Software Defined Radio

GNU Radio and the USRP

Overview

- What is Software Defined Radio?
- Advantages of Software Defined Radio
- Traditional versus SDR Receivers
- SDR and the USRP
- Using GNU Radio

2

Introduction

- What is Software Defined Radio (SDR)?
  - Getting code as close to the antenna as possible
  - Replacing hardware with software for modulation/demodulation
- Advantages:
  - Makes communications systems reconfigurable (adapting to new standards)
  - Flexible (universal software device - not special purpose)
  - Filters/Other Hardware
  - Cognitive Radio

3

Traditional Receiver

4

Traditional vs. SDR Receiver

5

SDR Receiver Using the USRP

6

Quadrature Signal Representation

The received signal, S(t), may be represented as follows:

$$S(t) = I(t)\cos(2\pi f_c t) + Q(t)\sin(2\pi f_c t)$$

f_c = carrier frequency

I(t) = in-phase component

Q(t) = quadrature component

Contain amplitude and phase information of baseband signal

•GNU Radio software uses I and Q components to demodulate signals

•USRP front end translates the signal to zero frequency and extracts I and Q

7

Extracting I(t) from S

$$S(t) = I(t)\cos(2\pi f_c t) + Q(t)\sin(2\pi f_c t)$$

Multiplying both sides by $\cos(2\pi f_c t)$:

$$\begin{aligned} S(t)\cos(2\pi f_c t) &= I(t)\cos^2(2\pi f_c t) + Q(t)\sin(2\pi f_c t)\cos(2\pi f_c t) \\ &= \frac{I(t)}{2}\left[1+\cos(4\pi f_c t)\right] + \frac{Q(t)}{2}\left[\sin(4\pi f_c t) + \sin(0)\right] \\ &= \frac{1}{2}I(t) + \frac{1}{2}I(t)\cos(4\pi f_c t) + + \frac{1}{2}Q(t)\sin(4\pi f_c t) \end{aligned}$$

Applying this signal to a low pass filter, the output will be:

$$\frac{1}{2}I(t)$$

8

Extracting Q(t) from S

$$S(t) = I(t)\cos(2\pi f_c t) + Q(t)\sin(2\pi f_c t)$$

Multiplying both sides by $\sin(2\pi f_c t)$:

$$\begin{aligned} S(t)\sin(2\pi f_c t) &= I(t)\cos(2\pi f_c t)\sin(2\pi f_c t) + Q(t)\sin^2(2\pi f_c t) \\ &= \frac{I(t)}{2}\left[\sin(4\pi f_c t) - \sin(0)\right] + \frac{Q(t)}{2}\left[1-\cos(4\pi f_c t)\right] \\ &= \frac{1}{2}I(t)\sin(4\pi f_c t) + \frac{1}{2}Q(t) - \frac{1}{2}Q(t)\cos(4\pi f_c t) \end{aligned}$$

Applying this signal to a low pass filter, the output will be:

$$\frac{1}{2}Q(t)$$

9

USRP Receiver Front End

10

Analog to Digital Converter (ADC)

- 12 bit A/D Converter (2^{12} levels)
- 2 volt peak-peak maximum input
- 64 Msamp/second

Sampling Interval:

$$\Delta t = \frac{1}{64 \times 10^6} = 0.0156 \mu S$$

Quantization Levels:

$$\Delta v = \frac{2}{2^{12}} = 0.488 mV$$

11

Decimation

- Original sampling rate is 64Msamp/sec
- Converts a portion of spectrum 32 MHz wide
- Generally we are interested is a narrower portion of the spectrum requiring a lower sampling rate
- USB cannot handle that high data rate
- Occurs in the FPGA of the USRP

f_s = 64Msamp/sec f_s = 64Msamp/sec f_s = 500Ksamp/sec

$$\frac{64M}{500K} = 128$$

12

SDR Receiver with USRP

13

USRP - Motherboard/Daughterboard

14

GNU Radio Software

- Community-based project started in 1998
- GNU Radio application consists of sources (inputs), sinks (outputs) and transform blocks
- Transform blocks: math, filtering, modulation/demodulation, coding, etc.
- Sources: USRP, audio input, file input, signal generator, …
- Sinks: USRP, audio output, file output, FFT, oscilloscope, …
- Blocks written in C++
- Python scripts used to connect blocks and form application

15

Design of a Receiver

- USRP: Set frequency of local oscillator (receive frequency), gain of amplifier, decimation factor
- GNU Radio application: use Python to specify and connect blocks that perform demodulation and decoding

16

Example: 400 - 500 MHz NBFM Receiver

- Problem: Receive an audio signal (up to 4 KHz) transmitted at 446 MHz using narrowband FM (NBFM) with a 16 KHz transmission bandwidth

17

Design Procedure

1. Plan the block diagram of system components
2. Determine block parameters
3. Determine decimation rates
4. Write Python script to specify the blocks and connect them together

18

NBFM Receiver: Block Diagram/Parameters

19

Determining the Decimation Factors

Total Decimation factor = 8000 = $D_1D_2D_3$

64Msamp/sec → 8Ksamp/sec

20

FPGA Decimation Factor, D_1

- Total Decimation factor = 8000 = $D_1D_2D_3$
- Maximize the decimation in FPGA
- Maximum decimation factor in FPGA = 256
- Select D_1 = 250 (factor of 8000)
- Output sample rate = 64Ms/s / 250 = 256Ks/s

21

Channel Filter Specification

Channel Filter

•Maximum frequency = 16 KHz → Reduce sample rate to 32 Ks/s

•256Ks/s / 32Ks/s ⟶ D_2 = 8

22

FM Demodulator

•Maximum frequency = 4 KHz → Reduce sample rate to 8 Ks/s

•32Ks/s / 8Ks/s ⟶ D_3 = 4

•FM Demodulator block "extracts" audio signal from FM waveform by operating on I and Q

23

Complete Application Design

•Total decimation ratio = 250*8*4 = 8000

•Problem: The audio card requires an input sample rate ≥ 44.1 Ks/s

•Solution: Use a Resampler to increase the output sample rate

24

Final Application Design

•Audio Card requires a sample rate ≥ 44.1 Ks/sec. Use 48 Ks/sec.

•Modify FM Demodulator to have a decimation factor of 1 (no change)

•Increase the sample rate to 48 Ks/sec with Resampler (x 3/2)

25

Implementing the Design

- Create a Python script to specify and connect the various GNU radio blocks
- Blocks are already written in C++
- USRP parameters are set within Python script
- # indicates that the line is a comment
- Refer to nbfm.py script

26

Setting the USRP Parameters

- The following code sets the USRP Parameters:

```
#Create USRP data source
u = usrp.source_c(decim_rate=250)

#Tell USRP what daughter board to use and displays name
#(0,0) means side A, 1st channel - picks daughterboard
rx_subdev_spec = (0,0)
u.set_mux(usrp.determine_rx_mux_value(u,rx_subdev_spec))
subdev = usrp.selected_subdev(u,rx_subdev_spec)
print "Using daughterboard",subdev.name()

#Tune it to supplied frequency
u.tune(0,subdev,frequency)
```

27

Channel Filter Design

- The following code specifies the channel filter and computes the coefficients

```
#create the channel filter coefficients
chan_taps = optfir.low_pass(
        1.0,    #Filter gain
        256e3,  #Sample Rate
        8000,   #one sided modulation BW (edge of passband)
        9000,   #one sided channel BW (edge of stopband)
        0.1,    #Passband ripple
        60)     #Stopband Attenuation
```

28

Channel Filter Creation

- The following code creates the channel filter using the coefficients computed:

```
#creates the channel filter with the coef found above
chan = gr.freq_xlating_fir_filter_ccf(
        8,      #Decimation rate
        chan_taps,  #coefficients
        0.0,    #Offset frequency - could be used to shift
        256e3)  #incoming sample rate
```

29

FM Demodulator

- The following code creates the FM demodulator.
- The demodulator block also includes a low pass filter.

```
#create the FM demodulator
demod = fm_demod_cf(self,
        32e3,  #Sample Rate at input
        1,  #Decimation
        5000,#Deviation
        3000, #edge of audio passband
        4000)  #edge of audio stopband
```

30

Resampler

- The following code creates the resampler.
- The resampler decimates and/or interpolates the data to adjust the sample rate.

```
#insert resampler to increase sample rate of 32K to 48K
# mult by 3 and divide by 2
rsamp = blks.rational_resampler_fff(self,3,2)
```

31

Connecting the Blocks

- The following code connects the blocks:

```
#connect the usrp output to the channel filter input
self.connect(u,chan)

#connect channel filter to demodulator
self.connect(chan,demod)

spkr = audio.sink(48000)

#connect demod to resampler and resampler to speaker
self.connect(demod,rsamp)
self.connect(rsamp,spkr)
```

Or, a single connect statement:

```
self.connect(u,chan,demod,rsamp,spkr)
```

32

Final Thoughts

- Demonstration
- Storing/creating data
- Transmitters
- Installing GNU radio
- Questions
- Where do we go from here?

33