

Mobile Computing (EEC 687) Project Report 1

Two channel transmitter/receiver

Robert Fiske and Malav Shah Department of Electrical and Computer Engineering CSU

Abstract

Software Defined Radio (SDR) is an emerging technology which supports different types of Radio Designs. It transfers all the signal processing and hardware complexity to software in wireless transmission and reception. Lots of Development is going on using GNU radio open source software and USRP hardware for transmission and reception on this. Based on this our plan is to implement Man made radio that is going to transfer 2 audio files and on receiver side we are going to capture, select and play that file.

Introduction

Software Defined Radio is the radio communication system where components implemented in hardware are implemented in software (ex. Mixer, filters, demodulator etc). It consists of PC with sound card, USRP (mother board and its daughter boards) and software GNU Radio. All signal processing is handled by general purpose processor rather than hardware. This design produces a radio that can receive and transmit a different form of radio protocol just by running different software. SDR are used in lots of military and cell phone services because both required changing in variety of protocols with time.

Explanation of USRP and GNU radio is given below.

USRP

Universal Software Radio Peripheral

- RF Front End Daughter Boards
- ADC-DAC
- User Programmable FPGA
- Programmable USB 2.0 Controller

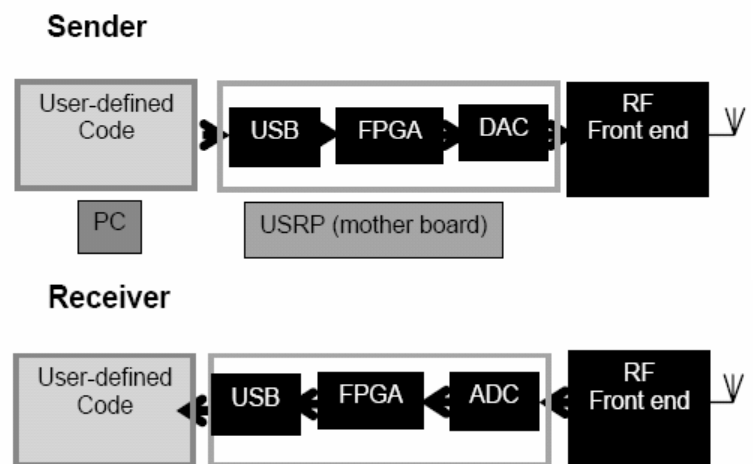
USRP motherboard contains USB 2.0 controller, FPGA and DAC-ADC. We need RF Front End Daughter to communicate with antenna Detailed figure is below in next page. (USRP with general purpose processor)

Block Diagram of sender and receiver is Explained below.

Sender: In our PC we have GNU driver to Transmit a file to USB. We are going to transmit a file from PC (using GNU

Radio Driver) to USB controller. The output goes to FPGA. FPGA digitally up converts the signal and gives input to DAC. DAC converts the digital signal to real world (Analog) signal and transmit it RF Front End. RF Front End changes that frequency to match with antenna input and transmit it to antenna. Antenna transmits that signal in the air.

Receiver: On Receiver side antenna captures the signal and transmits it to RF Front End. RF Front End changes the frequency to match it with ADC and give input to ADC. ADC converts the real world (Analog) signal to digital form and gives input to FPGA. FPGA digitally down converts the signal to match with USB USB transmits it to PC. In our PC we have GNU Radio Driver used to demodulate the received signal.



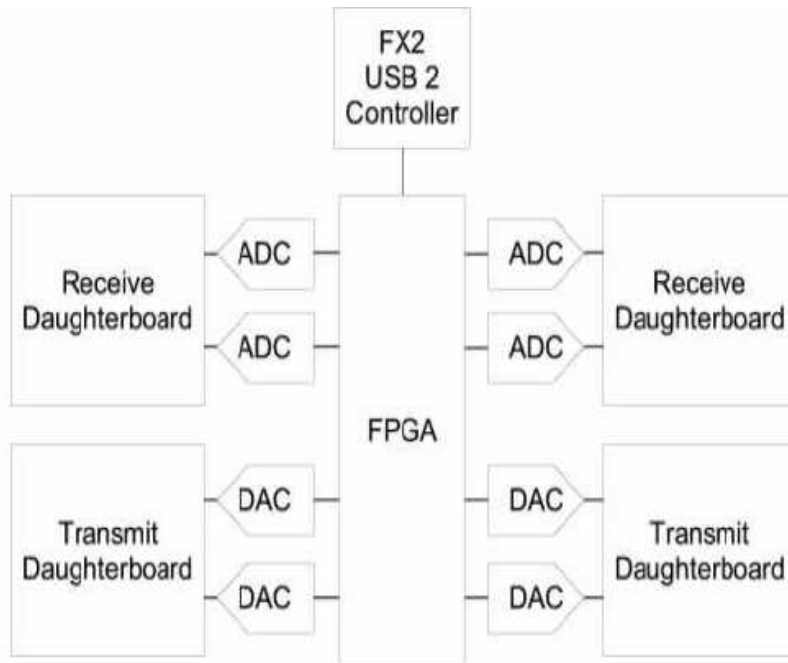


FIGURE : USRP on general purpose processor

GNU Radio

- Open source software toolkit for building software radios
- Defines the transmission waveform and demodulates received signal

GNU Radio Consists of

1. C++ classes which implement different signal processing functions
2. Python implement main application programming
3. SWIG works as glue between C++ classes and Python language. It is a Linux package that converts the C++ classes into Python compatible classes
SWIG allowed C++ code to be used for processing data since it is faster

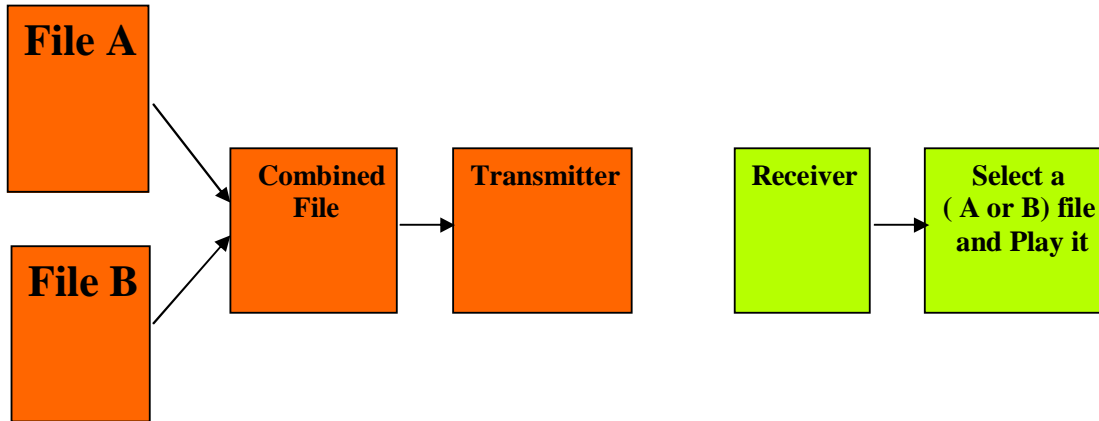
Project Overview

Use USRP and GNU Radio to transmit two audio signals from a server to a client. The client acts as a basic radio receiver, while the server acts as a radio station transmitting two separate "stations" on one set of hardware. This allows one set of hardware to act as two separate radio stations allowing for cheaper use of hardware.

Background

The GNU Radio/USRP option was chosen over the iPAQ device because it appeared to be more interesting to work with a project at a lower level, while working with iPAQ appeared to be more of an application layer device. The radio transmitter/receiver was chosen because its application seemed interesting, using this device could allow the development of a home wide radio station, allowing a server PC to stream multiple audio styles throughout a home, while receivers could be portable (or even simple radio devices) allowing a person to set a play list and listen to their preferred music while another person could do the same. The main requirement this project fulfills is, if we had access to the hardware at home, would it be something that we would find useful. It as felt that a home media system such as the proposed radio transmitter/receiver would convenient to have at home.

Block Diagram



Block Diagram Explanation

As shown in figure we are going to transmit 2 files. Say File A and File B. To transmit them we are going to combine them with the help of Decoding scheme. On Receiver side we are going to capture that files and select one of that files. After that we are going to split that files and play one of them.

Tasks

Project is divided in four basic tasks

1. Modify Receive file from USRP according to our needs
2. Add a C++ functionality to transmit signal
3. Find Decoding scheme to transmit 2 audio files and implement it
4. Capture that signal on receiver, select a file play that file

Decoding Scheme

- We are going to transmit 2 files and on receiver side we choose one of them
- To transmit them we are going to combine these files bit by bit
- For Ex. File A = First nibble A1 A2 A3 A4
- File B = First nibble B1 B2 B3 B4
- Combined Transmission file say File C
- First byte of File C would be A1 B1 A2 B2 A3 B3 A4 B4

Receiver (Code)

Receives packets from the channel, if the packet is received error free and the error detection version of the code is used an ACK packet is sent to the transmitter. The payload is then extracted from the received packet, the data is split into two separate arrays in C the code. The python code then calls a C function to save these to their respective files. Upon detection of an eof flag (0x01 followed by either 0xFD 0xFE or 0xFF, depending on which file(s) have ended) the python code calls the C code to close the previous file, and open a another. With error detection an incorrect packet reception causes the transmission of a NACK packet. Duplicate packets still generate an ACK but are not processed by the receiver in the error detecting version.

Transmitter (Code)

Opens two files, one called playlistA, the other playlistB, it then reads file names from these files (and loops back to the first when it reaches the end) the two current files then read either packet_size/2 bytes or read the maximum data left in one of the files (and the same amount of data from the other). The two data sets are then combined in the C code and transmitted after adding the header. The program then goes into an infinite loop. Upon reception of an ACK packet a flag is set which breaks the infinite loop and the program begins setting up for the next packet. Upon reception of a NACK, or a corrupted packet a retransmission takes place. Also if no reply is received a timeout occurs and the packet is retransmitted.

Note: packet_size is not the actual size of transmitted data, packet number header, as well as any control flags will add on to this size.

Problems faced and their Solutions :

1. *Understanding of GNU Radio*

Solution: GDB to follow the execution of Programs

2. *GNU Radio Example worked but USRP would not*

Solution : Looked at config.log and found that the usrp component of gnuradio had not been compiled, using config.log tracked down missing dependencies and recompiled gnuradio.

3. *Treatment of character arrays as null terminated strings by python*

Solution: Decoding scheme change according to it. 0 is 01 followed by negative count and 1 is 1 followed by number of 1s in that byte. By this was we got the solution of this problem. After getting the encoding scheme working implemented the changes into benchmar_rx.py and benchmark_tx.py.

4. *Playback of the audio file at receiver was too slow*

Solution: Switched project to file based approach (could find the reasons behind it)

5. *Wireless communication tried but PDR was 0%*

Solution : we tried using RFX 2400 transceiver boards which we found had a PDR of a bit more than 50% at it's highest, the RFX 1200 seemed to produce the best results. Still prone to errors however a second version of the code was written to add ACK/NACK to allow for guaranteed transmission

More Details about this in Appendix B

Functions Designed during Project work

1. compare two binary files pointing out where they differ
2. using a lookup table get the value that would be obtained when combining two alphabetic characters
3. generate a file containing a list of repeating characters
find any 0 values in a file.
4. extract the first X number of bytes from a file (used to make a file small enough to work with on paper)

Progress

1. Basic functionality is implemented; this code generally works if transmitted in an environment with low interference. Speed of transmission is good
2. Another version was created which can work with interference by using an ACK/NACK/Timeout system; this code creates better files in that all data is received successfully; however the overhead of this checking results in low throughput which significantly reduces the usefulness of the program if you wish stream the audio.
3. Each approach has it's own advantages/disadvantages, the performance of the error checking could be increased by using a sliding window protocol (instead of confirming every delivery)

References

- A. The class website, providing many links and recommendations.
http://academic.csuohio.edu/yuc/mobile08/08_week04_USRP.pdf
- B. The website of IEEE provides information on standards.
<http://spectrum.ieee.org/oct06/4654>
- C. The homepage of the manufacturer of the USRP hardware.
<http://www.ettus.com/>
- D. Provides documentation on the GNU Radio software package.
<http://www.wu.ece.ufl.edu/projects/softwareRadio/#Project%20paper>
- E. www.gnuradio.org/trac
- F. gnuradio.utah.edu/trac/browser/gnuradio/trunk/gnuradio-examples/python/digital/tunnel.py
- G. Ke-Yu, Chen, Zhi-Feng Chen, "GNU Radio,"
http://www.wu.ece.ufl.edu/projects/softwareRadio/documents/Project%20Report_James%20Chen.pdf

Appendix A (Glossary)

- **ADC** Analog to digital Converter
- **DAC** Digital to Analog converter
- **Daughter Boards** Daughter Boards with USRP for transmission and receive functions
- **DUC** Digital up-conversion
- **DDC** Digital down-converters
- **Demodulation** Demodulation is the act of removing the modulation from an analog signal to get the original baseband signal back.
- **FPGA** Field Programmable Gate Array
- **GNU Radio** Open source software toolkit for building software radios
- **Modulation** modulation is the process of varying a periodic waveform
- **Open source** is computer software for which the human-readable source code is made available under a copyright license
- **RF Front End** See Daughter Boards
- **SDR** Software Defined Radio
- **USB** Universal Serial Bus
- **USRP** Universal Software Radio Peripheral

Appendix B(Details of Work Done, Problems faced and its solutions)

▪ Downloaded gnuradio-3.1.1, successfully compiled, but found gnuradio would not work. Found the error to be cause by the environment variable PYTHONPATH not being set. After fixing this gnuradio exmaples would run, but usrp examples would not. Looked at config.log and found that the usrp component of gnuradio had not been compiled, using config.log tracked down missing dependencies and recompiled gnuradio. After getting the usrp demos to run began trying to receive signals using usrp_wfm.py. By touching the antenna to metal was able to receive a local Cleveland radio station. Next edited fm_tx4.py to transmit a sine wave to a radio, in order to do this a bug was fixed in fm_tx4.py. After fixing the bug found that gnuradio had released version 3.1.2, switched over to using this release for the project. Tried to transmit to a radio using tx_fm4.py, was never able to do this. Instead tried using usrp_wmf.py and fm_tx4.py together, was able to get this to work although the audio had severely low volume. We next turned my attention to the gnuradio code to try and get a feel for how it was working, don't remember the file name of the first file that we looked at extensively, but I believe it was similar to fm_modulate_cc.cc as well as the demodulate version. Was confused that the modulate function was called, but demodulate was not, decided trying to eyeball the code was inefficient, used cscope to find the main loop of the program, and then recompiled to add debugging support. Then used gdb to make a list of all the functions that were called, and the order in which they were called. After beginning to modify the gnuradio code found that audio playback not only had low volume, but was also severely slowed down, thought this may have been due to using the same usrp for transmission and receptions, after setting up two usrps at the same time however the playback was still extremely slow decided to switch to a file based approach. Monday in class learned from Eli about the narrow/wide band difference in usrp, this may have been the cause of the slow playback. After switching to file based approach decided to switch base programs from usrp_wfm.py and fm_tx4.py to benchmark_rx.py and benchmark_tx.py, this was done because a digital framework was already laid out in these demos and the audio card output of usrp_wfm.tx wouldn't be needed. Wrote C programs to combine given characters, as well as to split a given short. Looked for and found SWIG tutorials to see how to incorporate C code and python code. After successfully making some demo SWIG functions to test out the process began writing the algorithm to combine data in the SWIG code, then did the same for reception. At first used the C functions as normal C functions, including using pointers to save output, but found that this was not working, began writing functions to set and get data from C. Discovered that a lot of data was not being transmitted, realized this was because of data values of 0 being interpreted as the end of strings in the python code. This was discovered by printing out the sizes of packets. Devised a scheme to save the 0's as 1's as well as provide control flags to the code. To take transmission errors out of the equation implemented the new scheme using a file in place of outputting to the usrp. After getting the encoding scheme working implemented the changes into benchmar_rx.py and benchmark_tx.py. Although I don't remember at what various stages each program was done, throughout working on the project several utility programs were written to

analyze data, some of functions designed were:

- compare two binary files pointing out where they differ
- using a lookup table get the value that would be obtained when combining two alphabetic characters
- generate a file containing a list of repeating characters
- find any 0 values in a file.
- extract the first X number of bytes from a file (used to make a file small enough to work with on paper)

Initially we tried wireless communication using the BasicRx/BasicTx daughter boards but found that PDR was near 0% even for unmodified demo programs, after talking with Sachin and reading the README file we tried using RFX 2400 transceiver boards which we found had a PDR of a bit more than 50% at it's highest, the RFX 1200 seemed to produce the best results. Still prone to errors however a second version of the code was written to add ACK/NACK to allow for guaranteed transmission.

- Where it stands:
 - Basic functionality is implemented, this code generally works if transmitted in an environment with low interference. Speed of transmission is good
- Another version was created which can work with interference by using an ACK/NACK/Timeout system, this code creates better files in that all data is received successfully, however the overhead of this checking results in low throughput which significantly reduces the usefulness of the program if you wish stream the audio.

Each approach has it's own advantages/disadvantages, the performance of the error checking could be increased by using a sliding