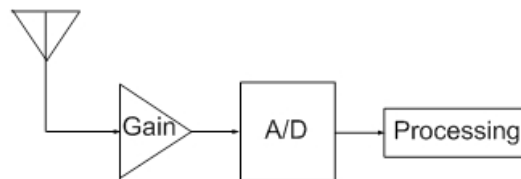


## Software Radio Introduction Guide - An FM Radio Primer

This guide assumes that the reader has completed a course covering the basics of frequency modulation and demodulation. It also assumes that the reader is familiar with signals theory.

A software radio is a communications device created partially or entirely using software. In the simplest case, software is used as a means of controlling some property of the radio such as gain, bitrate, frequency, etc. An example of this would be cell phone or wireless internet (802.11a/b/g/n). Such radios have a fatal flaw, and are doomed to eventual obsolescence because they cannot be upgraded to take advantage of newer technologies. Software Radios can utilize most new technologies with simple software upgrades, or different kinds of radios can be downloaded to provide added functionality.



A radio created entirely using software must have some supporting hardware for communication. The most minimal kind of software radio receiver has an antenna, an amplifier with a filter, and an A/D converter. All demodulation, decoding, and additional filtering are performed using digital signal processing algorithms implemented in software or with some sort of programmable integrated circuit. It is also common to demodulate the received signal using a digitally controlled tuner to an intermediate frequency to allow for a wider reception of signals. However, one downside of software radio is that to process high bandwidth signals in realtime requires a very powerful computer. For example, to receive over-the-air HDTV (1080i) signals using a software radio requires 6 hours of processing to receive a 1 hour signal using a modern PC. Your HDTV or cable tuner accomplishes this feat by using dedicated IC's at all stages of decoding/processing.

We will be using a software radio package called GNURadio that provides a large collection of signal processing blocks that can be connected in a signal flow graph to create a software radio. A signal processing block can function as a data source/sink or an I/O block.

Source:	1 to Many Outputs
Sink:	1 to Many Inputs
I/O:	1 to Many Inputs & Outputs

A signal processing block can implement a specific algorithm such as a filter, or connect to physical hardware such as a PC's sound card. By linking numerous blocks together, a macro block can be created which accomplishes some complicated task. Each signal processing block has an input/output data type to represent the kind of data being

processed, and these must be matched for a connection between signal processing blocks to occur.

GNURadio signal processing blocks are written in the C++ programming language and compiled into libraries. A wrapper to these libraries is provided by a program called SWIG that allows for the creation of the flow graph in a high level scripting language called Python. Typically, to create a software radio using GNURadio required knowledge of Python, C++, and a great deal of tolerance for Linux. Thankfully (or sadly), this situation has changed.

GRC or GNURadio Companion is a program written by Josh Blum. It allows signal processing blocks to be placed and connected using a GUI environment similar to LabView or Simulink. The main workspace contains all blocks used in the flow graph and each block's properties can be edited by double-clicking on them. Adding blocks is done by double-clicking on their name in the "Signal Blocks" tab located on the right side of the main GUI window. The final "Variables" tab contains global variables that can be used across the flow graph. Specifying a value for the max/min of a variable will bring up a slider GUI element that allows for this variable to be changed dynamically during flow graph runtime. The following is a step-by-step guide on how to create a broadcast band FM receiver using GRC.

1. Log into the machine called "Patrick" in CSEB 224 using the password given out during 520.465
2. Double-Click on the GRC shortcut on the Desktop
3. Click on the triangle next to "Sources" in the "Signal Blocks" tab. This will display a list of all the signal sources supported by GRC.
4. Double-click on the "USRP Source", and a new block should appear in the Workspace window. Double-click on this block to configure its properties.

The Universal Software Radio Peripheral (USRP) is a device designed specifically to work with GNURadio and supports several plug-in modules called daughterboards. These boards allow the USRP to receive signals over a wide range of frequencies. Currently, SRPL owns boards that receive in the ranges: 0-30MHz, 54MHz-930MHz, and 1GHz-2.3GHz. Transmit boards are more limited, and cover only 0-150MHz.

The daughterboards connect to high speed A/D and D/A IC's which have sampling rates of **64 & 128 MS/s**. Data then goes into an FPGA where the signal is decimated, buffered, and sent over USB2.0 to a computer for processing.

5. Double-click the "USRP Source" block, and it will display a list of parameters that can be edited.

**Output Type:** This is the data type that the signal block will output. The format is two 4-byte integers representing the real and imaginary (I & Q) components of the signal. Typically a complex type is desired, but for high bandwidth signals, a float may be preferred.

**Unit Number:** This indicates which USRP that the data will originate from. Multiple USRP's can be connected, and the "Unit Number" is assigned in the order that the devices were powered-up when connected to the computer via USB.

**Side/Subdevice:** There are 4 slots for daughterboards on the USRP (2 TX, 2 RX). This window indicates which board will be used. Selecting "Auto" will use the device connected to RXA, or alternatively RXB if there is no device connected in slot A.

**Frequency:** This number specifies the frequency in Hertz that the USRP will attempt to receive on. If the daughterboard you are using does not support this frequency, it will return an error. All of the details about the intermediate frequency used by the daughterboard are handled automatically.

**Decimation:** This number indicates the rate at which data comes out of the USRP. A higher decimation results in a slower rate of information, but over a narrow bandwidth. To calculate the effective sampling rate out of the USRP divide 64 million by the decimation factor. A factor of at least 4 must be used when interfacing with the USRP due to the bandwidth limitation of USB2.0, resulting a maximum signal bandwidth of 8MHz ( $64 / 4 / 2 = 8\text{MHz}$ )

**Gain:** Most daughterboards have some kind of amplification stage that can be controlled digitally. The value entered in here is the amplification in dB that the signal will experience before being sampled. The USRP board provides amplification of 20dB, but some daughterboards have additional amplifiers that can go up to 120dB.

The properties you will need for the reception of an FM signal are

Unit Number: 0

Side: Auto

Frequency: **105700000** or **105.7e6** or your favorite station<sup>1</sup>

Decimation: **200**

Gain: **90**

6. Click close to save the changes, and add the block "WFM Receive" located in the "Modulators" category.

---

<sup>1</sup> FM96.1 happens to be a strong signal in the basement of Barton, and is not indicative of the author's preference in music.

WFM is an example of a macro block because it contains several primitive GNURadio blocks inside of it. Inside this block are a Phase-Lock-Loop, a decimation filter, and something that de-emphasizes the FM signal. All of these details about actually receiving the FM signals are conveniently hidden, and only two simple parameters need to be configured for this to work properly. The output is a signal decimated to a frequency range in the audio domain.

7. Double-click on this block and enter the follow:  
    Quad Rate: 320000  
    Audio Decimation: 10
  
8. Click close to save the changes, and add the block “Audio Sink” located in the “Sinks” category. This block sends any data it receives directly to the soundcard, which if hooked up to speakers, produces an audible signal. Change the sampling rate to reflect the rate provided by the WFM receiver. A sound card will only support a few discrete sampling rates for playback, and a list of the supported rates if provided by a menu in the signal block.  
    Sampling Rate: 32KHz
  
9. To connect the signal blocks, single-click on the terminal of the destination and then single-click on the source to form a connection. Connect the blocks in the following order:  
  
    USRP Source -> WFM Receive -> Audio Sink  
  
    You can click and drag the signal blocks, or rotate them using the Left/Right icons just under the menubar to make the overall flow graph look aesthetically pleasing.
  
10. Once you are done, click the File menu, and choose Save. Enter your name and click save. This will place the file in your home directory where you can access it for future modifications. (NOTE: Individual home directories are not provided. Name the file with part of your name to distinguish it from others. Before you leave your session, copy your file(s) to a flash drive that you have brought for this purpose.)
  
11. Click on the Green Gear icon to Run the flow graph. The USRP firmware is stored in volatile memory, and must be dynamically loaded after each power cycle, so you may have to wait several seconds. Eventually a window should appear, and extremely faint audio will begin to come out of the speaker. Not bad for a first attempt, but not particularly rewarding either.

12. Click the red circle X icon next to the Run icon to stop the music, and add a “Multiply Constant” block from the “Operations” category. Click on the connection between the “Audio Sink” and the “WFM Receive” blocks and click delete. Now connect the “WFM Receive” block to the “Multiply Constant” block. Next connect the output of the “Multiply Constant” block to the “Audio Sink”. Double-click on the “Multiply Constant” to configure it to the following:  
Type: Float  
Constant: \$volume
13. Click close to save these changes and click on the Add button in the Variables tab. Single click on the name of the new variable and call it “volume”. Specify the default as 1, the min as 1, and the max as 10.
14. Click the Run icon, and now a window appears with a slider with the name Volume. By clicking and dragging on this slider to the left or right, the constant that multiplies the audio signal varies, which results in a higher output volume. This is better than before, but it most likely sounds terrible (unless you know the station frequency exactly).
15. Add an “FFT Sink” block located in the Sink category. Connect this block to the output of “USRP Source”. The output terminal of a block can be connected to many sinks, so this will allow you to hear, and see the FM signal. Leave all the parameters in the FFT sink as default except the following:  
Ref Level: 100  
Sampling Rate: 320000
16. Add a variable called “frequency” in the variables tab, and set the default as 96e6, with a min of 88e6, and a max of 106e6. These are the approximate frequency ranges of the broadcast FM band.
17. Lastly, double-click on the USRP source and change the value under the frequency category to “\$frequency”. Click close, and then click the run icon. Slowly move the Frequency slider until you see an FM signal on the frequency spectrum, and when you tune in onto the center frequency, the audio will suddenly sound really clear.

### **Congratulations, you’ve just made a Software Radio!**

At this point, if you’re feeling adventurous you can try the following:

- Change the max value for the frequency variable to 900e6, and you can go hunting for FM broadcasts across the spectrum. You’ll likely find something around the 400MHz range because broadcast TV uses FM to transmit audio.

A picture of the Completed Signal Flow Diagram

