RFSoC4x2 보드를 활용한 **Strathclyde SDR Book** 마스터 가이드**: 40**단계 실습 튜토리얼

서론

목적 및 대상 독자

본 보고서는 "Software Defined Radio with Zynq UltraScale+ RFSoC" 서적의 이론적 개념을 RFSoC4x2 개발 보드를 통해 실질적으로 체득하고자 하는 엔지니어 및 연구자를 위한 포괄적인 40단계 실습 튜토리얼을 제공하는 것을 목표로 합니다.[1] 본 가이드의 대상 독자는 SDR 및 FPGA 기술에 대한 이론적 배경은 있으나 RFSoC 플랫폼에 대한 실무 경험이 부족한 "응용 학습자(Applied Learner)"로 정의됩니다. 이들은 구조화된 실습 과정을 통해 이론과 실제 구현을 연결하며 가장 효과적으로 학습하는 공학 대학원생, 연구원 또는 현업 전문가입니다. 본 튜토리얼은 하드웨어의 초기 설정부터 시작하여 디지털 신호 처리(DSP)의 핵심 원리를 거쳐, 최종적으로 복잡한 OFDM(Orthogonal Frequency Division Multiplexing) 시스템을 구현하는 전 과정을 안내합니다.

StrathSDR 학습 생태계

스트래스클라이드 대학(University of Strathclyde)의 SDR 연구팀(StrathSDR)은 단순히 서적 한 권을 제공하는 데 그치지 않고, 완결된 형태의 학습 생태계를 구축했습니다.[3] 이 생태계는 이론의 근간을 이루는 RFSoC 서적, 서적의 내용과 직접적으로 연계되는 주피터 노트북(Jupyter Notebook) 예제, 기초 DSP 개념을 다루는 별도의 노트북 모음, 그리고 QPSK 및 OFDM과 같은 완전한 형태의 애플리케이션 데모로 구성됩니다.[5] 이러한 자원들은 개별적으로 존재하는 것이 아니라, 이론(Why), 실습(How), 그리고 응용(What for)을 유기적으로 연결하는 치밀한 교육 전략의 산물입니다. 따라서 본 튜토리얼은 이러한 자원들을 하나의 일관된 학습 경로로 엮어내는 가이드 역할을 수행할 것입니다. 사용자는 이 가이드를 통해 각 자원의 역할을

이해하고, 체계적으로 기술을 습득하게 될 것입니다.

이러한 학습 생태계의 중심에는 PYNQ 프레임워크가 있습니다. PYNQ는 Python을 사용하여 FPGA의 복잡한 하드웨어 설계를 추상화하고, "오버레이(Overlay)"라는 개념을 통해 사전 컴파일된 하드웨어 블록을 제어하는 생산성 중심의 프레임워크입니다.[8] 이 접근법은 사용자가 VHDL이나 Verilog와 같은 하드웨어 기술 언어(HDL)에 대한 깊은 지식 없이도, Python을 통해 RFSoC의 강력한 프로그래머블 로직(PL) 성능을 즉시 활용할 수 있게 해줍니다.[9] 본 튜토리얼의 대부분은 이러한 PYNQ의 "소프트웨어 우선(Software-First)" 철학을 따라 진행되며, 이를 통해 사용자는 SDR 및 DSP 개념 자체에 집중할 수 있습니다.

아래 표는 본 튜토리얼 전반에 걸쳐 활용될 StrathSDR 학습 생태계의 핵심 자원들을 정리한 것입니다.

표 **1: RFSoC4x2**를 위한 **StrathSDR** 학습 생태계

리소스 명칭	저장소/URL	핵심 목적	관련 튜토리얼 단계
The RFSoC Book	rfsocbook.com	RFSoC 기반 SDR의 핵심 이론을 제공하는 주 교재.	모든 단계
RFSoC Book Notebooks	strath-sdr/RFSoC-Book	서적의 장(Chapter) 구성과 직접 연계된 주피터 노트북 예제.	II, III, IV
DSP Notebooks	strath-sdr/dsp_notebooks	샘플링, FFT, 필터, 변조 등 기초 DSP 개념 학습용 노트북.	III, IV, V
RFSoC Spectrum Analyser	strath-sdr/rfsoc_sam	고성능 스펙트럼 분석기 오버레이.	III
RFSoC QPSK Demonstrator	strath-sdr/rfsoc_qpsk	완전한 단일 반송파 QPSK 송수신기 오버레이.	IV
RFSoC OFDM Demonstrator	strath-sdr/rfsoc_ofdm	완전한 다중 반송파 OFDM 송수신기 오버레이.	V

RFSoC-PYNQ Docs	rfsoc-pynq.io	하드웨어 문서, 시작 가이드, PYNQ 이미지 제공.	I

1단계: 환경 및 하드웨어 초기화 (1-5단계)

이 단계는 사용자가 실습을 진행하기 위한 안정적인 하드웨어 플랫폼을 구축하는 것을 목표로 합니다. 정확한 초기 설정은 이후 발생할 수 있는 문제들을 예방하는 데 매우 중요합니다.

1단계: 개봉 및 물리적 설정

RFSoC4x2 키트에는 보드 본체, Micro SD 카드, Micro USB 3.0 케이블, 그리고 12V 10A 전원 공급 장치가 포함되어 있습니다.[10] 보드의 주요 포트를 확인하는 것이 중요합니다. 전원 커넥터, 데이터 통신 및 시리얼 터미널 접속에 사용되는 USB 3.0 포트, RF 신호 입출력을 위한 SMA 커넥터, 그리고 운영체제 이미지가 저장될 Micro SD 카드 슬롯의 위치를 숙지해야 합니다.[11] 보드 레이아웃을 시각적으로 파악하여 ADC와 DAC 포트의 라벨(예: ADCA, DACB)을 확인합니다.

2단계: PYNQ SD 카드 준비

최신 기능과 안정성을 확보하기 위해 보드에 최신 PYNQ 이미지를 설치하는 것이 권장됩니다. rfsoc-pynq.io 웹사이트에서 RFSoC4x2 전용 최신 PYNQ 이미지를 다운로드합니다.[10] 다운로드한

.img 파일을 16GB 이상의 Micro SD 카드에 기록해야 합니다. 이를 위해 Balena Etcher나 Win32DiskImager와 같은 이미지 라이팅 소프트웨어를 사용합니다.[12] 이 과정은 SD 카드의 모든 기존 데이터를 삭제하므로, 중요한 데이터가 없는지 확인 후 진행해야 합니다.

3단계: 첫 부팅 및 연결

이미지가 기록된 Micro SD 카드를 보드의 슬롯에 삽입합니다. 보드 상의 부팅 모드 스위치를 'JTAG'이 아닌 'SD' 위치로 설정합니다.[10] 이후 USB 3.0 케이블을 PC와 보드에 연결하고 전원 케이블을 연결한 뒤, 전원 스위치를 켭니다. 부팅 과정은 다음과 같은 순서로 진행되며, 각 단계를 통해 보드의 상태를 확인할 수 있습니다 [10]:

1. 전원 인가 시 보드 우측 하단의 10개 전원 상태 LED가 켜집니다.
2. 약 40초 후, 보드 좌측 상단의 녹색 DONE 및 INIT LED가 켜집니다. DONE LED는 프로그래머블 로직(PL)에 비트스트림이 성공적으로 다운로드되었음을 의미합니다.
3. 몇 초 후, 보드 중앙 하단의 4개 백색 사용자 LED가 잠시 깜박인 후 켜진 상태를 유지합니다.
4. 보드 상의 OLED 디스플레이에 네트워크 정보와 함께 IP 주소가 표시됩니다.

4단계**:** 주피터 랩 환경 접속

RFSoC4x2 보드는 USB 3.0 포트를 통해 'USB 이더넷 가젯'으로 동작하여 별도의 이더넷 연결 없이 PC와 직접 통신할 수 있습니다. 이 경우 보드의 기본 IP 주소는 192.168.3.1입니다.[10] PC에서 웹 브라우저(성능을 위해 Chrome 사용이 권장됨 [15])를 열고 주소창에

http://192.168.3.1/lab을 입력합니다. 주피터 랩 로그인 화면이 나타나면, 기본 비밀번호인 xilinx를 입력하여 로그인합니다.[10]

5단계**:** 주피터 랩 및 **PYNQ** 프레임워크 탐색

로그인하면 주피터 랩 인터페이스가 나타납니다. 좌측의 파일 탐색기, 새 노트북이나 터미널을 열 수 있는 런처, 그리고 리눅스 셸 명령을 실행할 수 있는 터미널 창으로 구성되어 있습니다. PYNQ의 핵심 개념인 '오버레이'를 처음으로 경험해볼 차례입니다. 오버레이는 PL에 로드된 특정 하드웨어 설계를 나타내는 Python 객체입니다. 새 주피터 노트북을 열고 다음 코드를 실행하여 기본 base 오버레이를 로드하고, 소프트웨어 명령으로 보드의 물리적인 LED를 제어해봅니다.

Python

```
from pynq.overlays.base import BaseOverlay

# 기본 오버레이 로드
base = BaseOverlay("base.bit")

# 사용자 LED 0번을 켭니다.
base.leds.on()

# 사용자 버튼 0번의 상태를 읽습니다.
# 버튼을 누른 상태에서 이 셀을 실행하면 1이 출력됩니다.
print(base.buttons.read())
```

이 간단한 실습을 통해 Python 코드(PS, Processing System)가 FPGA 하드웨어(PL, Programmable Logic)를 직접 제어하는 PYNQ의 강력함을 확인할 수 있습니다.

2단계: RF 세계와의 인터페이스 (6-10단계)

이 단계에서는 RFSoC의 핵심 기능인 RF 데이터 컨버터(RFDC)를 다룹니다. 아날로그 RF 신호를 생성하고 캡처하는 기본적인 능력을 갖추는 것을 목표로 합니다.

6단계: RF 데이터 컨버터 API 소개

RFSoC4x2 보드에 탑재된 Zynq UltraScale+ RFSoC ZU48DR 칩은 여러 개의 고속 RF-ADC(최대 5 GSPS)와 RF-DAC(최대 9.85 GSPS)를 내장하고 있습니다.[11] PYNQ 환경에서는

rfsoc_pynq라는 Python 라이브러리를 통해 이러한 RFDC 블록들을 고수준에서 제어할 수 있습니다. 새 노트북에서 이 라이브러리를 임포트하고 RFDC 객체를 초기화하여 시스템이 RF 데이터 컨버터를 인식하는지 확인합니다.

Python

```
import rfsoc_pynq as rfsoc

# RFDC 객체 초기화
```

```
rfdc = rfsoc.RFDC()
```

7단계: 신호 생성 - 첫 번째 톤

RF-DAC의 가장 기본적인 기능 중 하나는 내장된 수치 제어 발진기(NCO, Numerically Controlled Oscillator)를 사용하여 특정 주파수의 연속적인 사인파(톤)를 생성하는 것입니다. 다음 코드는 특정 DAC 채널(예: DAC_B)을 활성화하고, 500 MHz의 톤을 생성하도록 설정합니다.

Python

```
# DAC Tile 0, Block 0 (보드 레이블상 DAC_B에 해당할 수 있음) 선택
dac_tile = rfdc.dac_tiles
dac_block = dac_tile.blocks

# DAC 블록 활성화
dac_block.Reset()

# NCO 주파수를 500 MHz로 설정
dac_block.UpdateEvent(rfsoc.EVNT_SRC_IMMEDIATE)
dac_block.NCO.set(frequency=500, event=rfsoc.EVNT_SRC_TILE)
dac_tile.SetupFIFO(enable=True)
dac_tile.DynamicPLLConfig(1, 409.6, 6553.6) # 샘플링 주파수 설정 예시
dac_block.Mixer.set(event=rfsoc.EVNT_SRC_TILE)

# DAC 채널 활성화
rfdc.dac_tiles.blocks.Run()
```

이 코드를 실행하면 해당 DAC의 SMA 출력 포트에서 500 MHz의 RF 신호가 출력됩니다.

8단계: RF 루프백 - 생성한 신호 캡처하기

송신된 신호를 수신하여 전체 RF 신호 체인이 정상 동작하는지 확인하는 루프백 테스트를 수행합니다. 물리적으로 SMA 케이블을 사용하여 7단계에서 신호를 출력한 DAC 포트(예:

DAC_B)와 신호를 입력받을 ADC 포트(예: ADC_D)를 연결합니다.[14] 이 때, 신호 감쇠를 위해 중간에 적절한 감쇠기(attenuator)를 연결하는 것이 좋습니다. 이후, Python 코드를 통해 해당 ADC 채널이 데이터를 캡처하도록 설정합니다.

Python

```
# ADC Tile 0, Block 0 (보드 레이블상 ADC_D에 해당할 수 있음) 선택
adc_tile = rfdc.adc_tiles
adc_block = adc_tile.blocks

# ADC 블록 활성화
adc_block.Reset()
adc_tile.SetupFIFO(enable=True)
adc_tile.DynamicPLLConfig(1, 409.6, 5000.0) # 샘플링 주파수 설정 예시
adc_block.Mixer.set(event=rfsoc.EVNT_SRC_TILE)

# ADC 채널 활성화
adc_block.Run()
```

9단계**:** 데이터 수집 및 버퍼링

ADC에서 디지털화된 고속 데이터 샘플은 PL에서 PS의 메인 메모리로 효율적으로 전송되어야 합니다. 이 과정은 DMA(Direct Memory Access) 컨트롤러가 담당하며, CPU의 개입 없이 대량의 데이터를 이동시켜 시스템 부하를 줄입니다.[18]

rfsoc_pynq 라이브러리는 이 DMA 과정을 추상화하여 간단한 함수 호출로 데이터를 수집할 수 있게 합니다. 다음 코드는 ADC로부터 일정량의 샘플을 캡처하여 NumPy 배열에 저장합니다.

Python

```
# ADC로부터 8192개의 샘플을 캡처
samples = adc_block.get_data(8192)
```

10단계: 시각화 - 파형 확인하기

데이터 수집의 성공 여부를 확인하는 가장 확실한 방법은 시각화입니다. 주피터 노트북의 강력한 기능 중 하나는 matplotlib과 같은 라이브러리를 사용하여 데이터를 즉시 플로팅할 수 있다는 것입니다. 9단계에서 수집한 NumPy 배열 samples를 플로팅하여 시간 영역 파형을 확인합니다.

Python

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

# 캡처된 데이터 플로팅 (처음 200개 샘플만)
plt.figure(figsize=(12, 6))
plt.plot(np.arange(200), samples[:200])
plt.title("Captured ADC Waveform (Time Domain)")
plt.xlabel("Sample Index")
plt.ylabel("ADC Value")
plt.grid(True)
plt.show()
```

만약 모든 과정이 성공적이었다면, 플롯에는 깨끗한 사인파 형태가 나타날 것입니다. 이는 디지털 신호 생성, DAC를 통한 아날로그 변환, RF 루프백, ADC를 통한 디지털 변환, 그리고 DMA를 통한 메모리 저장까지의 전 과정이 완벽하게 동작했음을 의미하는 중요한 "첫 성과(first light)"입니다.

3단계: 디지털 신호 처리의 기초 (11-20단계)

이 단계에서는 RFSoC 서적의 4장과 5장에 해당하는 핵심 DSP 이론을 실습을 통해 학습합니다.[1] StrathSDR의

dsp_notebooks와 강력한 rfsoc_sam 스펙트럼 분석기 오버레이를 활용하여 이론적 개념을 실제

신호에 적용해 봅니다.

11단계: 주파수 영역 - 스펙트럼 분석기 설치 및 사용

시간 영역 파형만큼이나 중요한 것이 주파수 영역에서의 신호 분석입니다. StrathSDR에서 제공하는 rfsoc_sam은 RFSoC를 고성능 스펙트럼 분석기로 변환하는 오버레이입니다. 주피터 랩의 터미널을 열고 다음 명령어를 실행하여 설치합니다.[20]

Bash

```
pip3 install git+https://github.com/strath-sdr/rfsoc_sam
```

설치가 완료되면 주피터 파일 탐색기에 rfsoc-sam 폴더가 생성됩니다. 해당 폴더의 노트북을 실행하여 스펙트럼 분석기를 구동합니다. 이 노트북은 시간 영역 신호를 FFT(Fast Fourier Transform)를 통해 주파수 성분으로 변환하고 그 결과를 시각적으로 보여줍니다.

12단계: 스펙트럼 분석의 실제

7단계에서 생성했던 500 MHz 톤을 스펙트럼 분석기의 입력으로 사용합니다 (SMA 케이블 연결 필요). 스펙트럼 분석기 GUI 또는 플롯에서 주파수 축의 500 MHz 지점에 날카로운 피크(peak)가 나타나는 것을 확인합니다. 이는 10단계에서 보았던 시간 영역의 사인파가 주파수 영역에서는 단일 주파수 성분으로 표현됨을 시각적으로 연결해주는 중요한 경험입니다.

13단계: 샘플링 및 에일리어싱(Aliasing) 이해

dsp_notebooks 저장소를 설치하여 기초 DSP 개념 학습을 시작합니다. 터미널에서 다음 명령어를 실행합니다.[21]

Bash

```
pip3 install git+https://github.com/strath-sdr/dsp_notebooks
```

rfsoc-studio/dsp-notebooks 폴더에서 nb_sampling_aliasing.ipynb 노트북을 엽니다.[6] 이 노트북은 나이퀴스트-섀넌 샘플링 정리(

fs>2fmax)를 설명합니다. 이 이론을 실제로 검증하기 위해, DAC의 NCO를 사용하여 ADC의 나이퀴스트 주파수(샘플링 주파수의 절반)보다 높은 주파수의 톤을 생성합니다. 스펙트럼 분석기로 이 신호를 관찰하면, 원래 주파수가 아닌 낮은 주파수 대역에 "접혀서(folded back)" 나타나는 에일리어싱 현상을 직접 목격할 수 있습니다.

14단계**:** 양자화**(Quantization)** 이해

dsp_notebooks의 nb_sampling_quantisation.ipynb 노트북을 실행합니다.[6] 이 노트북은 연속적인 아날로그 값을 유한한 개수의 이산적인 디지털 레벨로 변환하는 양자화 과정을 설명합니다. 양자화 비트 수가 신호의 정밀도에 미치는 영향을 시뮬레이션을 통해 확인하고, 양자화 잡음(quantization noise)과 신호 대 잡음비(SNR, Signal-to-Noise Ratio)의 관계를 학습합니다.

15단계**:** 디지털 필터링 소개

nb_digital_filtering.ipynb 노트북을 통해 디지털 필터의 기본을 학습합니다.[6] FIR(Finite Impulse Response) 필터와 IIR(Infinite Impulse Response) 필터의 차이점을 이해합니다. Python의

scipy.signal 라이브러리를 사용하여 원하는 주파수 응답을 갖는 간단한 저역 통과(low-pass) FIR 필터를 설계하는 코드를 실습합니다.

Python

```
from scipy import signal
```

```
# 필터 사양
num_taps = 64  # 필터 탭(계수)의 개수
cutoff_freq = 600e6  # 차단 주파수 (600 MHz)
sampling_freq = 5e9  # ADC 샘플링 주파수 (5 GHz)

# FIR 필터 계수 생성
taps = signal.firwin(num_taps, cutoff_freq, fs=sampling_freq)

# 필터의 주파수 응답 플로팅
w, h = signal.freqz(taps, worN=8000, fs=sampling_freq)
plt.plot(w, 20 * np.log10(abs(h)))
plt.title('Low-pass FIR Filter Frequency Response')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Magnitude (dB)')
plt.grid()
```

16단계**:** 실제 신호 필터링

DAC를 사용하여 두 개의 톤(예: 500 MHz와 800 MHz)을 동시에 생성합니다. ADC로 이 합성 신호를 캡처한 후, 15단계에서 소프트웨어로 설계한 저역 통과 필터(차단 주파수 600 MHz)를 적용합니다. 필터링 전후 신호의 FFT 결과를 비교하여 플로팅하면, 500 MHz 톤은 통과하고 800 MHz 톤은 크게 감쇠되는 것을 확인할 수 있습니다. 이를 통해 소프트웨어 필터가 실제 RF 신호에 어떻게 작용하는지 이해하게 됩니다.

17단계**:** 기저대역 변조 원리

nb_modulation_demodulation.ipynb 노트북을 통해 디지털 통신의 기본인 변조 개념을 학습합니다.[6] 비트(0과 1)를 특정 파형의 속성을 나타내는 심볼(symbol)로 매핑하는 과정을 이해합니다. BPSK(Binary Phase Shift Keying)와 QPSK(Quadrature Phase Shift Keying)의 원리를 배우고, 각 변조 방식의 성상도(constellation diagram)를 직접 그려봅니다.

18단계**:** 복소 신호와 직교 변조

RFSoC 서적 7장의 핵심 내용인 I/Q(In-phase and Quadrature) 신호 표현법을 학습합니다.[1] I/Q는 하나의 복소수(

I+jQ)를 사용하여 진폭과 위상을 동시에 표현하는 효율적인 방법입니다. Python을 사용하여 0과 1의 비트 스트림을 QPSK 심볼에 해당하는 복소수 스트림으로 변환하는 코드를 작성합니다. 예를 들어, '00'은 1+j, '01'은 -1+j, '11'은 -1-j, '10'은 1-j로 매핑될 수 있습니다.

19단계**:** 대역폭 제어를 위한 펄스 성형

nb_pulse_shaping.ipynb 노트북을 실행합니다.[21] 디지털 심볼을 사각 펄스로 전송할 경우 주파수 대역을 비효율적으로 많이 차지하게 되고, 이는 심볼 간 간섭(ISI, Inter-Symbol Interference)을 유발할 수 있음을 배웁니다. 이를 해결하기 위해 RRC(Root-Raised Cosine) 필터와 같은 펄스 성형 필터를 사용합니다. 18단계에서 생성한 QPSK 복소수 심볼 스트림에 RRC 필터를 적용하고, 그 결과 파형과 스펙트럼을 플로팅하여 대역폭이 효과적으로 제한되는 것을 확인합니다.

20단계**:** 아이 다이어그램**(Eye Diagram)**

아이 다이어그램은 펄스 성형된 디지털 기저대역 신호의 품질을 시각적으로 평가하는 강력한 도구입니다. 신호 파형을 심볼 주기에 맞춰 반복적으로 겹쳐 그리면 눈(eye) 모양의 패턴이 나타납니다. 눈이 크게 열려 있을수록 ISI가 적고 신호 품질이 좋음을 의미합니다. 19단계에서 생성한 펄스 성형된 QPSK 신호의 I 채널과 Q 채널에 대한 아이 다이어그램을 생성하여 ISI의 영향을 시각적으로 분석합니다.

4단계**:** 완전한 단일 반송파 송수신기 구축 **(21-35**단계**)**

이 단계는 본 튜토리얼의 핵심으로, 사용자가 직접 QPSK 라디오의 송신기와 수신기를 단계별로 구축하는 과정입니다. 특히 수신기 구현에서는 프레임 동기, 심볼 타이밍 복구, 반송파 복구 등 실제 통신 시스템의 난제들을 소프트웨어로 직접 해결해봄으로써 StrathSDR의 rfsoc_qpsk 데모가 내부적으로 어떤 복잡한 과정을 수행하는지 깊이 있게 이해하게 됩니다.[22] 이는 RFSoC 서적의 6, 7, 13장 내용과 밀접하게 연관됩니다.[1]

섹션 **A: QPSK** 송신기

21단계**:** 랜덤 비트 스트림 생성

송신할 데이터를 만듭니다. numpy를 사용하여 무작위 0과 1로 구성된 긴 비트 배열을 생성합니다.

22단계**:** 비트를 **QPSK** 심볼로 매핑

생성된 비트 스트림을 두 비트씩 묶어 QPSK 심볼(복소수)로 변환합니다. (예: '00' -> 1+j, '01' -> −1+j,...)

23단계**: RRC** 펄스 성형 적용

22단계에서 생성된 심볼 스트림에 RRC 필터를 적용하여 대역폭을 제한하고 ISI를 최소화합니다.

24단계**: RF** 주파수로 상향 변환

펄스 성형된 복소 기저대역 신호를 RF-DAC의 디지털 믹서(NCO와 곱셈기)를 사용하여 원하는 RF 반송파 주파수(예: 1 GHz)로 상향 변환합니다. 이 과정은 복소수 신호에 $e^{j2\pi f_c t}$를 곱하는 것과 같습니다.

25단계**:** 신호 송신

최종적으로 생성된 디지털 RF 신호 샘플을 RF-DAC 버퍼에 쓰고 송신을 시작합니다. 이 신호는 루프백 케이블을 통해 수신기로 전달됩니다.

섹션 **B:** 수신기의 도전 과제 **-** 비동기 수신

26단계**:** 비동기 신호 캡처

RF-ADC를 사용하여 송신된 신호를 캡처합니다. 이 시점에서 수신기는 패킷이 언제 시작하는지, 정확한 심볼 타이밍이 언제인지, 송신기와 수신기의 오실레이터 주파수/위상 차이가 얼마인지 전혀 알지 못합니다.

27단계**:** 원시 수신 신호의 성상도

캡처된 원시 신호 샘플을 I-Q 평면에 그대로 플로팅합니다. 결과는 네 개의 뚜렷한 점이 아니라, 노이즈가 끼고 빙글빙글 회전하는 점들의 구름처럼 보일 것입니다. 이 시각적 결과는 앞으로 해결해야 할 문제들(프레임 동기 부재, 타이밍 오차, 반송파 오프셋)을 명확하게 보여줍니다.

섹션 **C:** 동기화 달성 **(**핵심 학습 과정**)**

28단계**:** 프리앰블 상관을 이용한 프레임 동기

송신 데이터 패킷의 맨 앞에 수신기가 미리 알고 있는 특정 시퀀스인 프리앰블(preamble)을 추가합니다. 수신기에서는 수신되는 데이터 스트림과 이 프리앰블 시퀀스 간의 상호상관(cross-correlation)을 지속적으로 계산합니다.[23]

numpy.correlate 함수를 사용하여 이를 구현할 수 있습니다. 프리앰블이 수신되는 순간 상관 값이 급격히 치솟는 피크가 발생하며, 이 피크의 위치를 통해 데이터 페이로드의 시작점을

정확히 찾아낼 수 있습니다.

29단계**:** 대략적인 주파수 오차 보정

프리앰블의 반복적인 구조를 이용하여 송신기와 수신기 간의 큰 주파수 오차(Coarse Frequency Offset)를 추정하고 보정할 수 있습니다. 예를 들어, 프리앰블의 앞부분과 뒷부분의 위상 차이를 계산하여 주파수 오차를 알아내고, 수신 신호 전체에 보정 계수를 곱해줍니다.

30단계**:** 심볼 타이밍 복구 **-** 가드너 알고리즘

프레임 시작점을 찾았더라도, 각 심볼을 가장 이상적인 순간(ISI가 최소인 지점)에 샘플링하는 것은 또 다른 문제입니다. 가드너 알고리즘(Gardner Algorithm)은 데이터 심볼 값에 의존하지 않고(non-data-aided) 제로 크로싱(zero-crossing) 정보를 이용하여 최적의 샘플링 타이밍을 찾아내는 효율적인 피드백 루프 알고리즘입니다.[25] 심볼 당 2개의 샘플을 사용하여 타이밍 에러를 계산하고, 이 에러를 기반으로 샘플링 시점을 점진적으로 조정하는 가드너 루프를 Python으로 구현합니다.

31단계**:** 타이밍 보정 시각화

가드너 알고리즘을 통과한 신호의 아이 다이어그램과 성상도를 다시 그려봅니다. 이전과 달리 아이 다이어그램의 눈이 활짝 열려 있고, 성상도의 점들이 더 이상 퍼져있지 않고 특정 위치에 모여있는 것을 볼 수 있습니다. 다만, 아직 반송파 위상 오차로 인해 성상도 전체가 회전하고 있을 것입니다.

32단계**:** 반송파 위상 및 주파수 복구 **-** 코스타스 루프

남아있는 미세한 주파수 오차(Fine Frequency Offset)와 위상 오차를 보정하기 위해 코스타스 루프(Costas Loop)를 사용합니다.[24] 코스타스 루프는 위상 잠금 루프(PLL, Phase-Locked Loop)의 한 종류로, 수신된 심볼과 가장 가까운 이상적인 성상도 점 간의 위상 차이를 에러로 간주하여 NCO를 제어함으로써 위상을 잠급니다. QPSK에 맞는 2차 코스타스 루프를

Python으로 구현하여 타이밍이 보정된 심볼 스트림에 적용합니다.

33단계**:** 완전한 동기화 시각화

코스타스 루프까지 통과한 최종 신호의 성상도를 플로팅합니다. 마침내, 27단계에서 보았던 회전하는 노이즈 구름이, QPSK 성상도의 이상적인 위치(예: 45, 135, 225, 315도)에 안정적으로 고정된 네 개의 뚜렷한 점 군집으로 변한 것을 확인할 수 있습니다. 이것이 바로 디지털 수신기 설계의 "아하!" 순간입니다.

섹션 **D:** 루프 닫기

34단계**:** 복조 및 비트 복구

동기화가 완료된 심볼들을 가장 가까운 이상적인 성상도 점으로 결정(slicing 또는 de-mapping)하여 원래의 비트 스트림으로 복원합니다.

35단계**:** 성능 측정 **-** 비트 오류율 **(BER)**

복원된 비트 스트림을 21단계에서 생성한 원본 비트 스트림과 비교하여 오류 비트의 비율, 즉 BER(Bit Error Rate)을 계산합니다. 의도적으로 노이즈를 추가하거나 감쇠를 조절하며 BER이 어떻게 변하는지 실험해봄으로써 통신 시스템의 성능을 정량적으로 평가합니다.

5단계**:** 고급 주제 **-** 직교 주파수 분할 다중화 **(OFDM) (36-40**단계**)**

이 마지막 단계에서는 단일 반송파 방식보다 진보된 다중 반송파 변조 방식인 OFDM을 다룹니다. StrathSDR의 rfsoc_ofdm 데모를 활용하여 이전에 배운 DSP 개념들이 어떻게 복잡한

현대 통신 시스템에 적용되는지 확인하는 최종 프로젝트를 수행합니다.

36단계: OFDM 데모 설치 및 탐색

주피터 랩 터미널에서 rfsoc_ofdm 오버레이를 설치합니다. 이 데모는 설치 과정이 조금 더 복잡할 수 있으므로, GitHub 저장소의 README 파일을 주의 깊게 따릅니다.[15]

Bash

```
pip3 install https://github.com/strath-sdr/rfsoc_ofdm/releases/download/v0.3.5/rfsoc_ofdm.tar.gz
python -m rfsoc_ofdm install
```

설치 후, rfsoc_ofdm 폴더의 노트북을 실행하여 제공되는 GUI를 통해 OFDM 시스템의 파라미터(변조 방식, 부반송파 개수 등)를 변경하며 송수신을 테스트해 봅니다.

37단계: OFDM 해부 - IFFT, 부반송파, 순환 전치

OFDM의 핵심 원리를 학습합니다. OFDM은 고속의 데이터 스트림을 수백 또는 수천 개의 저속 병렬 데이터 스트림으로 나눈 뒤, 각각을 서로 직교하는(orthogonal) 여러 개의 부반송파(sub-carrier)에 실어 동시에 전송하는 방식입니다. 이 변조 과정은 IFFT(Inverse Fast Fourier Transform)를 통해 매우 효율적으로 구현됩니다. 또한, 다중 경로 페이딩(multipath fading)으로 인한 심볼 간 간섭을 막기 위해 각 OFDM 심볼의 앞부분에 심볼 뒷부분의 일부를 복사하여 붙이는 순환 전치(CP, Cyclic Prefix)의 역할에 대해 이해합니다.

38단계: Python으로 간소화된 OFDM 송신기 구축

rfsoc_ofdm 데모의 내부 동작을 이해하기 위해 numpy를 사용하여 간소화된 OFDM 송신기를 직접 코딩해 봅니다.

1. 부반송파에 할당할 복소수 데이터 배열을 생성합니다.

2. numpy.fft.ifft 함수를 사용하여 이 주파수 영역 데이터를 시간 영역 OFDM 심볼로 변환합니다.
3. 생성된 심볼의 뒷부분을 복사하여 앞에 붙여 순환 전치를 추가합니다.
4. 결과로 나온 시간 영역 파형을 플로팅하여 확인합니다.

39단계**: Python**으로 간소화된 **OFDM** 수신기 구축

송신기에 대응하는 수신기를 코딩합니다.

1. 수신된 OFDM 심볼에서 순환 전치 부분을 제거합니다.
2. numpy.fft.fft 함수를 사용하여 시간 영역 심볼을 주파수 영역 데이터로 변환합니다.
3. 결과로 나온 각 부반송파의 복소수 값을 성상도에 플로팅하여 데이터가 성공적으로 복원되었는지 확인합니다.

40단계**:** 최종 프로젝트 **- OFDM**을 이용한 이미지 전송

40단계의 대장정을 마무리하는 최종 프로젝트로, 간단한 흑백 로고 이미지를 OFDM을 통해 전송합니다.

1. 이미지 파일을 읽어 픽셀 데이터를 비트 스트림으로 변환합니다.
2. 이 비트 스트림을 38단계에서 만든 OFDM 송신기를 사용하여 변조하고, RFSoC4x2의 DAC를 통해 송신합니다.
3. 수신기에서는 ADC로 신호를 캡처하고, 39단계의 수신기 코드로 복조하여 비트 스트림을 복원합니다.
4. 복원된 비트 스트림을 다시 이미지 파일로 변환하여 화면에 표시합니다.

송신한 원본 이미지와 수신하여 복원된 이미지를 나란히 비교하며, 보드를 켜는 것부터 시작하여 복잡한 통신 시스템을 통해 실제 데이터를 성공적으로 전송하기까지의 전 과정을 성공적으로 완수했음을 확인합니다.

결론 및 다음 단계

성과 요약

본 튜토리얼을 통해 사용자는 RFSoC4x2 보드의 전원을 켜는 것부터 시작하여, PYNQ 환경 설정, RF 데이터 컨버터 제어, 핵심 DSP 이론의 실습, 완전한 QPSK 송수신기 소프트웨어 구현, 그리고 최종적으로 OFDM을 이용한 이미지 전송에 이르는 방대한 여정을 완수했습니다. 이 과정에서 "Software Defined Radio with Zynq UltraScale+ RFSoC" 서적의 이론적 지식이 실제 하드웨어에서의 실습을 통해 어떻게 구체화되는지 깊이 있게 체득했습니다. 특히, 동기화 과정을 직접 구현해봄으로써 상용 통신 시스템의 복잡성과 정교함에 대한 통찰을 얻었습니다.

서적과의 연계

이제 사용자는 RFSoC 서적에서 다루는 MIMO, 빔포밍, 5G 셀룰러 네트워크 응용과 같은 고급 주제들을 탐구할 수 있는 강력한 실무적 기반을 갖추게 되었습니다.[1] 서적의 17장과 18장에서 설명하는 다중 안테나 시스템의 원리를 이해하고, 이를 RFSoC4x2의 다중 ADC/DAC 채널을 활용하여 실험해 볼 수 있는 능력을 보유하게 된 것입니다.

향후 탐구 방향

본 튜토리얼은 PYNQ의 "소프트웨어 우선" 철학에 따라 진행되었지만, RFSoC의 진정한 잠재력은 하드웨어와 소프트웨어의 긴밀한 통합에 있습니다. 다음 단계로, 사용자는 다음과 같은 주제들을 탐구하며 전문성을 더욱 심화시킬 수 있습니다.

- **StrathSDR**의 다른 오버레이 탐색**:** 자동 이득 제어(AGC) 데모와 같은 다른 오버레이를 설치하고 분석하며 추가적인 SDR 구성 요소를 학습할 수 있습니다.[7]
- 사용자 정의 하드웨어 오버레이 제작**:** Vivado, Vitis HLS, System Generator와 같은 AMD-Xilinx 툴을 사용하여 자신만의 하드웨어 가속기(IP)를 설계하고 이를 PYNQ 오버레이로 패키징하는 방법을 학습할 수 있습니다.[9] 예를 들어, 본 튜토리얼에서 Python으로 구현했던 상관기나 필터를 하드웨어로 구현하여 성능을 극적으로 향상시키는 프로젝트를 진행할 수 있습니다.

이로써 사용자는 RFSoC라는 강력한 단일 칩 SDR 플랫폼을 자유자재로 활용하여 차세대 통신 시스템을 연구하고 개발할 수 있는 전문가로 나아갈 준비를 마쳤습니다.

참고 자료

1. The RFSoC Book, 9월 21, 2025에 액세스, https://www.rfsocbook.com/

2. Software Defined Radio with Zynq Ultrascale+ RFSoC - University of Strathclyde, 9월 21, 2025에 액세스, https://pureportal.strath.ac.uk/en/publications/software-defined-radio-with-zynq-ultrascale-rfsoc
3. strath-sdr repositories - GitHub, 9월 21, 2025에 액세스, https://github.com/orgs/strath-sdr/repositories
4. StrathSDR – University of Strathclyde 5G Software Defined Radio Research Lab, 9월 21, 2025에 액세스, https://sdr.eee.strath.ac.uk/
5. University of Strathclyde - Software Defined Radio Research Laboratory - GitHub, 9월 21, 2025에 액세스, https://github.com/strath-sdr
6. Educational Resources - RFSoC-PYNQ, 9월 21, 2025에 액세스, http://www.rfsoc-pynq.io/educational_resources.html
7. RFSoC-PYNQ overlays, 9월 21, 2025에 액세스, http://www.rfsoc-pynq.io/overlays.html
8. System-on-Chip Overlay Methodologies for Software Defined Radio - UKRI Gateway to Research, 9월 21, 2025에 액세스, https://gtr.ukri.org/projects?ref=studentship-2110783
9. RFSoC 4x2 transmit and receive help - Support - PYNQ, 9월 21, 2025에 액세스, https://discuss.pynq.io/t/rfsoc-4x2-transmit-and-receive-help/5633
10. Getting started with your RFSoC 4x2 | RFSoC-PYNQ, 9월 21, 2025에 액세스, http://www.rfsoc-pynq.io/rfsoc_4x2_getting_started.html
11. RFSoC 4x2 Overview, 9월 21, 2025에 액세스, http://www.rfsoc-pynq.io/rfsoc_4x2_overview.html
12. RFSoC 4x2 Reference Manual - Real Digital, 9월 21, 2025에 액세스, https://www.realdigital.org/downloads/15456c97a4fe5a1ee66208c5aa3894bb.pdf
13. Getting started with RFSoC-PYNQ, 9월 21, 2025에 액세스, http://www.rfsoc-pynq.io/getting_started.html
14. Quick-start guide — qick 0.2.367 documentation, 9월 21, 2025에 액세스, https://qick-docs.readthedocs.io/latest/quick_start.html
15. strath-sdr/rfsoc_ofdm: PYNQ example of an OFDM ... - GitHub, 9월 21, 2025에 액세스, https://github.com/strath-sdr/rfsoc_ofdm
16. RFSoC 4x2 Board and Kit - Announcements - PYNQ, 9월 21, 2025에 액세스, https://discuss.pynq.io/t/rfsoc-4x2-board-and-kit/4392
17. RFSoC 4x2 Reference Manual - #2 by joshgoldsmith - Support - PYNQ, 9월 21, 2025에 액세스, https://discuss.pynq.io/t/rfsoc-4x2-reference-manual/6882/2
18. Building a daq from RFSoC 4x2? - Support - PYNQ, 9월 21, 2025에 액세스, https://discuss.pynq.io/t/building-a-daq-from-rfsoc-4x2/5584
19. Building a daq from RFSoC 4x2? - #2 by marioruiz - Support - PYNQ, 9월 21, 2025에 액세스, https://discuss.pynq.io/t/building-a-daq-from-rfsoc-4x2/5584/2
20. strath-sdr/rfsoc_sam: RFSoC Spectrum Analyser Module on PYNQ. - GitHub, 9월 21, 2025에 액세스, https://github.com/strath-sdr/rfsoc_sam
21. strath-sdr/dsp_notebooks: A collection of Digital Signal ... - GitHub, 9월 21, 2025에 액세스, https://github.com/strath-sdr/dsp_notebooks
22. strath-sdr/rfsoc_qpsk: PYNQ example of using the RFSoC ... - GitHub, 9월 21, 2025에 액세스, https://github.com/strath-sdr/rfsoc_qpsk

23. How to perform timing synchronization for a signal with a preamble, 9월 21, 2025에 액세스, https://dsp.stackexchange.com/questions/98161/how-to-perform-timing-synchronization-for-a-signal-with-a-preamble
24. Synchronization | PySDR: A Guide to SDR and DSP using Python, 9월 21, 2025에 액세스, https://pysdr.org/content/sync.html
25. Symbol Timing Recovery - DSPIllustrations.com, 9월 21, 2025에 액세스, https://dspillustrations.com/pages/pages/courses/course_singlecarrier/html/05%20-%20Symbol%20Timing%20Recovery.html
26. 4.6 Gardner Algorithm, 9월 21, 2025에 액세스, https://picture.iczhiku.com/resource/eetop/sYKGhWgEKUghunnC.pdf
27. Gardner Timing Error Detector: A Non-Data-Aided Version of Zero-Crossing Timing Error Detectors | Wireless Pi, 9월 21, 2025에 액세스, https://wirelesspi.com/gardner-timing-error-detector-a-non-data-aided-version-of-zero-crossing-timing-error-detectors/
28. Simple phase-locked loop (Costas loop) - Github-Gist, 9월 21, 2025에 액세스, https://gist.github.com/jgaeddert/bdfd390f2637afd5cc7f3e10297350b1
29. Costas Loop - GNU Radio, 9월 21, 2025에 액세스, https://wiki.gnuradio.org/index.php/Costas_Loop