GNU Radio

Ke-Yu, Chen (8818-0493), Zhi-Feng Chen (1218-1197)
Dept. of Electrical Computer Engineering
University of Florida
Gainesville, Florida

ABSTRACT

***GNU radio* is a free/open-source software toolkit for building software radios, in which software defines the transmitted waveforms and demodulates the received waveforms. Software radio is the technique of getting code as close to the antenna as possible. It turns radio hardware problems into software problems. GNU Radio provides functions to support implementing spectrum analyzer, an oscilloscope, concurrent multichannel receiver and an ever-growing collection of modulators and demodulators. In this academic semester, we will build the environment of GNU radio, be familiar with it, and learn how to use the existing libraries to transmit a jpeg file with Differential Binary Phase Shift Keying (DBPSK)/ Differential Quadrature Phase Shift Keying (DQPSK) modulation between two end systems by TCP connection. The future objective of this project is to build the video/audio transmission over wireless network, including error correction (CRC) and QoS (Quality of Service) by using GNU Radio.**

Key words—, GNUradio, Python, , DBPSK/DQPSK, TCP socket

I. Introduction

The fundamental characteristic of software radio is that software defines the transmitted waveforms, and software demodulates the received waveforms. This is in contrast to most radios in which the processing is done with either analog circuitry or analog circuitry combined with digital chips. GNU Radio is a free software toolkit for building software radios.

Software radio is a revolution in radio design due to its ability to create radios that change on the fly, creating new choices for users. At the baseline, software radios can do pretty much anything a traditional radio can do. The exciting part is the flexibility that software provides you. Instead of a bunch of fixed function gadgets, in the next few years we'll see a move to universal communication devices. Imagine a device that can morph into a cell phone and get you connectivity using GPRS, 802.11 Wi-Fi, 802.16 WiMax, a satellite hookup or the emerging standard of the day. You could determine your location using GPS, GLONASS or both.

Perhaps most exciting of all is the potential to build decentralized communication systems. If you look at today's systems, the vast majority is infrastructure-based. Broadcast radio and TV provide a one-way channel, are tightly regulated and the content is controlled by a handful of organizations. Cell phones are a great convenience, but the features your phone supports are determined by the operator's interests, not yours. A centralized system limits the rate of innovation. Instead of cell phones being second-class citizens, usable only if infrastructure is in place and limited to the capabilities determined worthwhile by the operator, we could build smarter devices. These user-owned devices would generate the network. They'd create a mesh among themselves, negotiate for backhaul and be free to evolve new solutions, features and applications.

II. Description

A. *Basic Architecture in GNU Radio*

Figure 1 shows a typical block diagram for a software radio. To understand the software part of the radio, we first need to understand a bit about the associated hardware.

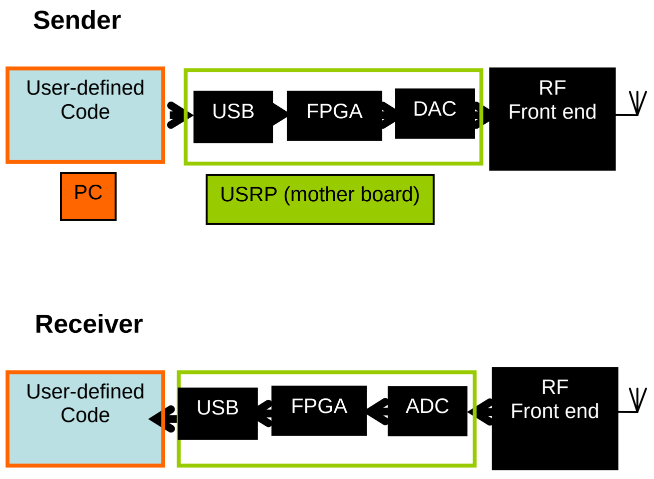

Figure 1. Architecture of GNU Radio

The USRP (Universal Software Radio Peripheral)[1], as described in figure 2, consists of a small motherboard containing up to four 12-bit 64M sample/sec ADCs, four 14-bit, 128M sample/sec DACs, a million gate-field programmable gate array (FPGA) and a programmable USB 2.0 controller. Each fully populated USRP motherboard supports four daughterboards, two for receive and two for transmit. RF front ends are implemented on the daughterboards. A variety of daughterboards is available to handle different frequency bands. For amateur radio use, low-power daughterboards are available that receive and transmit in the 440 MHz band and the 1.24 GHz band. A receive-only daughterboard based on a cable modem tuner is available that covers the range from 50 MHz to 800 MHz. Daughterboards are designed to be easy to prototype by hand in order to facilitate experimentation.

Examining the receive path in the figure, we see an antenna, a mysterious RF front end, an analog-to-digital converter (ADC) and a bunch of code. The analog-to-digital converter is the bridge between the physical world of continuous analog signals and the world of discrete digital samples manipulated by software.

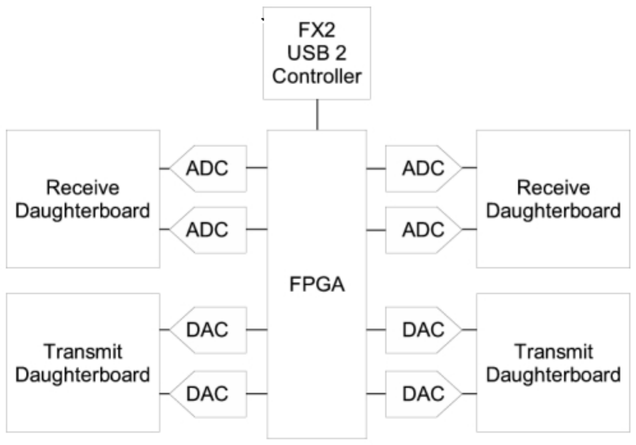

Figure 2. USRP (Universal Software Radio Peripheral)

ADCs have two primary characteristics, sampling rate and dynamic range. Sampling rate is the number of times per second that the ADC measures the analog signal. Dynamic range refers to the difference between the smallest and largest signal that can be distinguished; it's a function of the number of bits in the ADC's digital output and the design of the converter. For example, an 8-bit converter at most can represent 256 (2^8) signal levels, while a 16-bit converter represents up to 65,536 levels. Generally speaking, device physics and cost impose trade-offs between the sample rate and dynamic range.

Assuming we're dealing with low pass signals - signals where the bandwidth of interest goes from 0 to f_{MAX}, the Nyquist criterion states that our sampling frequency needs to be at least 2 * f_{MAX}. But if our ADC runs at 20 MHz, how can we listen to broadcast FM radio at 92.1 MHz? The answer is the RF front end. The receive RF front end translates a range of frequencies appearing at its input to a lower range at its output. For example, we could imagine an RF front end that translated the signals occurring in the 90 - 100 MHz range down to the 0 - 10 MHz range.

Mostly, we can treat the RF front end as a black box with a single control, the center of the input range that's to be translated. As a concrete example, a cable modem tuner module that we've employed successfully has the following characteristics. It translates a 6 MHz chunk of the spectrum centered between about 50 MHz and 800 MHz down to an output range centered at 5.75 MHz. The center frequency of the output range is called the intermediate frequency, or IF.

In the simplest-thing-that-possibly-could-work category, the RF front end may be eliminated altogether. One GNU Radio experimenter has listened to AM and shortwave broadcasts by connecting a 100-foot piece of wire directly to his 20M sample/sec ADC.

B. *Software development environment*

We choose Fedora Core 5 as our testing platform. RedHat decided stopping to develop the personal edition of Linux after RedHat 9 and transfer all the techniques of RedHat to the new project, called "Fedora Core". RedHat put some experimental packages used in enterprise edition into Fedora Core to test its stability. That is why Fedora Core still has some critical problems even though it has been developed into the 5th edition. We had spent much time to solve them.

Since Fedora still has so many "experimental packages" inside, why do we still choose this platform? The answer is very simple – sufficient and strong software support. Because many experienced RedHat users keep using Fedora Core, when any new problem occurs, it is more likely to find corresponding solutions on the website. There are also some useful forums to issue our problems. Designer or engineers can get quick responses usually in one day. These considerations are why we picked this platform.

C. *Co-work of C++ and Python*

There are two program languages used in GNU Radio, C++ and Python which play different roles in the whole system. All the signal processing and performance-critical blocks are written in C++ and Python is used to create a network or graph and glue these blocks together. So in this particular scenario, Python is a higher level language. Many useful and frequently used blocks have been provided by the GNU Radio project, so in many cases we don't need to touch C++, just using Python to finish your task. However, to do more sophisticated work, you have to use C++ to create your own block. In our demo program, because Python lacks of type conversion functions, we write a C++ block which is in charge of type converting between character and unsigned short.

There are two methods to using C++ modules in Python. The first one is to build the C++ into an executable file, and use Python built-in function *os.system()* to call this executable file, just like sending a command into the shell.

The second way is to use *SWIG*, the powerful tool in Python to 'glue' the C++ blocks to Python. SWIG generates the wrapper for C++ modules and generate the corresponding Python code (*.py) and library (*.so) so that we can include these classes and functions in Python. The details of what SWIG does and how to import C++ modules into Python is described in [5].

III. Evaluation and Demo

In the previous section, we know that GNU Radio with USRP has the ability to communicate with each other. Our objective is to transmit a JPEG file by using the modules in GNU Radio. Because lacking of the real hardware (USRP), we have to try to substitute it with other component.

The modified architecture is shown in figure 3. We first read a BMP file from hard disk, send it into a JPEG encoder implemented by C++ to get a compressed image file. After encoding, the transmitter read the JPEG file and starts the modulation. On the completion of modulation, the transmitter puts the bit stream into socket and sends it to the receiver.

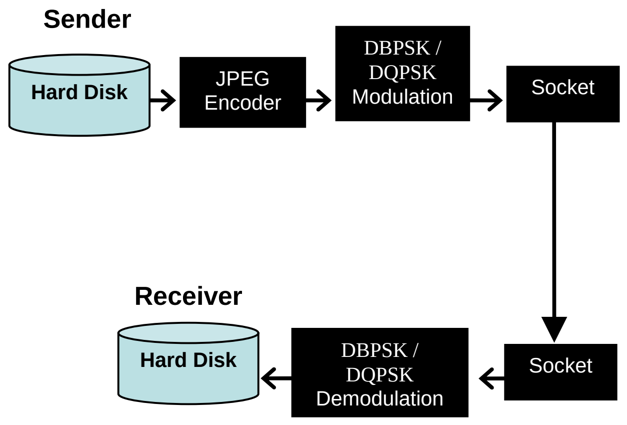

Figure 3. Architecture of our demo program

We add modulation and demodulation respectively in the transmitter and receiver in order to implement a practical wireless communication system, which use DBPSK/DQPSK as the modulation method. However, this setup needs the hardware support of Universal Software Radio Peripheral (USRP). In our demo, we will use TCP/IP socket to simulate the wireless transmission, which will be discussed in detail in the next section. So, without multiple the sinusoid carrier signals in our modulation, the final output is a baseband representation of the modulated signal, that is, the in-phase and quadrate (I/Q) signals. Receiver plays the role, the server, to listen the incoming bit stream and put the incoming bit stream into the next block 'demodulation' to restore the input data to the original JPEG file. All the works described above are done by software in PC, and we use socket to substitute USRP. We will describe the details in each block in the following paragraphs.

1) JPEG Encoder

JPEG Encoder reads a BMP file and compresses it into a JPEG file. The component is implemented in C++ and is called by the main program, written in Python. These two languages have different functions and there are two methods to coordinate these two languages. We will discuss it latter.

2) DBPSK/DQPSK Modulation and demodulation

The modulation and demodulation flow graph is show as below in figure 4

Modulation module

a) Get source input from JPEG file

To modulate the source file, we need to produce a binary stream and feed it into the modulation module. So, we first read the jpeg file, which is just produced above, as a byte stream and save it into a vector variable. There is a trick here, in the DBPSK modulation block, it assume that signal starts from zero phase. However, the first bit in the JPEG file does not necessary zero. So, we need to add one bit "0" ahead the JPEG file. To simplify the memory allocation, we add a byte "0x00"here.

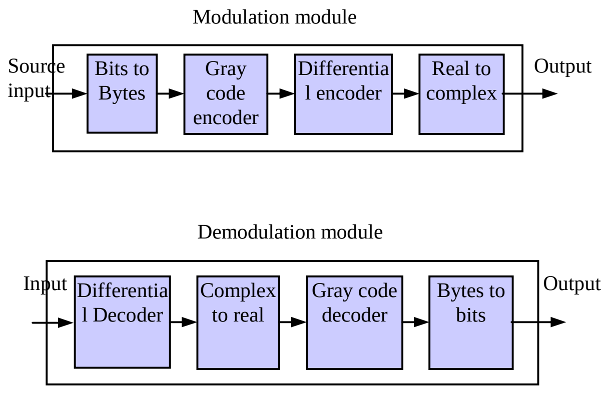

Figure 4 modulation and demodulation modules

b) Bits to Bytes converter

Now we will begin to process the data stream into DBPSK I/Q signals. In DBPSK modulation, each bit represents one symbol to be modulated. So, we will process data stream bit by bit. Since in computer it is easier to use byte operation than bit operation, we use one byte to represent each bit. While mapping each bit into one byte is quite straightforward, we need to take care of the selection of Big-Endian or Little-Endian, and the selection should keep same in both the modulation and demodulation side. In our program, we choose the Big-Endian, that is, the most significant bit first.

c) Gray code encoder

In modern digital communications, gray codes play an important role in error correction. Different from natural binary code, in gray code the signal's constellation diagram is arranged so that the bit patterns conveyed by adjacent constellation points differ by only one bit. By combining this with forward error correction (FEC) capable of correcting single-bit errors, it is possible for a receiver to correct any transmission errors that cause a constellation point to deviate into the area of an adjacent point. This makes the transmission system less susceptible to noise comparing with natural binary code.

In DQPSK, we may choose to use gray code or not, again, the option should be the same in both the modulation and demodulation side. In the GNUradio, there is also a function block in processing gray code for DBPSK, but actually, the map between natural binary code and gray code is the same.

d) Differential encoder

Instead of using the bit patterns to set the phase of the wave known as Phase Shift Keying (PSK), we can use it to denote the phase changes of the wave by Differential Phase Shift Keying (DPSK). Since the DPSK demodulator then determines the changes in the phase of the received signal rather than the phase itself, DPSK can be significantly simpler to implement than ordinary PSK.

Analysis shows that differential encoding approximately doubles the error rate compared to ordinary M-PSK. However this may be overcome by only a small increase in SNR. Furthermore, this analysis is based on a system in which the only corruption is additive white Gaussian noise. In communication system, the physical channel between the transmitter and receiver will, in general, introduce an unknown phase-shift to the PSK signal. In these cases the differential schemes can yield a better error-rate than the ordinary schemes relying on precise phase information[2].

e) Map the symbol into constellation point (Real number to Complex number)

Till now, each symbol is still represented by one byte (As mentioned above, in fact, only one bit of the byte is used in DBPSK, and only two bits is used in DQPSK). In DBPSK, there are only two points in the constellation diagram, so each point, i.e. each symbol, denote one bit baseband information. For the digital modulation, signal processing is based on complex number operation, and therefore, each symbol will use one complex number to represent and computation. In GNUradio, each complex number occupies 8 bytes, i.e. 64 bits and denotes one bit of baseband information. So, at the end of the modulation, we will see the output file become 64 times bigger than the original JPEG file.

The same, if using DQPSK, one symbol denote two bits baseband information, and for complex number computation, each complex number, i.e. 64 bits, represent one symbol. So, the output file is 32 times bigger than the original JPEG file.

Demodulation module

In our demodulation module, there are main 4 function blocks: Differential Decoder, Constellation mapping (Complex number to Real number), Gray code decoder, Bytes to bits converter. Other than Differential decoder, each function block performs the reverse procedure of their respective opposite function block in the modulation module.

The reason why the position of Differential Decoder in demodulation module is not exactly the reverse position of Differential encoder in the modulation is because the difference of DPSK from PSK. In PSK, demodulator need a complex carrier-recovery schemes to provide an accurate phase estimate and determine the symbols by mapping signals directly from constellation point. While in DPSK, as mentioned above, demodulator only determines the changes in the phase of the received signal rather than the phase itself. So, without a reference signal to compare the phase of the received signal, we cannot determine the symbols by directly mapping from constellation points. But in an easier way in QPSK, we can determine each symbol phase by knowing the phase change and the initial phase, which we set as “0” at the beginning. Then we map the symbol from constellation point (complex number) to real number.

When differential encoding is used in this manner, the scheme is known as differential phase-shift keying (DPSK). If demodulator still use reference signal to first demodulate each symbol, and then do the reverse of differential decoder, it is still a PSK demodulation with only baseband data being differential coded.

Limit in our modulation and demodulation modules:

As mentioned in the beginning of this section, in our demo the final output of modulation module is a baseband representation of the modulated signal. In the real wireless communication, we may need to fill out the high frequency components in the I/Q complex signal by a root raised cosine filter and then multiple it with a sinusoid carrier signal to get Intermediate Frequency (IF). In reverse, we also need to add some processes in the demodulation module, such as: automatic gain control, root raised cosine filter, symbol clock recovery.

3) Socket

As mentioned above, in our demo we simulate the wireless channel by using TCP/IP connection, which is done by python socket call. Python provide access to the Berkeley Software Distribution (BSD) socket interface. Python socket interface is a straightforward transliteration of the Unix system call and library interface[3]. For Internet connection, two kinds of most important socket types are SOCKET_STREAM and SOCKET_DGRAM, which respectively represent the standard Transmission Control Protocol (TCP) and User data Protocol (UDP) in transportation layer. TCP provides a logical full-duplex connection between two application layer processes across the Internet with connection-oriented, reliable, in-sequence service and flow control and congestion control, while UDP is a connectionless, unreliable protocol that provides only two additional services beyond IP: de-multiplexing and error checking on data[4].

In our demo, we choose the TCP connection based on two concerns: First, TCP connection is reliable and do not need extra application layer software to ensure the file being correctly received as in UDP. Second, we may want to extend the demo to support data stream transmission, such as video/audio stream, in the future, which requires a connection-orient service.

We implement the modulation together with TCP client in transmission side, and demodulation with TCP server in receiver side, and send the modulated JPEG file from client to server. First, we should assign a socket for TCP server with a fixed port number, and then TCP server will keep listening on the appointed port for any connection. Second, TCP client setup a connection with TCP server by latter’s IP address and appointed port. After the connection is established, server will keep listening on any data stream coming from client. As soon as server receives the whole modulated JPEG file, it will save the file for demodulation use next.

IV. Related and Future Work

The GNUradio project is still under developing, so source code release is updated at times. In our study, we find there are some bugs in the released software. For example, in the

DBPSK demodulation module, the clock recovery block does not work well. If we choose to add this function in the demodulation, after performing clock recovery, it will lose some data which represent the first 11 symbols in modulation. This definitely is not acceptable in transmitting a JPEG file, since any bit error in it will cause the JPEG decoder can decode it correctly. So, in our next work, we may go in detail to find the solution for this issue. Otherwise, it will cause DBPSK/DQPSK wrongly demodulate the received signal.

In our present demo, since we use the TCP socket to transmit the JPEG file, so we do not worry about any packets loss and bit error in the transmission channel since the TCP connection is a reliable connection. However, if we use USRP hardware with a practical wireless channel to transmit the JPEG file, things are quite different. We need to check whether received bit stream is correctly demodulated, and even if receiver's demodulation module functions well, we still have to check whether some signals lost or interfered, which are caused by noisy wireless channel. So, we may need to implement Forward error correction (FEC) or Automatic Repeat-reQuest (ARQ), and in error detection, CRC may be used.

At current stage, GNUradio also provide the functions support for implementing Frequency Shift Keying (FSK), Frequency Modulation (FM), Spectrum display (FFT sink) and Oscilloscope (Scope sink). Implementing these functions in our demo will give us more convenience to deploy future research project and stimulate us to develop more functions based on GNUradio environment.

V. SUMMARY AND CONCLUSIONS

In our paper, we first introduce the concept of software-defined radio and the capacity of GNUradio. Afterward, we describe the related hardware support and development environment respectively. Then, we explain in detail about what we have implemented based on our demo, and discuss the limits in this demo. At last we present some possible improvements and probable future work. Through this project, we build up a systematical knowledge and experience for developing GNUradio and it is very helpful for us to deploy a research-oriented project next.

REFERENCES

[1] http://www.comsec.com/wiki?UniversalSoftwareRadioPeripheral
[2] http://en.wikipedia.org/wiki/DPSK
[3] http://www.python.org/doc/current/lib/module-socket.html
[4] Communication Networks, Alberto Leon-Garcia, Indra Widjaja, McGraw-Hill 2003
[5] http://www.geocities.com/foetsch/python/extending_python.htm
[6] http://www.nd.edu/~jnl/sdr/docs/tutorials/
[7] http://staff.washington.edu/~jon/gnuradio.html
[8] http://lists.gnu.org/archive/html/discuss-gnuradio/
[9] http://jsp.dfes.tpc.edu.tw/py-book.jsp
[10] http://www.freebsd.org.hk/html/python/tut_tw/tut.html
[11] http://www.woodpecker.org.cn:9081/doc/abyteofpython_cn/chinese
[12] http://wiki.wxpython.org/index.cgi/Getting_Started