**Raspberry Pi - Cost Effective Digital Transmitter**

**Michael Botkin, Junsung Choi, Eric Sollenberger**

**Abstract**

The goal of this project, as stated in its proposal, was to implement a low cost SDR transmitter using the Raspberry Pi (RPi or Pi). The proposal goes on to discuss various applications that can be developed, some of the details of how it might be implemented, and how its performance might be optimized using this SDR transmitter. This work has been largely successful, but it has also been subject to a great number of limitations. The most significant limitations result from the modest computing power of the RPi and the limited control of the transmitted waveform. Over the past four months, the Raspberry Pi has proven to be a capable and useful SDR transmitter. Using the software developed during this term, it is possible to achieve reasonably high data rates while maintaining high fidelity communication. This work can be leveraged to enable a great variety of applications or even to work with other existing radio controlled systems. Several applications were developed to go along with this work as a demonstration of the radios utility. This document will discuss the success of this project, the limitations that were encountered, and how this work can be continued and improved upon.

The project started out based off the PiFM software created by Oliver Mattos and Oskar Weigl, as many other projects have. Rather than being used directly or with minor modifications, the PiFM code was broken down and new code was written from scratch. This newly written code uses the same fundamental concepts as PiFM, but with a focus on digital modulations and flexibility. This program could provide the transmitter backend for many applications. Two different modulation types have been implemented, OOK and FSK. Clearly, this transmitter is not as flexible as typical SDR transmitters that accept baseband data, enabling any modulation scheme to be used. That being said, the Raspberry Pi is still very useful, as will be shown later in this paper. Also, here is more room for improvement in error correction/detection coding, CDMA, and even reduction in CPU usage. The Raspberry Pi has more computer power available to use complex transmitting algorithms. We created a stable backbone that can be easily built on.

GNU Radio was selected as the platform for this receiver, to easily interface with the RTLSDR or other RF frontends. A custom GRC block was created to interpret the received waveform. This block was easily integrated into existing flowgraphs, which facilitated the development of applications. Once the transmitter and receiver were both adequately stable, a method for data packetization was added to easily manipulate and control data. Despite limitations, the RPi was able to achieve reasonably high throughput. The maximum symbol rate that was achieved was 300 ksps. Using asynchronous transmission, symbols were successfully received when sent in packets as long as 8kb.

In this short time, a number of applications were developed that demonstrate the utility and flexibility of the RPi as an SDR transmitter; text file transfer, an instant messaging GUI, simple DSA algorithm implementation, a cognitive radio network testbed, and RC car control. There are many potential applications, a number of which have already been implemented by others.

**Table of Contents**

1. Topic/Problem Statement ...................................................................................................... 4

2. Background and Motivation ................................................................................................... 5

2.1 Inspiration/Related Works ....................................................................................... 5

2.1.1 Transmitter ............................................................................................... 5

2.1.2 Receiver .................................................................................................... 6

2.2 Consulted Research ................................................................................................. 6

2.2.1 Main Development .................................................................................... 6

2.2.2 Advanced Applications .............................................................................. 8

3. Details of SDR Implementation ............................................................................................ 9

3.1 RPi Transmitter ....................................................................................................... 9

3.1.1 OOK Transmission .................................................................................. 11

3.1.2 FSK Transmission ................................................................................... 12

3.1.3 TCP Data Transfer and Control ............................................................... 12

3.2 GNU Radio Receiver .............................................................................................. 13

3.2.1 Custom Block - RPi Rx ............................................................................ 15

3.2.2 Receiver Parameters ............................................................................... 16

3.3 Packet and Header ................................................................................................. 16

4. Enabled Applications ............................................................................................................ 18

4.1 File Transfer ........................................................................................................... 18

4.2 Instant Messaging .................................................................................................. 18

4.3 DSA ........................................................................................................................ 18

4.4 Micro Cognitive Radio Network Testbed ................................................................ 19

4.5 Rc Car Control and other Functions ....................................................................... 19

5. Review of Work Plan ............................................................................................................. 19

5.1 Deviation from Proposal ......................................................................................... 20

5.2 Major Obstacles ..................................................................................................... 20

5.3 Limitations .............................................................................................................. 21

6. Performance Metrics ............................................................................................................ 21

6.1 Max Throughput ..................................................................................................... 21

6.2 BER ........................................................................................................................ 22

6.3 Tx Range ................................................................................................................ 23

6.4 Bandwidth vs M-FSK .............................................................................................. 23

6.5 Spectral Analysis .................................................................................................... 23

6.6 FCC Regulations .................................................................................................... 24

7. Potential Further Developments .......................................................................................... 27

7.1 Analog Transmission .............................................................................................. 27

7.2 Variable Packet/Header lengths ............................................................................. 27

7.3 Error Detection, Prevention, and Correction ........................................................... 27

7.4 Low Rate Wireless Personal Area Networks .......................................................... 27

7.5 ASK ........................................................................................................................ 28

7.6 FCC Regulations .................................................................................................... 28

7.7 Other Functions ...................................................................................................... 28

7.8 Filtering ................................................................................................................... 28

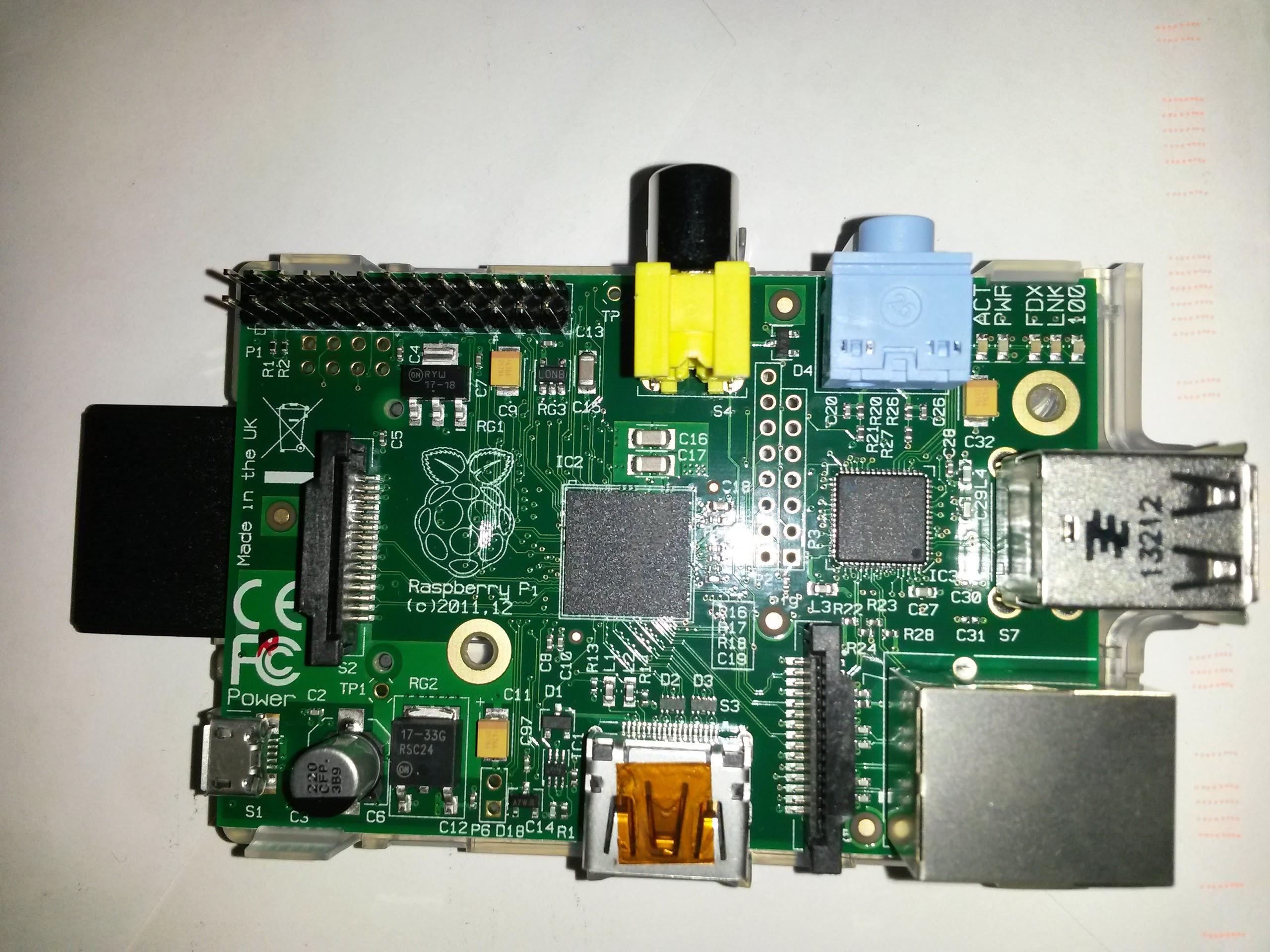
8. Conclusion ............................................................................................................................ 29

**List of Abbreviations**

|  |  |
| --- | --- |
| ADPLL - All Digital PLL  b/s - bits per second  b/S - bits per Symbol  BER - Bit Error Rate  CR - Cognitive Radio  DMA - Direct Memory Access  FCC - Federal Communications  Commission  FM - Frequency Modulation  FSK - Frequency Shift Keying  GPIO - General Purpose Input/Output  GRC - GNU Radio Companion  LAN - Local Area Network  OOK - On Off Keying  OS - Operating System | PLL - Phase Locked Loop  PWM - Pulse Width Modulation  RPi or Pi - Raspberry Pi  RF - Radio Frequency Rx - receive  S/s - Symbols per second  SDR - Software Defined Radio  SSH - Secure Shell  SSTV - Slow Scan Television  TCP - Transport Control Protocol  Tx - Transmit  UART - Universal Asynchronous  Receiver/Transmitter  USB - Universal Serial Bus |

**1. Topic/Problem Statement**

The initial proposal was to use the Raspberry Pi as an SDR transmitter based on work of researchers at the University of Cambridge’s Computing Laboratory who were the first to explore this idea. They created open source software, PiFM, that took advantage of the RPi’s DMA (direct memory access) and a GPIO[1] pin. PiFM software replicates a FM transmitter and can be listened to on any home stereo just like a radio station. The program reads in a wav file and uses the clock signal generating capabilities of I/O pin 4 to create a carrier. The clock divisor control register can modify the fractional divider of the PLL. This, in turn, will change the frequency and can create FM signal. The audio stored by the wav file is transmitted as an FM signal by dynamically changing the control register described.

The backbone for the project was built on the Raspberry Pi, shown to the right. It is a barebones computer used to encourage Linux, hard-coding development in the educational environment [2]. In the end, the Raspberry Pi is capable of many different uses from a media center to a low power desktop for $25 - $35. This computer is not as powerful or as expensive as the Beaglebone or some Arduino’s, but its initial purpose was to be capable and cheap enough to encourage risky development for new and experienced programmers. The Pi, itself, is limited on memory and processor speed which provides complications on determining its capabilities for our application. The model we used, Model B, has a 700MHz ARM11 core processor with 512 Mb SDRAM. An 8GB external SD card stored the OS. The Raspberry Pi can be operated using a dedicated mouse and display or over a LAN using the SSH protocol. Our RPi’s are networked directly to a Ubuntu 12.04 PC’s Ethernet port.

Our project started on the Pi by adapting PiFM to build a digital transmitter. Next, a receiver was built in GNURadio, using the RTL as the SDR frontend. As more capabilities were included, the transmitter and receiver developed simultaneously. Coding milestones, set based off of the expected difficulty and time frame were all complete ahead of schedule; Enable GPIO clock, Modulate samples (OOK), Map bits to frequencies (FSK), and increase modulation rate by using the DMA.

After the initial milestones were met, more time was devoted to developing a receiver, creating a more stable transmitter, and enabling more applications. The real challenges were faced with controlling the DMA, used to reduce CPU usage, and building a receiver from scratch in GNU radio. Despite these challenges, enhancements were achieved as planned and will be explained throughout this document.

**2. Background and Motivation**

**2.1 Inspiration/Related Works**

The Raspberry Pi opened the door for a lot of low power electronic home projects and has been utilized for more than the original creators could have imagined. There were already a couple of applications that used the RPi as a transmitter, and they were each based off of work by researchers at the Imperial College Robotics Society. The original code created a personal FM radio transmitter, used to play wav files over the air on any home stereo.

**2.1.1 Transmitter**

PiFM, created a little over 2 years ago, opened the door for hobbyists to create their own personal transmitters. Although most applications were analog or geared towards very specific problems, our project ventures into uncharted territory, creating a universal digital transmitter. Other works provided insight to the variety of options for not only how the transmitter might be implemented, but also the vast number of purposes for which it might be employed.

The creators of weed pi took this a step further, adding the functionality of an internet radio including an IR remote control [6]. This was a neat idea, but it just utilized pre-existing software, we wanted to implement an actual SDR transmitter. Oliver Mattos and Oskar Weigl, in the book *Raspberry Pi Hacks,* mention building a text transmitter that uses recorded sounds for morse code. They even suggested another stop, still only sounds through PifM, use a microphone decoder program as the receiver [7].

One of the most advanced/practical adaption from PiFM is used for SSTV, slow scan television. The Raspberry Pi has been used in place of expensive computers for many applications, where microprocessors may be harder to implement. In “Long Range Image Radio Transmitter,” they used the Pi to handle the image acquisition, which uses its USB port camera driver. This gave it an advantage over microprocessors, but they still implemented very efficient coding to convert the image to a sound file for transmission. Unfortunately, other hardware was used to handle the transmission [8]. Gerrit Polder, creator of the PiSSTV program, used an efficient C program to capture the image from the camera and convert to a sound file. He then adapted the PiFM program and, after a few modifications to help with image skew and bandwidth changes, was able to create a portable SSTV [9].

SSTV was a very creative use of PiFM, but it was still essentially an analog application. Jon Petter Skagmo created an ASK program that could control Nexa Power Switches. He did not implement use of the DMA, but built his project based off of the clock controls used in PiFM. Skagmo did one thing in particular that stood out; although the Pi is very limited in the range of frequencies it can transmit, he still implemented a 433.93MHz transmitter by using a harmonic of the clock frequency. He took advantage of the powerful third harmonic from a 144.64MHz signal. To be within regulations, Skagmo used a surface acoustic wave bandpass filter around 433.92MHz for under $10 [10]. Adapting this method, with proper filtering, we could expand the range we can transmit without using of an upconverter.

**2.1.2 Receiver**

Despite the Raspberry Pi’s limited processing power, it has been used as a receiver for several existing applications. One example, found in *Pi Hacks*, uses the Pi with an RTLSDR to listen to Airplane transponders. This code is prebuilt and a set up process is supplied to implement it. The code will list the flights within range. From another PC you can pull up a browser and visually see these planes on a map [7].

Unfortunately, the Raspberry Pi was not well suited to be a standalone receiver for this work. The receiver was implemented in GNU Radio, which has issues running on the RPi. It can be installed and opened up, but whenever any level of DSP is implemented in a flow graph, it will crash. A good alternative is to use the RPi as a server for an RTLSDR and run the receiver DSP in GNU Radio on another, more capable platform. Stephen Cass, was working with the RTLSDRs and decided that they were too cumbersome for his office, which also had poor reception. His goal was to have a cheap SDR receiver compared to the motherboard, daughterboard combination that can cost up to $875. By running rtl\_tcp, the RPi can be placed anywhere with access to the LAN, along with an RTLSDR. The complex baseband samples received by the RTLSDR can then be accessed by any computer on the LAN by starting a tcp client and connecting to the RPi. This allows for a great deal of flexibility in the placement of the RTLSDR and the main PC, making setup much more convenient.

**2.2 Consulted Research**

While working through this project, it was important to know more information about the modulation schemes being used, packetizing protocols, and other SDR techniques. First we had to get a stable system, including packetizing data, error checking and testing. After we can prove the functionality of our system we are able to advance it. The two sections below describe the research needed for the fundamental work of this project, followed by research that was useful for developing advanced applications.

**2.2.1 Main Development**

Before approaching this work, it was important that both modulation schemes were understood thoroughly. “A Comparison Between OOK/ASK and FSK Modulation Techniques for Radio Links,” written by Darrell L. Ash, was consulted to gain a greater understanding of the two modulation schemes and to understand the background for FSK transmitters and receivers more fully. In this paper, Darrell L. Ash discusses short range radios, typically below 100 meters. He talks about the applications of these radios, including keyless entries, security, and data links. His assessment of the two systems are from the view of an implementation and industry standpoint. This provided insight into the practicality of our device. After comparing the two receivers and transmitters for bandwidth, noise, co-channel interference, complexity, and even a practical comparison of tire pressure monitoring, his conclusions supported using an OOK transmission over FSK for short range devices. There was some concern of flutter in the transmitters and how FSK receivers are more immune to this, but this shouldn’t affect our project. He claimed that OOK receiver can equal if not outperform an FSK receiver while remaining simpler [5].

For this reason, an OOK transmitter/receiver pair was implemented first. The OOK transmitter is a very simple and power efficient design. The article “I’m OOK, Your OOK?” explains a lot about OOK transmitters and receivers and supplied a solid background of understanding as to how the transmitter will work, conceptually, and many useful applications for it. OOK is appealing because its bandwidth is directly proportional to the signaling rate (modulation rate). Theoretical bandwidth is the same as the modulation rate but the waveform can increase the actual bandwidth [4].This was a good start, but the ability to transmit FSK signals was also highly desirable, so both were implemented.

After a short time, both FSK and OOK transmitters were functional. The OOK signals seemed to be limited though because the timing of its transitions between on and off were not as accurate as originally hoped. It is believed that this was caused by a caveat of the GPIO, wherein it cannot be disabled during a cycle of the fractional PLL. This causes a delay when turning off the GPIO. As a consequence, OOK was placed on the backburner, and work was continued using almost exclusively FSK.

It was also essential that the data be appropriately formatted for it to be actually useful. A major reference used for calculations and different packet protocols was *Data Communications and Networking*. This book is for a Telecommunications class at Virginia Tech. It helps explain how networks operate and the details for their implementation. Since we created a small, crude wireless network we used a simplified version of an Ethernet header and referred to digital transmission section for OOK and FSK calculations [4].

Once our packets we properly being transmitted and received we decided to add error checking to the header, to make sure at least the first 21 bits were received correctly. We decided to add a CRC check for the header alone. Since we are limited on processing power, we needed a memory efficient method for the CRC. Gam Nguyen’s “Fast CRCs” gave use a good idea on how to implement this. We reinforced the concept of CRCs and multiple methods for implementation using this article. Fast CRCs can be very useful for large amount of data. Importance is placed on the CRC length since computers like to work in bytes or tuples [12]. Our CRC is only for the header, we used this information and adapted our own method for implementing a CRC algorithm.

The final step for our initial system is to prove its functionality and reliability. Bit Error Rate testing is one useful method for showing how accurate our combined transmitter and receiver are. “Analyze BER Performance of Wireless FSK Systems,” targeted our project specifically on how to test and theoretically derive characteristics of our transmitted waveform. This document makes sure to point out the effect of data rate and the importance of bandwidth in the band limited spectrum. They also provide theoretical and simulated BER values that we can compare our system to [13]. This gives us a background on how to interpret our BER tests.

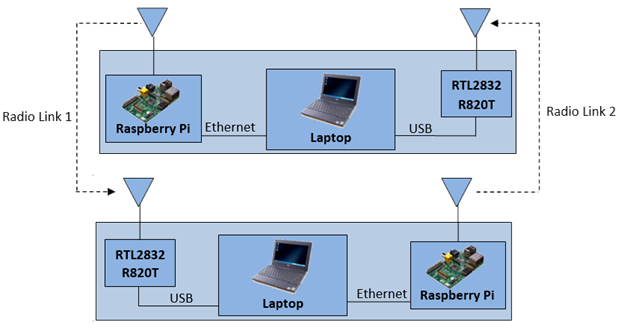
**2.2.2 Advanced Applications**

After achieving a stable transmitter and receiver pair, more advanced SDR systems/networks can be deployed. A topic gaining increasing recognition and interest is dynamic spectrum access. Since the conception of the radio there has been a consistent increase in the demand for bandwidth across the board in both military and private applications. This trend shows no sign of stopping, yet it is a finite spectrum that is available. Dynamic spectrum access is a method that has been proposed for mitigating this issue. The basic concept is that radios will change their carrier frequency so as to operate in a channel that is unoccupied and to avoid interfering with users that own specific part of the spectrum. One group member, Eric Sollenberger, has already researched this topic for another project. Since this system runs asynchronous, peer to peer, with no central control, we consulted “Asynchronous Dynamic Spectrum Access.” It discusses the importance of using DSA and how independent devices will interact. Game theory suggest that each individual will make their own decisions and eventually come to an equilibrium. An optimal equilibrium can be achieved, but there is a fine relationship between sensing time and overall throughput [15]. We have to account for this in DSA applications.

To create network control of the transmitter and allow the DSA functionality to run on a computer with more processing power, a TCP server transmitter can be utilized. *Linux Socket Programming* was a useful book for how to create a socket server and how to handle endianness differences from the Pi and laptop [16]. TCP control proved to be more useful than initially predicted.

Another potential application for our transmitter, with some adaptation, can fall into IEEE 802.15.4g. “Low Power FSK transmitter using all-digital PLL for IEEE 802.15.4g” explains the standards and use of an ADPLL for FSK transmission. Unfortunately the frequency band for this increasingly important low power radio system is 902 - 928 MHz. This frequency is directly out of our range, but we might be able to utilize a harmonic with a bandpass filter at the right frequency. Three data rates that should be supported include 50/150/200 kbps with specific deviation settings [17]. We have successfully tested these three rates and have a variable deviation setting. With a little effort, our project could work for this standard.

**3. Details of SDR Implementation**

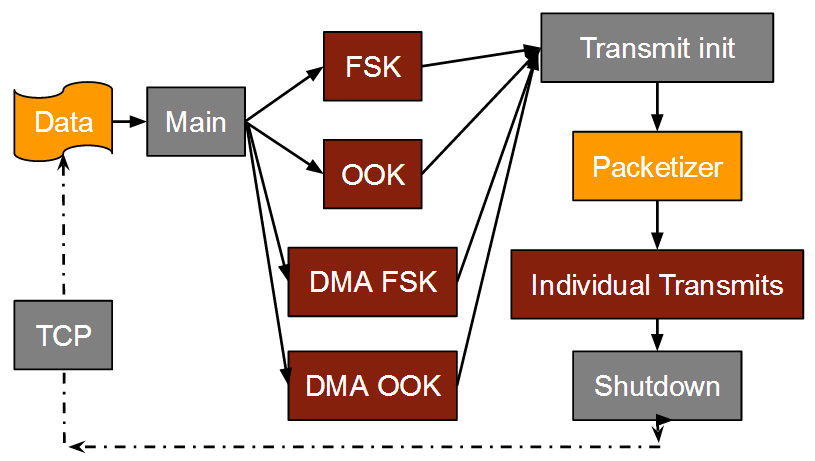
This section will discuss the details of how the transmitter and receiver were implemented. Specifically it will describe the code that was written to run on the Raspberry Pi, the receiver GNU Radio flow graphs and custom block, and how data was packetized to enable the different applications we had planned. Below is a block diagram describing how the final system was set up. For development we would only transmit from the Pi to the controlling PC. For application level, practical implementation, two independent computers would be connected only by the radio link between the Raspberry Pi and the receiver using an RTLSDR.

**3.1 RPi Transmitter**

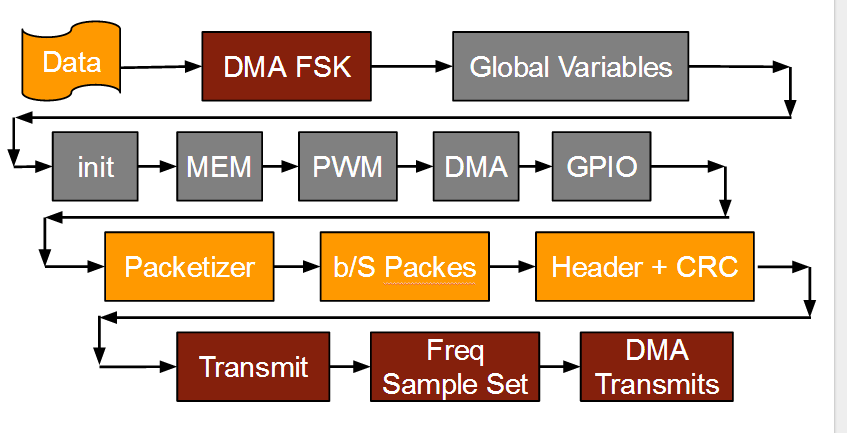
This section will discuss the code built and run on the Raspberry Pi. The breakdown of PiFM left a very confusing but necessary DMA and PWM setup process before ever transmitting a bit. The initial code had very few comments and was not modularized in any way. To fully understand how the code works, the program was scoured and commented thoroughly while verifying functionality with the processor documentation. After understanding the code, it was recreated in a new project, organized, modularized, and memory efficiency was increased.

The following page has a block diagram for the overview of the main transmitter program. Upon program call, a number of options can be set to change the transmitter properties. Otherwise its global parameters will be defaults. The options can change the following: Center Frequency, Frequency Deviation, Modulation Rate (S/s), Bits per Symbol (b/S), Packet Length, Address, Packetizing Off, File to transmit, and different transmit Flags.

Once started, main will ask which modulation type the user wants to use; DMA FSK (direct memory access), DMA OOK, FSK (Computer Intensive without DMA), OOK. These functions take hard coded data or an input text file, manipulates and packetizes it based off of global variable settings, then transmits using the desired method. The TCP main will not ask for a transmission type, but that can be change over the socket with a CONFIGURE command.

The main program calls the respective transmission program passing it a binary int array of data (containing only 1’s and 0’s) and the length of the data. All transmission programs, if packetizing is turned on, pass the data to a pacterizer function that returns a double int array and a number of packets. After packetizing, a transmit initialization is called then data is then transmitted using methods discussed in the following sections.

Below, and discussed on the following page, is a description for the DMA-FSK Transmitter.



After data has been passed to any transmitter functions, it must go through a set up process which controls the transmission parameters and packetizes the data. The global variables control the transmitter initialization and packetization process. The first step in the initialization process was to allocate a number of pages for the program to safely use and allow the DMA to access them. Understanding how the DMA and PWM work is a very important step for a reliable transmitter. PifM initially used sleeps between each bit transmission, which can result in non-exact modulation rate. Slight differences do not really affect a listener, but precision is important in a digital communication device. The DMA controller runs parallel to the processors.

To set up, a series of control blocks, connected to the DMA control registers, were initialized. The DMA will pass through a memory page, copying data from the designated source to a destination. For each sample to be transmitted, the DMA had two blocks to read through; the digital signal (frequency or on/off), depending on the modulation type, and a generic write to the PWM. The PWM was essential for timing. At initialization, the DMA control register was set to wait for writes. The PWM rate is set to the modulation rate. This, in turn will slow down the DMA and control the rate at which digital samples are to be transmitted.

The final, and essential step for the transmitter set-up process, after memory has been allocated, DMA and PWM both initialized, the clock had to be connected to GPIO4, the pin used for transmission. Each GPIO for the Raspberry Pi has multiple functions and setting the correct setting is critical. This function is used to provide a clock signal to an external peripheral but the signal can be radiated by attaching a 15 - 20cm wire to act as an antenna.

The data now must be prepared to transmit in a way that the receiver can properly interpret it.

To packetize data:

1. If the bits per symbol is greater than one, the data is packed into symbols
2. The packetizer allocates memory for the number of packets based off of the packet length, header length (if on), and necessary start/stop bits.
3. Each packet is stepped through
   1. The start and stop bits are set, discussed in section 3.2.1
   2. The header is set by a separate function, discussed in section 3.3
4. The number of packets and new data is passed back to the transmission function

After the transmitter is initialized, different for each type, and the data has been packetized, the individual transmitter functions can be called (discussed below).

**3.1.1 OOK Transmission**

The easiest transmission type to implement is the OOK transmitter. We did successfully create an OOK transmitter, but we did not develop the OOK receiver fully. The OOK data rate was too limited to invest more time in to perfect. The transmitter is straightforward, but the receiver was left in its early states and development of the more powerful FSK receiver was increased.

The FSK transmitter and the OOK transmitter have the same setup, but OOK transmits on the carrier for a 1 and doesn’t transmit at all for a 0. This would be a very power efficient plan for the already low power Raspberry Pi. OOK is very useful for home automation and small electronics but is limited in data rate since it is a binary modulation type. OOK receivers are also simple in design and price [3], but we wanted to push the data rate or our transmitter past 1 b/S.

To create the OOK aspect of the transmitter, enable/disable bit is repeatedly set or cleared depending on the data at the modulation rate. When using the DMA, the destination for each data sample was the clock control register and the default value has the clock disabled. This will connect the clock and disconnect it creating an OOK waveform. When the OOK transmit function is running the samples, sources for the DMA controller are set on or off, depending on the data passed to the function. Depending on the packet length, a series of data can be stored in the sample space for the DMA to transmit. This enables the ability to sleep while waiting for transmissions to complete.

**3.1.2 FSK Transmission**

The differences between FSK and OOK transmitters/receivers have often been compared and both methods have their own advantages. The minimum bandwidth needed for BFSK signal is 1.5 times greater than that of OOK, which means it is spectrally less efficient. Since bandwidth is an extremely scarce resource, this is a very critical metric when using the RF Spectrum [5]. Even though OOK receivers perform in noisy situations a little bit better than FSK receivers, there is a limiting factor of binary transmission versus m-ary transmission that M-FSK can provide.

Our implementation of FSK is based off of altering the clock divisor register to control the frequency. For higher speed applications, the DMA can be used with a destination register for digital samples set to the frequency divisor register. This is a very crude method for changing the frequency since complex algorithms should be used to steadily change frequencies, applying pulse shaping, instead of an instantaneous shift from one frequency to the next.

FSK can be flexible with a couple different parameters. Initially the FSK is set to a 0, which depends on the frequency deviation and number of bits per symbol. If a carrier frequency of 125 MHz was used, 2 bits per symbol, and frequency deviation of 100 kHz, then the 0, 1, 2, and 3 would be at 124.85, 124.95, 125.05, 125.1 MHz respectively.

While in the transmitter function, the different data samples, from the packetizer, are read and set the frequency based off of a frequency deviation. The new clock divisors are stored in the sample space, sources for the DMA, and in thus will be written to the clock divisor control register when the DMA controller gets to those samples.

**3.1.3 TCP Data Transfer and Control**

A separate main function, that uses the same transmission back end already discussed, creates a TCP server that waits for data or commands. The TCP function works in a similar manner as the file transfer functions. Any byte based data sent over the TCP socket will be broken down into binary then passed to the transmitter. If the data on the socket contains CONFIGURE followed by a series of options (ex. CONFIGURE -f 128 -m 80000 -d 200 -r off -p 1024), then the socket will not transmit that packet and will then reconfigure the transmitter, changing the global control variables. TCP data transfer and control opened the door to cognitive radio functionality, after discovering that receiving on the raspberry pi was not practical.

Since the control of the transmitter is as easy as sending a short packet over a TCP socket, any number of cognitive functions could be used. The limitations for this would be from the receiver. The receiver is being ran on a laptop, as long as it can sense the environment and decide what actions to take, a simple cognitive or DSA radio can be implemented by sending TCP commands. The Pi has a lot more computing power than is currently being used to transmit FSK through the DMA.

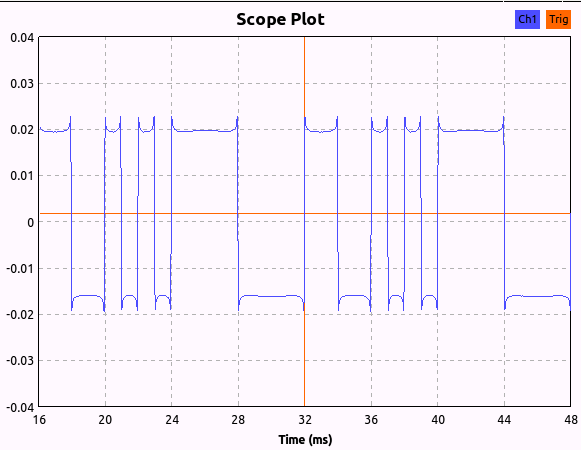
Using a TCP transmit method, the Raspberry Pi could run the program on boot and just wait for configuration commands of data to transmit. There would be no reason to actually SSH into the Pi if the boot time running was set up correctly. Any socket could be created to pass data. We have created a simple IM program, discussed later, that passes data to socket, which in turn gets transmitted. This program was developed in a day and seamlessly was able to transmit IM messages, without modification.

The TCP transmit server code include client software. There is a client file transmit program, that breaks a file down, block by block, and passes it to the transmitter. There is also client software that can be used to simply control the transmitter. Some functions include set\_freq or set\_mod\_rate. We created this so that it can be included into a GRC custom block for DSA control. The idea for a true DSA transmitter is to have the TCP transmit server start on boot time for the Raspberry Pi. Then the transmitter will be available at any time without the need to ssh into the Pi and start the program.

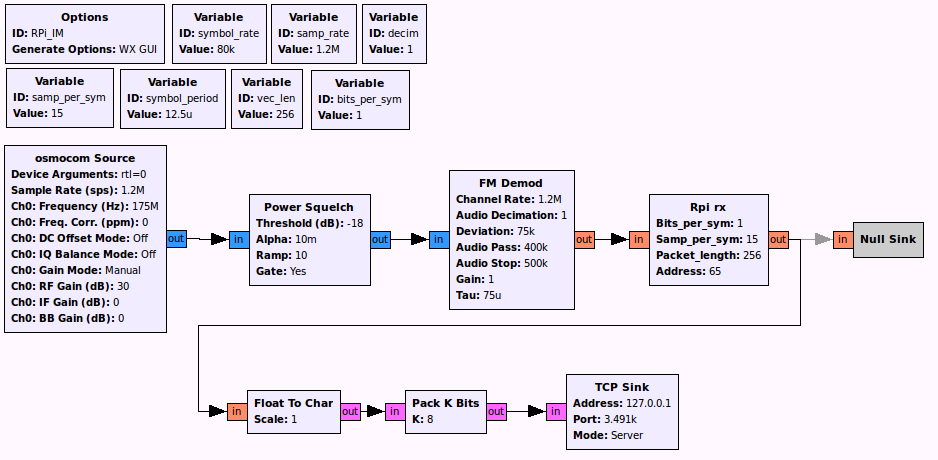
**3.2 GNU Radio Receiver**

One of the more challenging aspects of this work was determining how to accurately receive data from the RPi transmitter. At first, it seemed reasonable to implement some synchronization in the receiver. The first attempt to make a working receiver used an M&M clock synchronizer in an attempt to extract the baseband data correctly. This approach never produced the correct data though. Admittedly, it may have been possible if someone on the team had more experience with implementing synchronization for a receiver, but this was not the case. Not much progress was made on this issue for a little while until eventually it was decided that an asynchronous demodulation method would be better suited to this work.

The final receiver is implemented in GNU Radio and consists of an osmocom source block which reads in complex baseband samples from an RTL2832u Digital Video Broadcast USB dongle. These samples are first passed through a gating squelch which has a threshold set to impede the signal flow when there is no carrier present in the channel being received. Next the signal is demodulated either with an FM demodulator for M-FSK signals or with an average magnitude squared conversion for OOK signals. The FM demodulator has a built in low pass filter which helps diminish the effect of any noise outside the channel. Once the signal has been demodulated, the result is an oversampled baseband digital signal as shown on the next page.

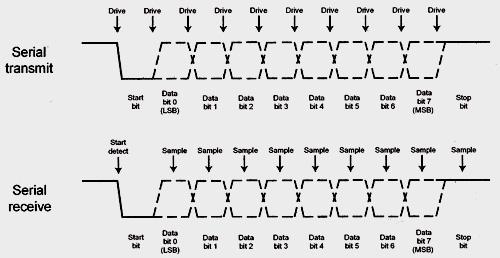


This signal is fed to a custom block which implements a UART receiver, symbol slicing, interpretation of a packet header (defined later), and some additional functionality such as: file writing, displaying the data, and the standard method of passing data to the output port of the block. A more in-depth description of this block is in the following section. For the standard functions that were built into the receiver, the output of the custom block is simply fed to a null sink. For other applications, such as the Instant Messaging App, the output of the block is fed to a TCP sink, which sets up a server. This server will allow applications to retrieve the data as it was transmitted and use it as necessary. The GRC flow graph is shown below.

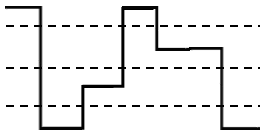


**3.2.1 Custom Block - RPi Rx**

The custom block that was created is essentially a UART receiver with some added functionality. The first step for this block is to look for a rising edge. For this to work correctly, the transmitted signal must turn on at the high then low frequency first (2 bits - 1, 0). Once a rising edge has been detected, the receiver looks at the samples received during the next two bits. It then checks to see that the first bit is an acceptably high value and that the second bit is an acceptably low value. This implies that the center frequency of the receiver must be sufficiently aligned with that of the transmitter so that the high start bit will be interpreted as a positive frequency and the low start bit will be interpreted as a negative frequency. For the frequency spacing that was used during testing, this is not an issue, though the frequency error between the two devices is observable. Once the start bits have been successfully identified, the receiver counts samples and interprets the sample closest to the center of each symbol as the symbols value. This concept is depicted below.



The two start bits not only alert the block to begin receiving symbols but also act as a way to mitigate errors caused by the frequency offset between the two devices and a way to enable the decoding of multiple bits per symbol. This is accomplished by defining the levels at which the received symbols should be sliced. The average value received in the first bit is taken to be maximum symbol value, so a 0b1 for single bit symbols or a 0b11 for two bit symbols. Likewise the average value received in the second bit is taken to be zero. This allows the receiver to slice the symbols according to the signal being received, rather than at theoretical points. The symbol slicing for two bit per symbol data is shown below. Even though the signal is not centered about zero, the slicing points are defined based on the high and low values received, and therefore fit the data much better than theoretically chosen points.





**3.2.2 Receiver Parameters**

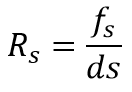
In order for the receiver to function properly, there are a number of parameters which must be set appropriately. Beginning at the first block, the Osmocom Source block, the important parameters are the sample rate, the center frequency, and the RF gain. The sample rate is important because if it is not an integer divisor of the RTLSDR’s reference oscillator, the signal will not remain in sync for very long when the custom block attempts to decode it. When the receiver was first created, this was not understood, and it caused the maximum packet length to be only around 40 symbols long. Once this issue was understood, the maximum packet length exceeded 8,000 symbols in length. The center frequency is of course important in that it must match the frequency that the transmitter is using. Finally, the RF gain became very important once a shorter, less efficient antenna was used, because the received signal was much weaker, requiring a greater gain.

The next block is the squelch, in which it is important to set an appropriate threshold and to enable the gate. The threshold needs to be set so that it will gate the signal when there is no carrier in the channel, but allow it to pass when there is a carrier in the channel. This value needed to be modified based on the RTLSDR used and the RF gain setting of the RTLSDR.

The next block is the FM demodulation block. The only significant parameter here is the cutoff frequency of the low pass filter, which needs to be beyond the maximum bandwidth required for the signal to retain its information. For FSK this is the frequency deviation (from the center frequency) plus one half the symbol rate.

The next and final significant block is the custom RPi Rx block. For this block, it is basically important that all of the parameters match the parameters that the transmitter is using.

In addition, a very important check is whether or not the sample rate, decimation, sample per symbol, and symbol rate make sense together. Basically, this relationship must be true for the receiver to work correctly.



Where Rs is the symbol rate, fs is the sample rate of the RTLSDR, d is the decimation of the FM demodulator, and s is the samples per symbol used for the custom block.

**3.3 Packet and Header**

Once the receiver and transmitter both were at the same point, there needed to be a way to control how data was interpreted. The first step was to break data up into packets to allow control over groups of data and to achieve asynchronous transmission, which by definition cannot be continuous. Next, a header was added to the transmitted packets. Most transmission schemes, wireless and wired, send packets with some form of a header. It allows for addressing, options, organization of data, and error checking.

The packet header not only provided better functionality like addressing, but also it was an easy fix to throwing away bad packets. For most transmissions at relative higher data speeds, around 80k Sps, there was consistently a bad packet in the beginning of transmission with either all zeroes or a short line of ones. We hypothesized that these packets are the result of the set up process for the DMA and the clock connect functions that occur before any actual data is transmitted or similar transients in the Raspberry Pi. These didn’t happen at low speeds, so it is hard to discover what is actually causing the error. The layout of each packet and the header is shown below.

**Packet Layout**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Start Bits (2) | Header (24) | Extra Header  (Default 0) | Payload  (PKT\_LEN - 24) | Stop Bit (1) |
| 1,0 | bits 2 - 26 | 27 - (27 + Extra Header) | 27 - (PKT\_LEN - Header) | 0 |

**Header Layout**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| IP Destination (0-7) | Packet Number (8-10) | Extra Header Length (11-15) | Flags (16-20) | CRC (21-23) |
| XXXX - XXXX | XXX | XXXXX  (Bytes) | XXXXX  (Bit Flags) | XXX |

The packet layout and the header are pretty straight forward when compared to other packet headers. Our unique section is the extra header length. Since making headers and packets variable lengths requires a more dynamic memory allocation for the transmitter, an extra header length was used to pass more header data in the payload of the packet. This was used on the first packet of a file transfer to send the file name and was only intended to be used on the first packet of any transmission. The extra header data is put in front of the full payload, memory allocated based off of that length, then broken up into individual packets.

We used five flags to control how our receiver interprets or displays the data, they are as follows: File transfer, GRC output data, GRC display data as hex, GRC display data as characters, and a DSA page flag. The three middle flags control the flowgraphs data control while the first and last flags have other functions programmed in.

The final three bits of the header is a CRC. This CRC is only for the first 21 bits of the header. It does not include any extra header length or payload. This made a big difference on the receiver side, because it was now able to throw away any miscellaneous or out of sync packets before all the data was interpreted. Another CRC could be added later at the end of the packet.

**4. Enabled Applications**

The Raspberry Pi is capable of complex computing and can create a more complex transmitter following the stable FSK transmitter developed for this project. The transmitter has been set up to be very flexible and include a number of different options upon function call or via configure commands sent over TCP. These variables include the center frequency, modulation rate, bits per symbol, frequency deviation of the carriers in FSK, and so on. Below, the applications that were implemented as part of this project are discussed.

**4.1 File Transfer**

One of the first applications that was implemented and frequently used to verify that the system was setup correctly was file transfer from the RPi or computer to another computer. File transfer required very little additional code to be written other than the transmitter/receiver code that was already complete. After successfully testing the file transfer, in terms of the transmitter code stability and robustness, any front end can pass data to the transmitter.

The code works by reading in a file 1024 characters at a time, converting them to binary and storing in a long integer array. This binary data and its length can be passed to any of the four transmit functions. The receiver block will identify that the “write to file” function flag is set, and will then write each character it receives in the payload to the designated text file. The filename is transmitted as the extra header section in the first packet. The receiver interprets this data and creates a new file with the correct lengths in a predefined directory. This application has been proven to work without error with an eighty line file that repeated “Hello World” followed by a line number.

**4.2 Instant Messaging**

A very simple GUI was created that enables two users on independent networked computers to send text messages back and forth. The GUI executes a GNU Radio receiver which it retrieves data from via a TCP socket every second. If anything is in the socket, the characters are displayed on the screen. Below, there is a separate window that the user can input a message and send it by clicking on the send button. The GUI will then read the text that the user typed and send it to the RPi TCP server, where it will be transmitted. This application was coded using wxpython. Also, any configure command can be sent from this GUI to the transmitter using methods described in the TCP section.

**4.3 DSA**

A simple Dynamic Spectrum Access algorithm based on sensed channel power can be implemented using this setup. The algorithm surveys all possible channels using the RTLSDR. Once the channel powers have been measured, a packet is transmitted. After each transmission the channel power is again measured with the RTLSDR to verify that it is still empty. The basic operation of this algorithm has been tested. The transceiver will correctly pick the channel on which to transmit, and when another transmitter is detected it shuts off and looks for a different channel. A TCP command can follow the channel selection to tell the transmitter what channel to transmit on.

**4.4 Micro Cognitive Radio Network Testbed**

Micro Cognitive Radio Network Testbed (MICRONET), is a small scale, low-cost testbed intended for education, demonstration, and experimentation of various SDR and CR concepts and technologies. This testbed is essentially a large number of Raspberry Pi’s and RTLSDR’s connected to a single central server, which controls their operation. Clearly there is a great parallel between this project and the work done for MICRONET. The addition of digital transceiver capabilities to this testbed would make it much more effective and useful than before.

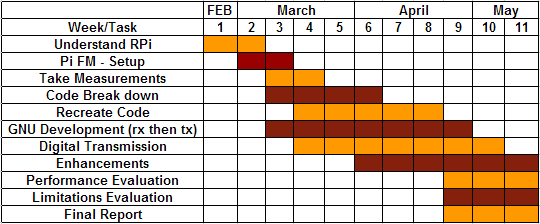
**4.5 Rc Car Control and Other Functions**

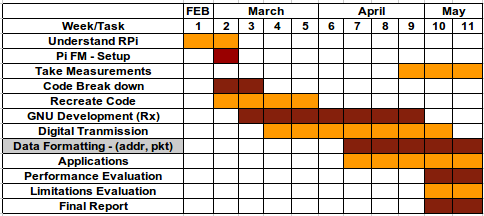
In proof that the transmitter can easily be adjusted for a number of uses, we worked with another team in the class working on controlling a pre-built RC car from Target, without modifications. This team, Francis Nguyen, Doug Hogan, Deirdre Beggs, gave us the OOK modulation rate, (1/0.00057) b/s, carrier frequency, 49.8 MHz, and data vector to define a command. The global modulation rate and carrier frequency were easily set to these values and the data was then passed to the DMA OOK or OOK transmit functions and we successfully commanded the car to go forward. No other commands were tested, just proof of concept for the RC car control.

Any application can use the provided functions for the file transfer and TCP transmitter which can break down bytes to binary data and then pass this data on for easy transmission.

**5. Review of Work Plan**

The proposed schedule and completed schedule are both shown below and on the next page. Clearly, there were a number of deviations resulting from different tasks being more or less complicated than anticipated. Initially the capabilities and limitations were not feasible. They were discovered after development and testing which manipulated our areas of development. The time required to understand the RPi, break down the PiFM code, and rebuild it took much less time than anticipated. Work on the transmitter and receiver were fairly consistent with the plan, and spanned most of the term. The performance evaluation, limitations evaluation, and measurements were all pushed back to the very end of the term. Also, another category for data formatting was added to include packetizing and the work for implementing a header.





**5.1 Deviation From Proposal**

At the time the proposal was written, the team had limited knowledge of the Raspberry Pi and the PiFM software. Unfortunately some of the proposed enhancements were not feasible. Initially, the hope was to use the raspberry pi as a standalone SDR transceiver with no need for external control or processing. This turned out to be much more complicated than first thought because of the complexity of running GNU Radio on the RPi. With some additional steps, the raspberry pi is able to compile and run GNU Radio. Unfortunately it is unable to execute any flowgraphs that implement any real DSP such as FM demodulation, FIR filters, etc. Since our receiver is built in GRC, this means that the RPi could not function as our complete receiver. Despite this limitation the RPi can still be used to receive the data from an RTLSDR dongle, and forward it to a central processor to be interpreted via TCP sockets. The advantage of this over using an RTLSDR connected directly to the processing node is that the receiver and transmitter can be packaged as a unit and placed anywhere on the LAN rather than being tethered to the central processor. Using this approach, any number of these SDR transceivers could be placed throughout a LAN, provided the central processor was powerful enough to handle processing all of the received data.

After testing the capabilities of the RPI as a digital OOK and FSK transmitter, OOK had to be scrapped because of hardware limitations. At merely 100 b/s (bits per second) the on and off times were not consistent. The OOK transmitter and receiver worked but could not compete with the very capable 200k M-FSK transmitter and receiver. This limitation results from the RPi’s inability to instantaneously connect or disconnect the GPIO pin. Instead it waits until the end of a cycle to cut off the pin safely. A cycle is based on the fraction used by the PLL.

**5.2 Major Obstacles**

A major obstacle on the receive side, as mentioned before, was determining how to correctly retrieve the symbols from the baseband signal. Synchronous approaches were investigated but proved unfruitful and so an asynchronous approach was adopted.

Another issue that was encountered in this work was in debugging issues when the transmitter/receiver were not working correctly. It became increasingly complex as more code was written for both sides of the radio. There were several occasions where it was unclear whether the issue lied with the transmitter or receiver, which made it very difficult to diagnose.

Finally, it was a challenge to properly measure the spectrum of the transmitted waveform to verify FCC part 15 compliance. None of the team members had any experience with measuring the power of a signal as it propagates through the air. There was also limited equipment for this sort of measurement. Other methods based on theoretical antenna gains and propagation models were used instead.

**5.3 Limitations**

Most of the limitations that were encountered in this work have already been discussed. For the sake of understanding them all at once, they will be restated here. The first major limitation was that the OOK transmitter was limited by its inability to instantaneously disable the GPIO pin, which led to timing issues in the waveform. A limitation of the receiver was that the GNU Radio flow graph had to be run on a higher performance computer rather than directly on the RPi due to its limited processing capabilities. Another limitation that is discussed later is the transmit distance, which was significantly decreased when a new antenna was used to meet FCC compliance.

The transmitter was more powerful than the receiving computer that was trying to real time decode the data. Trying to push our peak data rate was limited by the receiver computing power. Laptops were used as the receiver instead of a powerful desktop computer. The transmitter’s actual data rate could be higher than shown later, just limited testing due to the receiver.

**6. Performance Metrics**

The following section describes tests carried out to measure the performance of the digital transmission system we created. The Raspberry Pi was used as the transmitter and GNU Radio with the RTLSDR as the receiver. Default testing for development was at a modulation rate of 80kb/s, 1 bit per symbol, 128 MHz, with frequencies 50kHz on either side of the carrier. Tests using a 200ksps (symbols per second) M-FSK transmitter and receiver were successful. The transmitter is flexible and can choose a variety of frequencies and modulation rates, specified by the user at run time or through TCP socket messages. Packet lengths varied from 32 bits to a 8kb packet. At short distances, we successfully received full 8 kS packets without error.

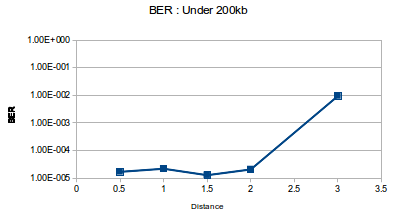
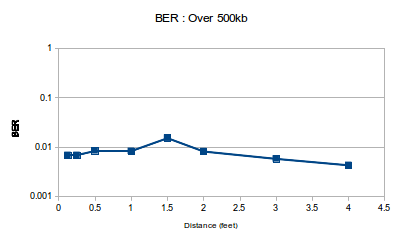
**6.1 Max Throughput**

While testing the maximum throughput of this system, one issue that encountered was the computing power of the laptop receivers. As it turns out, the custom block in GRC is extremely computationally expensive. When running the system at 200 kS/s, the receiver would correctly receive the first two or three packets, but would then get bogged down and shut off. In order to more accurately define the maximum throughput of this system, a more powerful desktop computer was used. Once this change was made, 200 kS/s and even 300 kS/s worked without any issues. The system did fall apart at 400 kS/s though, with only the very first data being received correctly followed by a large number of errors. The maximum throughput is then reported to be 300 kS/s. Using two bits per symbol this can achieve a throughput of 600 kb/s.

**6.2 BER**

One of the more important metrics of this transmitter/receiver is what bit error rate (BER) could be achieved. To test this metric, a pseudo random sequence of 1000 bits was generated and written to a text file. These bits were transmitted in a 1024 packet, with the other 24 bits being the header. On the receive side, the same file was read into an integer array and as each bit was received, it was compared to the appropriate bit. The transmit code was written so that the same packet would be transmitted continuously until the program was closed. This allows a large number of bits to be received, which is important in determining an accurate BER. The transmitter was able to put the packet in the DMA and then sleep while transmitting, using almost 0% CPU usage.

There was however an error in the measurement that would occur every test shortly after 200 kb were transmitted. Before this point, the BER would be extremely low, with only a few total errors. But somewhere around 200 kb, the number of errors would jump up by several hundred in a single packet. Unfortunately, there was not sufficient time to determine the cause for this problem. Data was taken nonetheless, with the understanding that it was not as accurate as it should have been. Below there are two graphs, both showing BER as a function of distance (in feet) between the transmitter and receive antenna. The first graph shows the BER when less than 200 kb were transmitted, the second when more than 500 kb were transmitted. Several important things to note are that the BER in the first graph increased with distance in a manner that should be expected based on the decrease in SNR from the increased distance. Also, the error, identified above, is clearly visible in the second plot. As the BER is several orders of magnitude higher than the first plot.



**6.3 Tx Range**

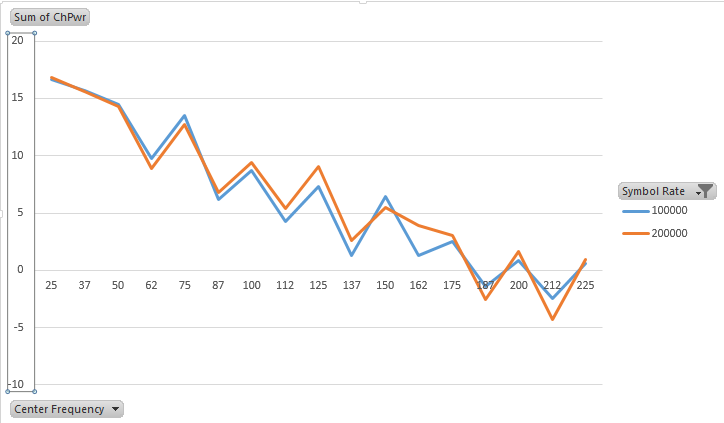
Using the short antenna that was required to meet FCC regulations (according to calculated transmitted power), the maximum distance at which the system was still able to function was approximately three feet. At this distance there are still a non-trivial number of errors. Using the original antenna that was much longer (approximately 16 cm), the transmit range was greatly increased. The maximum distance using this antenna was never explicitly measured since it would not be the final implementation, but the system worked correctly from across a classroom of about 30 feet.

**6.4 Bandwidth vs M-FSK**

When changing the number of bits in each symbol, the bandwidth changes proportionally. We used a default frequency deviation of 100 kHz, but the transmitter can use any different frequency for the deviation. When the number of b/S increased from 2 to 3, the bandwidth increases from (±150) 300 kHz to (±350) 700 kHz, not including the effect of the data rate. This is more than double the band width without a sufficient data rate increase. We decided to not test/use more than 2 bits per symbol because of this.

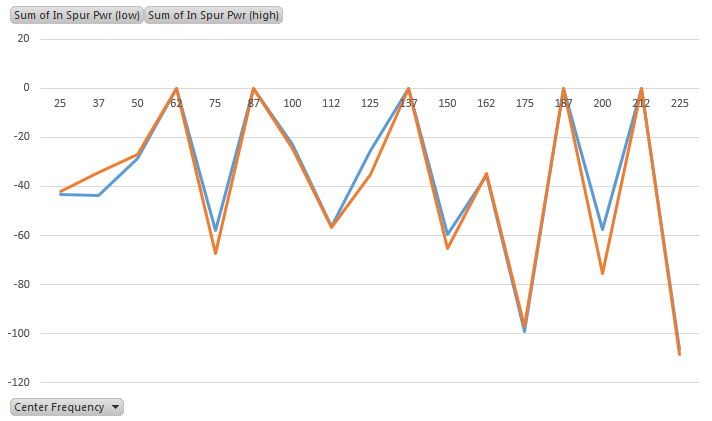
**6.5 Spectral Analysis**

The purpose of measuring spectrum is to find which frequencies perform the best; the fundamental frequency that has high channel power at fundamental frequency and low power at spurs. Also, based on those fundamental frequencies, we can calculate field strength to compare whether it is possible to meet the FCC part 15 regulation. We measured transmitter power directly from output pin with setup of 200000 and 100000 symbol rates, FSK, and range of 25~225 MHz frequency. The span for in-band spurs is 3 MHz and for out-band spurs is 50~300 MHz. Because at some fundamental frequencies, the out-band spurs appears wider than others.

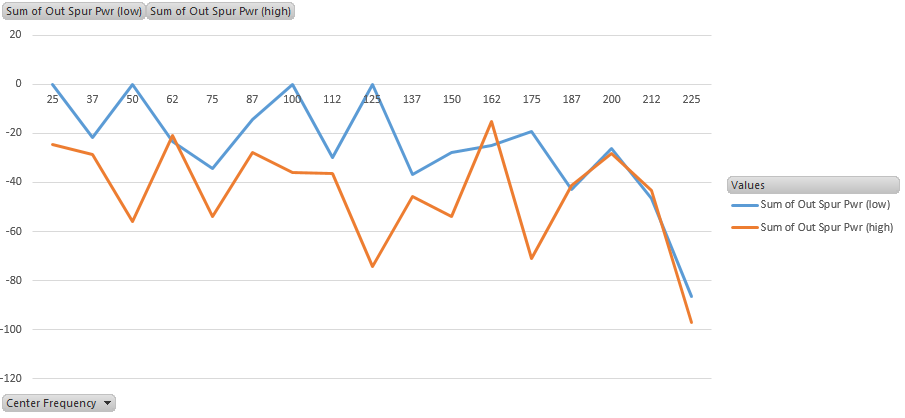


<Channel Power at Fundamental Frequencies>





<In-band Spurs Power>



<Out-band Spurs Power>

Based on these three graphs, we concluded that 75, 150, and 175 MHz are the good fundamental frequencies to use because the carrier power is at a local peak and both, the in band and out of band spurs, are at local minimums.

**6.6 FCC Regulations**

The Federal Communications Commission (FCC) has rules to limit transmitter power for harmful interference to licensed transmitters. In the regulations, the transmitter power is listed by different frequencies and different purposes. When we run the Raspberry Pi transmitter, our transmit power should be lower than the regulated power at 3 m. Since 75 and 150 MHz regulated as ‘Spurious Emissions Only’, we decided to use 76 and 156 MHz. Under regulation, our transmitter categorized as ‘Periodic Transmissions’, and for 76 MHz, power should be under 500 uV/m @3 m, for 156 MHz, power should be under 977 uV/m @ 3m, and for 175 MHz, power should be under 1500 uV/m @3m.

For accurate calculation, we do not know with 100% certainty, for the antenna shorter than the quarter-wave antenna, how to determine the radiated power. Therefore, we measured channel power with quarter-wave antenna with 3rd harmonics. The reason of measuring power at 3rd harmonic is that each 3rd harmonics has less power than the fundamental frequencies but still can be strong enough for transmission. Only some filtering would need to be applid to block out the fundamental.

The calculations for the quarter-wave antenna are shown below.

• Pwr [W]= 10^((Pwr [dBm]-30)/10)

• G= 10^((G [dBi])/10) (G[dBi] = 5.19 dBi for quarter-wave antenna)

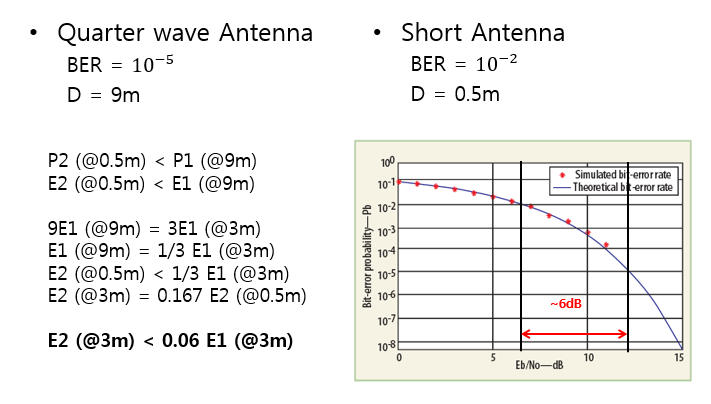
• EIRP=P[W]∗G

• E= √(30∗EIRP)/D (D= 3m for FCC regulation)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Freq [MHz] | Ch Pwr [dBm] | Pwr [W] | E [V/m] | FCC [V/m] |
| 75 | 6 | 0.004 | 0.20976 | 0.0005 |
| 150 | -1 | 7.963e-4 | 0.09359 | 0.000977 |
| 175 | -2.6 | 5.5e-4 | 0.07778 | .0015 |
| 225 (3rd) | -6 | 2.512e-4 | 0.05257 | .0015 |
| 450 (3rd) | -15.8 | 2.63e-5 | 0.01701 | .004667 |
| 525 (3rd) | -25.2 | 3.02e-6 | 0.00576 | 0.005 |

<Table of Field Strength at Fundamental and 3rd Harmonic Frequencies>

The frequencies marked with (3rd) are the third harmonics of the three fundamental frequencies shown in the beginning of the table. Based on table, none of frequencies with quarter antenna follow FCC regulation. However, 3rd harmonic of 175 MHz, 525 MHz, has field strength close to the FCC regulation. To obtain an antenna length that is within regulation, we estimated the field strength at the furthest distance we could receiver/transmit. Then, using the BER, we compared the SNR at that point. The next page shows and explains the process we used.



BER Graph originally from [13]

To reduce the field strength within regulations we had to decrease the size of our antenna. Since the field strengths are proportional to distance we can set up a ratio and estimate our field strength for a smaller antenna. Based on calculation that placed shown above, E2, field strength of short antenna, will have upper limit as 0.06 ratio of E1, field strength of quarterwave antenna. Unfortunately, this is only a maximum power at 3 meters comparison which was still above the regulations.

Additionally, to prove short antenna has lower field strength than quarterwave antenna is using BER vs SNR graph. We measured BER for quarterwave antenna and short antenna (~5 cm). For quarterwave antenna we got BER of 10-5 at distance of 9m,and for short antenna we got BER of 10-2 at distance of 0.5m. Based on the graph, the difference between BER of quarterawave antenna and short antenna is more than 6 dB. Therefore, we can conclude that short antenna will have lower field strength than quarterwave antenna. This 6dB reduction with the shorter antenna puts us below the FCC regulation. This limits the Raspberry Pi to a point where it can no longer transmit more than 0.5m while the FCC regulation are considered. Using a better antenna, filters, and a proper measuring technique, a more accurate/powerful antenna could be found

**7. Potential Further Developments**

The following section describes more applications or improvements that was not able to be completed. These improvements are feasible and some may only require minimal amount of work.

**7.1 Analog Transmission**

This project has been based off of PiFM which transmitted an analog FM signal. It was later modified to transmit digital data. With some modification our transmitter could be able to transmit analog signals again. Some examples could be to transmit the original FM wav transmission or other analog signals.

**7.2 Variable Packet/Header Lengths**

Part of the header includes a section that allows additional header information to be appended as needed. Currently the only use for this additional header is to send a file name to the receiver so that it knows what to call a file before writing it. Other possible uses of the additional header could be other control information, such as a pattern to use for spread spectrum frequency hopping, security information, or updates on performance to be used in some low level cognitive engine.

Different memory allocation methods can be used to control the length of the packet and the header, eliminating the confusion of our “Extra Header Length”. There is room for a longer header to be implemented in the packet or even a longer packet. We chose to use a finite size so that a number of packets could be allocated all at once.

**7.3 Error Detection, Prevention, and Correction**

Based on the BER measurements taken (below 200 kb), it appears that the transmitter/receiver combination has reasonably high fidelity as they currently exist. As of now, there is only error detection for the header. A much better BER should be possible if more advanced forward error correction (FEC) algorithms were used for the payload data. The tradeoff would be that more of the information bits would be used for error correction rather than for actual data transmission, decreasing the throughput.

Similarly, a type of automatic repeat query (ARQ) could be used in conjunction with error detection to handle errors in the received data. The simplest form of ARQ is known as stop and wait ARQ, where each packet must be acknowledged before the next is transmitted. This method can be highly inefficient though, so other methods are often implemented such as Go Back N ARQ or Selective Repeat ARQ [4]. Some combination of these two error correction strategies would make this system much more robust to errors.

**7.4 DSA Improvement**

The current DSA application controls the receiver and transmitter from a single process, which acts as a sort of backbone network. In this case, the transmitter and receiver are not independent from one another. Some thought was put into how two transceivers might actually operate independently using the same basic DSA approach. The method that had been considered most was for one of the radios to be a master and the other a slave. The master would decide which frequency the two radios are to use based on received signal strength in the channels. It then begins to transmit a DSA page in this channel. The slave, is constantly polling each of the channels, searching for DSA pages. Once it detects a page, it transmits back to the master immediately after the end of a page. The master detects the acknowledgement and then communication can occur. In order for this to work the DSA pages would need to be adequately separated in time so that the slave would be able to send an acknowledgement back, accounting for the round trip time between the two radios. This approach should certainly be feasible. Had there been more time, there is little doubt that this would have been a success. For now this remains a potential future improvement of the system.

**7.5 ASK**

It may also be possible to implement Amplitude Shift Keying with more levels than simple on or off as in OOK. In order to achieve this, some kind of digitally stepped attenuator or amplifier would need to be used. Some number of the other GPIO pins available on the RPi could be used to set the gain/attenuation of this device, giving direct control of the amplitude of the transmitted signal. There are plenty of GPIO pins available for control as well as other methods like serial comm ports or USB ports.

**7.6 Other Improvements**

Two major downfalls to the transmitter code on the Raspberry Pi is its memory efficiency and CPU usage. The DMA no longer helps with the CPU usage, unless repeating for BER tests, only for accurate symbol rates. To create a transmitter that was simpler to use and would work in a relative high rate digital environment, control over the transmitter must be very accurate. Certain loops that check and wait until the DMA has enough free space to safely write another packet to its sample space use up CPU usage. Sleeps could be put there for quarter of a packet length to reduce CPU usage from constantly checking the DMA current position.

We use an int array to hold the symbols that are being transmitter which limits the size of the array. Typically this array just holds 1’s and 0’s using only 1/32 of its possible memory storage. Either shorter variables can be used or a method for masking bits into sequential bit locations for each int.

**7.7 Filtering**

One of the issues with using a clock signal as an RF carrier is that clock signals are extremely spurious. Particularly at harmonic frequencies but other spurs across the spectrum exist from the way that the fractional PLL works. To be truly compliant with FCC standards, there would need to be some sort of filtering on the signal. If harmonics are being used as the carrier than a high pass filter may suffice. Otherwise, a bandpass filter would most likely be required.

**8. Conclusion**

In conclusion, this project has demonstrated the capabilities of the Raspberry Pi as a low cost SDR transmitter. The Pi Coupled with an RTLSDR and the code written for this project, a full SDR transceiver can be easily implemented. This transceiver may find use in a number of applications beyond what has been presented here. Its current performance is satisfactory for what has been required, achieving reasonably high throughput and fidelity (600kb/s - 300kS/s). That being said, there are many ways that performance might be improved including the ideas discussed earlier in this report. We hope that our transmitter and receiver code is stable enough for other people to come along and use for any digital application they want to implement.

**9. References**

[1] "What is a Raspberry Pi?" Raspberry Pi. RASPBERRY PI FOUNDATION, n.d. Web. 05 May 2014. <<http://www.raspberrypi.org/>>.

[2] Verry, T. (2012). What is the raspberry pi? *ExtremeTech.Com,* Retrieved from <http://ezproxy.lib.vt.edu:8080/login?url=http://search.proquest.com/docview/1009161165?accountid=14826>

[3] Holenarsipur, P. (2008). I'm OOK, you're OOK? *Electronic Engineering Times,* (1544), 41-42. Retrieved from h[ttp://ezproxy.lib.vt.edu:8080/login?url=http://search.proquest.com/docview/208078116?accountid=14826](http://ezproxy.lib.vt.edu:8080/login?url=http://search.proquest.com/docview/208078116?accountid=14826)

[4] Forouzan, Behrouz A. *Data Communications and Networking*. 5th ed. New York, NY: McGraw-Hill, 2012. Print.

[5] "A Comparison Between OOK/ASK and FSK Modulation Techniques for Radio Links.” Darrell L. Ash, RF Monolithics, Inc., Dallas, Texas. Web. 4 May 2014. <<http://www.rfm.com/products/apnotes/ookvsfsk.pdf>>.

[6] "Create an Internet Radio FM Transmitter." Weed Pi - All about Raspberry Pi. Weed Pi, 2014. Web. 06 May 2014. <<http://www.weedpi.com/how-to/create-an-internet-radio-fm-transmitter/>>.

[7] Suehle, Ruth, and Thomas L. Callaway. *Raspberry Pi Hacks: Tips & Tools for Making Things with the Inexpensive Linux Computer*. N.p.: O'Reilly Media, n.d.*Safaribooksonline*. O’Reilly Media, Inc., 12 June 2013. Web. 5 May 2014.

[8] Hapenciuc, A.; Svasta, P., "Long range image radio transmitter," *Design and Technology in Electronic Packaging (SIITME), 2013 IEEE 19th International Symposium for* , vol., no., pp.99,102, 24-27 Oct. 2013

URL: [http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6743652&isnumber=6743631](http://ieeexplore.ieee.org.ezproxy.lib.vt.edu:8080/stamp/stamp.jsp?tp=&arnumber=6743652&isnumber=6743631)

[9] "Raspberry Pi SSTV Camera." Agri Vision -. N.p., 19 Apr. 2014. Web. 06 May 2014. <http://www.agri-vision.nl/CMS/index.php?option=com\_content&view=article&id=54#Links>.

[10] Skagmo, Jon P. "PiHAT - Raspberry Pi Home Automation Transmitter." Skagmo.com -. N.p., n.d. Web. 06 May 2014. <http://www.skagmo.com/page.php?p=projects/22\_pihat>.

[11] Cass, S., "A $40 software-defined radio [Resources\_Hands On]," *Spectrum, IEEE* , vol.50, no.7, pp.22,23, July 2013

URL: [http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6545114&isnumber=6545094](http://ieeexplore.ieee.org.ezproxy.lib.vt.edu:8080/stamp/stamp.jsp?tp=&arnumber=6545114&isnumber=6545094)

[12] Nguyen, Gam D., "Fast CRCs," *Computers, IEEE Transactions on* , vol.58, no.10, pp.1321,1331, Oct. 2009

URL: [http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5089317&isnumber=5226180](http://ieeexplore.ieee.org.ezproxy.lib.vt.edu:8080/stamp/stamp.jsp?tp=&arnumber=5089317&isnumber=5226180)

[13] Ismail, W., Kadhum, E., Mandeep, J. S., & Hamid, H. S. (2009). Analyze BER performance of wireless FSK systems.*Microwaves & RF, 48*(11), 80-83,85-89. Retrieved from <http://ezproxy.lib.vt.edu:8080/login?url=http://search.proquest.com/docview/217124153?accountid=14826>

[15] Yu-Yu Lin; Kwang-Cheng Chen, "Asynchronous Dynamic Spectrum Access," *Vehicular Technology, IEEE Transactions on* , vol.61, no.1, pp.222,236, Jan. 2012

URL: [http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6095384&isnumber=6129945](http://ieeexplore.ieee.org.ezproxy.lib.vt.edu:8080/stamp/stamp.jsp?tp=&arnumber=6095384&isnumber=6129945)

[16] Walton, Sean. *Linux Socket Programming*. Indianapolis, IN: Sams, 2001.*Safari Books Online*. 26 Jan. 2001. Web. 6 May 2014. <http://su8bj7jh4j.search.serialssolutions.com.ezproxy.lib.vt.edu:8080/?sid=sersol&SS\_jc=TC0000075995&title=Linux%20socket%20programming>.

[17]Hongjin Kim, SoYoung Kim, and Kang-Yoon Lee, "Low power FSK transmitter using all-digital PLL for IEEE 802.15.4g application", Analog Integrated Circuits and Signal Processing, vol. 74, no. 3, pp. 599-612, Mar 2013.

URL: <http://link.springer.com.ezproxy.lib.vt.edu:8080/article/10.1007%2Fs10470-012-0020-8>