Getting Started with GNU Radio

Originalartikel

Backup

<html> Software Defined Radio (SDR)–the ability to process radio signals using software instead of electronics–is undeniably fascinating. However, there is a big gap from being able to use off-the-shelf SDR software and writing your own. After all, SDRs require lots of digital signal processing (DSP) at high speeds. Not many people could build a modern PC from scratch, but nearly anyone can get a motherboard, some I/O cards, a power supply, and a case and put together a custom system. That’s the idea behind GNU Radio and SDR. GNU Radio provides a wealth of Python functions that you can use to create sophisticated SDR application (or, indeed, any DSP application). If Python is still not up your alley (or even if it is), there’s an even easier way to use GNU Radio: The GNU Radio Companion (GRC). This is a mostly graphical approach, allowing you to thread together modules graphically and build simple GUIs to control you new radio. Even though you usually think of GRC as being about radios, it is actually a good framework for building any kind of DSP application, and that’s what I’ll show you in the video below. GRC has a signal generator block and interfaces to your sound card. It even has the ability to read and write data to the file system, so you can use it to do many DSP applications or simulations with no additional hardware.
UPDATE: Don’t miss the follow-up post that uses SDRPlay to build a GNU Radio based receiver. <iframe class=„youtube-player“ type=„text/html“ width=„800“ height=„480“ src=„https://www.youtube.com/embed/ufxBX_uNCa0?version=3&rel=1&fs=1&autohide=2&showsearch=0&showinfo=1&iv_load_policy=1&wmode=transparent“ allowfullscreen=„true“ style=„border:0;“>[embedded content]</iframe> <h2>Building Blocks</h2> There are several key building blocks that combine to make SDR possible. The first is some input device (a source) that is sampled at some sampling rate. For an audio device, the samples will be real numbers. However, radio devices will more likely provide complex numbers with an I and Q component. If you aren’t familiar with expressing signals as I and Q components (sometimes known as quadrature data), that’s a big topic (with a great 3D explanation, one from Tektronix, and another one from National Instruments). However, you don’t need to directly understand the theory behind quadrature signals to get started with GRC. Just know that the I and Q signals can combine to express any waveform and, conversely, any waveform can break up into a series of I and Q values. With GRC, though, it isn’t that important (in most cases) to understand that, just like you can use a video card without knowing exactly what signals are on the PCI express bus.
<h2>A Starter GRC Project</h2> <img data-attachment-id=„177192“ data-permalink=„https://hackaday.com/2015/11/11/getting-started-with-gnu-radio/grc0-cropped/“ data-orig-file=„https://hackadaycom.files.wordpress.com/2015/10/grc0-cropped.png?w=800&h=261“ data-orig-size=„800,261“ data-comments-opened=„1“ data-image-meta=„{"aperture":"0","credit":"","camera"

:"","caption":"","created_timestamp":"0","copyright":"","focal_length":"0","iso":"0","shutter_speed":"0","title":"","orientation":"0"}“ data-image-title=„grc0-cropped“ data-image-description=„“ data-medium-file=„https://hackadaycom.files.wordpress.com/2015/10/grc0-cropped.png?w=800&h=261?w=400“
data-
large-
file=„https://hackadaycom.files.wordpress.com/2015/10/grc0-cropped.png?w=800&h=261?w=800“ class=„aligncenter wp-image-177192 size-full“
src=„https://hackadaycom.files.wordpress.com/2015/10/grc0-cropped.png?w=800&h=261“
alt=„“ width=„800“ height=„261“
srcset=„https://hackadaycom.files.wordpress.com/2015/10/grc0-cropped.png 800w,
https://hackadaycom.files.wordpress.com/2015/10/grc0-cropped.png?w=250&h=82 250w,
https://hackadaycom.files.wordpress.com/2015/10/grc0-cropped.png?w=400&h=131 400w,
https://hackadaycom.files.wordpress.com/2015/10/grc0-cropped.png?w=768&h=251 768w“
sizes=„(max-width: 800px) 100vw, 800px“/>You can find a simple starter project on the video below. GRC uses a block diagram (a flow graph) to represent your project. When you create a new flow graph, you’ll see two blocks already present: one for options, and a variable named samp_rate. The most important part of the options block sets the graphical toolkit you want to use (WX graphical widgets or the QT widgets). For our purpose, it doesn’t really matter, but in the video I’m going to select Qt. The sample rate is so integral to your design that there is a special variable for it. Most other blocks will pick up the value of this variable and use it for their sample rate. However, this isn’t always what you want, but it is a good starting point. For this example, I’m eventually going to get input from a sound card, so I wanted a sample rate most sound cards will support (48 kHz). To start with, though, I kept the example very simple as you can see in the diagram above. <img data-attachment-id=„175730“ data-permalink=„https://hackaday.com/2015/11/11/getting-started-with-gnu-radio/fft1/“ data-orig-file=„https://hackadaycom.files.wordpress.com/2015/10/fft1.png“ data-orig-size=„708,780“ data-comments-opened=„1“ data-image-
meta=„{"aperture":"0","credit":"","camera":"","caption":"","created_timestamp":"0","copyright":"","focal_length":"0","iso":"0","shutter_speed":"0","title":"","orientation":"0"}“ data-image-title=„fft1“ data-image-description=„“
data-medium-file=„https://hackadaycom.files.wordpress.com/2015/10/fft1.png?w=363&h=400“
data-large-file=„https://hackadaycom.files.wordpress.com/2015/10/fft1.png?w=567“ class=„wp-image-175730 size-medium alignright“
src=„https://hackadaycom.files.wordpress.com/2015/10/fft1.png?w=363&h=400“ alt=„fft1“
width=„363“ height=„400“
srcset=„https://hackadaycom.files.wordpress.com/2015/10/fft1.png?w=363&h=400 363w,
https://hackadaycom.files.wordpress.com/2015/10/fft1.png?w=227&h=250 227w,
https://hackadaycom.files.wordpress.com/2015/10/fft1.png 708w“ sizes=„(max-width: 363px) 100vw, 363px“/>To start with, you’ll place a signal source that generates IQ data (as complex numbers) on the flow graph. The signal generator block can actually generate too much data and slow down the CPU. Since the sample rate is 48 kHz, it doesn’t make sense to generate more than 48,000 samples per seconds. To make sure that happens, you’ll add a throttle block and connect it to the generator. Connecting ports that have the same color (and, thus, the same data type) is simple. Just click on one port and then click on the other. The order doesn’t matter and you can connect more than one input to a single output. If the port data type

doesn’t match, you’ll need to use a type converter (and the example video will show that later). The final part of my simple example is just an FFT display block that uses QT (the same graphics library you set in the options block). Once it is all connected together you can press the play button and you should see an FFT of the signal. <h2>Adding More</h2> Granted, that isn’t terribly exciting, but you have to walk before you run. In the video you’ll see that you can add other graphical elements like sliders to change the frequency and selectors to pick the type of signal. Finally, you’ll add a sound card input (which does not need a throttle; in fact, in this case it is harmless to have both, but usually you’d remove the throttle block when using a real hardware block). What it does need, however, is a type conversion to change the real data to complex. The final project appears to the left. I’d encourage you to watch the video with GRC open. You can get most of the way through without any special hardware and the sound card source should work with your sound card (surely you have a sound card in your PC, I’m guessing). Once you are comfortable with the example, try extending it. Add a sound card sink to get audio out. Try filtering the audio from the test source. For example, what does a low pass filtered square wave sound like at different frequencies? If you are adventurous, try filtering sound card audio and writing it to a wave file for playback. All the tools you need are on the GRC tool palette. <h2>If You Use PulseAudio</h2> By the way, if you use Linux (like I do) and have PulseAudio as your sound system, you may get choppy audio playback. The fix for this is simple. Just create a file in your home directory: ~/.gnuradio/config.conf (or edit it if it is already there). You need the following: <pre>[audio_alsa]
 nperiods = 16
 period_time = 0.100
 </pre> You can increase nperiods even further if that doesn’t make it better. <h2>Next Time</h2> UPDATE: Continue on to build a radio receiver by example. Next time I’ll break out the SDRPlay device and build a real radio. If you want a lot of detailed theory, check out [Michael Ossman’s] tutorial series. It is aimed at the HackRF, but the theory still applies. Of course, GNU Radio (and GRC) aren’t the only way to do DSP software, as we’ve noted

before.</p> </html>