VIP Final Progress Report

SDR Team

Daniel Alejandro Tejada

1. Introduction

The goal of the Software Defined Radio project is to enable a point-to-point link between the two laptops provided by Epiq Solutions. In order to achieve this goal the following project subsystems were proposed after discussing with the advisor:

1. Implementation of a communication system in Gnu-Radio.
2. Development of linear error correcting code.
3. Configuring a Network Interface and convert PDU’s into samples.
4. Enable transmission of data between Sidekiqs(USRP) provided by Epiqs solutions.

The team consisted of six members who were divided into groups. First group was conformed by Charles Li, Kevin Choi, and Zi Jin Lu. They worked on the implementation of a communication system in Gnu-Radio. Second group consisted of Shannon Sremac, and Htoo Thein. They worked in the development of linear error correcting code and understanding of last semester BFSK flowgraph. Since I returned from last semester team, I collaborated with each group on using Gnu-Radio software and understanding of the project goals. Additionally, my contribution was on enabling transmission of data between the Sidekiqs and converting Protocol Data Units (PDU) into samples.

This final report will consist of a summary of the weekly progress, and a discussion of the flow graphs developed during the semester.

1. Weekly Progress

**Week 01/18-01/22**

Week Goals: Attend Meeting with Tom Krauss to install drives for sidekiqs

|  |
| --- |
| Summary:  Tom came to campus to install drives for the sidekiqs. We also had a discussion on how to use the sidekiqs blocks in gnu radio. We went over some examples on how to transmit waveforms using sidekiq. |

**Week 01/25-01/29**

Week Goals: Complete L01 and present last semester flow graph to Professor Wang

|  |
| --- |
| Summary:  First Official SDR team meeting we introduced ourselves and talked about how will we work over the semester.  *Group tasks* –  Complete tasks handout 01 |

**Week 02/01-02/05**

Week Goals: Complete L02 and L03

|  |
| --- |
| Summary:  We went over our understanding of the first homework.  *Group tasks* –  Read and complete L02 and L03 |

**Week 02/08-02/12**

Week Goals: Work on the BFSK scheme with higher sample rate and Complete L04

|  |
| --- |
| Summary :  Increase sample rate on the audio flowgraph to use sidekiq blocks.  *Group tasks* -  Learn about gnuradio software, how to use flowgraphs and the theory behind Binary Frequency Shift Key.  *Individual tasks* - Increase sample rate in audio BFSK to use sidekiq blocks. Finish L04. |

**Week 02/15-02/19**

Week Goals: Decide team members for two different sub teams

|  |
| --- |
| Summary :  This week meeting we talked about splitting into two teams that will work on different aspects of the project  *Group tasks* -  Analyze the theory behind Binary Frequency Shift Key modulation flowgraph and demodulation flowgraph  *Individual tasks* -  Arrange a meeting to explain how last semester work. Specifically teach every block function in the modulation and demodulation flowgraph. |

**Week 02/22-02/28**

Week Goals: learn how to use Github, GNU radio software, set Milestones

|  |
| --- |
| Summary :  This week we discuss the goals for the project and setting the milestones to achieve these goals.  *Group tasks* -  Define the milestones for the semester, keep learning about the gnuradio flowgraph. Learn about github and create accounts to obtain access to the github repository  *Individual tasks* -  Assign readings about BFSK theory. Attempt to use sidekiq blocks to transmit and receive frequency-modulated signals. |

**Week 02/29-03/06**

Week Goals: Define milestones for the rest of the semester, help recreate audio demo.

|  |
| --- |
| Summary:  We discuss about what are the goals for the semester, there was also a discussion about the errors that could appear when transmitting over air.  *Group tasks* -  Recreate the audio BFSK flow graph to understand how the communication system works.  *Individual tasks* -  Set milestones for the rest of the semester.  Finish modulation and demodulation of flow graph with higher sample rate on the same computer. |

**Week 03/07-03/11**

Week Goals: Understand why demodulation scheme doesn’t work with sidekiq blocks

|  |
| --- |
| Summary :  We discuss the factors that could be causing the wrong demodulation of the frequency modulated signal.  *Group Tasks* -  Work in each subteam task. Attend other teams meetings to help in using gnuradio flowgraphs and also theory behind it.  *Individual tasks* -  Debug the demodulation flowgraph.  Consider different factors that can cause a high bit error rate.  Factor 1: Verify that the frequency modulated signal is correct. Observe output using a Oscilloscope and Spectrum Analyzer.  Factor2: The integration period does not match in the demodulation scheme. Causes could be latency of USRP and different clocks of USRP.  Factor3: Frequency error, there are a lot of periods, a little difference in frequency at such high frequency (906.25MHz) could cause the two sinusoidals to be orthogonal. |

**Week 03/21-03/25**

Week Goals: Investigate factor 1 for high bit error rate and define reason to set up a network interface

|  |
| --- |
| Summary:  We discuss on specifying the goal of converting PDU messages into samples.  *Group Tasks* -  Work in each subteam task. Attend other teams meetings to help in using gnuradio flowgraphs.  *Individual tasks* -  Investigate factor 1 of possible causes of high bit error rate. Use oscilloscope and spectrum analyzer to anlyze output of modulation scheme. |

**Week 03/28-04/01**

Week Goals: Investigate using DPBSK for the communication system between the two sidekiqs

|  |
| --- |
| Summary :  We talked about testing the communication system involving differential phase binary shift key  *Group Tasks* -  Work in each sub team task. Attend other teams meetings to help in using gnuradio flowgraphs.  *Individual tasks* -  Investigate how to implement DBPSK as the communication system between the two radios. |

**Week 04/04-04/08**

Week Goals: Understand how the DBPSK system works. Figure out how the blocks manipulate samples

|  |
| --- |
| Summary :  We were able to send a file through air using DBPSK modulation and demodulation blocks in gnuradio.  *Group Tasks* -  Arrange a meeting during the week to work on the poster and include every sub team contribution to the poster  *Individual tasks* -  Finish the poster before the deadline and investigate the communication system with DBPSK. |

**Week 04/11-04/15**

Week Goals: Investigate ways to convert PDU packets that come from Tun/Tap into samples

|  |
| --- |
| Summary :  We discussed ideas on how to correctly transform PDU messages into samples to inject into the DBPSK system.  *Group Tasks* -  Work on understanding how each sub team project works in order to present on next week’s poster session  *Individual tasks* -  Investigate how to convert PDU into samples using the post on eventstream and packet transmission that is located in the sidekiq support forum |

**Week 04/18-04/22**

Week Goals: Develop a working demo to present on the poster session

|  |
| --- |
| Summary:  We talked about what is the idea behind converting PDUs into samples and also on having a demonstration for the poster session.  *Group Tasks* -  Work on having a working demo for the poster session  *Individual tasks* -  Install blocks that help managing asynchronous messages. Maybe find a way to write code for the block that converts PDU’s into samples. |

**Week 04/25-04/29**

Week Goals: Finish final progress report and update Github repository

|  |
| --- |
| Summary:  We had our last meeting and discussed how the final report should be submitted. Additionally we discussed next semester ideas because many team members are returning.  *Group Tasks* –  Finish each sub team report |

1. Gnu Radio Flow Graphs
   1. Radio Transmitter with DBPSK

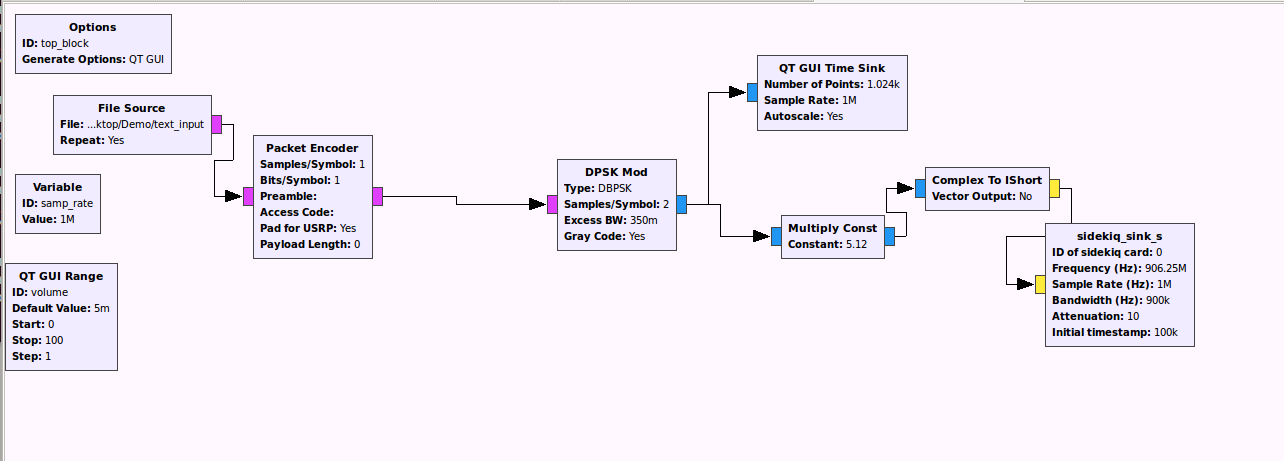


Figure 3.1.1 DBPSK transmission using Sidekiq blocks

The communication system between the two software-defined radios inside the computers was done using the above flow graph. One group of the SDR team worked on implementing Differential Phase Shift Keying to modulate the signal of the transmitter. I helped on the implementation of the sidekiq sink and source blocks in gnu radio. The idea is to take the modulated complex samples give them some gain with the multiply constant and turn the complex samples into short. After that, take the short samples into the sidekiq\_sink block to use the software-defined radio inside the computer to transmit the modulated samples using radio waves.

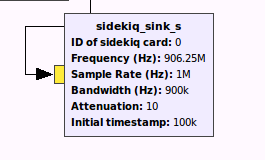


Figure 3.1.2 sidekiq\_sink gral block

This block was provided by Epiq solutions. The block was obtained from the Purdue/Epiq support site under gr-ral version0.1. Samples flow into the block and then the sidekiq converts the samples into radio waves. The ID of sidekiq card label defines which radio to use. To use the radio inside the sidekiq laptop use 0 for external sidekiq use 1. The Frequency label is to choose the carrier frequency of the generated radio wave. Sample rate label is used to choose how fast to sample the signal. Attenuation is used to reduce the gain of the output signal to avoid saturation. This block is used at the same time with sidekiq source to send and receive radio waves.

* 1. Radio Receiver with DBPSK

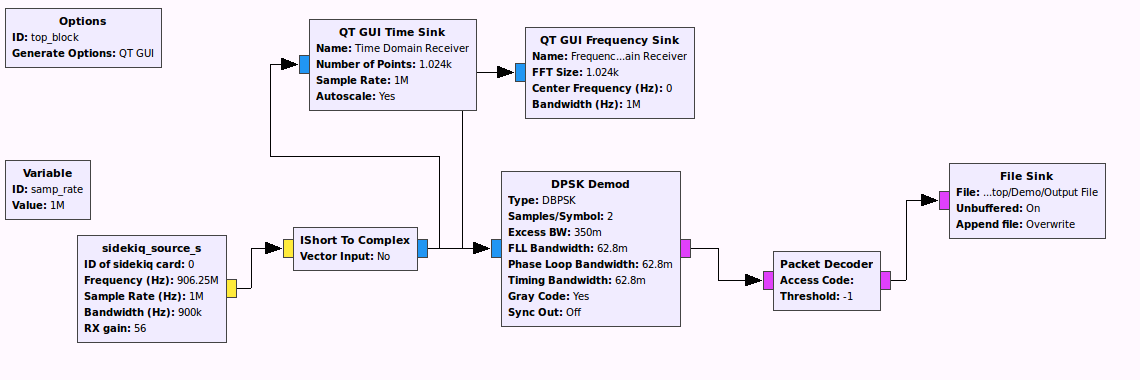


Figure 3.2.1 DBPSK receiver with sidekiq blocks

The flow graph is an implementation of the receiver for the transmitter flow graph in section 3.1. The sidekiq\_source block will receive the radio wave put the modulated signal into baseband and obtain samples. These samples will be injected into the demodulation system for DBPSK.

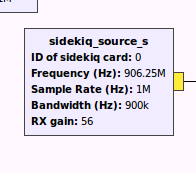


Figure 3.2.2 Sidekiq Source gral block

This block was also obtained from the Purdue/Epiq solutions support site under gr-ral version01. The sidekiq\_source block will generate short samples at a sample rate specified at the label sample rate. The samples will be generated according to the carrier frequency specified in the frequency label. The Rx gain label amplifies the samples to used them in the flow graph.

* 1. Transforming PDU’s into samples

To enable a point-to-point link a network interface has to be configured on the host system. This could be a Tun/Tap device that will be used to obtain packets. These packets need to be converted into samples to inject in our implementation of the communication system. On the other side the samples should be interpreted correctly to convert them into packets that are used to connect to another Tun/Tap device configured on the other computer.

A Tun/Tap interface is a virtual interface created in Linux. A network interface usually has a physical device associated with it to put packets on the wire. However the Tun or Tap devices are virtual and managed by a kernel. User space applications can interact with TUN or TAP devices as if they were associated with a physical device and the operating system will inject packets into the network stack.

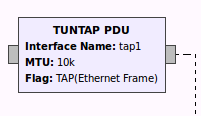


Figure 3.2.1 TUNTAP PDU block

There is a block in gnuradio that allow us to set a tuntap device that outputs PDUs. PDU is the term for protocol data unit, which refers to a group of information added or removed by a layer in the OSI model. For layer 2, which is the Ethernet level, the output is a frame. The MTU label is used to set the maximum performance that the network interface can achieve this means the maximum amount of bytes that the interface can output.

To correctly manage these packets that come from the network interface the post from the Purdue/Epiq support site under notes about EventStream and Transmitting Packets.

In the support site there is a clear description on how to download the required blocks to manage packets from an interface. The theory behind this process can be found in the following link: <https://oshearesearch.com/2015/04/02/burstpskmodem/>

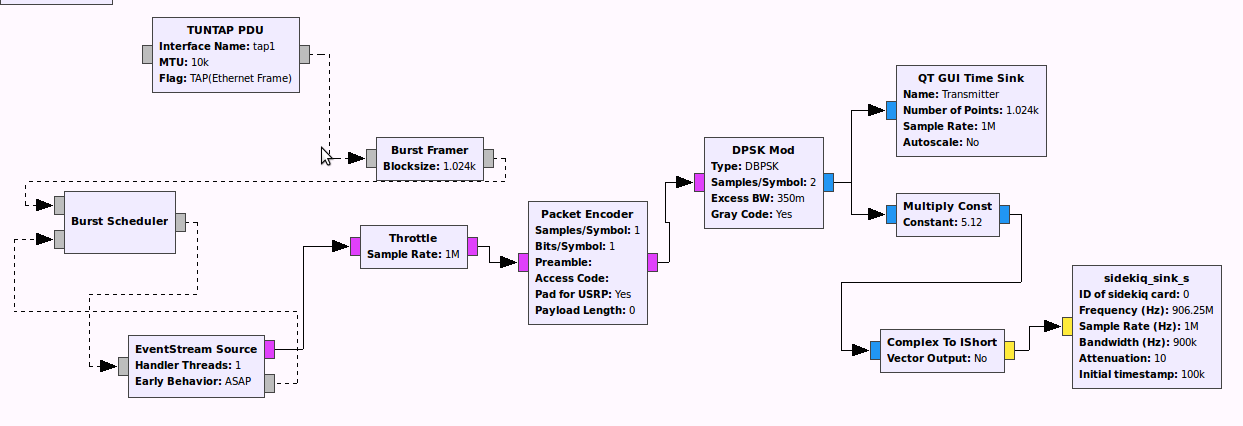


Figure 3.2.2 Point-to-Point link Transmitter implementation

The flow graph above is a tentative idea on how the implementation of the point-to-point link can be done. The packages will be supplied from the Tun/Tap interface into the burst framer block. The framer block will add a length field, a header checksum, and a payload checksum to verify on the demodulation part. The burst scheduler decides when to schedule a burst. A burst is defined as amount of data sent or received in one intermittent operation. The burst is scheduled into the stream on to the event stream source block and receives asynchronous feedback from the same eventstream block. The output of the eventstream source will be a sample of complex zeros with events copped into the correct offsets specified by their event time.

Finally the demodulation will have the same steps but in a reversed manner for this the trigger rising edge event block can be used as shown in the discussion of how to transmit packets and referenced site.

1. Conclusions/Future Work

First of all, the DBPSK communication system needs a more reliable linear error control coding. The current flow graph uses the packet encoder and decoder blocks to check for errors. The packet encoder takes a byte and puts 8 bytes at the beginning of the byte as a preamble. On the demodulator the packet decoder checks for the exact 8 bytes in the preamble and outputs the data only if the 8 bytes of the preamble match. For the setup used in the DBPSK with 500 000 symbols per second the amount of data that has to match in the preamble is considerable for every second. This causes considerable data loss and can cause disruption when trying to establish the point-to-point link between the two computers. The implementation of the hamming code or another type of error control coding is needed for a more reliable communication system.

Moreover, the Binary Frequency Shift Keying flow graph worked with audio signals at low sample rates but when the sample rate was increased to use the sidekiq blocks the communication system didn’t work. The main problem with BFSK is that when the sample rate is increased there is frequency error when transmitting over air. Since the communication system works with checking exact frequencies there were always mistakes when trying to lock into a frequency. There could be an implementation of Phase Lock Loop to avoid this problem and have the BFSK system transmit over air with the USRP.

Finally, during the semester I have learned about gnu radio software, communication systems, network engineering and the Open Systems Interconnection. Moreover, I got an idea on how important is to have clear objectives before starting to work on a project. I have also learned the importance of communication in a team and being able to explain a project to other people. Overall, the experience I gathered in vertically integrated projects (VIP) will be very useful in the future as an engineer and a person.