GNU Radio

SDR for the masses

Marcus Müller

Software Defined Radio Academy 2015

Who am I?

- GNU Radio contributor and user
- Spent too much time on the `discuss-gnuradio@gnu.org` mailing list
- Got hired by Ettus Research™ A National Instruments Company

Who am I?

- GNU Radio contributor and user
- Spent too much time on the `discuss-gnuradio@gnu.org` mailing list
- Got hired by Ettus Research™ A National Instruments Company

…and who is Ettus?

- Producer of the USRP series of SDR frontends
- `gr-uhd` integrates directly in GNU Radio
- `http://www.ettus.com`

A short overview

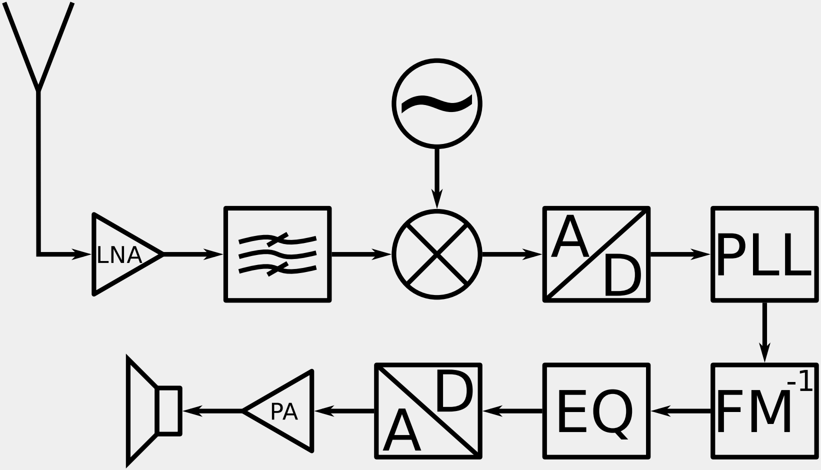
1. Introduction
2. What is GNU Radio
3. Core Concepts
4. Typical Work Flow 1: Flow graphs in GRC
5. The USRP B2x0: A direct mixing SDR architecture
6. Questions and Answers
7. Typical Work Flow 2: How to build a talking clock
8. Useful Resources

What is GNU Radio?

. . . and more importantly: *Why would I want that?*

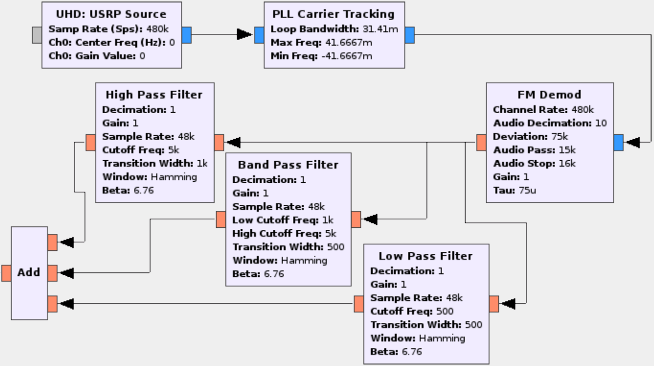
A Sample Flow Framework

Radio signal processing often looks like a chain of steps:



A Sample Flow Framework

GNU Radio is a signal flow graph oriented framework:



UHD: USRP Source
Samp Rate (Sps): 480k
Ch0: Center Freq (Hz): 0
Ch0: Gain Value: 0

PLL Carrier Tracking
Loop Bandwidth: 31.41m
Max Freq: 41.6667m
Min Freq: -41.6667m

FM Demod
Channel Rate: 480k
Audio Decimation: 10
Deviation: 75k
Audio Pass: 15k
Audio Stop: 16k
Gain: 1
Tau: 75u

High Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 48k
Cutoff Freq: 5k
Transition Width: 1k
Window: Hamming
Beta: 6.76

Band Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 48k
Low Cutoff Freq: 1k
High Cutoff Freq: 5k
Transition Width: 500
Window: Hamming
Beta: 6.76

Low Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 48k
Cutoff Freq: 500
Transition Width: 500
Window: Hamming
Beta: 6.76

Add

Audio Sink
Sample Rate: 48k

GNU Radio comes with a toolbox of useful blocks

- hardware interface blocks
- basic mathematical operations
- filters
- digital modulators
- analog modulators
- network tools
- example implementations (digital TV, pagers, Sat images. . .)
- . . .

. . . **But it’s not a receiver/transmitter for any particular standard.**

What people built with GNU Radio

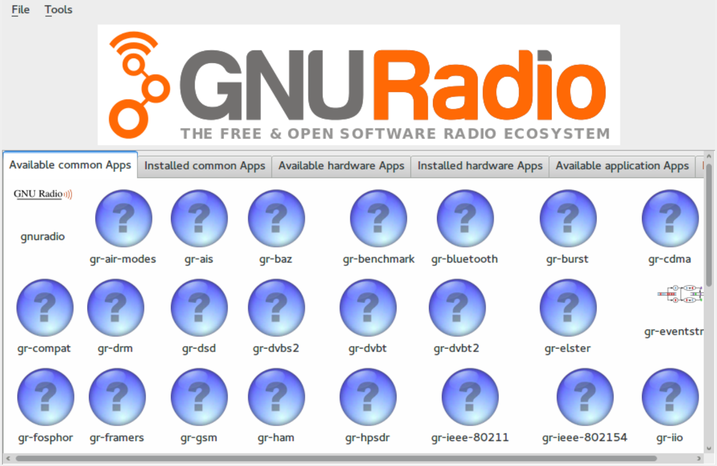
GNU Radio can rely on an active community with many useful modules, among those

- GQRX
- gr-radar
- gr-ieee802-11
- and much more

Have a look at CGRAN, `http://cgran.org`

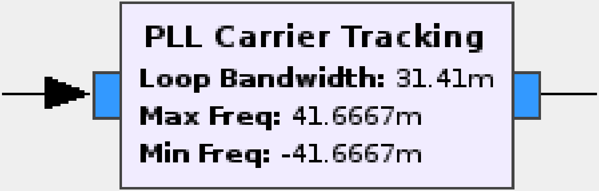
GNU Radio has an application installer!

PyBOMBS: convenient installer for GNU Radio, build dependencies and out-of-tree modules



Core concept: Block

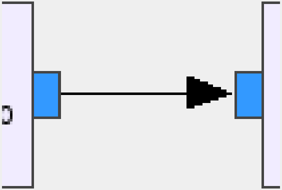
A block represents a signal processing step



- can have 0 or more inputs
- can have 0 or more outputs
- can either
  - do its own signal processing
  - or contain different blocks in itself

Core concept: Item Stream

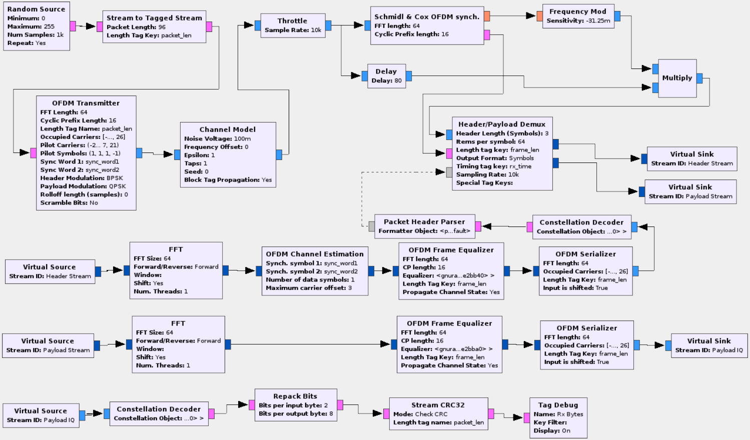
A stream is the connection between two blocks



- data is exchanged between blocks on predefined connections, *streams*
- these streams represent the output buffer of the producing block, and
- the input buffer(s) of the consuming block(s)

Core concept: Flow Graph

A Flow graph is the abstract representation of how GNU Radio connects different blocks



Random Source
Minimum: 0
Maximum: 255
Num Samples: 1k
Repeat: Yes

Stream to Tagged Stream
Packet Length: 96
Length Tag Key: packet_len

Throttle
Sample Rate: 10k

Schmidl & Cox OFDM synch.
FFT length: 64
Cyclic Prefix length: 16

Frequency Mod
Sensitivity: -31.25m

Delay
Delay: 80

Multiply

OFDM Transmitter
FFT Length: 64
Cyclic Prefix Length: 16
Length Tag Name: packet_len
Occupied Carriers: [-..., 26]
Pilot Carriers: (-2... 7, 21)
Pilot Symbols: (1, 1, 1, -1)
Sync Word 1: sync_word1
Sync Word 2: sync_word2
Header Modulation: BPSK
Payload Modulation: QPSK
Rolloff length (samples): 0
Scramble Bits: No

Channel Model
Noise Voltage: 100m
Frequency Offset: 0
Epsilon: 1
Taps: 1
Seed: 0
Block Tag Propagation: Yes

Header/Payload Demux
Header Length (Symbols): 3
Items per symbol: 64
Length tag key: frame_len
Output Format: Symbols
Timing tag key: rx_time
Sampling Rate: 10k
Special Tag Keys:

Virtual Sink
Stream ID: Header Stream

Virtual Sink
Stream ID: Payload Stream

Packet Header Parser
Formatter Object: <p...fault>

Constellation Decoder
Constellation Object: ...0> >

Virtual Source
Stream ID: Header Stream

FFT
FFT Size: 64
Forward/Reverse: Forward
Window:
Shift: Yes
Num. Threads: 1

OFDM Channel Estimation
Synch. symbol 1: sync_word1
Synch. symbol 2: sync_word2
Number of data symbols: 1
Maximum carrier offset: 3

OFDM Frame Equalizer
FFT length: 64
CP length: 16
Equalizer: <gnura...e2bb40> >
Length Tag Key: frame_len
Propagate Channel State: Yes

OFDM Serializer
FFT length: 64
Occupied Carriers: [-..., 26]
Length Tag Key: frame_len
Input is shifted: True

Virtual Source
Stream ID: Payload Stream

FFT
FFT Size: 64
Forward/Reverse: Forward
Window:
Shift: Yes
Num. Threads: 1

OFDM Frame Equalizer
FFT length: 64
CP length: 16
Equalizer: <gnura...e2bba0> >
Length Tag Key: frame_len
Propagate Channel State: Yes

OFDM Serializer
FFT length: 64
Occupied Carriers: [-..., 26]
Length Tag Key: frame_len
Input is shifted: True

Virtual Sink
Stream ID: Payload IQ

Virtual Source
Stream ID: Payload IQ

Constellation Decoder
Constellation Object: ...0> >

Repack Bits
Bits per input byte: 2
Bits per output byte: 8

Stream CRC32
Mode: Check CRC
Length tag name: packet_len

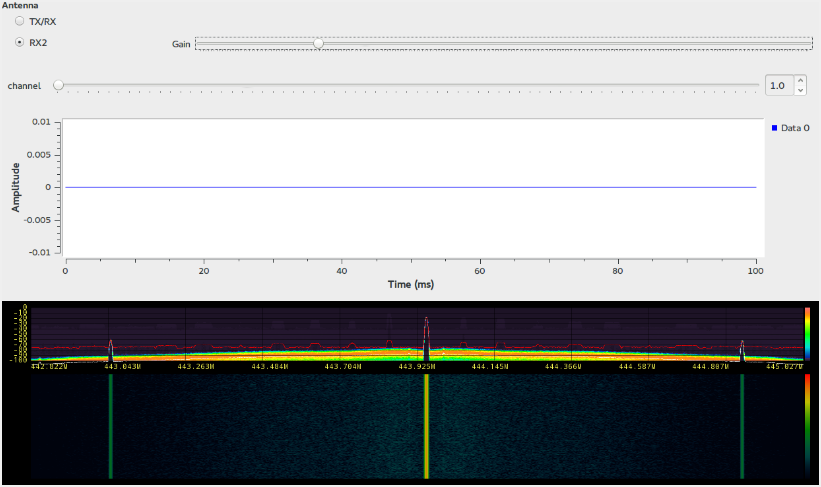
Tag Debug
Name: Rx Bytes
Key Filter:
Display: On

Let’s build a Demo!

Observing the whole LDP433 spectrum

- 25 channels, 433.075 MHz to 434.775 MHz
- channel spacing: 25 kHz
- Modulation: FM
- receiving and visualizing the whole spectrum
- demodulation and playback of a single channel

Spectrum Visualizer



Antenna

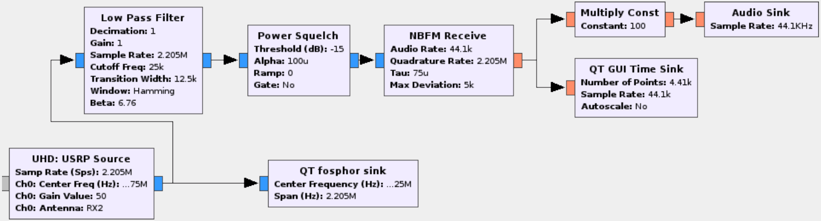
○ TX/RX

● RX2

Gain

channel 1.0

Spectrum Visualizer



Low Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 2.205M
Cutoff Freq: 25k
Transition Width: 12.5k
Window: Hamming
Beta: 6.76

Power Squelch
Threshold (dB): -15
Alpha: 100u
Ramp: 0
Gate: No

NBFM Receive
Audio Rate: 44.1k
Quadrature Rate: 2.205M
Tau: 75u
Max Deviation: 5k

Multiply Const
Constant: 100

Audio Sink
Sample Rate: 44.1KHz

QT GUI Time Sink
Number of Points: 4.41k
Sample Rate: 44.1k
Autoscale: No

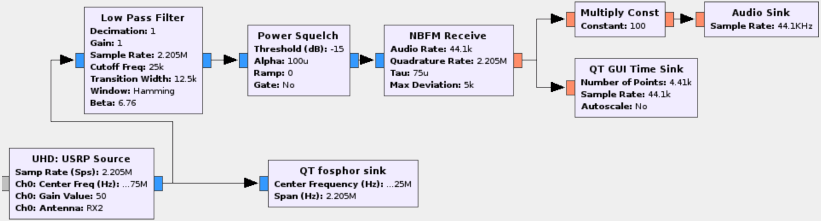
UHD: USRP Source
Samp Rate (Sps): 2.205M
Ch0: Center Freq (Hz): ...75M
Ch0: Gain Value: 50
Ch0: Antenna: RX2

QT fosphor sink
Center Frequency (Hz): ...25M
Span (Hz): 2.205M

Demo: What's happening behind the scenes?

- GNU Radio Companion (GRC) converts graphically defined flow graph to python file
- As python gets executed, blocks get instantiated and connections defined by calling GNU Radio methods
- The flow graph is started: GNU Radio starts calling the blocks' `work` methods
- `work` methods consume input and produce output
- GNU Radio calls the surrounding blocks again, now that there's new output space / new input

What's happening in the individual blocks?



Low Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 2.205M
Cutoff Freq: 25k
Transition Width: 12.5k
Window: Hamming
Beta: 6.76

Power Squelch
Threshold (dB): -15
Alpha: 100u
Ramp: 0
Gate: No

NBFM Receive
Audio Rate: 44.1k
Quadrature Rate: 2.205M
Tau: 75u
Max Deviation: 5k

Multiply Const
Constant: 100

Audio Sink
Sample Rate: 44.1KHz

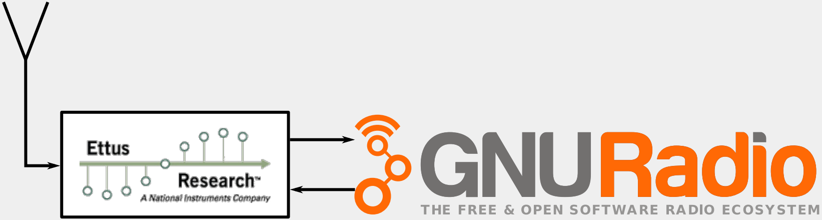
QT GUI Time Sink
Number of Points: 4.41k
Sample Rate: 44.1k
Autoscale: No

UHD: USRP Source
Samp Rate (Sps): 2.205M
Ch0: Center Freq (Hz): ...75M
Ch0: Gain Value: 50
Ch0: Antenna: RX2

QT fosphor sink
Center Frequency (Hz): ...25M
Span (Hz): 2.205M

What's happening on the hardware side of things?

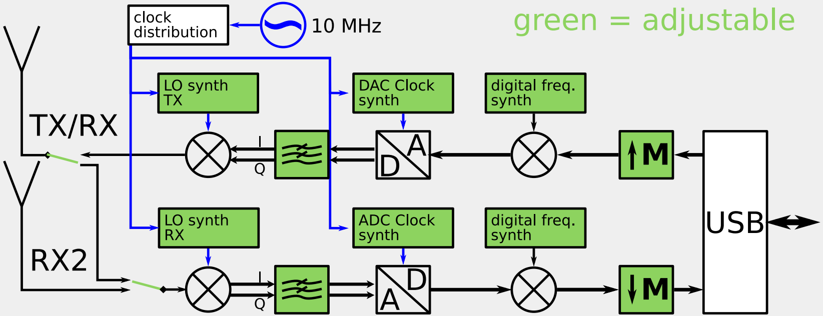
The Ettus USRP B210 is the interface between software and RF:


Ettus
Research™
A National Instruments Company
GNURadio
THE FREE & OPEN SOFTWARE RADIO ECOSYSTEM

Features of the B200/B210

Channels	B200: 1 TX & 1 RX, B210: 2 TX & 2 RX
Coverage	Seamless 70 MHz to 6 GHz
f_{ADC}	flexible, up to 56 MHz
user f_{sample}	flexible, $\frac{f_{\text{ADC}}}{N}, N \in 1, \ldots, 512$
Duplex	Full duplex
Analog Filters	Adjustable, up to f_{sample}
Digital Filters	Automatically chosen to optimize signal
Connectivity	USB3
Host Driver	Open Source, `https://github.com/EttusResearch/uhd/`
Firmware	Open Source
FPGA	Open Source
Schematics	Online, `http://files.ettus.com/schematics/b200/`

structure of the B2x0 direct receiver



Questions? Answers!

Now is the time for some questions and some answers, before we move on.

Demo: A talking clock

What? A clock that, every N seconds, says the current time.

Demo: A talking clock

What? A clock that, every N seconds, says the current time.

How? Using an existing text-to-speech program and python blocks.

Demo: A talking clock

What? A clock that, every N seconds, says the current time.

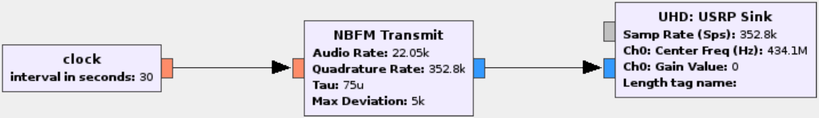
How? Using an existing text-to-speech program and python blocks.

Why? Yes.

Design Approach

- Using existing blocks, we can
  - Have an interface to the USRP
  - Generate FM out of audio samples
  - Have control over volume
- What we still need is a block that generates the voice samples
  - `itemize` is an established Text-to-Speak program
  - we need to prepend the message with a “ping”

Step I: Constructing a flow graph with missing components



clock
interval in seconds: 30

NBFM Transmit
Audio Rate: 22.05k
Quadrature Rate: 352.8k
Tau: 75u
Max Deviation: 5k

UHD: USRP Sink
Samp Rate (Sps): 352.8k
Ch0: Center Freq (Hz): 434.1M
Ch0: Gain Value: 0
Length tag name:

Step II: Adding a python block stub

gr_modtool allows us to create a module, and add a block stub:

Step II: Adding a python block stub

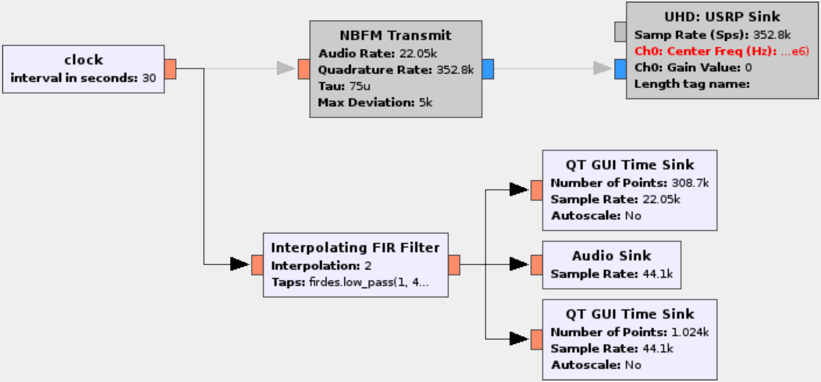
gr_modtool allows us to create a module, and add a block stub:

```
gr_modtool newmod talkingclock
Creating out-of-tree module in
./gr-speakingclock...Done.
>Use 'gr_modtool add' to add a new block to this
currently empty module.
cd gr-talkingclock
gr_modtool add
GNU Radio module name identified:  speakingclock
Enter block type:  source
Language (python/cpp):  python
...
```

Step III: Adding functionality

- Most important about our block is the `work` method:
  - gets called repeatedly
  - has the job of filling the output buffer, and returning how many output items were produced
- We add `tx_time` and start-of-burst *stream tags* so that the USRP knows when to transmit
- in the constructor, we make sure everything is set up correctly

Step IV: Putting it all together



clock
interval in seconds: 30

NBFM Transmit
Audio Rate: 22.05k
Quadrature Rate: 352.8k
Tau: 75u
Max Deviation: 5k

UHD: USRP Sink
Samp Rate (Sps): 352.8k
Ch0: Center Freq (Hz): ...e6)
Ch0: Gain Value: 0
Length tag name:

Interpolating FIR Filter
Interpolation: 2
Taps: firdes.low_pass(1, 4...

QT GUI Time Sink
Number of Points: 308.7k
Sample Rate: 22.05k
Autoscale: No

Audio Sink
Sample Rate: 44.1k

QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 44.1k
Autoscale: No

Useful Links

GNU Radio project `http://gnuradio.org`

Guided Tutorials `https://gnuradio.org/redmine/projects/gnuradio/wiki/Guided_Tutorials`

CGRAN `http://cgran.org`

PyBOMBS `http://pybombs.info`

GNU Radio mailing list discuss-gnuradio@gnu.org
Registration & Archive: `https://lists.gnu.org/mailman/listinfo/discuss-gnuradio`

Ettus `http://www.ettus.com`

UHD Manual `http://files.ettus.com/manual/`