessary to program the FPGAs on the component boards.

Also to use the firmware JTAG programmer, the code was stored in a different part of the Metis Board from the regular communication code. This bootloader code in Metis need to be separate from the communication code. This brought us to the point where there was a need to develop custom software to use these firmware programming features.

Bootloader-Programmer

The first effort at producing a custom programmer was by John, G0ORX. Because of the dicated USB cable between Ozymandis board and the computer, the board would pull the firmware from the computer at every power up.

In the Ethernet context, the connection is a shared link and there needs to be software to filter the Ethernet packets for the radio. There can be other devices on the subnet, so both ends of the connection need to provide filtering. Also in tcpip protocols in the radio needs that understand ip addressing and packet types. This turned on to be a considerable programming task in the in Verlog in FPGA. There are commercial proprietary libraries for this but that interfered with the projects commitment to open source code. In the openHPSDR project, communications code was programmed from scratch to abide with the open source approach to our code base.

There is a Ethernet protocol that requires less code that is a basic protocol. It is the “pcap” protocol, which uses MAC addressing instead of IP addressing. This is the same protocol used in the Wireshark program and because of the opportunity for this protocol to impact the code at each end of the connection the client computer must be run as administrator or root account.

Additionally because the job of the Bootloader-Programmer is to erase or insert new code in the FPGA, it is possible to complete the code erase without the accompanying replacement. A simple bit of firmware was created that speaks to the a Bootloader-Programmer in “pcap” protocol. Bootloader firmware is loaded at manufacturing and testing on the board and should never need to be replaced. This provides a way for one to recover from hardware or operator errors that erase firmware. It is possible to erase the bootloader firmware but not with the openHPSDR programs. Only by using the original Altera software.

The original code would allow the user to replace the Metis board firmware or by using the bootloader firmware as a JTAG programmer to program the firmware in other component Altas bus boards. This code is fully functional and could replace all firmware that a normal user should need to replace. This programmer was coded in C++ programming language using the QT graphics library for the GUI (Graphical User Interface). The big disadvantages were that it used “pcap” Ethernet protocol requiring the Bootloader-Programmer to be run as administrator or root user, it also required the bootloader jumper to be set on the board to run the bootloader firmware, and thirdly the programmer worked differently if programming the connected board or using the connected board as a JTAG to program a secondary board (See Figure 1). Also we discovered a quirk that the board to be programmed with JTAG worked best if the Metis board was in Atlas slot furtherest from the power supply and the board to be programmed with JTAG protocol in the slot next to the Metis board. This was due to a quirk in how the Atlas bus was wired in the PCB long before this procedure was imagined.

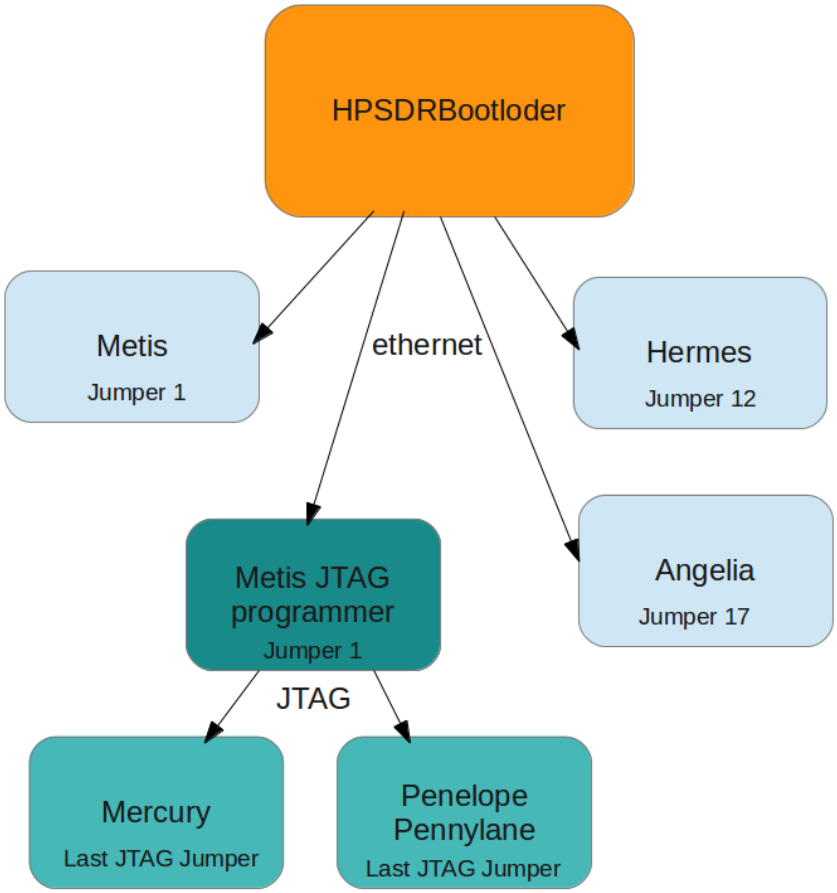


Figure 1 is a diagram of how the HPSDRBootloader access the component boards of the HPSDR system.

We also found that some occasional users found the work flow confusing mainly because they did not clearly understand the information described above. This lead to the point where I started working on the firmware programming code. John G0ORX is a very energetic programmer and had moved on to other problems as the Bootloader-Programmer was stable from his point of view. At work, I had been using C++ and Qt for the previous 10 years, so I volunteered to take on this task.

Spliting the G0ORX code

The first idea was to split the Bootloader-Programmer into two programs both to work more or less like the original but with a cleaner GUI from a user perspective. This program would keep using the pcap library and the requirements to run as administrator or root user, set the bootloader jumper. It provided information on if the expected board was found and if it was programmed successfully. This program was labeled the HPSDRBootloader program. And was intended to recover from situations with firmware

corruption issues. A second program was defined that only programmed Ethernet connected boards and the was called the HPSDRProgrammer(See Figure 3).

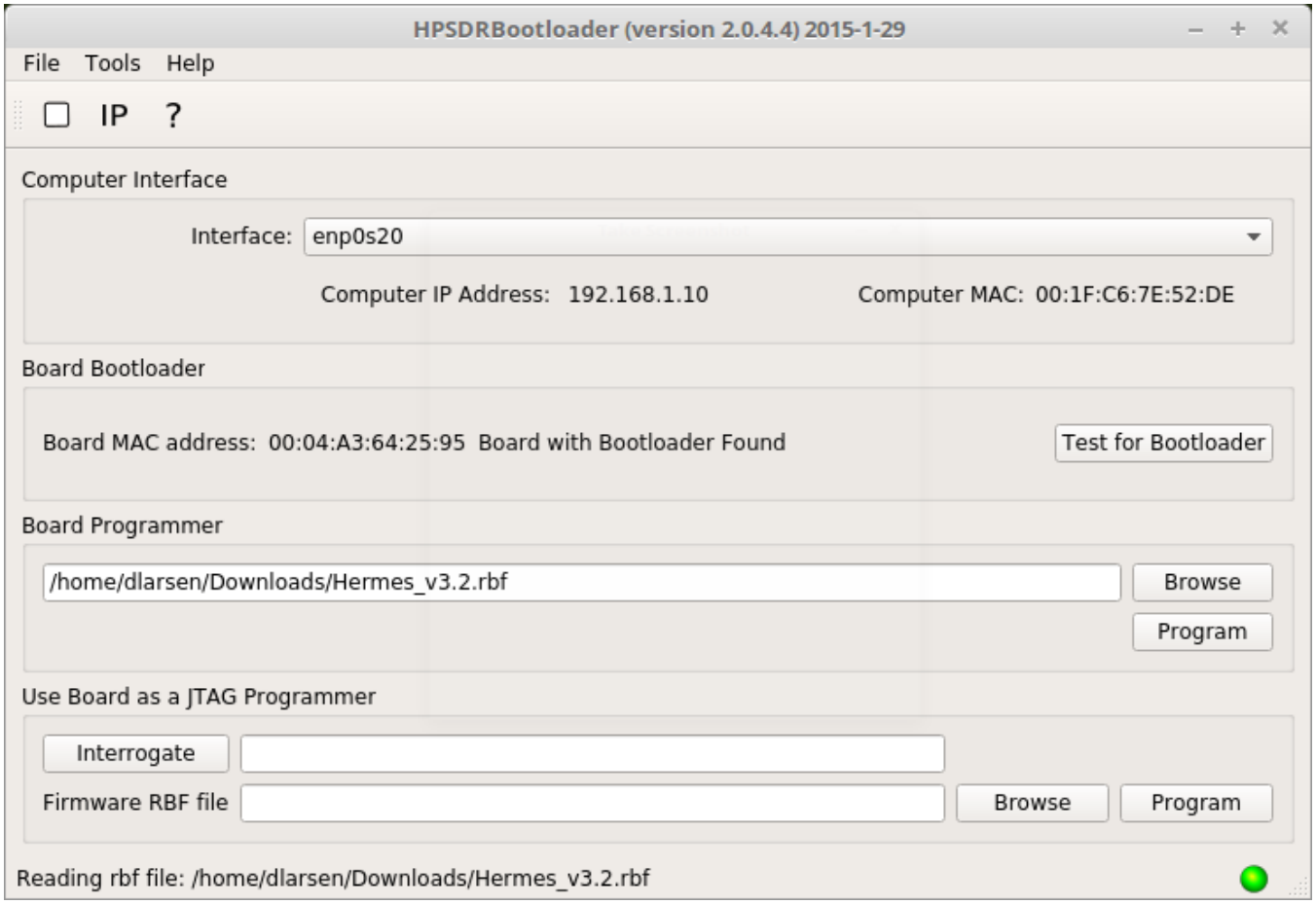


Figure 2 is a diagram of the HPSDRBootloader program after Board discovery and prepared to program the Hermes board..

HPSDRProgrammer

The first HPSDRProgrammer was designed to only changed the firmware in the connected board. This version still used pcap but only communicated to the boards directly connect to the client computer. This was intended to be an interim version as I was reorganizing the code.

HPSDRProgrammer_V2

During this time the Hermes board was released. Most users of the single board radios want to put them in a box and this made accessing jumper pins much less convenient. Also all the functions were fit into a single larger FPGA, eliminating the need for a software JTAG programmer. The HPSDRProgrammer_V2 used a different Ethernet protocol UDP to communicate with the radio board. This is the same Ethernet Protocol used by the radio communication software uses, so it was already in the firmware of the FPGA. It did **not** need to be run as administrator or root user, it did not have to change bootloader jumper and it did not use the pcap library. The HPSDRProgrammer_V2 depends that the current version of the firmware has the code to talk to the HPSDRProgrammer_V2, which works well until there is a programmer failure or the user loads very old firmware. This software became popular with the advent of the single board radios starting with the Hermes board (See Figure 3).

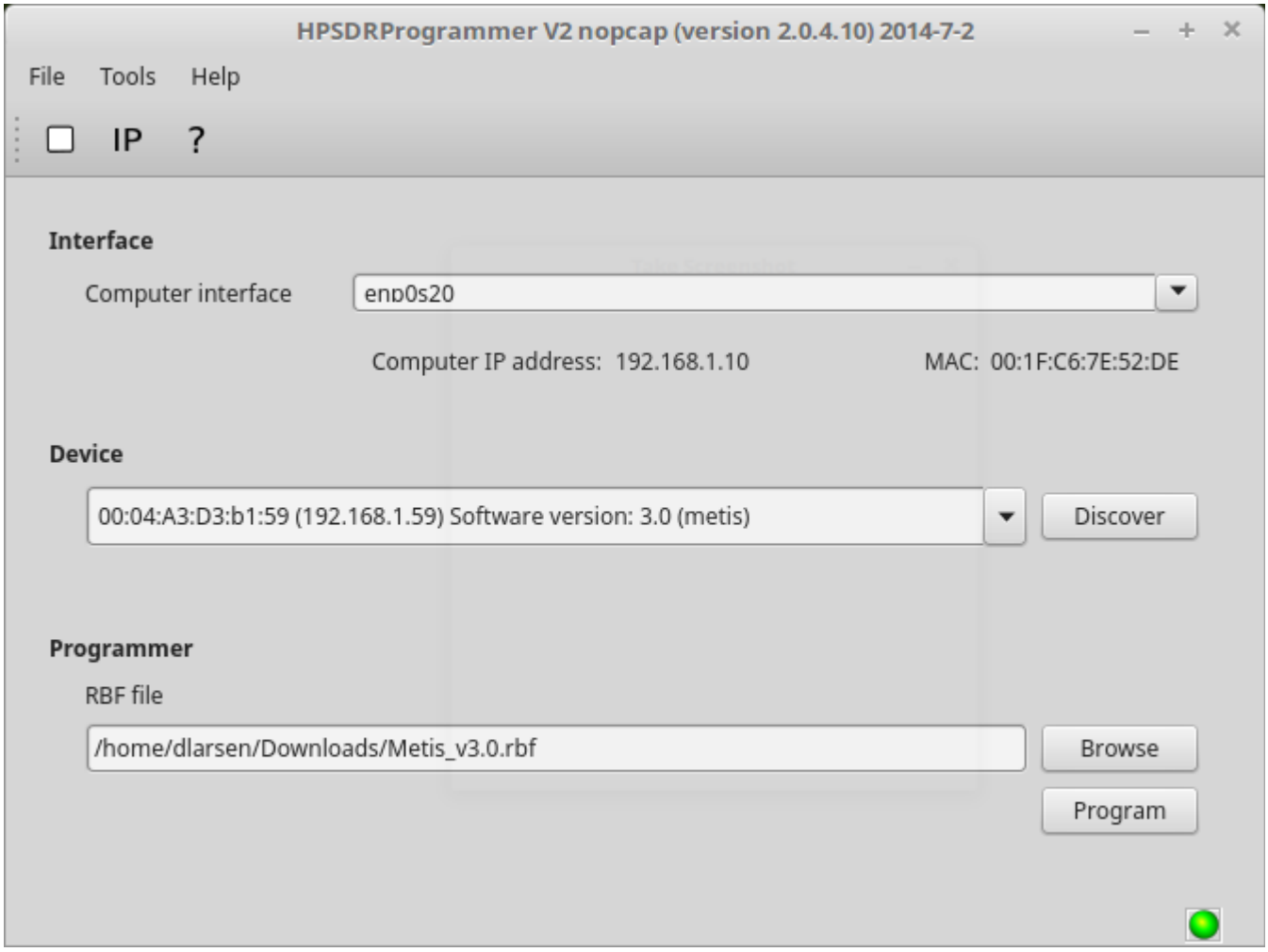


Figure 3 is a example of the HPSDRProgrammer_V2 program with the metis board discovered and ready to load the firmware into the board.

A new way of implementing HPSDR radio protocol

Over the years, there have been several nagging issues that seem to be corner cases for the majority of users but they are very important to the individuals that report thse cases. Most of these were hard or impossible to fix due the the lack of information on what was actually happening with the transferred packets. The previous software was not helpful as producing feedback. So most user could not provide and information that was useful in debugging their problems.

This situation brought about the need to upgrade the protocol starting in 2015, this new protocol is now labeled protocol2 in the code repository. The first protocol, labeled protocol1, was a simple statement of the information being sent back and forth and was in part based on the older USB protocol. Since the starting point on the computer client software was a version of PowerSDR shared by FlexRadio 10 years ago over the years the software used by openHPSDR and FlexRadio have diverged.

In protocol2 the protocol has been vetted over a number of years by many people writing computer client software. It attempts to better use the bandwidth in a way the client program can use. And provide debugging information of the current status of the communication process.

As for my own interest, I had taken a 2 year break from working on the HPSDRProgrammer as the code was stable. As you can imagine with the sequence of revisions programmer code accumulates unused bits of code that are either bypassed or saved as they might be useful later. I wanted to write a clean new set of code the I fully understand as I would write it all myself. Writing this code is not my job, it is my hobby, so I want to have a clear idea of the process in my coding style. As the protocol2 was evolving at the time, I decided to start with the protocol1. Also since I am Linux programmer, I thought I would start with a command line program for simplicity and building the complete programmer with the least amount of code. It is important to know that about 30% of the older code is necessary to program the radio board the other 70% of the code is to communication the the program user.

HPSDRProgrammer in a new language

Also at the time, I had learned a new programming language at work, so I thought I would apply it to this problem. With the C++ code, I was frustrated with my ability to test the bits of code out of context of the program. The new language I decided to use is Go (often referred to as golang) (golang.org). Go is a open source language started at Google in 2007 since 2012 is has stable and quite usable. Go had a good linage for a starting place. Ken Thompson (Plan9 OS, The B programming language a predecessor of the C programming language) and Rob Pike (worked on the practice of “Programming and the Unix operating system manual”, and Plan9 with Ken Thompson) both work for many years a Bell Labs and both developed UTF-8 and were latter hired to work at Google. Robert Griesemer the third co-creator that worked on database structures. After producing a stable Linux implementation they open-sourced the code and since then over half of the code developments have come from a dedicated open-source community.

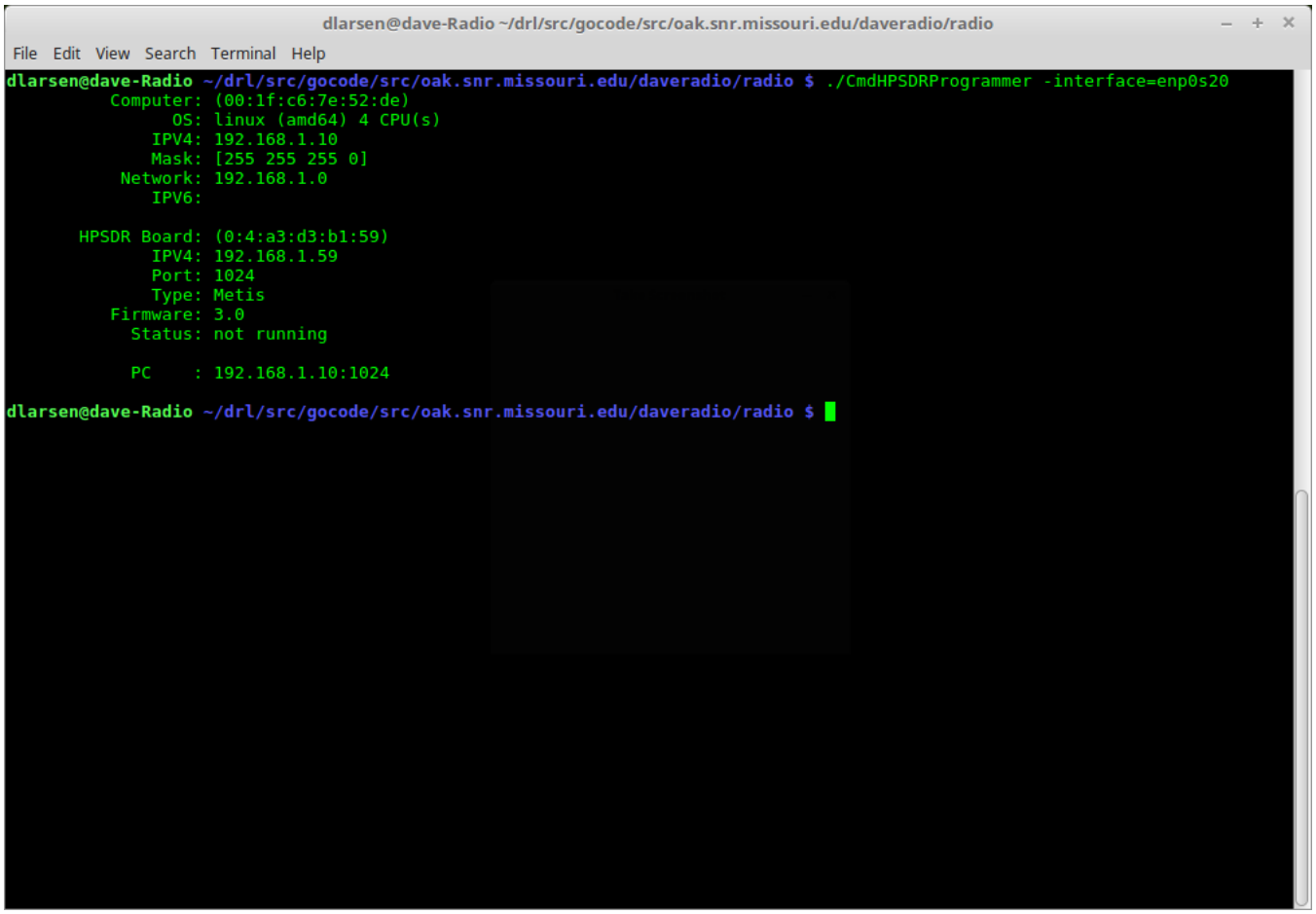
Go, while a robust language in most ways, has no GUI in the normal sense. It does have a very robust networking package include normal tcpip, pcap, and support for making web severs. It also has a robust testing environment. So that you can be comfortable that each subcomponent is working correctly and as new releases of the language come along you can easily retest your code.

I first wrote a set of functions to perform all the basic steps in the programming process. With these functions, I built tests for each. Also, I added a debug mode that can be turned on by the user, that allows the code to report all bytes sent to the radio and all bytes returned. Additionally, these bytes can be displayed as decimal or hex numbers. This feature was quite helpful as it allow me to find an obscure error the the new code.

Go also can be compile on most of the common operating systems and some not so common operating systems. See the golang.org download web site for a list of available operating systems. Additionally the executable code produced by the compilers is statically linked so the program is a single executable file and reducing the need for program installers. Each operating system has a different approach to dealing with program installers and this make producing and distributing cross platform code much easier. Also the compiler system is moving to make the cross platform compiler much more robust.

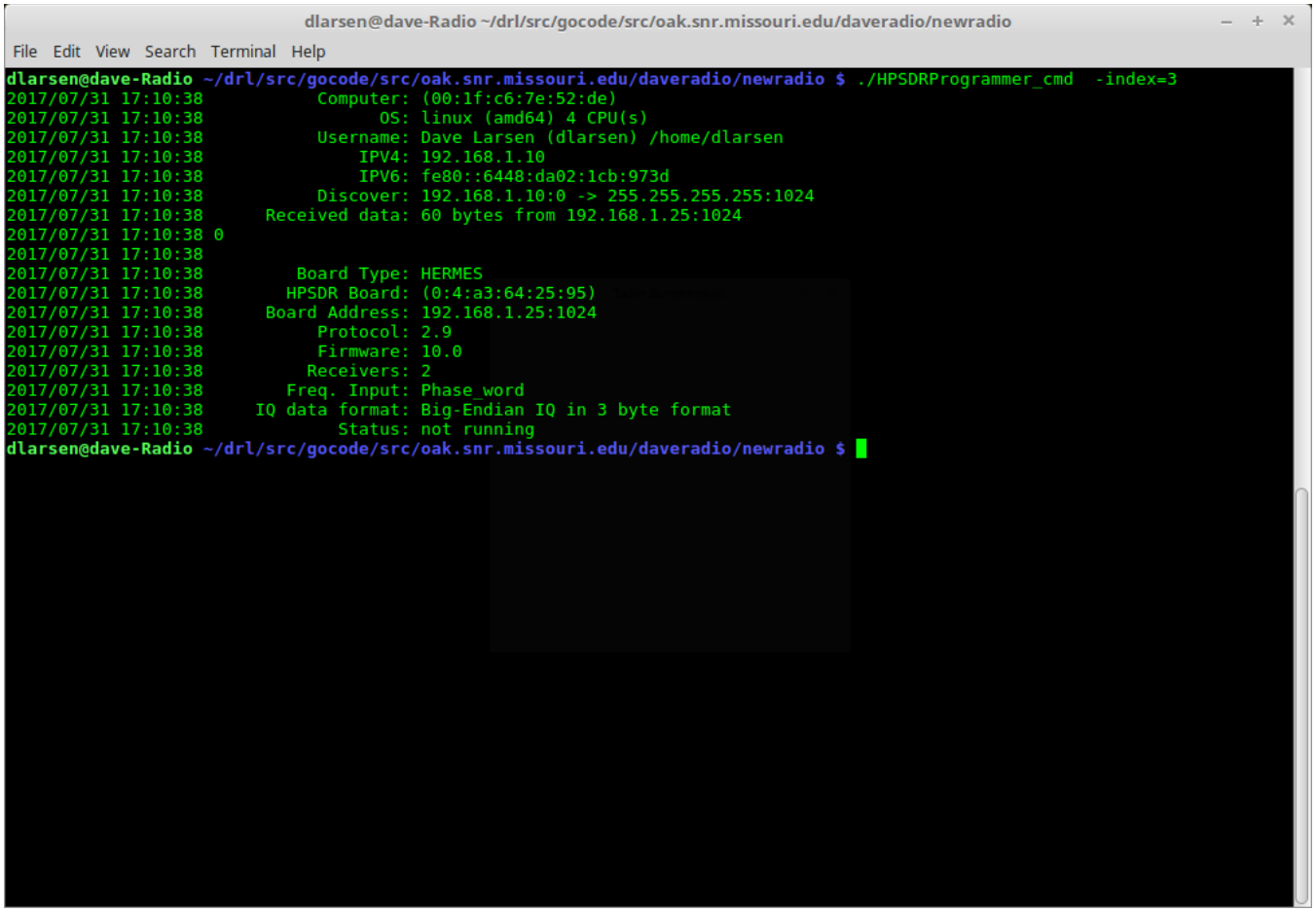
The disadvantage is that since the advent of computer GUIs, people have become less familiar with command line programs especially on windows.

All this being said I moved forward with a set of protocol1 functions to detect, erase, program and set the IP address on openHPSDR radios. The program was very fast and gave feedback of the status of the process (see Figure 4, for a screen capture of the program output).



Figures 4 is the output of the Protocol 1 command line programmer after discovery of a Metis board.

Once I completed the protocol1, I started the protocol2 Programmer as the protocols have several differences from the protocol1 code was working, I developed the new code as separate program. When I was working on the early version of the code the firmware was very rudimentary. I wrote the code to the protocol specifications using the testing functions so I was reasonably sure the code was working correctly. Then after the firmware was available it was very easy to get the last details working correctly (see Figure 5).



Figures 5 is the output of the Protocol 2 command line programmer after discovery of a Hermes board.

Now I had functions that completed the HPSDRProgramming task correctly for both protocols and I shared it with other project developers. It became clear, I needed and interface for a wider group of users. Go has builtin libraries that make building web servers almost trivial. So I decide to make the HPSDRProgrammer program with a web server interface. The user would start the program and control the tasks with a web browser. This became a good choice as most everybody knows how to use a web browser. I also made the program with the web pages compiled into the single executable file so that it was the only file needed to perform the programming process (see Figure 6).

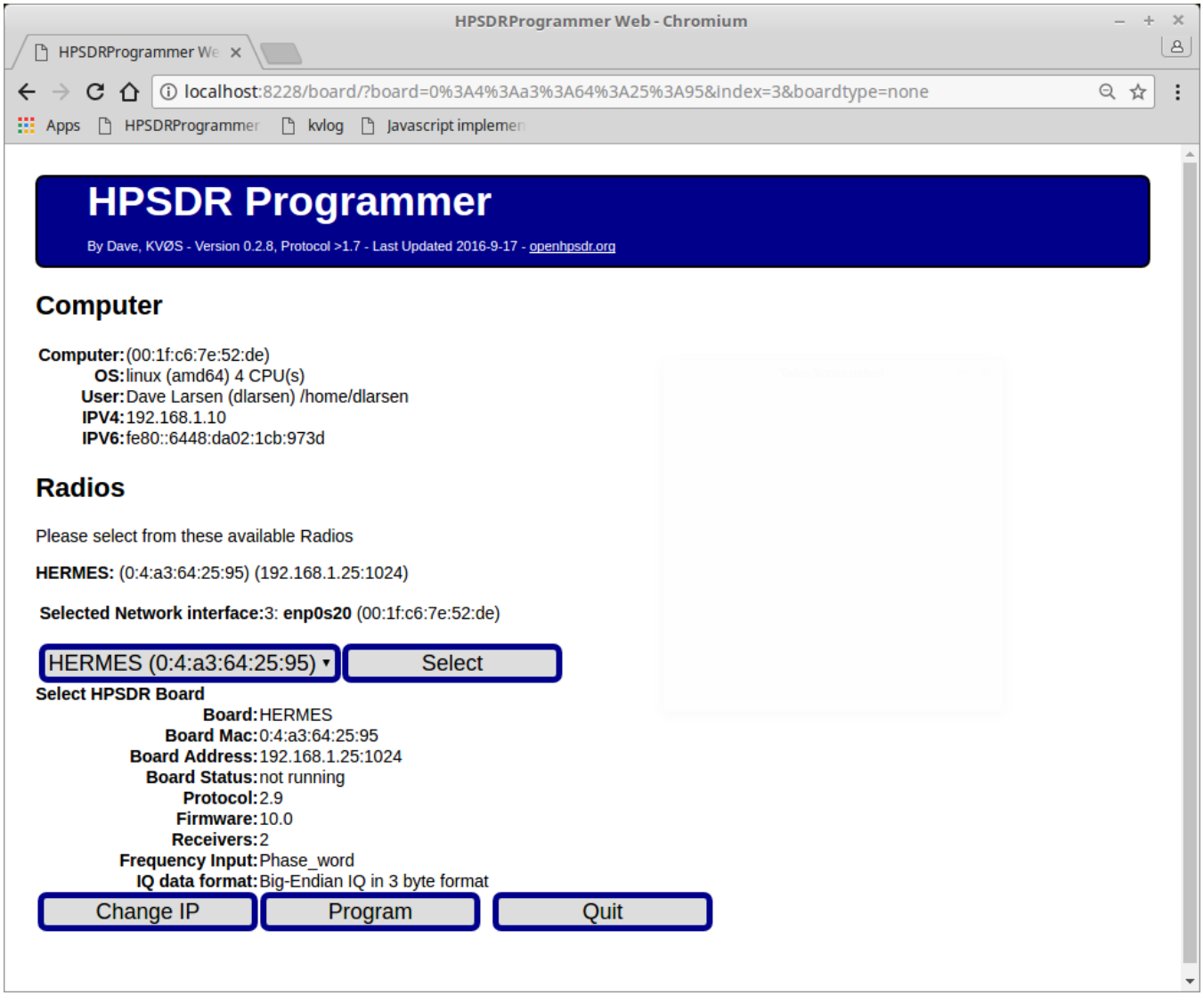


Figure 6 is a Diagram of how the HPSDRProgrammer web interface after discovery of a Hermes board.

One of the nice features of using a web browser as the interface to a program is the web browser can also access help files or a manual on the programs operation. Most of the issues that made the program difficult to implement were related to limits web browsers have on their ability to interact with the client computer. The limits are for the safety of the client computer when using the Internet but made it more difficult to access the radio firmware files and to let the user know the status of the programming process.

Conclusion

The development of the openHPSDR radios was a community process that focused on the development of very high quality radio systems. This allowed Amateur Radio operators the access to these

technologies in a open-source environment. It is a collection of volunteers that do this work as an extension of their Amateur Radio hobby.

My development of the HPSDRProgrammer programs is one small part of the project. There are many projects that can be developed in this project. You do not have to be an expert in the whole system to contribute. It is very satisfying to be able to say that in some way I help create parts of this project. If you would like to be a part of the process, come and join us.

References

The Go Programming Language, (golang.org) is an open source programming language started in 2007 by Ken Thompson, Rob Pike, and Robert Griesemer all who worked at Google. The project has been open sourced and there are now Go programming teams at most corporations including Microsoft and Intel.

C++ with Qt (https://en.wikipedia.org/wiki/Qt_(software)) The original GUI programs were written in C++ with Qt.

HPSDR USB protocol (http://openhpsdr.org/support/Ozy/USB_protocol_V1.57.pdf)

Protocol1 (Often called the original Ethernet or Metis Protocol) (http://svn.tapr.org/repos_sdr_hpsdr/trunk/Metis/Documentation/Metis-%20How%20it%20works_V1.33.pdf)

Protocol2 (Often called the new protocol) (https://github.com/TAPR/OpenHPSDR-Firmware/tree/master/Protocol2/Documentation)

The openHPSDR project web site (http://openhpsdr.org/) the web site of material on the original openHPSDR projects.

Protocol2 resources web site (http://openhpsdr.org/beta/). This page is used by those beta testing the protocol2.

OpenHPSDR repository (https://github.com/TAPR/) the projects are organized by the authors of the code. The SVN on this page is a historical copy to the old SVN in March 2017.