ANTSDR E310 플랫폼을 활용한 PySDR.org 완벽 정복: 종합 실습 튜토리얼

파트 I: 플랫폼 소개 및 환경 구성

소프트웨어 정의 라디오(SDR)의 세계는 이론적 지식과 실제 구현 능력의 조화를 요구합니다. PySDR.org는 파이썬을 기반으로 디지털 신호 처리(DSP)와 무선 통신의 핵심 개념을 학습할 수 있는 훌륭한 교육 자료를 제공합니다.[1] 본래 이 커리큘럼은 Analog Devices의 ADALM-PLUTO를 기준으로 작성되었으나, 본 튜토리얼은 한 단계 더 나아가 더욱 강력한 하드웨어 플랫폼인 ANTSDR E310을 사용하여 PySDR.org의 모든 실습 과정을 완벽하게 수행할 수 있도록 설계되었습니다. 이 보고서는 ANTSDR E310의 하드웨어적 특성을 심도 있게 분석하고, 실습에 필요한 개발 환경을 완벽하게 구축하며, PySDR의 파이썬 코드를 E310에 맞게 조정하고 최적화하는 모든 과정을 빠짐없이 안내할 것입니다.

1. ANTSDR E310: 아키텍처와 성능

ANTSDR E310은 단순한 학습용 SDR을 넘어, 현장 배치 및 상용 제품 적용까지 고려하여 설계된 고성능 임베디드 SDR 플랫폼입니다.[3] 이 장치의 핵심은 강력한 처리 능력과 유연성을 동시에 제공하는 자일링스 Zynq-7020 시스템 온 칩(SoC)입니다. Zynq-7020은 듀얼코어 ARM Cortex-A9 CPU와 7-시리즈 FPGA를 하나의 칩에 통합하여, 리눅스 운영체제 기반의 범용 컴퓨팅과 고속 실시간 신호 처리를 동시에 수행할 수 있습니다.[3]

RF 프론트엔드에는 Analog Devices의 고성능 AD9361 RF 트랜시버가 탑재되어 있습니다.[3] 이 트랜시버는 2개의 수신 채널과 2개의 송신 채널(2x2 MIMO)을 지원하며, 70 MHz부터 6 GHz에 이르는 넓은 주파수 범위와 최대 56 MHz의 순간 대역폭을 제공합니다.[4] 이는 PySDR.org의 기본 실습 범위를 넘어 훨씬 더 광범위한 무선 통신 실험을 가능하게 합니다. 또한, 1 GByte에 달하는 넉넉한 DDR3 RAM은 대용량 IQ 샘플 데이터를 버퍼링하고 처리하는 데 있어 탁월한 성능을 보장합니다.[4]

ANTSDR E310과 **ADALM-PLUTO**의 비교 분석

PySDR.org 실습을 진행하기에 앞서, 우리가 사용할 ANTSDR E310과 커리큘럼의 기준 장비인 ADALM-PLUTO의 차이점을 명확히 이해하는 것은 매우 중요합니다. ANTSDR E310은 ADALM-PLUTO에서 영감을 받아 설계되었으며 높은 수준의 펌웨어 호환성을 유지하지만, 하드웨어적으로는 상당한 업그레이드가 이루어졌습니다.[4] 이러한 차이점들은 실습 과정에서 성능의 이점으로 작용하거나, 때로는 미묘한 설정 변경을 요구하기도 합니다. 아래 표는 두 장치의 핵심 사양을 비교하고, 이러한 차이가 PySDR 실습에 미치는 영향을 분석한 것입니다.

기능	ANTSDR E310 (AD9361 버전)	ADALM-PLUTO	PySDR 실습에 미치는 영향
SoC	Xilinx Zynq XC7Z020 (85k 로직 셀) [4]	Xilinx Zynq Z-7010	더 강력한 FPGA와 CPU는 복잡한 실시간 신호 처리나 대용량 데이터 버퍼 사용 시 성능 저하 없이 안정적인 실행을 보장합니다.
RF 트랜시버	Analog Devices AD9361 [3]	Analog Devices AD9363 (기본)	E310은 더 넓은 주파수(70 MHz - 6 GHz)와 대역폭(최대 56 MHz)을 지원하여, 기본 Pluto의 성능을 넘어서는 고급 실험을 가능하게 합니다.[4]
RAM	1 GByte DDR3 [4]	512 MByte DDR3	더 큰 RAM 용량은 긴 시간 동안의 신호 캡처나 광대역 스펙트럼 분석과 같이 대규모 IQ 샘플 블록을 처리하는 실습에서 결정적인

			이점을 제공합니다.
클럭 정밀도	0.5ppm TCXO [4]	표준 크리스탈 발진기	고정밀 온도 보상 수정 발진기(TCXO)는 주파수 드리프트를 최소화하여 더 정확한 측정을 가능하게 하고, 수신 알고리즘에서 빈번한 주파수 보정의 필요성을 줄여줍니다.
I/O 포트	기가비트 이더넷, USB 2.0 OTG, USB-JTAG [4]	USB 2.0	기가비트 이더넷은 USB보다 잠재적으로 더 높은 데이터 전송률을 제공할 수 있으나, CPU 성능에 의해 제한될 수 있습니다.[9] 내장 JTAG은 고급 개발을 용이하게 합니다.
RF 프론트엔드	서브-밴드 처리, 필터 뱅크 [3]	단순한 프론트엔드	E310의 더 복잡한 RF 프론트엔드는 더 나은 선택도를 제공하지만, Pluto 대비 약 10dB의 기본 감쇠가 발생할 수 있어 소프트웨어에서 이득 보상이 필요할 수 있습니다.[12]

이 비교표에서 드러나는 중요한 사실은 ANTSDR E310이 단순히 '더 좋은 Pluto'가 아니라는 점입니다. 기가비트 이더넷과 온보드 USB-JTAG 포트의 존재는 이 장치가 처음부터 독립적인 임베디드 개발 플랫폼으로 설계되었음을 시사합니다.[4] PySDR 커리큘럼은 SDR을 PC나 가상 머신에 연결된 단순한 USB 주변 장치로 다루지만 [10], E310은 그 자체로 완전한 컴퓨터로서 작동할 수 있는 잠재력을 가지고 있습니다. 예를 들어, 호스트 PC 없이 E310의 ARM

프로세서에서 직접 GNU Radio나 파이썬 스크립트를 실행하는 것이 가능합니다. 본 튜토리얼에서는 PySDR 커리큘럼에 맞춰 E310을 주변 장치로 활용하는 방법에 집중하지만, 이러한 고급 기능들은 사용자가 기본 과정을 마친 후 더 깊이 있는 탐구를 할 수 있는 길을 열어줍니다.

2. 실험실 환경 설정

안정적이고 올바르게 구성된 개발 환경은 성공적인 SDR 실습의 첫걸음입니다. 이 섹션에서는 PySDR.org에서 권장하는 환경을 ANTSDR E310에 맞게 구축하는 과정을 단계별로 상세히 안내합니다. 이 과정을 통해 하드웨어, 네트워크, 드라이버, 파이썬 API가 모두 원활하게 연동되는 완벽한 실습 플랫폼을 마련하게 될 것입니다.

가상화된 리눅스 환경 구축

PySDR.org에서 권장하는 바와 같이, 우리는 VirtualBox를 사용하여 우분투 22(Ubuntu 22) 가상 머신(VM) 환경을 구축할 것입니다.[10] 이는 호스트 운영체제의 종류와 상관없이 일관된 개발 환경을 보장하는 가장 효과적인 방법입니다.

1. **VirtualBox** 및 우분투 **22** 설치**:** 최신 버전의 VirtualBox를 설치하고, 우분투 22 데스크톱(.iso) 이미지를 다운로드합니다.
2. 가상 머신 생성**:** VirtualBox에서 새 가상 머신을 생성합니다. 메모리 크기는 호스트 PC RAM의 50% 정도를 할당하고, 가상 하드 디스크는 동적 할당으로 최소 15 GB 이상을 확보하는 것이 좋습니다.
3. 우분투 설치**:** 생성된 VM을 시작하고, 다운로드한 우분투 22.iso 파일을 설치 미디어로 선택하여 설치를 진행합니다. 설치 옵션은 대부분 기본값을 따릅니다.
4. **VM** 설정 최적화**:** 설치 완료 후 VM을 종료하고, 설정 메뉴에서 시스템 > 프로세서 탭으로 이동하여 CPU 코어를 최소 3개 이상 할당합니다. 이는 컴파일 및 신호 처리 성능에 큰 영향을 미칩니다.
5. 게스트 확장 설치**:** VM을 다시 시작한 후, VirtualBox 메뉴의 '장치' > '게스트 확장 CD 이미지 삽입'을 선택하여 게스트 확장을 설치합니다. 이는 공유 클립보드, 화면 크기 조절 등 VM 사용 편의성을 크게 향상시킵니다.

하드웨어 연결 및 검증

이제 ANTSDR E310을 호스트 PC에 연결하고 VM과의 통신을 확인할 차례입니다.

1. **USB** 연결**:** ANTSDR E310의 여러 USB 포트 중 가운데에 위치한 **USB 2.0 OTG** 포트를 사용하여 호스트 PC와 연결합니다. 다른 포트는 전원 전용이므로 주의해야 합니다.[10] 연결이 성공하면, ADALM-PLUTO와 마찬가지로 E310은 호스트 PC에서 USB 이더넷 네트워크 어댑터로 인식됩니다.
2. 네트워크 연결 확인**:** E310은 기본적으로 192.168.2.1 IP 주소를 가집니다. 먼저 호스트 **PC**의 터미널(또는 명령 프롬프트)에서 다음 명령을 실행하여 하드웨어와의 기본적인 네트워크 연결을 확인합니다.
   Bash
   ```
   ping 192.168.2.1
   ```
   응답이 오지 않는다면, 드라이버 문제나 케이블 연결을 먼저 확인해야 합니다.
3. **VM** 연결 확인**:** 다음으로, 우분투 **VM** 내의 터미널에서 동일한 명령을 실행합니다.
   Bash
   ```
   ping 192.168.2.1
   ```
   VM에서도 성공적으로 응답을 받는다면, 하드웨어와 실습 환경 간의 통신 경로가 완벽하게 구축된 것입니다.[10] 이 확인 과정은 이후 발생할 수 있는 복잡한 문제들을 사전에 방지하는 매우 중요한 단계입니다.

드라이버 및 **API** 설치

ANTSDR E310을 파이썬에서 제어하기 위해서는 Analog Devices에서 제공하는 핵심 소프트웨어 스택을 설치해야 합니다. 이 스택은 여러 계층으로 구성되어 있으며, 각각의 역할은 다음과 같습니다.

- **libiio:** 하드웨어의 IIO(Industrial I/O) 커널 드라이버와 통신하는 저수준 크로스플랫폼 라이브러리입니다. 모든 하드웨어 제어의 기반이 됩니다.[10]
- **libad9361-iio:** AD9361/AD9363 RF 트랜시버 칩에 특화된 기능들을 담고 있는 보조 라이브러리입니다.[10]
- **pyadi-iio:** 본 튜토리얼에서 직접 사용하게 될 고수준 파이썬 API입니다. libiio의 복잡한 C 인터페이스를 사용하기 쉬운 객체 지향 파이썬 클래스로 추상화하여 제공합니다.[10]

우분투 VM의 터미널에서 다음 명령들을 순서대로 실행하여 필요한 모든 패키지와 라이브러리를 설치합니다.[10]

Bash

```
# 필수 빌드 도구 및 라이브러리 설치
sudo apt-get update
sudo apt-get install build-essential git libxml2-dev bison flex libcdk5-dev cmake python3-pip
libusb-1.0-0-dev libavahi-client-dev libavahi-common-dev libaio-dev

# libiio 설치 (v0.23 기준)
cd ~
git clone --branch v0.23 https://github.com/analogdevicesinc/libiio.git
cd libiio
mkdir build
cd build
cmake -DPYTHON_BINDINGS=ON..
make -j$(nproc)
sudo make install
sudo ldconfig

# libad9361-iio 설치
cd ~
git clone https://github.com/analogdevicesinc/libad9361-iio.git
cd libad9361-iio
mkdir build
cd build
cmake..
make -j$(nproc)
sudo make install

# pyadi-iio 설치 (v0.0.14 기준)
cd ~
git clone --branch v0.0.14 https://github.com/analogdevicesinc/pyadi-iio.git
cd pyadi-iio
pip3 install --upgrade pip
pip3 install -r requirements.txt
sudo python3 setup.py install
```

최종 검증 **(Smoke Test)**

모든 설치가 완료된 후, 최종적으로 파이썬 환경에서 ANTSDR E310이 올바르게 인식되는지

확인해야 합니다. VM에서 새 터미널을 열고 다음 파이썬 코드를 실행합니다.

Python

```
# python3 인터프리터 실행
python3

# adi 라이브러리 임포트 및 장치 연결
import adi
sdr = adi.Pluto('ip:192.168.2.1')

# 연결된 장치의 URI 출력
print(sdr.uri)
```

오류 없이 ip:192.168.2.1과 같은 출력이 나타난다면, 모든 설정이 성공적으로 완료된 것입니다.[10]

여기서 주목할 점은, 서로 다른 하드웨어인 ANTSDR E310이 어떻게 ADALM-PLUTO용 드라이버 및 API와 완벽하게 호환되는가 하는 점입니다. 이는 우연이 아니라, 두 장치 모두 리눅스 커널의 표준화된 하드웨어 인터페이스인 산업용 **I/O(Industrial I/O, IIO)** 프레임워크를 따르도록 설계되었기 때문입니다. libiio 라이브러리는 이 IIO 프레임워크와 사용자 공간 애플리케이션 사이의 다리 역할을 하며, pyadi-iio는 이를 다시 파이썬 친화적으로 감싼 것입니다.[14] 즉, E310의 설계자들이 의도적으로 리눅스 IIO 표준을 채택했기 때문에, 우리는 Pluto용으로 개발된 풍부한 소프트웨어 생태계를 그대로 활용할 수 있는 것입니다. 이 아키텍처에 대한 이해는 "마법처럼 작동한다"는 인식을 "표준을 따르기 때문에 작동한다"는 공학적 이해로 바꾸어 줍니다.

3. 하드웨어의 잠재력 극대화

기본 ADALM-PLUTO는 출고 시 RF 칩의 전체 성능보다 제한된 주파수 범위와 샘플링 속도로 설정되어 있습니다. 사용자들은 간단한 명령어를 통해 이 제한을 해제하여 칩의 본래 성능을 모두 활용하는데, 이를 흔히 "Pluto 핵(Hack)"이라고 부릅니다.[10] ANTSDR E310은 이미 더 우수한 AD9361 칩을 사용하며 70 MHz ~ 6 GHz의 전체 주파수 범위를 지원한다고 광고되지만 [4], 이 단계를 통해 장치의 펌웨어 환경 변수가 최대 성능을 내도록 올바르게 설정되었는지 확인하고 보장할 수 있습니다.

1. 장치에 **SSH**로 접속**:** VM의 터미널에서 다음 명령을 실행하여 E310의 내장 ARM CPU에 직접 접속합니다. 기본 비밀번호는 analog입니다.[10]
   Bash

```
ssh root@192.168.2.1
```

2. 환경 변수 확인 및 설정**:** 접속 후, 다음 명령어를 실행하여 장치가 AD9361의 확장된 기능을 사용하도록 설정합니다. 이는 Pluto에서 AD9364 호환 모드를 활성화하는 것과 유사한 과정입니다.
   Bash
   ```
   fw_setenv compatible ad9364
   fw_setenv attr_name compatible
   fw_setenv attr_val ad9361
   ```
   참고: 펌웨어 버전에 따라 fw_setenv 명령어의 문법이 약간 다를 수 있습니다. 오류 발생 시 fw_setenv attr_name compatible 과 fw_setenv attr_val ad9361 조합을 시도해 보십시오.[10]
3. 재부팅**:** 설정 변경을 적용하기 위해 장치를 재부팅합니다.
   Bash
   ```
   reboot
   ```

이 과정을 통해 우리는 ANTSDR E310이 물리적으로 가진 모든 RF 성능을 소프트웨어에서 온전히 활용할 준비를 마쳤습니다. 이제부터 진행될 모든 파이썬 실습은 이 극대화된 하드웨어 성능을 기반으로 합니다.

파트 **II:** 파이썬을 이용한 **SDR** 기본 운용

완벽하게 구성된 환경 위에서, 이제 본격적으로 파이썬 코드를 통해 ANTSDR E310을 제어하는 방법을 배웁니다. 이 파트에서는 pyadi-iio API를 사용하여 RF 신호를 수신하고 송신하는 가장 기본적인 작업을 마스터하고, E310의 하드웨어적 특성을 코드 수준에서 다루는 방법을 익히게 됩니다.

4. 첫 만남**:** 신호 수신 및 시각화

SDR의 "Hello, World!"는 주변의 RF 신호를 수신하여 그 스펙트럼을 눈으로 확인하는 것입니다. 이 과정을 통해 우리는 설정된 환경이 실제로 작동함을 확인하고, SDR 프로그래밍의 가장 기본적인 흐름을 익힐 수 있습니다.

SDR의 **"Hello, World!"** 스크립트

아래는 특정 주파수 대역의 신호를 수신하여 파워 스펙트럼 밀도(Power Spectral Density, PSD)를 계산하고 그래프로 그리는 완전한 파이썬 스크립트입니다.

Python

```python
import adi
import numpy as np
import matplotlib.pyplot as plt

# --- 1. 장치 연결 및 파라미터 설정 ---
sdr_ip = "ip:192.168.2.1"
sample_rate = 2.5e6  # 샘플링 속도: 2.5 MS/s
center_freq = 915e6  # 중심 주파수: 915 MHz (ISM 대역)
num_samps = 16384    # 수신할 샘플 개수

# ANTSDR E310에 연결
sdr = adi.Pluto(sdr_ip)

# 수신 파라미터 설정
sdr.sample_rate = int(sample_rate)
sdr.rx_lo = int(center_freq)
sdr.rx_rf_bandwidth = int(sample_rate)
sdr.rx_buffer_size = num_samps

# --- 2. 이득(Gain) 설정: E310 특성 고려 ---
# E310은 Pluto 대비 약 10dB 감쇠가 있을 수 있으므로, 수동으로 높은 이득 설정
sdr.gain_control_mode_chan0 = 'manual'
sdr.rx_hardwaregain_chan0 = 70.0 # dB, 0-70dB 범위에서 조절 가능

# --- 3. IQ 데이터 수신 ---
print("Receiving samples...")
samples = sdr.rx()
print(f"Received {len(samples)} samples.")

# --- 4. 스펙트럼 시각화 ---
# FFT를 사용하여 주파수 영역으로 변환
fft_result = np.fft.fftshift(np.fft.fft(samples))
```

```
psd = np.abs(fft_result)**2 / (num_samps * sample_rate)
psd_db = 10 * np.log10(psd)

# 주파수 축 생성
freqs = np.linspace(center_freq - sample_rate/2, center_freq + sample_rate/2, num_samps)

# 그래프 그리기
plt.figure(figsize=(10, 6))
plt.plot(freqs / 1e6, psd_db)
plt.title("Received Signal Spectrum (915 MHz ISM Band)")
plt.xlabel("Frequency (MHz)")
plt.ylabel("Power Spectral Density (dB/Hz)")
plt.grid(True)
plt.show()

# 장치 연결 해제 (스크립트 종료 시 자동으로 처리되지만 명시적으로 추가)
del sdr
```

코드 해설

1. 장치 연결 및 파라미터 설정: adi.Pluto(sdr_ip)를 통해 E310에 연결합니다.[10] pyadi-iio 라이브러리는 E310을 Pluto 호환 장치로 인식하여 동일한 Pluto 클래스를 사용합니다. 이후 sample_rate(샘플링 속도), rx_lo(수신 중심 주파수), rx_rf_bandwidth(아날로그 필터 대역폭), rx_buffer_size(한 번에 수신할 샘플 수) 등 핵심 파라미터를 설정합니다.[10]
2. 이득(Gain) 설정: 이 부분이 ANTSDR E310을 위한 핵심적인 조정 과정입니다. 연구에 따르면 E310은 Pluto에 비해 약 10dB의 수신 감쇠를 보일 수 있습니다.[12] 이는 E310의 더 정교한 RF 프론트엔드 설계(필터 뱅크 등)에서 비롯된 특성일 수 있습니다. 따라서 자동 이득 제어(AGC)를 끄고(gain_control_mode_chan0 = 'manual'), 수동으로 하드웨어 이득을 높은 값(예: 70.0 dB)으로 설정하여 약한 신호도 충분히 수신할 수 있도록 보상해 줍니다.[10] 이 값은 주변 환경에 따라 최적화가 필요할 수 있습니다.
3. IQ 데이터 수신: sdr.rx() 메소드를 호출하면, 설정된 rx_buffer_size만큼의 복소수(complex) IQ 샘플 데이터가 NumPy 배열 형태로 반환됩니다. 이 한 줄의 코드가 실제로 하드웨어로부터 데이터를 가져오는 역할을 수행합니다.
4. 스펙트럼 시각화: 수신된 시간 영역의 샘플 데이터는 그 자체로는 의미를 파악하기 어렵습니다. numpy.fft를 사용하여 고속 푸리에 변환(FFT)을 수행하면, 신호가 주파수 영역에서 어떻게 분포하는지(즉, 어떤 주파수 성분이 얼마나 강한지)를 나타내는 스펙트럼을 얻을 수 있습니다.[20] 이 스펙트럼을 그래프로 그려보면, 주변의 Wi-Fi, 휴대폰,

방송 등 다양한 RF 신호들을 시각적으로 확인할 수 있습니다.

이 간단한 스크립트가 성공적으로 실행되고 스펙트럼 그래프가 나타난다면, 복잡한 SDR 하드웨어를 파이썬이라는 고수준 언어로 완벽하게 제어하고 있음을 의미합니다. 여기서 pyadi-iio 라이브러리의 설계 철학이 빛을 발합니다. sdr.rx_lo = 915e6과 같이 하드웨어의 물리적 파라미터를 마치 파이썬 객체의 속성(property)처럼 간단하게 다룰 수 있게 한 것은, SDR과 DSP에 익숙하지 않은 입문자들도 하드웨어 제어의 복잡성에 얽매이지 않고 신호 처리 자체에 집중할 수 있도록 하는 매우 효과적인 추상화 방식입니다.[1] 본 튜토리얼은 이러한

pyadi-iio의 장점을 적극적으로 활용하여 학습 효과를 극대화할 것입니다.

5. pyadi-iio 인터페이스 심층 탐구: 버퍼와 송신

신호를 수신하고 시각화하는 데 성공했다면, 이제 pyadi-iio API의 더 깊은 기능들을 탐구할 차례입니다. 특히 데이터가 하드웨어와 파이썬 스크립트 사이를 어떻게 오가는지 이해하는 '버퍼 관리'와, 수동적인 수신을 넘어 능동적으로 전파를 생성하는 '송신' 기능은 모든 SDR 애플리케이션의 핵심입니다.

버퍼 관리의 이해

sdr.rx()를 호출할 때마다 데이터가 즉시 생성되는 것처럼 보이지만, 실제로는 여러 단계의 버퍼를 거칩니다. sdr.rx_buffer_size는 파이썬 스크립트가 한 번의 rx() 호출로 가져올 데이터 덩어리(chunk)의 크기를 정의합니다.[19] 이 값이 너무 작으면

rx() 호출이 너무 빈번해져 오버헤드가 발생하고, 너무 크면 메모리 사용량이 늘어나고 지연 시간(latency)이 길어집니다.

실시간 애플리케이션(예: 실시간 음성 통신)에서는 파이썬의 처리 속도가 SDR의 데이터 생성 속도를 따라가지 못할 경우 버퍼 오버플로우(수신) 또는 언더플로우(송신)가 발생할 수 있습니다. 이는 데이터 손실로 이어지므로, rx_buffer_size를 적절히 조절하고 효율적인 신호 처리 코드를 작성하는 것이 중요합니다.[19]

송신 기능 소개

수신이 세상의 소리를 듣는 것이라면, 송신은 세상에 나의 목소리를 내는 것입니다. pyadi-iio를 이용한 송신 과정은 수신과 대칭적인 구조를 가집니다.

1. 송신 파라미터 설정: 수신과 마찬가지로 송신을 위한 중심 주파수와 이득을 설정합니다.
   - sdr.tx_lo = int(center_freq)
   - sdr.tx_hardwaregain_chan0 = -10.0 (송신 이득은 보통 0 이하의 음수 값으로 설정하여 출력을 감쇠시킵니다. 과도한 출력은 장비 손상이나 전파법 위반의 원인이 될 수 있으므로 낮은 값에서 시작하는 것이 안전합니다.)
2. 송신 신호 생성: NumPy를 사용하여 송신할 디지털 IQ 샘플을 생성합니다. 가장 간단한 예는 복소 정현파(complex sinusoid)입니다.

   ```
   Python
   fs = int(sdr.sample_rate)
   N = 10000 # 보낼 샘플 수
   fc = 100000 # 100 kHz 주파수 톤 생성
   ts = 1.0/fs
   t = np.arange(0, N*ts, ts)
   i = np.cos(2*np.pi*t*fc) * 2**14
   q = np.sin(2*np.pi*t*fc) * 2**14
   iq_samples_to_tx = i + 1j*q
   ```

   중요: 송신 샘플의 진폭은 DAC(Digital-to-Analog Converter)의 최대 범위를 넘지 않도록 조절해야 합니다. pyadi-iio는 내부적으로 215−1 범위의 정수로 스케일링하므로, 생성된 샘플에 2**14 정도의 상수를 곱해주는 것이 일반적입니다.[10]
3. 단일 버스트(Burst) 송신: 생성된 샘플 배열을 sdr.tx() 메소드에 전달하여 송신합니다.

   ```
   Python
   sdr.tx(iq_samples_to_tx)
   ```

   이 명령은 iq_samples_to_tx 배열에 담긴 모든 샘플을 한 번만 송신합니다.

연속 송신을 위한 순환 버퍼 (Cyclic Buffer)

스펙트럼 분석기나 다른 수신기로 송신 신호를 안정적으로 관찰하기 위해서는 신호를 계속해서 반복적으로 송신해야 합니다. 파이썬 루프 안에서 sdr.tx()를 계속 호출하는 것은 높은 샘플링 속도에서는 비효율적이며 버퍼 언더플로우를 유발하기 쉽습니다.

이러한 경우를 위해 pyadi-iio는 순환 버퍼 기능을 제공합니다.[19] 순환 버퍼를 활성화하면,

sdr.tx()로 한 번 보낸 데이터가 파이썬의 개입 없이 하드웨어 레벨에서 무한히 반복 송신됩니다.

Python

```
# 1. 순환 버퍼 활성화
sdr.tx_cyclic_buffer = True

# 2. 샘플 전송 (이 시점부터 하드웨어에서 무한 반복 시작)
sdr.tx(iq_samples_to_tx)

print("Transmitting a 100 kHz tone continuously. Press Ctrl+C to stop.")
try:
    while True:
        pass # 파이썬 스크립트가 종료되지 않도록 대기
except KeyboardInterrupt:
    print("Stopping transmission.")

# 3. 송신 중지 및 버퍼 파괴 (필수)
sdr.tx_destroy_buffer()
sdr.tx_cyclic_buffer = False
```

순환 버퍼를 사용한 후 새로운 데이터를 송신하려면, 반드시 sdr.tx_destroy_buffer()를 호출하여 기존의 하드웨어 버퍼를 먼저 파괴해야 합니다. 그렇지 않으면 오류가 발생합니다. 이 기능은 안정적인 테스트 신호원을 생성하는 데 매우 유용하며, 다음 장에서 만들 FM 수신기의 성능을 테스트하는 데에도 활용될 수 있습니다.

파트 **III:** 고급 애플리케이션 및 **PySDR** 프로젝트 구현

기본적인 SDR 운용 기술을 익혔으니, 이제 PySDR.org 커리큘럼의 핵심 프로젝트들을 ANTSDR E310 플랫폼 위에서 직접 구현해 볼 차례입니다. 이 파트에서는 지금까지 배운 모든 기술과 DSP 이론을 통합하여, 실세계의 신호를 처리하는 완전한 SDR 애플리케이션을 처음부터 만들어 봅니다.

6. 프로젝트 **1:** 광대역 **FM** 방송 수신기 제작

이 프로젝트의 목표는 ANTSDR E310을 사용하여 우리 주변에 항상 존재하는 FM 라디오 방송 신호를 수신하고, 이를 실시간으로 복조하여 컴퓨터 스피커로 청취 가능한 오디오를

만들어내는 것입니다. 이 프로젝트는 PySDR의 주파수 영역, 필터, IQ 샘플링 등 여러 챕터의 개념을 종합적으로 활용하는 훌륭한 실습입니다.

FM 수신기 신호 처리 체인

FM 방송 수신은 다음과 같은 단계적인 신호 처리 과정을 거칩니다. 각 단계별 이론과 함께 완전한 파이썬 코드를 제공합니다.

Python

```
import adi
import numpy as np
from scipy.signal import firwin, resample_poly, lfilter
import sounddevice as sd

# --- 1. 튜닝 및 수집 (Acquisition) ---
sdr_ip = "ip:192.168.2.1"
center_freq = 99.5e6  # 수신할 FM 방송 주파수 (예: 99.5 MHz)
sample_rate = 240e3   # FM 방송 대역폭을 충분히 포함하는 샘플링 속도
num_samps_per_frame = 16384 # 프레임 당 샘플 수

sdr = adi.Pluto(sdr_ip)
sdr.rx_lo = int(center_freq)
sdr.sample_rate = int(sample_rate)
sdr.rx_rf_bandwidth = int(sample_rate)
sdr.rx_buffer_size = num_samps_per_frame
sdr.gain_control_mode_chan0 = 'manual'
sdr.rx_hardwaregain_chan0 = 60.0 # 주변 환경에 맞게 조절

# --- 2. 광대역 FM 복조 (Demodulation) ---
# 이 함수는 IQ 샘플을 받아 FM 복조를 수행합니다.
def fm_demod(iq_samples):
    # 위상차 계산을 통한 주파수 편이 추출
    x = iq_samples[1:] * np.conj(iq_samples[:-1])
    # np.angle()을 사용하여 위상(주파수 정보)을 오디오 신호로 변환
    audio = np.angle(x)
    return audio
```

```
# --- 3. 저역 통과 필터 (Low-Pass Filter) ---
# 모노 오디오 신호(최대 15kHz)만 통과시키는 FIR 필터 설계
cutoff_freq = 15e3  # 15 kHz
num_taps = 101
lpf_taps = firwin(num_taps, cutoff_freq, fs=sample_rate)

# --- 4. 디엠퍼시스 (De-emphasis) ---
# FM 방송의 고주파 강조를 원상 복구하는 필터
# 시간 상수 tau = 75us (북미 표준)
tau = 75e-6
deemphasis_b =
deemphasis_a = [1, -np.exp(-1/(tau * sample_rate))]

# --- 5. 리샘플링 (Resampling) ---
# 오디오 카드에 맞는 샘플링 속도(48kHz)로 변환
audio_sample_rate = 48000
resample_factor_up = int(audio_sample_rate / 1000)
resample_factor_down = int(sample_rate / 1000)

# --- 6. 실시간 오디오 출력 ---
# sounddevice 스트림 설정
audio_stream = sd.OutputStream(samplerate=audio_sample_rate, channels=1, dtype='float32')
audio_stream.start()

print("Starting FM radio receiver. Press Ctrl+C to exit.")
try:
    while True:
        # 1. IQ 샘플 수신
        iq_data = sdr.rx()

        # 2. FM 복조
        audio_raw = fm_demod(iq_data)

        # 3. 저역 통과 필터링
        audio_filtered = lfilter(lpf_taps, 1.0, audio_raw)

        # 4. 디엠퍼시스 필터링
        audio_deemphasized = lfilter(deemphasis_b, deemphasis_a, audio_filtered)

        # 5. 리샘플링
        audio_resampled = resample_poly(audio_deemphasized, resample_factor_up,
resample_factor_down)
```

```
        # 6. 오디오 출력
        audio_stream.write(audio_resampled.astype(np.float32))

except KeyboardInterrupt:
    print("Stopping receiver.")
finally:
    audio_stream.stop()
    audio_stream.close()
    del sdr
```

처리 과정 상세 설명

1. 튜닝 및 수집: sdr.rx_lo를 원하는 FM 방송국의 중심 주파수로 설정합니다. FM 방송 채널 하나의 대역폭은 약 200 kHz이므로, 나이퀴스트 이론에 따라 이를 충분히 포함할 수 있는 240 kHz (240e3) 정도의 샘플링 속도를 사용합니다.[23]
2. 광대역 **FM** 복조: FM 신호의 정보는 반송파의 '주파수 변화'에 담겨 있습니다. 이를 복조하는 가장 간단하고 효율적인 디지털 방식은 연속된 샘플 간의 '위상 변화율'을 계산하는 것입니다. iq_samples[1:] * np.conj(iq_samples[:-1]) 코드는 현재 샘플과 한 샘플 이전의 켤레 복소수(conjugate)를 곱하여 두 샘플 간의 위상 차이를 계산하고, np.angle() 함수가 이 복소수의 위상각을 추출하여 순간 주파수 편이를 나타내는 오디오 신호로 변환합니다.[23]
3. 저역 통과 필터링: 복조된 신호에는 우리가 듣고자 하는 모노 오디오(0~15 kHz)뿐만 아니라, 스테레오 파일럿 톤(19 kHz), 스테레오 차 신호(38 kHz 주변), RDS 데이터(57 kHz 주변) 등 다양한 정보가 섞여 있습니다. scipy.signal.firwin을 사용하여 차단 주파수가 15 kHz인 저역 통과 필터(LPF)를 설계하고, 이 필터를 통과시켜 모노 오디오 신호만 깨끗하게 분리합니다.[24]
4. 디엠퍼시스: FM 방송은 송신 시 잡음이 많은 고주파수 오디오 성분을 의도적으로 증폭(프리엠퍼시스, pre-emphasis)해서 보냅니다. 수신기에서는 이와 반대로 고주파수 성분을 감쇠시키는 디엠퍼시스 필터를 적용해야 원래의 평탄한 음색을 복원할 수 있습니다.[24] 북미/한국 표준인 75µs 시간 상수에 해당하는 간단한 1차 IIR 필터를 구현하여 적용합니다.
5. 리샘플링 **(Decimation):** 현재 오디오 신호의 샘플링 속도는 240 kHz로, 일반적인 오디오 장치에서 사용하는 44.1 kHz나 48 kHz에 비해 매우 높습니다. scipy.signal.resample_poly 함수를 사용하여 신호의 왜곡 없이 샘플링 속도를 효율적으로 48 kHz로 낮춥니다. 이는 불필요한 데이터를 줄여 CPU 부하를 감소시키는 중요한 과정입니다.[23]
6. 실시간 오디오 출력: 최종적으로 처리된 오디오 샘플을 컴퓨터의 사운드 카드로 보내기 위해 sounddevice 라이브러리를 사용합니다.[27] sd.OutputStream을 생성하고, 메인 루프 안에서 지속적으로 IQ 샘플을 수신-처리한 후, 그 결과를 audio_stream.write()를 통해 스피커로 출력합니다.

이 스크립트를 실행하면, 여러분의 ANTSDR E310과 컴퓨터는 완벽하게 작동하는 FM 라디오가 됩니다.

7. 프로젝트 2: BPSK/QPSK 디지털 송수신기 구현

이 프로젝트는 SDR의 진정한 힘을 보여주는 실습입니다. "Hello World"와 같은 간단한 텍스트 메시지를 BPSK 또는 QPSK와 같은 디지털 변조 방식으로 변환하여 ANTSDR E310의 송신기로 전송하고, 동시에 수신기로 그 신호를 받아 다시 원래의 텍스트로 복원하는 완전한 통신 시스템을 구현합니다. 이 과정은 PySDR의 디지털 변조, 펄스 성형, 동기화 챕터의 핵심 이론들을 코드로 구현하는 결정체입니다.[30]

송신기 구현 (tx.py)

송신기는 텍스트 데이터를 RF 파형으로 변환하는 역할을 합니다.

Python

```
import adi
import numpy as np
import matplotlib.pyplot as plt

# --- 송신기 설정 ---
sdr_ip = "ip:192.168.2.1"
center_freq_tx = 915e6
sample_rate_tx = 1e6
sdr = adi.Pluto(sdr_ip)
sdr.tx_lo = int(center_freq_tx)
sdr.sample_rate = int(sample_rate_tx)
sdr.tx_rf_bandwidth = int(sample_rate_tx)
sdr.tx_hardwaregain_chan0 = -10 # -80 ~ 0 dB

# --- 1. 데이터 준비 및 심볼 매핑 ---
message = "Hello PySDR!"
```

```
# 텍스트를 비트 스트림으로 변환
bits = np.unpackbits(np.array([ord(char) for char in message], dtype=np.uint8))
# BPSK 변조: 0 -> -1, 1 -> 1
symbols = np.array([-1 if bit == 0 else 1 for bit in bits])

# --- 2. 펄스 성형 (Pulse Shaping) ---
# 심볼 간 간섭(ISI)을 줄이고 대역폭을 제한하기 위함
sps = 8 # 심볼 당 샘플 수 (oversampling factor)
num_taps = 101
beta = 0.35 # Roll-off factor
# Root-Raised Cosine (RRC) 필터 설계
t = np.arange(-num_taps//2, num_taps//2 + 1)
h_rrc = (np.sin(np.pi*t/sps*(1-beta)) + 4*beta*t/sps*np.cos(np.pi*t/sps*(1+beta))) /
(np.pi*t/sps*(1-(4*beta*t/sps)**2))
h_rrc[num_taps//2] = (1 + beta*(4/np.pi - 1))
h_rrc[t == sps/(4*beta)] = beta/np.sqrt(2)*((1+2/np.pi)*np.sin(np.pi/(4*beta)) +
(1-2/np.pi)*np.cos(np.pi/(4*beta)))
h_rrc[t == -sps/(4*beta)] = h_rrc[t == sps/(4*beta)]

# 심볼을 업샘플링하고 RRC 필터와 컨볼루션
symbols_upsampled = np.zeros(len(symbols) * sps)
symbols_upsampled[::sps] = symbols
tx_samples = np.convolve(symbols_upsampled, h_rrc, mode='full')

# --- 3. 전송 ---
# DAC 범위에 맞게 스케일링
tx_samples = tx_samples * 2**14 / np.max(np.abs(tx_samples))

print("Transmitting message...")
sdr.tx(tx_samples.astype(np.complex64))
print("Transmission complete.")

del sdr
```

1. 데이터 준비 및 심볼 매핑: 전송할 문자열을 np.unpackbits를 사용해 비트의 배열로 변환합니다. 그 후, BPSK(Binary Phase Shift Keying) 변조 규칙에 따라 비트 0은 -1로, 1은 1로 매핑하여 심볼 스트림을 생성합니다.
2. 펄스 성형: 디지털 심볼을 그대로 전송하면 사각파 형태가 되어 매우 넓은 대역폭을 차지하게 됩니다. 이를 방지하고 심볼 간 간섭(ISI)을 최소화하기 위해 펄스 성형 필터를 사용합니다.[31] 여기서는 통신 시스템에서 널리 사용되는 RRC(Root-Raised Cosine) 필터를 설계하고, 업샘플링된 심볼 스트림과 컨볼루션하여 대역폭이 제한된 부드러운 파형을 만듭니다.
3. 전송: 최종 생성된 복소수 샘플 배열을 DAC 범위에 맞게 정규화한 후, sdr.tx()를 통해

공중으로 송신합니다.

수신기 구현 **(rx.py)**

수신기는 공중의 RF 신호를 받아 원래의 텍스트 메시지를 복원하는 복잡한 과정을 수행합니다.

Python

```
import adi
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import correlate

# --- 수신기 설정 ---
sdr_ip = "ip:192.168.2.1"
center_freq_rx = 915e6
sample_rate_rx = 1e6
num_samps_rx = 500000 # 송신 신호를 포함할 만큼 충분히 길게 수신

sdr = adi.Pluto(sdr_ip)
sdr.rx_lo = int(center_freq_rx)
sdr.sample_rate = int(sample_rate_rx)
sdr.rx_rf_bandwidth = int(sample_rate_rx)
sdr.rx_buffer_size = num_samps_rx
sdr.gain_control_mode_chan0 = 'manual'
sdr.rx_hardwaregain_chan0 = 20.0 # 송수신 거리에 따라 조절

# --- 1. 수신 및 정합 필터링 ---
rx_samples = sdr.rx()
# 송신기에서 사용한 것과 동일한 RRC 필터 (정합 필터)
# (tx.py의 RRC 필터 생성 코드와 동일)
#... h_rrc 생성 코드...
rx_filtered = np.convolve(rx_samples, h_rrc, mode='full')

# --- 2. 동기화 (간략화된 버전) ---
# 타이밍 동기화: 심볼 에너지 최대 지점 찾기
sps = 8
```

```
energy = np.abs(rx_filtered)**2
timing_est = np.argmax(correlate(energy, np.ones(sps), mode='valid'))

# --- 3. 심볼 디코딩 ---
# 최적의 타이밍에 샘플링
rx_symbols = rx_filtered[timing_est::sps]
# BPSK 복조: 실수부의 부호로 0과 1 판단
rx_bits = (np.real(rx_symbols) > 0).astype(int)

# --- 4. 데이터 검증 ---
# 비트를 바이트로 묶고 텍스트로 변환
try:
    # 패딩된 0을 제거하기 위해 마지막 null 바이트를 찾음
    null_idx = np.where(rx_bits == 0)[-1]
    if (null_idx+1) % 8 == 0:
        rx_bits_packed = rx_bits[:null_idx+1]
    else:
        rx_bits_packed = np.packbits(rx_bits)

    message_decoded = "".join([chr(byte) for byte in rx_bits_packed])
    print(f"Decoded Message: {message_decoded}")
except Exception as e:
    print(f"Decoding failed: {e}")

del sdr
```

1. 수신 및 정합 필터링: 송신된 신호를 포함하도록 충분히 긴 시간 동안 샘플을 수신합니다. 그 후, 송신기에서 사용한 것과 동일한 **RRC** 필터를 사용하여 컨볼루션을 수행합니다. 수신기에서 송신기와 동일한 필터를 사용하는 것을 '정합 필터링(Matched Filtering)'이라고 하며, 이는 수신 신호의 신호 대 잡음비(SNR)를 최대화하는 최적의 방법입니다.[31]
2. 동기화: 실제 통신 환경에서 가장 어려운 부분입니다. 수신기는 언제 신호가 시작되었는지(타이밍 동기), 송신기와 수신기 간의 미세한 주파수 차이는 얼마인지(주파수 동기)를 알아내야 합니다. 위 코드는 sps 길이의 창을 이동시키며 에너지가 최대가 되는 지점을 찾아 심볼의 시작점을 추정하는 간단한 타이밍 동기화 기법을 보여줍니다. 실제 시스템은 훨씬 더 정교한 알고리즘을 사용합니다.[32]
3. 심볼 디코딩: 동기화를 통해 찾은 최적의 샘플링 시점에서 신호를 추출하여 심볼을 복원합니다. BPSK의 경우, 샘플의 실수(real) 부분이 양수이면 1, 음수이면 0으로 간단히 결정할 수 있습니다.
4. 데이터 검증: 복원된 비트 스트림을 np.packbits를 사용하여 8비트 바이트로 묶고, 각 바이트를 아스키(ASCII) 문자로 변환하여 원래의 메시지를 복원합니다.

이 디지털 송수신기 프로젝트를 성공적으로 완수하는 것은 SDR의 핵심 패러다임을 체득했음을

의미합니다. ANTSDR E310이라는 하드웨어는 단지 'RF 신호를 비트로', '비트를 RF 신호로' 변환해주는 고성실도의 변환기 역할을 할 뿐입니다. 이 하드웨어가 FM 라디오가 될지, QPSK 송수신기가 될지, 아니면 스펙트럼 분석기가 될지를 결정하는 것은 전적으로 호스트 컴퓨터에서 실행되는 파이썬 코드, 즉 '소프트웨어'입니다. 이것이 바로 소프트웨어 정의 라디오의 본질이며, 이 튜토리얼을 통해 사용자가 얻게 될 가장 중요한 통찰입니다.[1]

파트 **IV:** 결론 및 추가 탐구

지금까지 ANTSDR E310 플랫폼을 사용하여 PySDR.org의 핵심 실습 과정을 성공적으로 완수했습니다. 이 마지막 파트에서는 지금까지의 학습 내용을 정리하고, 본 튜토리얼에서 다루지 않은 ANTSDR E310의 고급 기능들을 소개하며, 여러분의 SDR 여정을 한 단계 더 발전시킬 수 있는 방향을 제시합니다.

8. 요약 및 다음 단계

성취 내용 검토

본 튜토리얼을 통해 여러분은 다음과 같은 핵심 역량을 습득했습니다.

- 완전한 **SDR** 개발 환경 구축**:** 고성능 SDR 하드웨어(ANTSDR E310)를 가상화된 리눅스 환경에 연결하고, libiio와 pyadi-iio를 포함한 필수 드라이버 및 API 스택을 성공적으로 설치하고 검증했습니다.
- **pyadi-iio**를 통한 하드웨어 제어**:** 파이썬을 사용하여 SDR의 수신 및 송신 파라미터를 설정하고, IQ 데이터를 효율적으로 처리하는 방법을 마스터했습니다. 특히, E310의 하드웨어적 특성(이득 차이)을 코드 수준에서 보상하는 실질적인 문제 해결 능력을 갖추게 되었습니다.
- 실용적인 **SDR** 애플리케이션 제작**:** FM 라디오 수신기와 BPSK 디지털 송수신기라는 두 개의 완전한 무선 시스템을 파이썬과 NumPy/SciPy만을 사용하여 처음부터 구현했습니다. 이를 통해 DSP 이론이 실제 코드와 어떻게 연결되는지 깊이 있게 이해하게 되었습니다.

PySDR를 넘어서**: ANTSDR E310**의 잠재력

본 튜토리얼은 PySDR 커리큘럼에 집중하기 위해 ANTSDR E310의 모든 기능을 활용하지는 않았습니다. 여러분의 다음 목표는 이 강력한 하드웨어의 숨겨진 잠재력을 탐구하는 것이 될 수 있습니다.

- **MIMO (Multiple-Input Multiple-Output):** E310은 2개의 송신 안테나와 2개의 수신 안테나를 동시에 사용할 수 있습니다.[4] 이는 PySDR의 후반부 챕터에서 다루는 빔포밍(Beamforming), 공간 다중화(Spatial Multiplexing)와 같은 고급 MIMO 기술을 실험할 수 있는 완벽한 플랫폼을 제공합니다.[2]
- 기가비트 이더넷**:** USB 2.0 연결 외에도, 기가비트 이더넷 포트를 사용하여 더 높은 데이터 처리량이나 원격 네트워크를 통한 SDR 제어가 가능합니다.[4] 다만, E310 내장 ARM CPU의 처리 능력에 따라 최대 전송률이 제한될 수 있다는 점은 유의해야 합니다.[9]
- 독립형**(Standalone)** 운용**:** ANTSDR E310의 가장 큰 특징은 그 자체가 완전한 임베디드 컴퓨터라는 점입니다. 호스트 PC 없이, E310의 ARM 프로세서에서 직접 GNU Radio나 여러분이 작성한 파이썬 스크립트를 실행할 수 있습니다. 이는 드론, 원격 센서 등 독립적으로 작동해야 하는 애플리케이션을 개발할 때 필수적인 기능입니다.

추가 프로젝트 아이디어

여러분이 습득한 기술은 수많은 최첨단 오픈소스 무선 통신 프로젝트의 문을 열어줍니다. ANTSDR E310의 Pluto 호환 펌웨어는 다음과 같은 프로젝트들을 실험하기에 이상적입니다.

- **srsRAN:** 오픈소스 LTE/5G 네트워크 스택으로, E310을 사용하여 소규모 사설 LTE 또는 5G 기지국을 구축하고 실험해 볼 수 있습니다.[6]
- **Openwifi:** FPGA 기반의 완전한 오픈소스 Wi-Fi 칩 설계 및 소프트웨어 스택입니다. E310의 Zynq SoC를 활용하여 Wi-Fi 프로토콜의 가장 깊은 수준까지 탐구할 수 있습니다.[6]

본 튜토리얼은 여러분의 SDR 학습 여정의 시작점입니다. ANTSDR E310이라는 강력한 도구와 PySDR를 통해 다진 탄탄한 기초를 바탕으로, 무한한 무선 통신의 세계를 자유롭게 탐험하고 창조해 나가시길 바랍니다.

참고 자료

1. PySDR: A Guide to SDR and DSP using Python - TIB AV-Portal, 9월 21, 2025에 액세스, https://av.tib.eu/media/53319
2. PySDR: A Guide to SDR and DSP using Python, 9월 21, 2025에 액세스, https://pysdr.org/
3. USRP E310 Datasheet - Ettus Research, 9월 21, 2025에 액세스, https://www.ettus.com/wp-content/uploads/2019/01/USRP_E310_Datasheet.pdf
4. MicroPhase ANTSDR E310 AD9361 Software Defined Radio Open Source SDR ZYNQ7020, 9월 21, 2025에 액세스, https://www.ebay.com/itm/156564730461

5. MicroPhase ANTSDR E310 AD9363 SDR Development Board for ADI Pluto Openwifi - eBay, 9월 21, 2025에 액세스, https://www.ebay.com/itm/375459506423
6. MicroPhase ANTSDR E310 Software Defined Radio Demo Board transceiver ZYNQ 7000 SoC ADI AD9361 SDR transmitter and receiver - AliExpress, 9월 21, 2025에 액세스, https://www.aliexpress.com/item/1005003181244737.html
7. MicroPhase ANTSDR E310 AD9363 SDR Development Board for ADI Pluto Communication- | eBay, 9월 21, 2025에 액세스, https://www.ebay.com/itm/126254547389
8. MicroPhase ANTSDR E310 AD9363 SDR Development Board for ADI Pluto ... - AliExpress, 9월 21, 2025에 액세스, https://www.aliexpress.com/item/3256809376273708.html
9. ANTSDR: Low-cost SDR Alternatives for Ettus Research and Lime Microsystems | Hacker News, 9월 21, 2025에 액세스, https://news.ycombinator.com/item?id=41213514
10. PlutoSDR in Python | PySDR: A Guide to SDR and DSP using Python, 9월 21, 2025에 액세스, https://pysdr.org/content/pluto.html
11. MicroPhase ANTSDR E310 Software Defined Radio transceiver ZYNQ 7000 SoC ADI AD9361 AD9363 MIMO SDR transmitter and receiver - AliExpress, 9월 21, 2025에 액세스, https://www.aliexpress.com/i/1005003181244737.html
12. ANTSDR E310 Features - Satsagen - Groups.io, 9월 21, 2025에 액세스, https://groups.io/g/satsagen/topic/antsdr_e310_features/96870466
13. IIO buffer setup - The Linux Kernel Archives, 9월 21, 2025에 액세스, https://www.kernel.org/doc/html/v4.19/driver-api/iio/buffers.html
14. Analog Devices Hardware Python Interfaces 0.0.19 documentation, 9월 21, 2025에 액세스, https://analogdevicesinc.github.io/pyadi-iio/
15. Pyadi Iio - conda install - Anaconda, 9월 21, 2025에 액세스, https://anaconda.org/conda-forge/pyadi-iio
16. Python for the Rest of Us - GNU Radio, 9월 21, 2025에 액세스, https://www.gnuradio.org/grcon/grcon20/grcon20_mthoren_Python_for_the_rest_of_us.pdf
17. Connectivity — Analog Devices Hardware Python Interfaces 0.0.3 documentation, 9월 21, 2025에 액세스, https://pyadi-iio.readthedocs.io/en/latest/guides/connectivity.html
18. Examples — Analog Devices Hardware Python Interfaces 0.0.3 documentation, 9월 21, 2025에 액세스, https://pyadi-iio.readthedocs.io/en/latest/guides/examples.html
19. Buffers — Analog Devices Hardware Python Interfaces 0.0.19 documentation, 9월 21, 2025에 액세스, https://analogdevicesinc.github.io/pyadi-iio/buffers/index.html
20. Frequency Domain | PySDR: A Guide to SDR and DSP using Python, 9월 21, 2025에 액세스, https://pysdr.org/content/frequency_domain.html
21. FOSDEM 2021 - PySDR: A Guide to SDR and DSP using Python, 9월 21, 2025에 액세스, https://archive.fosdem.org/2021/schedule/event/fsr_pysdr_guide_to_sdr_and_dsp_using_python/
22. Clear IIO Buffer · analogdevicesinc libiio · Discussion #1161 - GitHub, 9월 21,

2025에 액세스, https://github.com/analogdevicesinc/libiio/discussions/1161
23. End-to-End Example | PySDR: A Guide to SDR and DSP using Python, 9월 21, 2025에 액세스, https://pysdr.org/content/rds.html
24. Python FM - listening to radio with Python - PU5EPX, 9월 21, 2025에 액세스, https://epxx.co/artigos/pythonfm_en.html
25. EmphasisFilter - DSP Concepts Documentation, 9월 21, 2025에 액세스, https://documentation.dspconcepts.com/awe-designer/8.D.2.3/emphasisfilter
26. Pre-emphasis and De-emphasis - GeeksforGeeks, 9월 21, 2025에 액세스, https://www.geeksforgeeks.org/electronics-engineering/pre-emphasis-and-de-emphasis/
27. bastibe/SoundCard: A Pure-Python Real-Time Audio Library - GitHub, 9월 21, 2025에 액세스, https://github.com/bastibe/SoundCard
28. Two important libraries used for audio processing and streaming in Python - Medium, 9월 21, 2025에 액세스, https://medium.com/@venn5708/two-important-libraries-used-for-audio-processing-and-streaming-in-python-d3b718a75904
29. Play and Record Sound with Python — python-sounddevice, version 0.5.1, 9월 21, 2025에 액세스, https://python-sounddevice.readthedocs.io/
30. Digital Modulation | PySDR: A Guide to SDR and DSP using Python, 9월 21, 2025에 액세스, https://pysdr.org/content/digital_modulation.html
31. Pulse Shaping | PySDR: A Guide to SDR and DSP using Python, 9월 21, 2025에 액세스, https://pysdr.org/content/pulse_shaping.html
32. Synchronization | PySDR: A Guide to SDR and DSP using Python, 9월 21, 2025에 액세스, https://pysdr.org/content/sync.html