Senior Design Project

*Report #1*

Project Title: Software Defined Radio
Sponsor: IPFW Wireless and Technology Center
Team Members: Jeff Tosch
               Ansel Aiken
               Simer Dasson
Faculty Advisor: Dr. Chao Chen
Date: December 14, 2009
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acknowledgment</td>
<td>Page 2</td>
</tr>
<tr>
<td>Abstract</td>
<td>page 3</td>
</tr>
<tr>
<td>Section I</td>
<td>page 4</td>
</tr>
<tr>
<td>Section II</td>
<td>page 7</td>
</tr>
<tr>
<td>Section III</td>
<td>Page 15</td>
</tr>
<tr>
<td>Section IV</td>
<td>page 19</td>
</tr>
<tr>
<td>Conclusion</td>
<td>page 21</td>
</tr>
<tr>
<td>References</td>
<td>page 22</td>
</tr>
</tbody>
</table>
Acknowledgment

We would like to extend our gratitude to our faculty advisor Dr. Chao Chen for her continuing support and dedicated guidance in the development and formulation of this project. We would also like to thank the IPFW Wireless Technology Center for giving us the opportunity to further discover the potential of a Software Defined Radio, also choosing us as students from the IPFW Engineering Department and Dr. Todor Cooklev for his feedback and recommendations throughout the semester. Without their contribution, this project would have never been fully realized. Dr. Carlos Raez is another honorable mention for all his timely and valuable input to our many questions. And finally, a thank you to the entire Department of Engineering faculty at IPFW for giving us the opportunity to earn and apply the prerequisite knowledge and education that was put to work during this project.

We would also like to acknowledge the companies listed below for their contributions in technical information and project components.
Abstract

A software defined radio (SDR) is an exciting new field for the wireless industry. A software defined radio system is a radio communication system where components such as mixers, filters, amplifiers, modulators/demodulators, detectors etc. that have typically been implemented in hardware are instead implemented using software on a personal computer or other embedded computer devices. Software defined radio uses programmable digital devices to perform the signal processing necessary to transmit and receive baseband information at radio frequency. Devices such as digital signal processors (DSPs) and field programmable gate arrays (FPGAs) use software to provide them with the required signal processing functionality. This technology offers greater flexibility and potentially longer product life, since the radio can be upgraded very cost effectively with software.

The students went through several steps to come to a design which they consider appropriate for their job in accordance with the requirements of the project. For the first half of the project, the students worked on the design process and analysis by going through a number of steps. First, the problem was stated. Then, analysis and research was done on the different conceptual designs. A final design was selected by evaluation and screening. Finally, detailed design analysis and cost estimation of the chosen design was done. For the second half of the project, the final selected design was actually supposed to be built and programmed in stages and then testing and evaluation was done to ensure that it worked as close to specifications as was originally stated. Since none of us had RF design experience our project was simplified and our second phase several changes were made in order to meet our desired goals. Instead of transmitting and receiving signal students were asked to focus only on one side. Instead of building the whole RF front end, using the USRP platform and implementing an amplifier design, students were asked to focus on the measurement and analysis.

It should be noted that the goal of this project is not to make a pioneering product or an enhanced version of an existing technology. It is to gain knowledge. Knowledge of how a signal is received wirelessly at a required frequency. The students will seek to get a thorough understanding of the design process and implementation and ultimately recognize what is meant by ‘to engineer’.
SECTION I
Figure 1.a. The SDR transmitter design with a FPGA chip and a DAC that samples at a lower frequency. Figure 1.a demonstrates the design of a SDR transmitter for sending a digital audio file. The digital audio signal is read from the computer through the USB port. The FPGA chip uses software to modulate the signal to an intermediate frequency (IF), which is lower than the carrier frequency. The digital IF signal is converted from digital to analog via a DAC, up-converted to the radio frequency (RF) by a mixer and a local oscillator (LO). The analog RF signal is then filtered by a band-pass filter and the power amplifier, and finally transmitted wirelessly through the antenna.

Figure 1.b. The SDR receiver design with a FPGA chip and an ADC that samples at a lower frequency. Figure 1.b demonstrates the design of a SDR receiver for receiving the analog signal. The analog RF signal is first captured by the antenna, and then filtered through the band-select filter before going to the low-noise amplifier. The analog RF signal is down-converted to an intermediate frequency (IF) through a mixer and a local oscillator (LO). In this case, an image reject filter needs to be added before the mixer, so that the frequency component that would generate the same IF signal can be removed before passing through the mixer. The ADC (Analog-to-Digital Converter) digitizes the analog signal through at the IF band. The FPGA chip then demodulate and down-convert the digitized RF signal from the IF band to the baseband through software programming. In the end, the baseband digital signal is transmitted to the computer through the USB port and played using the PC speaker.
FPGA is used for our software-defined radio prototype mainly because of FPGAs capability to process at a higher speed and maintain a higher data rate. Our SDR system is expected to use transmit in the 900MHz RF band, with high-rate sampling speed of the ADC, the data rate is expected to be at the level of a few hundred samples per second.
SECTION II
During the summer of 2009 our advisors changed our design, because none of us have any RF design experience. The changes that were made are to design a transmitter and receiver with the help of the URSP platform.

**Universal Software Radio Peripheral (USRP)**

The USRP board is a high speed hardware component which is appropriate to help us implement real time software radio applications. The board is very integrated and incorporates, RF front end, an FPGA board to do the high-speed operations like digital down and up conversions, and Digital to Analog and Analog to Digital converters. This board is mainly used for developing software radios, so modulation and demodulation are done in the processing unit. A USRP board consists of one mother and it can contain up to four daughter boards, it also requires a MAC or PC with a USB2 interface.

![USRP Mother Board with Two RX and Two TX Daughter Boards](image)

Figure 2.1: USRP mother board with two RX and two TX daughter boards.

In this figure one can see the picture of a USRP board with two Receive and two Transmit daughter boards. The next figure, figure 2.2 shows the schematic.
Figure 2.2 Schematic of USRP board.

In the schematic one can see that a USRP board has four high-speed analog to digital converters, each of them at 12 bits per sample, and 64 million samples per second. There are also four high-speed digital to analog converters, each of them at 14 bits per sample, and 128 million samples per second. These inputs and outputs are connected to an Altera Cyclone EP1C12 FPGA. The FPGA connects to a USB2 interface chip so this can connect to the computer.

**The Daughter Boards**

The daughter boards are used to hold the RF receiver interface or tuner and the RF transmitter. Each daughter board slot has access to two of the four high-speed Analog to Digital / Digital to Analog converters. This will allow each of the daughter boards which use real sampling to have two independent RF sections, and two antennas.
Software Installation

Setting up and installing GNU radio software environment including GRC (GNU Radio Companion) is the most difficult part of the installation. Since the GNU radio software can integrate with different types of operating systems available. There is different instruction depending on the type of OS that is chosen. From doing research it was determined that Windows OS would be the biggest challenge for a successful setup and installation for the GNU radio platform. One of the main reasons is that the GNU radio software is Linux based. There was no Linux based computer available, so one of our team members installed Ubuntu 8.04 on their Laptop. This was obtained by downloading a Linux OS ‘live CD’. This CD was used to install Linux on the Window OS machine and configured for dual boot functionality, so this gives the machine the ability to boot up in Windows or Linux.

The following steps are the steps taken for a successful installation of the GNU radio software with USRP on Ubuntu 8.04 version of Linux.

1. A ‘Live CD’ that contains the Ubuntu was downloaded and created. This can be done by visiting the Ubuntu website (www.ubuntu.com). Instructions for downloading and creating a ‘Live CD’ can be found on the Ubuntu website.
2. This next step would be to install Ubuntu from the obtained ‘Live CD’. First one partitions their hard drive in Windows using Windows management (allocating about 16GB of memory for installation). After that insert the ‘Live CD’ in the CD drive for installation. This will provide a set of instruction how to do so. Follow these instructions and then selecting the portioned part of the hard drive to install Ubuntu. When this completes, the computer will then has the ability to dual boot either Ubuntu or Windows.
3. Since Ubuntu is now installed, the next steps to be taken is to install the required packages for the GNU radio software. All the following is a list of the packages to compile and run various components of the GNU Radio Software on Ubuntu. Most of these packages can be installed true the synaptic package manager, but boost must be installed from the source.

- g++
- subversion
- make
- autoconf
- automake
- libtool
- sdcc (from “universe”; version 2.4 or newer)
- guile (version 1.6 or newer)
- ccache (this package is not required but is recommended if you compile frequently)
- python-dev
- libfftw3
- libfftw3-dev
- libcppunit
- libcppunit-dev
- boost (version 1.35 or later)
- libusb
- libusb-dev
- wx-common
- python-wxgtk2.8
- python-numpy-ext
- alsa-base
- libasound2
- libasound2-dev
- libqt4-dev
4. Now boost must be downloaded and installed from the source
   a. Download `boost_1_37_0.tar.bz2`
   b. Unpack it somewhere and `cd` into the resulting directory:
      i. `cd boost_1_37_0`
   c. Use the following prefix to install it into:
      i. `BOOST_PREFIX=/opt/boost_1_37_0/include/boost-1_37`
   d. Enter the following command to configure:
      i. `./configure --prefix=$BOOST_PREFIX --with-libraries=thread,date_time,program_options`
   e. Enter the following command to compile the package:
      i. `make`
   f. Enter the following command to install the package:
      i. `sudo make install`

5. Now that boost is successfully installed, now one can start downloading and installing the GNU Radio Software.
   a. Enter the following command in a terminal to download GNU radio version 3.1.0
      i. `svn co http://gnuradio.org/svn/gnuradio/releases/3.1 gnuradio`
   b. Unpack it somewhere and `cd` into the resulting directory:
      i. `cd gnuradio`
   c. Set the LD_LIBRARY_PATH environmental variable:
      i. `export LD_LIBRARY_PATH=$BOOST_PREFIX:/usr/local/lib`
   d. Enter the following command to configure and compile:
      i. `./bootstrap`
      ii. `./configure --with-boost=$BOOST_PREFIX`
      iii. `make`
   e. Enter the following command to install the GNU radio software
      i. `sudo make install`
   f. The following GNU radio software packages should be listed as installed (if any of these packages are not listed as being successfully installed than GNU Radio/GRC is not guaranteed to work properly)
      i. `config`
      ii. `omnithread`
      iii. `gnuradio-core`
      iv. `usrp`
      v. `gr-usrp`
      vi. `gr-audio-also`
      vii. `gr-audio-oss`
      viii. `gr-atsc`
      ix. `gr-gpio`
      x. `gr-gsm-fr-vocoder`
      xi. `gr-pager`
After that step success, the next step is to set up communication between the USRP and the PC through USB. The non-root access to the USRP must also be set up. A script file must be created in order to do so. Enter the following command in the terminal window so the script file can be created.

- `sudo addgroup usrp`
- `sudo addgroup <YOUR_USERNAME> usrp`
- `echo 'ACTION=="add", BUS=="usb", SYSFS{idvendor}=="fffe", SYSFS{idProduct}=="0002", GROUP=="usrp", MODE=="0660"' > tmpfile`
- `sudo chown root.root tmpfile`
- `sudo mv tmpfile /etc/udev/rules.d/10-usrp.rules`

Now we verify that the communication is set up properly by trying to ping the USRP, for this to take place the USRP must be connected to the computer.

- `ls –lR /dev/bus/usb | grep usrp`

A message will then appear to verify the connection similar to the following:

- `crw-rw---- 1 root usrp 189, 514 Mar 24 09:46 003`

Now that the connection is verified between the computer and the USRP, the next step is to verify that the GNU radio software is functioning properly. The following should be entered in the terminal window:

a. The `gnuradio-examples` directory provides an example setup to verify that the GNU radio software was installed correctly. Enter the following:

- `cd gnuradio-examples/python/audio`
- `./dial_tone.py`

This should produce an audible dial tone from the computer speakers.
9. After that is verified, the following step is to download and install the GRC software:


   b. Unpack it somewhere and enter the following command to verify it functions properly:

   ```
   • cd grc-0.70/src/Editor.py
   ```

   This should launch the GRC GUI.

At this point we have had come across the problem where we could not get the GUI to launch, research was done on the internet and also to try to reach out the other Engineers in the industry, but no help was found. So we contacted Chris Pflieger from Logikos here in town and he recommended us to the OSSIE OS which already has a GUI build in which will make communicating to the USRP much simpler.

**OSSIE 0.7.4**

OSSIE is an open source Software Defined Radio development effort that is based at Wireless@Virginia Tech. It is intended to do research and enable education in SDR and wireless communications. This package includes SDR core framework based on the JTRS Software communications Architecture, tools to develop component and waveform applications, also in evolving library of already built components and waveform applications. There are also tutorial available for free to familiarize with the environment.

The OS the OSSIE is built in is Ubuntu 9.04. In this case since we did not have this available, we downloaded this from the OSSIE website ([http://ossie.wireless.vt.edu/trac/wiki/Downloads](http://ossie.wireless.vt.edu/trac/wiki/Downloads)).

The following steps that are going to be taken will allow one to run and have a valid connection with the USRP

1. Boot from CD drive after inserting OSSIE ‘Live CD’
   - There will be an option screen, one would hit enter so Ubuntu can load.
   - Then there will be a Login screen
     - Login Information:
       - Username: ossie, Password: wireless

2. This next step is to install all the dependencies so that Ubuntu can properly communicate with the USRP, use the terminal window and enter the following:

   ```
   $ sudo aptitude install gcc build - essential
   $ sudo aptitude -y install omniorb4 libomniORB4 -dev omniidl4 - python \  
   omniorb4 - nameserver python - omniORB2 libgdk2.0- dev freeglut3 -dev \  
   libxercesc27 -dev libxercesc27 python - wxgtk2.8 python - wxversion \  
   ```
After this was done we started to trouble shoot our design.
SECTION III
Testing

As our design changed our new goal is to transmit an audio signal from Computer 1 to Computer 2 through use of the Universal Software Radio Peripheral 1 (USRP1) boards.

![Signal Transmission](image)

![Signal Reception](image)

**Figure 3. Transmission and Reception using USRP**

To make this possible we contacted Carl Dietrich at Virginia Tech to help us brainstorm around this project, but due to time constraints we had to take another route. We decide to focus on getting an Am signal using the USRP board. In the following steps this will be explained.

**Software Components:**
- **USRP_Commander:** This will allow communication between Computer and the USRP device, allowing us to set parameters like frequency, gain, and the decimation factor. The decimation will be performed by the FPGA’s Digital Down Converter, where the maximum factor is 250KHz.
- **Decimator:** This makes sure the frequency rate is compatible with the pc sound card, around 48KHz.
- **AutomaticGainControl:** This will help keep the signal at a consistent level.
- **Am demod:** This component uses an algorithm to subtract the information from the carrier frequency

**Hardware Devices:**
- **GPP:** PC processor
- **USRP**
- **Sound card:** board inside the PC for reproducing the signal in the speakers

1. **Set up**
   a. When in Ubuntu open a terminal window
   b. Now we test if the COBRA naming service is running. Type `ps –e | grep omniNames`
This should already be running, and the omniNames will be listed.

2. Build Waveform using Eclipse
   a. Open Eclipse click File, the New and select other. A list appears, expand the OSSIE
      group and select OSSIE waveform and then click “NEXT”.
   b. The next wizard will allow you to give the waveform project a name and then click
      finish.
   c. Now add the next components to the design, drag these components to the
      waveform window in this specific order:

      USRP_Commander, Decimator, AutomaticGainControl, and am_demod

   d. Now we connect these components together:

      I. USRP_Commander: RX_Control (Use) -> Devices: USRP: RX_Control (Provide)
      II. Decimator: inData (Provide) -> Devices: USRP: RX_Data_1 (Use)
      III. Decimator: outData (Use) -> Components: AutomaticGainControl: data_in
           (Provide)
      IV. AutomaticGainControl: data_out (Use) -> Components: am_demod:
           Rx_In_from_USRP_or_Decimator (Provide)
      V. am_demod: Out_to_sound_card (Use) -> Devices: soundCardPlayback:
           soundOut (Provide)

   e. At this step we drag all the components to the GPP device in the platform window under
      the default GPP_USRP_sound_node.
      In our case the GPP device was names GPP1.

      This is what the platform should look like:

      ![Platform Diagram]

   f. Now we go back to the waveform window and right click the USRP_Commander and
      select the “Set Assembly controller.”
   g. After setting that, we right click again on the USRP_Commander and select “Edit
      Component Properties”. At this point we get to set specific properties. So we can fine
      tune the radio.

      1. Click the rx_freq under value, we entered the frequency 1090000, this is
         in Hertz
      2. Now we close this property by using the “OK” button.
h. At this point we need to edit the properties in the Decimator, by right clicking on it in the waveform window and the selecting “Edit Component Properties”.
   1. Click on the value in the list of properties under the DecimateBy, set the value to 10.
   2. Close the property editor.
   i. Now similarly we need to set the AutomaticGainControl properties
      
      a) \textit{energy\_lo}: 1000
      b) \textit{energy\_hi}: 1000
      c) \textit{k\_attack}: 0.00001
      d) \textit{k\_release}: 0.00001
      e) \textit{g\_min}: 0.01
      f) \textit{g\_max}: 1000
      g) \textit{rssi\_pass}: 10

      Now close the property and component editor, and save your work.

j. Now we want that we a rate of the decimated signal, we want to set the sound card at that same rate.
   
i. Use the terminal window to get to /sdr/xml/soundcardPlayback/
   
ii. Open soundCardPlayback.prf.xml by typing the following in the terminal window:

\begin{verbatim}
Gedit soundCardPlayback.prf.xml
\end{verbatim}

Now look for “Sample rate of sound card playback”. The number between the \textit{<Value>} brackets needs to be 25000. This will set the sound card sampling rate to 25KHz.

iii. Now save the file.

3. Connecting the USRP

a) A basic RX board is need to be connected to the USRP in the RXA slot.

b) Now connect the loop antenna to the RX-A connector on the RX board.

c) Plug the USRP to the computer using the USB cable and then power the USRP by plugging the power cord.

4. Running the Waveform

i. Now in Eclipse, select OSSIE, and Run NodeBoofter. Select Browse, and navigate to the file
   
   /sdr/nodes/default\_GPP\_USR\_sound\_node/DeviceManager.dcd.xm

   ii. Select OK to run the NodeBoofter.

   iii. Now similarly we run ALF, by navigating to OSSIE, RunALF.

   iv. In ALF double click on the name that you saved your waveform. And her you should here an AM signal.

   v. In the “manage waveform” panel, right click on your project and click display. Your waveform will be displayed in a block diagram form.

   vi. Now on any of the block diagrams you can right click and select plot and it will plot the desired waveform.
SECTION VI
Evaluations and Recommendations

In this project first we wish that we had more time, the difficulties that presented themselves that we did not know how to solve; it would have been great if there was some one that knew how the program works. I think if our group had that type of help at our disposal, it would have been a better learning experience meaning that we could have learned from some one who already has experience on software defined radio. Another recommendation that we have, is it would been a great idea to know in advance what the project entailed, this would have given us the opportunity to take the Software Defined Radio class on campus.
Conclusion

This Software Defined Radio System that we were able to work on this semester has been a challenging design problem to work on. The times spend researching and trying to accomplish the goal of communicating between two SDR’s has been extensive over the past semester. We each had to do individual research, but overall it was a group oriented project. Every group was required to do their own research and then bring their ideas to the table on how to overcome the challenging objectives in this project.

There has been a lot of programming issues that presented their selves during the semester. So the World Wide Web has been a useful for us.

The lesson we learned as a group during this time of research is that without enough recourses a project like this is not possible. What we were able to accomplish is a successful install of Ubuntu. Also we were able to receive a AM signal after extensive trial and error, having the PC communicate with the USRP. We were unable to run the GNU Radio Companion software, since this was a difficult objective for our group, we took a different route by acquiring information about OSSIE Wireless at VirginiaTech and using their software setup.

This project caused us to take a lot of time in researching and understanding the USRP concept. Now that we all are graduating we have a better understanding on how to approach challenging projects like these. Even though it was a challenging project it was a great learning experience for each team member. It was a pleasure to work on a project like this.
References

www.ubuntu.com

www.joshknows.com

“Prototyping a Software Radio Receiver Based on USRP and OSSIE”, S.M. Shajedul Hasan; P. Balister, December 2005


Chris Plieger, personal communication, October 9 – October 14 2009.