GNU Radio를 이용한 AM Receiver 구현

이봉준

2008-07-25

1. 연구목적

Software Radio를 이해하고 GNU Radio와 Universal Software Radio Peripheral (USRP)를 이용한 AM Receiver를 구현한다.

2. GNU Radio and USRP

GNU Radio는 GNU General Public License하에 배포되는 signal processing package로서 electromagnetic spectrum을 이용해 radio spectrum을 이해하고 쉽게 사용할 수 있게 해준다.

또한 modular pipeline structure와 python언어의 쉬운 configuration과 flexibility 성질을 이용하여 프로그래밍 하기 쉽도록 만들어 졌다.

USRP는 software radio를 만들기 위한 high speed USB기반의 장치로서 4 개의 high speed digital to analog converter와 4 개의 high speed digital to analog converter, FPGA, glue logic으로 구성되어 있다. USRP는 GNU Radio와 무료로 제공되는 라이브러리를 이용해 자유롭게 디자인 할 수 있다.

3. Amplitude Modulation

3.1 Introduction of modulation

Modulation은 정보를 저장, 전송 하기 위해 baseband signal을 전기적 신호로 변환하는 것을 말한다. 원하는 정보에 따라 carrier 신호의 amplitude, frequency, phase 정보를 변경하여 변조된 신호를 얻는다.

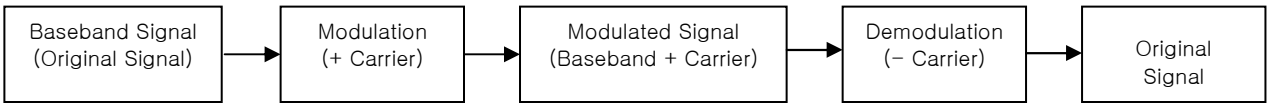
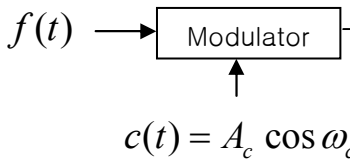


그림 1. Modulation Processing Block.

3.2 Amplitude Modulation with Large Carrier (DSB-LC or AM)



$$\phi(t) = (f(t) + A_c)\cos\omega_c t$$
$$= f(t)\cos\omega_c t + A_c \cos\omega_c t$$

DSB-LC방식은 DSB-SC (Suppressed Carrier) 방식에 large carrier를 더해주는 방식으로 일반적인 AM Radio방식으로 사용된다. $A_c \cos(\omega_c t)$를 carrier signal $f(t)$를 modulating signal이라 했을 때 다음과 같다.

$$F(\omega) = \mathrm{F}\{\mathrm{f(t)}\} \quad \Phi(\omega) = \mathrm{F}\{\phi(\mathrm{t})\} \tag{1}$$

$$\Phi(\omega) = \mathrm{F}[\mathrm{A_c}\cos(\omega_c t) + f(t)\cos(\omega_c t) \tag{2}$$

$$= \mathrm{F}\left(\frac{A_c}{2}e^{j\omega_c t} + \frac{A_c}{2}e^{-j\omega_c t} + \frac{f(t)}{2}e^{j\omega_c t} + \frac{f(t)}{2}e^{-j\omega_c t}\right) \tag{3}$$

$$= \pi A_c \delta(\omega - \omega_c) + \pi A_c \delta(\omega + \omega_c) + \frac{1}{2}\mathrm{F}(\omega - \omega_c) + \frac{1}{2}\mathrm{F}(\omega + \omega_c) \tag{4}$$

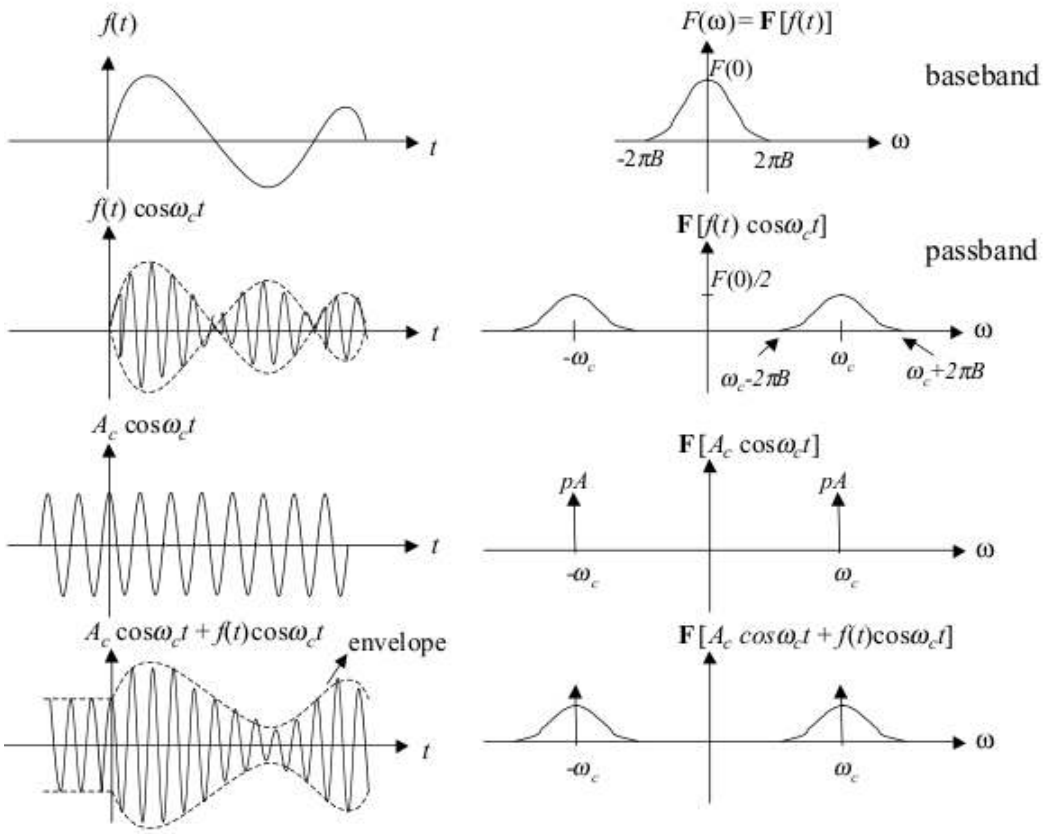


그림 2. DBS-LC Modulation Process.

3.3 Demodulation of modulated signal

DSB-LC 신호의 demodulation에는 주로 envelope detection방법이 이용되는데, 이것은 modulated signal를 diode 및 RC 회로를 통과시켜 양의 값의 envelope을 detection하여 original signal을 복구하는 방법이다. software signal processing에서는 magnitude processing을 통해 demodulation하게 된다.

4. AM Receiver

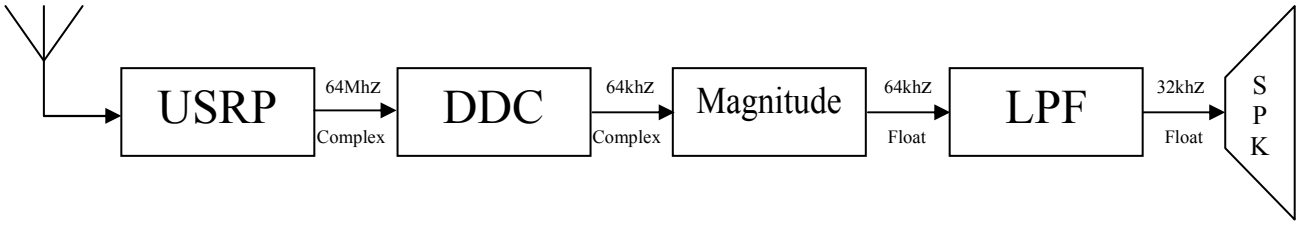


그림 3. AM Receiver Signal Process.

4.1 USRP Block

USRP의 RF단을 통해 Radio Signal을 수신하기 위해서는 interface설정이 필요하다. USRP를 설정하기 위한 코드는 다음과 같다.

```
src = usrp.source_c ()
src.set_decim_rate(usrp_decim)
subdev = usrp.selected_subdev(src, pick_subdevice(src))
usrp.tune(src, 0, subdev, 1107e3+64000)
```

usrp.source_c() 함수는 USRP Rx path interface로 연결하는 기능을 한다. 이때 '_c'는 complex로 데이터를 수신함을 의미한다. USRP내부의 ADC sampling rate는 64Mhz로 수행된다.

다음으로 usrp.tune() 함수를 이용해 center frequency를 setting하게 되는데 함수 인자 및 각각의 의미는 다음과 같다.

usrp.source_x.tune(u, chan, subdev, target_freq)

Parameters
- u: instance of usrp.source_*
- chan: DDC number
- subdev: daughterboard subdevice
- target_freq: frequency in Hz

위 sample source에서는 1107khz+64khz를 중심 주파수로 설정된 상태이다. 아래 그림은 sample source가 실행될 때 USRP를 통해 들어오는 signal를 frequency domain에서 본 모습니다.

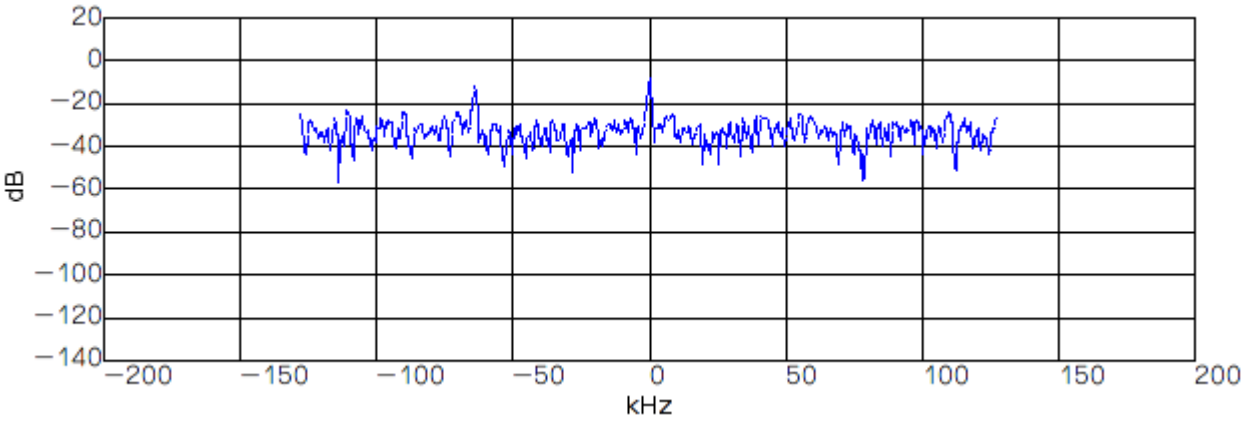


그림 4. Signal from USRP in frequency domain. Frequency centered at 1171khz.

4.2 Digital Down Converter (DDC) Block

USRP를 통해 64Mhz로 sampling된 signal은 DDC기능을 하는 gr.freq_xlating_fir_ccf() 함수를 통해 down converting되게 된다.

gr. freq_xlating_fir_filter _xxx (decimation, taps, center_freq, sampling_freq)

Parameters **decimation**: integer
taps: depends on function
center_freq: double
sampling_freq: double

gr.freq_xlating_fir_ccf() 함수는 내부적으로 frequency translation, FIR filter, decimation 과정을 수행하게 된다. 이런 기능을 통해 원하는 channel을 효과적으로 selection하고 wide bandwidth input을 narrow band signal로 decimate하게 된다. '_ccf'는 complex input complex output을 의미한다.

```
channel_coeffs = gr.firdes.low_pass (1, if_rate, 8000, 1000, gr.firdes.WIN_HANN)
ddc = gr.freq_xlating_fir_filter_ccf (if_decim, channel_coeffs, 64000, if_rate)
```

위 sample source에서는 중심 주파수를 64khz로 설정하고 sampling_freq 265khz, cutoff_freq 8khz, transition_with 1khz인 Low Pass Filter (LPF)와 계수가 4 인 decimation을 통해 최종적으로는 64khz로 sampling된 signal이 나오게 된다.

아래 그림은 DDC를 통과한 신호의 frequency domain에서의 모습이다.

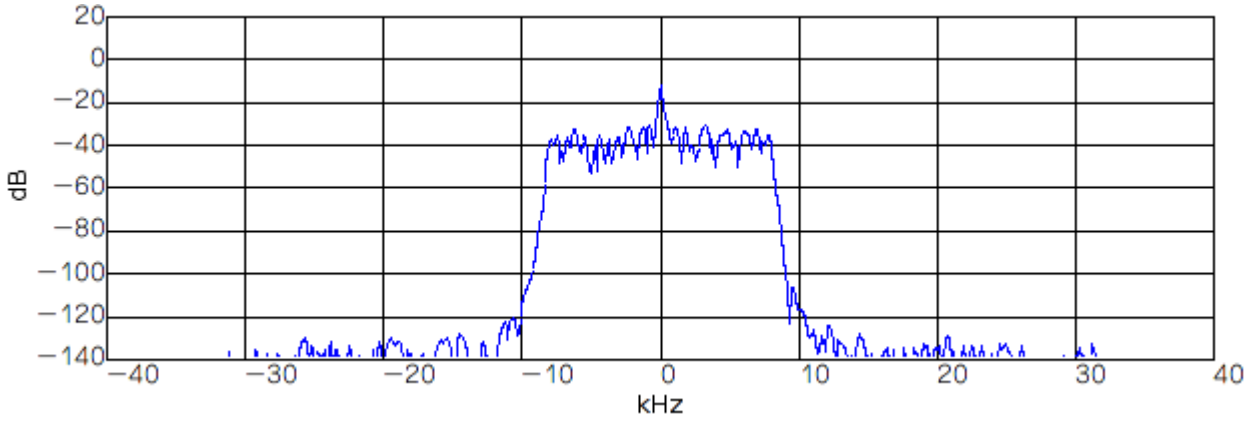


그림 5. Post-DDC Signal in frequency domain.

cutoff_freq와 transition_with를 각각 8khz, 1khz로 설정하는 이유는 AM Radio의 대여폭이 9khz이기 때문이다.

4.3 Magnitude Block

DSB-LC방식의 AM signal을 demodulation 하기 위해서 주로 envelope detection을 거치게 된다. software frequency에서는 magnitude를 통해 같은 과정을 수행하게 된다. 이때 사용되는 함수가 gr.complex_to_mag()이다.

```
magblock = gr.complex_to_mag()
```

gr.complex_to_mag() 함수는 complex signal을 받아 float형식의 magnitude를 출력으로 내보내는 역할을 한다. 이 과정을 거치면서 modulated signal이 demodulation이 되면서 original signal이 복원되게 된다.

4.4 LPF Block

Magnitude를 거쳐 복원된 original signal에서 불필요한 부분을 줄이고 가청 주파수 대역만 남기기 위해 LPF를 이용한다. gr.fir_filter_fff() 함수가 그 역할을 하게 된다.

gr. fir_filter_xxx (decimation, taps)

Parameters **decimation**: integer
taps: depends on function

gr.fir_filter_fff()는 float 형식의 Signal을 입력 받아 low pass filter와 decimation을

수행하고 float형식으로 내보내게 된다.

```
audio_coeffs = gr.firdes.low_pass (1.0, demod_rate, 9e3, 4e3, gr.firdes.WIN_HANN)
audio_filter = gr.fir_filter_fff (audio_decimation, audio_coeffs)
```

위 sample source에서는 sampling_freq 64khz, cutoff_freq 8khz, transition_with 4khz인 Low Pass Filter와 계수가 2 인 audio_decimation을 지나면서 최종적으로 32khz로 sampling된 signal이 출력된다. 아래 그림은 audio_filter을 통과한 signal의 frequency domain에서의 모습이다.

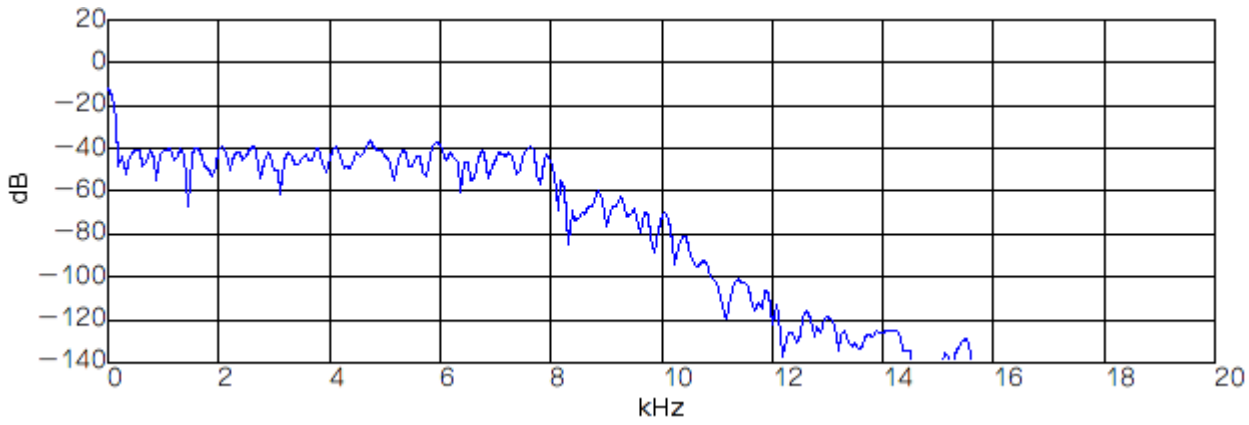


그림 6. Post-LPF Signal in frequency domain.

출력되는 sound signal의 volume을 설정하기 위해서는 gr.multiply_const_ff() 함수를 이용한다.

```
volumecontrol = gr.multiply_const_ff(.5)
```

gr.multiply_const_ff() 함수는 float형식의 constant signal을 입력 받아 크기를 증가시켜 float형식으로 출력하게 된다. 이렇게 나온 signal는 sound_sink() 함수를 통해 PC의 sound card로 입력되어 PC와 연결된 스피커를 통해 signal을 들을 수 있게 된다.

REFERENCES

[1] http://gnuradio.org/trac
[2] Ferrel G. Stremler, *Introduction to Communication Systems*, Addison-Wesley Pub. Co..