

SDR ile RF Sinyal Analizi ve Adaptif, Konvansiyonel ve Hibrit Beamforming Yöntemleriyle Yön Bulma Çalışmaları

Saadet Büşra ÇAM
Karel İleri Teknolojiler A.Ş

28 Temmuz - 22 Ağustos 2025

Kurum / Proje:	SDR Tabanlı RF Sinyal Analizi ve Uygulamaları
Kullanılan Donanım:	ANTSDR E310 (AD9361), RTL-SDR Dongle
Kullanılan Yazılımlar:	Python (PyADI-IIIO, NumPy, Matplotlib, Tkinter), GNU Radio, Maia SDR Ubuntu 24.04 VM, Windows 11
Stajyer:	Saadet Büşra ÇAM
Süre:	28.07–22.08.2025

İçindekiler

1	Giriş	3
2	Teorik Arka Plan	3
2.1	SDR ve Örnekleme	3
2.2	Downconversion (Aşağı Dönüşüm) ve LO	3
2.3	DC Spike (LO Sızıntısı) ve Offset Tuning	3
2.4	Anten Temelleri (Özet)	4
3	Donanım ve Yazılım Kurulumları	4
3.1	Donanım	4
3.2	Yazılım	4
4	RF Sinyal Ölçümü ve Spektrum Analizi	4
4.1	Geniş Bant Tarama ve FFT	4
4.2	Peak Birleştirme	4
4.3	Python ve GNU Radio Karşılaştırması	5
5	Beamforming ve DOA Çalışmaları	5
5.1	Model ve Notasyon	5
5.2	Konvansiyonel (Bartlett) Beamforming	5
5.3	Adaptif (MVDR/Capon) Beamforming	5
5.4	Faz Farkından AoA	5
5.5	Hibrit Yöntem (Faz + Genlik)	6
6	Kalibrasyon ve Stabilizasyon	6
7	Karşılaştırmalar ve Gözlemler	6
7.1	Python vs. GNU Radio	6
7.2	MVDR ve Bartlett Benzerliği	6
7.3	Yanlış Yön Tespiti ve Yansımalar	6
8	Sonuç ve Öneriler	7

1 Giriş

Bu stajın amacı Yazılım Tanımlı Radyo (SDR) altyapısı ile RF sinyallerinin *alınması, işlenmesi, görselleştirilmesi ve iletilmesi* konularında pratik deneyim kazanmaktır. Çalışma kapsamında:

- Python ve GNU Radio ile veri alma/işleme akışları kuruldu,
- Geniş bant tarama (70 MHz–6 GHz) yapıldı ve spektrum analizleri gerçekleştirildi,
- Teorik olarak Bartlett (konvansiyonel), MVDR (adaptif) ve *hibrit* (faz+genlik) DOA/beamforming yöntemleri incelendi,
- Kalibrasyon (faz/genlik ofset düzeltmesi) ve CFO (taşıyıcı frekans kayması) düzeltmesi uygulandı,
- GUI prototipiyle kullanıcı etkileşimi sağlandı.

2 Teorik Arka Plan

2.1 SDR ve Örnekleme

“*SDR 2 MHz örnek hızında çalışıyor*” ifadesi, **saniyede iki milyon adet IQ örneği** alındığı anlamına gelir; yani 2 MS/s *kompleks* (I ve Q) numune akışı. Kompleks örnekleme, bant-*baseband* işaretlemeye olanak tanır ve dar bantlı sinyallerin düşük hızlarda sayısallaştırılmasını mümkün kılar.

2.2 Downconversion (Aşağı Dönüşüm) ve LO

Yüksek taşıyıcı frekanslı (ör. 2.4 GHz) bir RF dalgasını doğrudan bu hızlarda örnekleme *ADC* için maliyetli/zordur. SDR içindeki **mikser** ve **yerel osilatör (LO)** sayesinde sinyal ara frekansa (IF) veya doğrudan sıfır frekansa (zero-IF) *downconvert* edilir. LO frekansı hedef taşıyıcıya (ör. 435 MHz) ayarlanarak sıfır IF elde edilir; LO aynı zamanda I/Q ayrışması için *kuadratür* sinyaller üretir ve IQ örneklemesini mümkün kılar.

2.3 DC Spike (LO Sızıntısı) ve Offset Tuning

Sıfır-IF mimarisinde merkez frekansta görülen iğne benzeri çıkıntı **DC spike / LO leakage** olarak adlandırılır; ortamda gerçek bir sinyal varlığına işaret etmek zorunda değildir. DC ofsetin etkisini azaltmak için pratikte *oversample + off tune* yaklaşımı kullanılır: alım merkezi, hedefin biraz yanına kaydırılır (*offset tuning*), ardından sayısal *frequency shift* ve *decimation* ile istenen bant merkeze taşınır.

2.4 Anten Temelleri (Özet)

- **Radyasyon diyagramı (pattern):** Antenin uzaya enerjiyi nasıl yaydığının açısal gösterimidir; ana lob yönü, yan loblar ve 3 dB *yarı güç hüzme genişliği* (HPBW) gibi metrikler kullanılır.
- **E/H düzlemleri:** E-düzlemi (elektrik alan yönü + yayılım doğrultusu), H-düzlemi (manyetik alan yönü + yayılım doğrultusu) kesitlerini ifade eder.
- **Uzak alan (Fraunhofer) koşulu:** $r \gg \frac{L^2}{\lambda}$; uzak alanda alanlar $\propto \frac{1}{r}$ azalır ve yalnızca *açısal* fonksiyon ölçülür.
- **Giriş empedansı:** Tipik olarak 50 Ω ; eşleşme verimliliği ve güç transferini belirler.

3 Donanım ve Yazılım Kurulumları

3.1 Donanım

ANTSDR E310 (AD9361): 2 RX / 2 TX kanal, 70 MHz–6 GHz çalışma aralığı. **RTL-SDR:** Tek kanal, düşük maliyetli alıcı; temel karşılaştırmalar için kullanıldı.

3.2 Yazılım

- **Python** (pyadi-iio, NumPy, SciPy, Matplotlib, Tkinter): veri alma, FFT, tepe (peak) tespiti ve GUI.
- **GNU Radio:** blok tabanlı akış şemasıyla hızlı prototipleme ve spektrum görselleştirme.
- **Maia SDR:** Ubuntu 24.04 VM üzerinde bağımlılıklar kuruldu; derleme sürecinde hata nedeniyle çalıştırılamadı (Windows tarafında da başarısız oldu).

4 RF Sinyal Ölçümü ve Spektrum Analizi

4.1 Geniş Bant Tarama ve FFT

ANTSDR ile 70 MHz–6 GHz aralığında ölçümler alınmıştır. Zaman alanındaki IQ örnekleri, pencereleme ve FFT ile frekans alanına dönüştürülmüş; *peak detection* ile yakın frekanslı çoklu tepeler birleştirilerek gereksiz kalabalık azaltılmıştır.

4.2 Peak Birleştirme

Yakın tepe noktalarını tek bir olaya indirgemek için minimum frekans aralığı (*distance*) ve minimum genlik eşiği (*height*) parametreleri ayarlanmıştır. Şüpheli tekil tepe yerine kümelenmiş tepe özeti sunulmuştur.

4.3 Python ve GNU Radio Karşılaştırması

Python özelleştirilebilir algoritmalar ve GUI açısından esnek bulunmuştur; GNU Radio ise hızlı testler ve görsel akış kurulumunda pratiktir. İkisi birlikte iteratif geliştirme için tamamlayıcı rol oynamıştır.

5 Beamforming ve DOA Çalışmaları

5.1 Model ve Notasyon

- **Dizi yanıt vektörü:** $\mathbf{a}(\theta)$, geliş açısı θ için anten dizisinin faz/genlik tepkisini temsil eder.
- **Kovaryans matrisi:** $\mathbf{R} = E\{\mathbf{x} \mathbf{x}^H\}$; burada \mathbf{x} alınan çok kanallı örnek vektörüdür.

5.2 Konvansiyonel (Bartlett) Beamforming

Geciktir-topla yöntemi için hüzme gücü

$$P_{\text{Bartlett}}(\theta) = \mathbf{w}^H(\theta) \mathbf{R} \mathbf{w}(\theta), \quad \text{genelde } \mathbf{w}(\theta) = \frac{\mathbf{a}(\theta)}{M}, \quad (1)$$

şeklinde yazılır (M anten sayısı). Maksimizasyonla ana lob yönü bulunur.

5.3 Adaptif (MVDR/Capon) Beamforming

$$P_{\text{MVDR}}(\theta) = \frac{1}{\mathbf{a}^H(\theta) \mathbf{R}^{-1} \mathbf{a}(\theta)}. \quad (2)$$

MVDR, parazite minimum geçirgenlik sağlayıp, istenen yönde bozunumsuz (distortionless) kazanç hedefler. Sayısal kararlılık için *diagonal loading* ve *ileri-geri (forward-backward) ortalama* kullanılmıştır.

5.4 Faz Farkından AoA

İki antenli durumda faz farkı $\Delta\phi$ ile geliş açısı

$$\Delta\phi = \frac{2\pi d}{\lambda} \sin \theta \quad \Rightarrow \quad \hat{\theta} = \arcsin\left(\frac{\lambda \Delta\phi}{2\pi d}\right), \quad (3)$$

şeklinde kestirilebilir (d anten aralığı, λ dalga boyu). Faz belirsizliği için $|\Delta\phi| \leq \pi$ koşulu ve $d \leq \lambda/2$ tercihi doyum/çakışmayı azaltır.

5.5 Hibrit Yöntem (Faz + Genlik)

Hibrit yaklaşımda faz farkı ve genlik (anten element *pattern* bilgisi) birlikte kullanılmıştır. Spirallerin radyasyon paternlerinden elde edilen CSV dosyasıyla genlik puanı hesaplanmış ve birleşik skor

$$S(\theta) = \alpha S_{\text{phase}}(\theta) + (1 - \alpha) S_{\text{amp}}(\theta), \quad \alpha \approx 0.75 \quad (4)$$

olarak birleştirilmiştir. Böylece yansıma ve faz sarmalanmasına karşı daha dayanıklı bir AoA tahmini elde edilmiştir.

6 Kalibrasyon ve Stabilizasyon

- **Faz/Genlik Ofset Düzeltmesi:** Kanal kazanımları ve RF zincir ofsetleri için *calibrate/measure* modlarında referans yakalama yapıp farklar telafi edilmiştir.
- **CFO Düzeltmesi:** Taşıyıcı frekans kayması, zaman içinde faz artımı (*phase increment*) olarak modellenmiş; yakalanan pilot/ton ile *lock-in* fazör tekniği kullanılarak giderilmiştir.
- **Pencereleme ve Hüzme Genişliği:** *Window tapering* ile yan loblar bastırılmış, dizi hüzme genişliği istenen seviyeye getirilmiştir.
- **Geometri:** Spiral antenler arası mesafe ve çubuk antenin düşey konumu ölçülerek *AoA doğrulama* testleri yapılmıştır.

7 Karşılaştırmalar ve Gözlemler

7.1 Python vs. GNU Radio

Python: algoritma ve GUI açısından esnek; *peak filtering*, özel FFT parametreleri ve dosya kaydı kolaydır. GNU Radio: hızlı görsel prototipleme ve gerçek zamanlı izleme için pratiktir.

7.2 MVDR ve Bartlett Benzerliği

Yüksek SNR ve tek güçlü kaynak durumlarında \mathbf{R} yaklaşık olarak tek baskın özvektöre hizalandığından Bartlett ve MVDR *benzer hüzme şekilleri* üretebilir. Çoklu kaynak, düşük SNR veya korele parazit varlığında MVDR'nin üstünlüğü daha belirginleşir.

7.3 Yanlış Yön Tespiti ve Yansımalar

- **Çok yollu yayılım** (yansımalar) sahte tepeler doğurabilir; hibrit skor bu duruma karşı daha karardır.

- **Kalibrasyon hataları** (faz/gain ofsetleri) küçük açısal hataları büyütebilir; düzenli yeniden kalibrasyon önerilir.
- **CFO ve saat kaymaları** zamanla fazın sürüklenmesine yol açar; periyodik CFO takibi gerekir.
- **Anten aralığı** $d > \lambda/2$ olduğunda faz belirsizlikleri artar; mümkünse $d \leq \lambda/2$ seçilmelidir.

8 Sonuç ve Öneriler

Bu çalışma ile SDR tabanlı RF alma, spektrum analizi ve yön bulma için temel/orta seviye bir altyapı kurulmuştur. Öneriler:

- **Gerçek zamanlılaştırma:** FPGA hızlandırma ve *streaming* işleyiş için *zero-copy* tamponlar.
- **Gelişmiş DOA:** MUSIC/ESPRIT gibi yüksek çözünürlüklü yöntemlerin stabilize edilmesi ve hibrit skorla birleştirilmesi.
- **Ölçüm Hijyeni:** Offset tuning, korunaklı RF ortamı ve periyodik kalibrasyon.
- **GUI:** Kaydet–yeniden yükle, CSV patern editörü ve otomatik tepe gruplama seçenekleri.

Not

Bu raporun yazım sürecinde yapay zekâ tabanlı bir asistanın desteğinden faydalanılmıştır.

Kaynaklar

- [1] PySDR: A Guide to SDR and DSP Using Python, *Sampling Fundamentals* bölümü. (<https://pysdr.org/>)
- [2] C. A. Balanis, *Antenna Theory: Analysis and Design*, Wiley, 3rd Ed.
- [3] MathWorks Documentation, *Element and Array Radiation Patterns and Responses*.

Ekler

A. Örnek Parametreler

Örnekleme Hızı	2 MS/s (kompleks IQ)
Taşıyıcı Frekans	2.4 GHz / 435 MHz testleri
Anten Aralığı	$d \approx \lambda/2$ tercih edildi
Pencere	Hann / Blackman
Korelasyon	Forward-Backward, diagonal loading & medyan yumuşatma
Hibrit Ağırlık	$\alpha = 0.75$ (faz) / 0.25 (genlik)

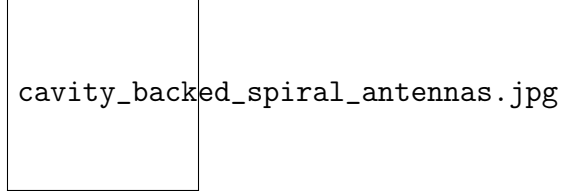
B. Kısa Terimler Sözlüğü

SDR	Yazılım Tanımlı Radyo
LO	Yerel Osilatör; downconversion için referans üretir
DC Spike	Zero-IF mimaride merkez frekanstaki ofset/sızıntı piki
CFO	Taşıyıcı Frekans Kayması (Carrier Frequency Offset)
DOA	Geliş Açısı (Direction of Arrival)

Ek C: Görsel Ekler

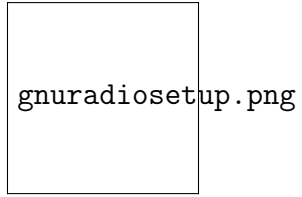
Şekil 1: ANTSDR port girişleri (genel görünüm).

Şekil 2: ANTSDR port girişleri (detay).

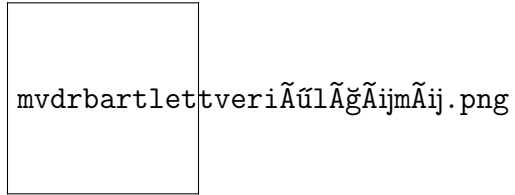


Şekil 3: Cavity-backed spiral antenler.

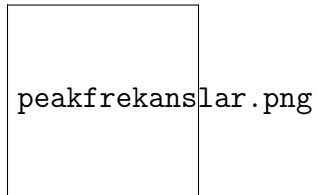
Şekil 4: GNU Radio canlı spektrum ekranı.



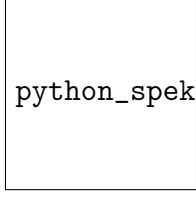
Şekil 5: GNU Radio blok diyagramı.



Şekil 6: MVDR–Bartlett veri ölçümü.

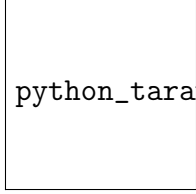


Şekil 7: Peak frekans grupları.



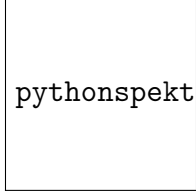
python_spektrum_analizi.png

Şekil 8: Python spektrum analizi.



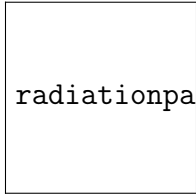
python_tarama.png

Şekil 9: Python bant tarama uygulaması.



pythonspektrumanalizi.png

Şekil 10: Python spektrum analizi (alternatif).



radiationpaternoFantennas.png

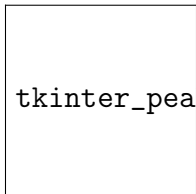
Şekil 11: Antenlerin radyasyon paterni.

Şekil 12: Deney düzeneği (genel).

Şekil 13: Çubuk antenle farklı açı testi.

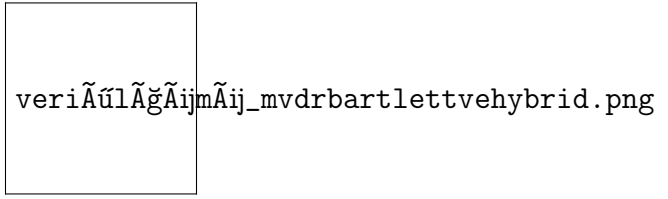
Şekil 14: Çubuk antenle uzak/near-far senaryosu.

Şekil 15: Anten aralığı değiştirilerek yapılan test.



tkinter_peakfinder.png

Şekil 16: Tkinter tabanlı peak finder GUI.



Şekil 17: Veri ölçümü: MVDR, Bartlett ve Hibrit karşılaştırması.

Ek D: Örnek Python Kodları

Listing 1: HybridMVDRBartlett beamforming

```
# -*- coding: utf-8 -*-
"""
Created on Fri Aug 15 11:06:52 2025

@author: stjyer1
"""

# -*- coding: utf-8 -*-
"""
E310 (AD9361) Hybrid DF + Bartlett/MVDR (Stabilized)      with CSV
    logging
- Lock-in phasor with CFO correction
- Phase & amplitude offset calibration
- Optional amplitude pattern CSV
- Robust averaging, forward-backward covariance, diagonal loading
- Multiple measurements per run with running-median smoothing
- CSV logging of every measurement
"""

import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import get_window, savgol_filter
import csv, json, os, time
from datetime import datetime
import adi

# ===== USER SETTINGS =====
SDR_URI      = "ip:192.168.2.1"
FS           = 2_000_000                # sample rate
FC           = 2_400_000_000            # RF
N_SAMP       = 200_000                  # samples per capture
RX_GAINS     = [35, 35]                 # dB
TX_ATTEN     = -30                      # dB
TONE_BB_HZ   = 100_000                  # baseband tone for TX
D_LAMBDA     = 0.5                      # element spacing / lambda
CAL_FILE     = "./df_cal.json"
PATTERN_CSV  = "./pattern_diffgain_vs_angle.csv"      # e.g., "./pattern_diffgain_vs_angle.csv"
MODE         = "measure"                # "calibrate" or "measure"

# Stability / averaging
AVERAGES     = 8                        # captures averaged per
    measurement
MEAS_TIMES   = 5                        # how many measurements to do in
    a row
SLEEP_BETWEEN_S = 0.15                  # sleep between captures
RUNNING_MED_N = 5                        # median window over AoA (odd)
CFO_CORRECT  = True
CFO_BLOCKS   = 16
WIN_NAME     = "hann"

# MVDR/Bartlett settings
PLOT_BEAMS   = True
```

```

SCAN_RES_DEG      = 0.25                # scan resolution
LOADING_A         = 1e-2                # diagonal loading
FORWARD_BACKWARD = True                 # forward-backward covariance
    averaging

# Logging
LOG_FILE          = "./df_measure_log.csv"

# ===== CONSTANTS =====
c = 3e8
lam = c/FC
d = D_LAMBDA * lam

def setup_sdr():
    sdr = adi.ad9361(SDR_URI)
    sdr.sample_rate = int(FS)
    sdr.rx_rf_bandwidth = int(min(FS, 0.8*FS))
    sdr.tx_rf_bandwidth = int(min(FS, 0.8*FS))
    sdr.rx_lo = int(FC)
    sdr.tx_lo = int(FC)
    sdr.rx_enabled_channels = [0, 1]
    sdr.gain_control_mode_chan0 = "manual"
    sdr.gain_control_mode_chan1 = "manual"
    sdr.rx_hardwaregain_chan0 = int(RX_GAINS[0])
    sdr.rx_hardwaregain_chan1 = int(RX_GAINS[1])

    sdr.tx_cyclic_buffer = True
    sdr.tx_enabled_channels = [0]
    sdr.tx_hardwaregain_chan0 = float(TX_ATTEN)

    t = np.arange(N_SAMP)/FS
    tx = 0.5*np.exp(1j*2*np.pi*TONE_BB_HZ*t)
    sdr.tx(tx.astype(np.complex64))

    sdr.rx_buffer_size = N_SAMP
    time.sleep(0.15)
    return sdr

def capture_iq(sdr):
    _ = sdr.rx()
    iq = sdr.rx()
    return iq[0].astype(np.complex64), iq[1].astype(np.complex64)

def phasor_lockin(x, fs, f0, win="hann"):
    N = len(x); t = np.arange(N)/fs
    osc = np.exp(-1j*2*np.pi*f0*t)
    if win:
        w = get_window(win, N, fftbins=True)
        ph = np.sum(x*osc*w)/np.sum(w)
    else:
        ph = np.mean(x*osc)
    return ph

def estimate_cfo(x, fs, f0, blocks=16, win="hann"):
    N = len(x); L = N//blocks
    if L < 64: return 0.0
    phs = []
    for i in range(blocks):

```

```

        seg = x[i*L:(i+1)*L]
        phs.append(np.angle(phasor_lockin(seg, fs, f0, win)))
    phs = np.unwrap(np.array(phs))
    dt = L/fs
    slope = np.polyfit(np.arange(blocks)*dt, phs, 1)[0] # rad/s
    return float(slope/(2*np.pi))

def estimate_tone_phasor_stable(x, fs, f0, win="hann", cfo_correct=True,
    blocks=16):
    if cfo_correct:
        df = estimate_cfo(x, fs, f0, blocks=blocks, win=win)
        ph = phasor_lockin(x, fs, f0+df, win)
    else:
        df = 0.0; ph = phasor_lockin(x, fs, f0, win)
    return ph, df

def load_cal():
    if os.path.exists(CAL_FILE):
        with open(CAL_FILE, "r") as f:
            return json.load(f)
    return None

def save_cal(cal):
    with open(CAL_FILE, "w") as f:
        json.dump(cal, f, indent=2)

def load_pattern_csv(csv_path):
    if not csv_path or not os.path.exists(csv_path): return None
    data = np.genfromtxt(csv_path, delimiter=",", names=True)
    return data

def amp_ratio_to_angle(diff_gain_db, pattern_data):
    ang = pattern_data["angle_deg"]; dif = pattern_data["diff_gain_dB"]
    idx = np.argmin(np.abs(dif - diff_gain_db))
    return float(ang[idx])

def fb_average(R):
    """Forward-backward averaging for 2-element ULA."""
    J = np.array([[0,1],[1,0]])
    return 0.5*(R + J@R.conj()@J)

def estimate_bartlett_mvdr(X_snapshots, scan_deg, loading_alpha=1e-2, fb
= True):
    M, K = X_snapshots.shape
    R = (X_snapshots @ X_snapshots.conj().T) / K
    if fb: R = fb_average(R)
    R += np.eye(M, dtype=complex) * (loading_alpha * np.trace(R).real /
        M)
    Rinv = np.linalg.pinv(R)

    PB, PM = [], []
    for th in np.radians(scan_deg):
        a = np.array([1.0, np.exp(-1j*2*np.pi*d*np.sin(th)/lam)], dtype=
            np.complex128).reshape(-1,1)
        pb = np.real((a.conj().T @ R @ a).squeeze())
        denom = (a.conj().T @ Rinv @ a).squeeze()
        pm = np.real(1.0 / denom) if np.abs(denom) > 1e-12 else 0.0
        PB.append(pb); PM.append(pm)

```

```

PB = 10*np.log10(np.maximum(np.array(PB), 1e-12)); PB -= PB.max()
PM = 10*np.log10(np.maximum(np.array(PM), 1e-12)); PM -= PM.max()
return PB, PM

def ensure_log_header():
    if not os.path.exists(LOG_FILE):
        with open(LOG_FILE, 'w', newline='') as f:
            w = csv.writer(f)
            w.writerow(["timestamp", "phase_diff_rad", "amp_ratio_dB",
                        "CF00_Hz", "CF01_Hz", "phase_AoA_deg",
                        "amp_AoA_deg", "hybrid_AoA_deg"])

def log_row(phase_diff, amp_ratio_db, df0, df1, th_phase, th_amp,
            th_hybrid):
    with open(LOG_FILE, 'a', newline='') as f:
        w = csv.writer(f)
        w.writerow([datetime.now().isoformat(timespec='seconds'),
                    f"{phase_diff:.6f}", f"{amp_ratio_db:.3f}",
                    f"{df0:.2f}", f"{df1:.2f}",
                    f"{th_phase:.2f}", " " if th_amp is None else f"{
                        th_amp:.2f}",
                    f"{th_hybrid:.2f}"])

def main():
    sdr = setup_sdr()
    pattern = load_pattern_csv(PATTERN_CSV)
    cal = load_cal()
    phs_of, amp_of = (0.0, 0.0) if cal is None else (float(cal["
        phase_offset_rad"]), float(cal["amp_offset_db"]))

    ensure_log_header()
    scan_deg = np.arange(-90, 90+SCAN_RES_DEG, SCAN_RES_DEG)
    aoa_series = []

    for m in range(MEAS_TIMES):
        P0, P1, df0s, df1s = [], [], [], []
        for _ in range(AVERAGES):
            x0, x1 = capture_iq(sdr)
            p0, df0 = estimate_tone_phasor_stable(x0, FS, TONE_BB_HZ,
                WIN_NAME, CFO_CORRECT, CFO_BLOCKS)
            p1, df1 = estimate_tone_phasor_stable(x1, FS, TONE_BB_HZ,
                WIN_NAME, CFO_CORRECT, CFO_BLOCKS)
            P0.append(p0); P1.append(p1); df0s.append(df0); df1s.append(
                df1)
            time.sleep(SLEEP_BETWEEN_S)

        P0m = np.mean(np.array(P0)); P1m = np.mean(np.array(P1))
        phase_diff_raw = np.angle(P1m/P0m)
        phase_diff = np.angle(np.exp(1j*(phase_diff_raw - phs_of)))
        amp_ratio_db_raw = 20*np.log10(np.abs(P0m)/np.abs(P1m))
        amp_ratio_db = amp_ratio_db_raw - amp_of

        theta_phase_rad = np.arcsin(np.clip((phase_diff) * lam / (2*np.
            pi*d), -1.0, 1.0))
        theta_phase_deg = float(np.degrees(theta_phase_rad))

        # ==== NEW: Amplitude AoA with pattern or fallback model ====
        if pattern is not None:

```



```

        theta_amp_deg = amp_ratio_to_angle(amp_ratio_db, pattern)
    else:
        # Basit bir lineer model rnei (gerekirse ger ek
        lme g re ayarla)
        theta_amp_deg = float(np.clip(amp_ratio_db * 3.0, -90, 90))
        # rnek katsay : 3 deg/dB

    # Hybrid AoA
    theta_hybrid_deg = 0.75 * theta_phase_deg + 0.25 * theta_amp_deg
    aoa_series.append(theta_hybrid_deg)

    # Running median smoothing
    if len(aoa_series) >= RUNNING_MED_N and RUNNING_MED_N % 2 == 1:
        med = float(np.median(aoa_series[-RUNNING_MED_N:]))
    else:
        med = theta_hybrid_deg

    print(f"[{m+1}/{MEAS_TIMES}] AoA phase={theta_phase_deg:+.2f} ,
        "
        f"hybrid={theta_hybrid_deg:+.2f} | median={med:+.2f} |
        "
        f"CF00={np.mean(df0s):+.1f} Hz, CF01={np.mean(df1s):+.1f} Hz")

    log_row(phase_diff, amp_ratio_db, np.mean(df0s), np.mean(df1s),
            theta_phase_deg, theta_amp_deg, theta_hybrid_deg)

# ---- Plot once using last capture for beam patterns ----
if PLOT_BEAMS:
    x0, x1 = capture_iq(sdr)
    K = min(8192, len(x0))
    X = np.vstack([x0[:K], x1[:K]])
    PB, PM = estimate_bartlett_mvdr(X, scan_deg, loading_alpha=
        LOADING_A, fb=FORWARD_BACKWARD)

    th = np.radians(scan_deg + 90)
    fig = plt.figure(figsize=(8,8))
    ax = plt.subplot(111, projection='polar')
    ax.plot(th, PB, label="Bartlett (norm, dB)")
    ax.plot(th, PM, label="MVDR (norm, dB)")

    def ang2pol(a_deg): return np.radians(a_deg + 90)
    rmin = min(PB.min(), PM.min())
    ax.plot([ang2pol(aoa_series[-1])*2, [rmin, -1.0], linestyle='--
        ', label=f"Hybrid AoA {aoa_series[-1]:+.1f} ")

    ax.set_theta_zero_location('N'); ax.set_theta_direction(-1)
    ax.set_title("Bartlett & MVDR (2-eleman ULA) + AoA i aretleri")
    ax.legend(loc="lower left", bbox_to_anchor=(1.05, 0.1))
    plt.tight_layout()
    plt.show()

    try: sdr.tx_destroy_buffer()
    except Exception: pass

if __name__ == "__main__":
    main()

```

Listing 2: BartlettMVDR comparison

```

# -*- coding: utf-8 -*-
"""
Created on Fri Aug 15 10:36:29 2025

@author: stajyer1
"""

# -*- coding: utf-8 -*-
"""
E310 (AD9361) ile Hybrid Amplitude/Phase Comparison DF (Patent tarz )
- 1 TX sabit ton gnderir, 2 RX e zamanl rnekler
- Faz fark + genlik oran -> AoA
- (Opsiyonel) Pattern CSV ile amplitude-AoA e le tirme (
hibrittle tirme)
- (Opsiyonel) Bartlett ve MVDR polar plot
"""

import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import get_window
from scipy.linalg import eigh
import json, os, time
import adi

# ===== KULLANICI AYARLARI =====
SDR_URI = "ip:192.168.2.1"
FS = 2_000_000 # rnekleme
FC = 2_400_000_000 # ta y c
N_SAMP = 200_000 # RX rnek say s / capture
RX_GAINS = [35, 35] # dB
TX_ATTEN = -30 # dBFS benzeri; ADALM i in
tx_hardwaregain dB (negatif)
TONE_BB_HZ = 100_000 # baseband tone (TX), R X te de
bu tonu arayaca z
D_LAMBDA = 0.7 # eleman aral / lambda (ULA
i in)
CAL_FILE = "./df_cal.json" # faz/genlik ofset kalibrasyonu
PATTERN_CSV= None # rn : "./
pattern_diffgain_vs_angle.csv" (angle_deg, diff_gain_dB)
MODE = "measure" # "calibrate" ya da "measure"
AVERAGES = 4 # capture tekrar (ortalama)
PLOT_BEAMS = True # Bartlett/MVDR polar plot iz
FREQ_BIN_WINDOW = "hann" # tonu lerken pencere
SEED = 42

np.random.seed(SEED)

# ===== YARDIMCI =====
c = 3e8
lam = c/FC
d = D_LAMBDA * lam

def setup_sdr():
    sdr = adi.ad9361(SDR_URI)
    # Ortak
    sdr.sample_rate = int(FS)
    sdr.rx_rf_bandwidth = int(min(FS, 0.8*FS))

```

```

sdr.tx_rf_bandwidth = int(min(FS, 0.8*FS))
sdr.rx_lo = int(FC)
sdr.tx_lo = int(FC)

# RX
sdr.rx_enabled_channels = [0, 1]
sdr.gain_control_mode_chan0 = "manual"
sdr.gain_control_mode_chan1 = "manual"
sdr.rx_hardwaregain_chan0 = int(RX_GAINS[0])
sdr.rx_hardwaregain_chan1 = int(RX_GAINS[1])

# TX
sdr.tx_cyclic_buffer = True
sdr.tx_enabled_channels = [0] # tek TX kullan
sdr.tx_hardwaregain_chan0 = float(TX_ATTEN) # dB (genelde negatif)
# Baseband ton ret ve ykle
t = np.arange(N_SAMP)/FS
tx = 0.5*np.exp(1j*2*np.pi*TONE_BB_HZ*t)
sdr.tx(tx.astype(np.complex64))

# RX buffer derinli i vs
sdr.rx_buffer_size = N_SAMP
time.sleep(0.1)
return sdr

def capture_iq(sdr):
    # FIFO'yu temizlemek i in bir dump
    _ = sdr.rx()
    iq = sdr.rx()
    x0 = iq[0].astype(np.complex64)
    x1 = iq[1].astype(np.complex64)
    return x0, x1

def estimate_tone_phasor(x, fs, tone_hz, win="hann"):
    """
    Tek bir dar tonun kompleks genlik/faz n tahmin et.
    Yntem: pencere -> FFT -> en yak n bin -> ortalama kompleks de er
    """
    N = len(x)
    if win:
        w = get_window(win, N, fftbins=True)
        xw = x * w
    else:
        xw = x

    # En yak n FFT bin
    k = int(np.round(tone_hz * N / fs)) % N
    X = np.fft.fft(xw)
    phasor = X[k] / (np.sum(w) if win else N) # pencere d zeltmesi
    return phasor

def load_cal():
    if os.path.exists(CAL_FILE):
        with open(CAL_FILE, "r") as f:
            return json.load(f)
    return None

```

```

def save_cal(cal):
    with open(CAL_FILE, "w") as f:
        json.dump(cal, f, indent=2)

def load_pattern_csv(csv_path):
    """
    CSV: angle_deg, diff_gain_dB
    diff_gain_dB = 20*log10(|X0|/|X1|)          lmn          beklenen de eri
    """
    if not csv_path or not os.path.exists(csv_path):
        return None
    data = np.genfromtxt(csv_path, delimiter=",", names=True)
    # beklenen kolon adlar : angle_deg, diff_gain_dB
    return data

def amp_ratio_to_angle(diff_gain_db, pattern_data):
    """
    Pattern tablosundan diferansiyel gain -> a          (interpolasyon).
    E er tablo monoton de ilse, en yak n e le meyi se iyoruz.
    """
    ang = pattern_data["angle_deg"]
    dif = pattern_data["diff_gain_dB"]
    # Ters e leme: mutlak fark minimize eden a
    idx = np.argmin(np.abs(dif - diff_gain_db))
    return float(ang[idx])

def hybrid_fusion(theta_phase, theta_amp, w_phase=0.7, w_amp=0.3):
    if theta_amp is None:
        return theta_phase
    return w_phase*theta_phase + w_amp*theta_amp

def safe_arcsin(x):
    return np.arcsin(np.clip(x, -1.0, 1.0))

def estimate_bartlett_mvdr(X_snapshots, d_lambda, scan_deg=np.linspace(-90,90,721)):
    """
    X_snapshots: shape (M, K) -> M anten, K snapshot (zaman)
    2 elemanl ULA varsay yoruz.
    """
    M, K = X_snapshots.shape
    R = (X_snapshots @ X_snapshots.conj().T) / K
    # Bartlett:  $P_B(\theta) = \mathbf{a}^H \mathbf{R} \mathbf{a}$ 
    # MVDR:  $P_M(\theta) = 1 / (\mathbf{a}^H \mathbf{R}^{-1} \mathbf{a})$ 
    #  $\mathbf{a}(\theta) = [1, \exp(-j 2 \pi d \sin(\theta) / \lambda)]^T$ 
    d = d_lambda * lam
    Rinv = np.linalg.pinv(R)
    pb_list, pm_list = [], []
    for th in np.radians(scan_deg):
        a = np.array([1.0, np.exp(-1j*2*np.pi*d*np.sin(th)/lam)]), dtype=
            np.complex128).reshape(-1,1)
        pb = np.real((a.conj().T @ R @ a).squeeze())
        denom = (a.conj().T @ Rinv @ a).squeeze()
        pm = np.real(1.0 / denom) if np.abs(denom) > 1e-12 else 0.0
        pb_list.append(pb)
        pm_list.append(pm)
    PB = 10*np.log10(np.maximum(np.array(pb_list), 1e-12))
    PM = 10*np.log10(np.maximum(np.array(pm_list), 1e-12))

```

```

# normalize for display
PB -= PB.max()
PM -= PM.max()
return scan_deg, PB, PM

# ===== ANA AKI =====
def main():
    sdr = setup_sdr()
    print(f"[i] SDR haz r . FC={FC/1e9:.3f} GHz , FS={FS/1e6:.1f} Msps ,
          tone={TONE_BB_HZ/1e3:.1f} kHz")

    # Kalibrasyon dosyas n oku
    cal = load_cal()
    pattern = load_pattern_csv(PATTERN_CSV)

    if MODE == "calibrate":
        print("[i] Kalibrasyon ba l yor . Antenleri referans (bilinen
              a ) konumuna koy .")
        phs_list, amp_list = [], []
        for i in range(AVERAGES):
            x0, x1 = capture_iq(sdr)
            p0 = estimate_tone_phasor(x0, FS, TONE_BB_HZ, win=
                                     FREQ_BIN_WINDOW)
            p1 = estimate_tone_phasor(x1, FS, TONE_BB_HZ, win=
                                     FREQ_BIN_WINDOW)
            phs_list.append(np.angle(p1/p0))
                                # rad
            amp_list.append(20*np.log10(np.abs(p0)/np.abs(p1)))
                                # dB
        phs_off = float(np.angle(np.mean(np.exp(1j*np.array(phs_list))))
                        ) # sarmal ortalama
        amp_off = float(np.mean(amp_list))
        cal = {
            "fc": FC, "fs": FS, "d_lambda": D_LAMBDA,
            "phase_offset_rad": phs_off,
            "amp_offset_db": amp_off,
            "timestamp": time.time()
        }
        save_cal(cal)
        print(f"[ok] Kalibrasyon kaydedildi : {CAL_FILE}")
        print(f"phase_offset = {phs_off:.4f} rad , amp_offset = {
              amp_off:.3f} dB")
        return

    # ---- MEASUREMENT ----
    if cal is None:
        print("[!] Uyar : Kalibrasyon bulunamad , offsetler 0 kabul
              edilecek .")
        phs_off = 0.0
        amp_off = 0.0
    else:
        phs_off = float(cal["phase_offset_rad"])
        amp_off = float(cal["amp_offset_db"])

    # Averaging
    p0_all, p1_all = [], []
    for i in range(AVERAGES):
        x0, x1 = capture_iq(sdr)

```

```

p0 = estimate_tone_phasor(x0, FS, TONE_BB_HZ, win=
    FREQ_BIN_WINDOW)
p1 = estimate_tone_phasor(x1, FS, TONE_BB_HZ, win=
    FREQ_BIN_WINDOW)
p0_all.append(p0); p1_all.append(p1)

P0 = np.mean(np.array(p0_all))
P1 = np.mean(np.array(p1_all))

# Faz ve genlik fark (kalibrasyon d zeltmeli)
phase_diff_raw = np.angle(P1/P0) # rad
phase_diff = np.angle(np.exp(1j*(phase_diff_raw - phs_off))) # -pi
    ..pi
amp_ratio_db_raw = 20*np.log10(np.abs(P0)/np.abs(P1))
amp_ratio_db = amp_ratio_db_raw - amp_off

# Fazdan AoA (iki elemanlı ULA; sin(theta) = * / (2 d))
theta_phase_rad = safe_arcsin((phase_diff) * lam / (2*np.pi*d))
theta_phase_deg = float(np.degrees(theta_phase_rad))

# Amplit dden AoA (pattern varsa)
theta_amp_deg = None
if pattern is not None:
    theta_amp_deg = amp_ratio_to_angle(amp_ratio_db, pattern)

# Hibrit
theta_hybrid_deg = hybrid_fusion(theta_phase_deg, theta_amp_deg,
    w_phase=0.7, w_amp=0.3)

print(f"[meas] phase_diff={phase_diff:.4f} rad, amp_ratio={
    amp_ratio_db:.2f} dB")
print(f"[AoA] phase-only={theta_phase_deg:.1f} ")
if theta_amp_deg is not None:
    print(f"[AoA] amplitude={theta_amp_deg:.1f} (pattern)")
print(f"[AoA] HYBRID={theta_hybrid_deg:.1f} ")

# ---- (Opsiyonel) Bartlett/MVDR G rsellesine tirme ----
if PLOT_BEAMS:
    # Snapshot matrisi: (M=2, K) ayn alyan bir pencere
    ekelim
    x0, x1 = capture_iq(sdr)
    K = min(4096, len(x0))
    X = np.vstack([x0[:K], x1[:K]])
    scan_deg, PB, PM = estimate_bartlett_mvdr(X, D_LAMBDA, scan_deg=
        np.linspace(-90,90,721))

    # Polar izim (radyan eksenini: g )
    th = np.radians(scan_deg + 90) # 0 yukar olsun diye +90
        kayd r yoruz
    fig = plt.figure(figsize=(7,7))
    ax = plt.subplot(111, projection='polar')
    ax.plot(th, PB, label="Bartlett (norm, dB)")
    ax.plot(th, PM, label="MVDR (norm, dB)")
    # Tahmin izgileri
    def angle_to_polar(theta_deg):
        return np.radians(theta_deg + 90)
    ax.plot([angle_to_polar(theta_phase_deg)]*2, [PB.min(), 0],
        linestyle='--', label=f"Phase AoA {theta_phase_deg:.1f} ")

```

```

        if theta_amp_deg is not None:
            ax.plot([angle_to_polar(theta_amp_deg)]*2, [PB.min(), -1],
                    linestyle='--', label=f"Amp_AoA_{theta_amp_deg:.1f} ")
        ax.plot([angle_to_polar(theta_hybrid_deg)]*2, [PB.min(), -2],
                linestyle='--', label=f"Hybrid_AoA_{theta_hybrid_deg:.1f} ")

    ax.set_theta_zero_location('N')
    ax.set_theta_direction(-1)
    ax.set_title("Bartlett & MVDR (2-eleman ULA) + AoA i aretleri")
    ax.legend(loc="lower_left", bbox_to_anchor=(1.05, 0.1))
    plt.tight_layout()
    plt.show()

    # Temizlik
    sdr.tx_destroy_buffer()
    del sdr

if __name__ == "__main__":
    main()

```

Listing 3: Transmit and Receive

```

# -*- coding: utf-8 -*-
"""
AM Mod lasyonlu Sweep Sinyali G nderimi ve Spektral Analizi (435 MHz
    ta y c )
@author: stjyer1
@date: 2025-07-30
"""

import numpy as np
import matplotlib.pyplot as plt
from scipy.fft import fft, fftfreq, fftshift
import adi
import time

# SDR Cihaz Ayarlar
sdr = adi.ad9361("ip:192.168.2.1")
sdr.sample_rate = int(2e6)
sdr.tx_rf_bandwidth = int(2e6)
sdr.rx_rf_bandwidth = int(2e6)
#test frekans 435MHz
sdr.tx_lo = int(435e6)
sdr.rx_lo = int(435e6)
sdr.tx_enabled_channels = [0]
sdr.rx_enabled_channels = [0]
sdr.rx_buffer_size = 4096
sdr.tx_cyclic_buffer = False
sdr.gain_control_mode = "manual"
sdr.rx_hardwaregain_chan0 = 50

# Sinyal retimi Ayarlar
duration = 0.5 # saniye
fs = sdr.sample_rate
N = int(fs * duration)
t = np.arange(N) / fs

```

```

#           Se enek: Sweeping sin s (4 kHz           6 kHz)
f0 = 4000 # ba lang frekans (Hz)
f1 = 6000 # biti frekans (Hz)
baseband = 0.5 * np.sin(2 * np.pi * (f0 + (f1 - f0) * t / duration) * t)

# AM Mod lasyon (baseband + ta y c )
carrier = np.exp(2j * 2 * np.pi * 0 * t) # ta y c 0 Hz (baseband)
tx_signal = (1 + 0.8 * baseband) * carrier

# TX G nderimi
print("TX_ba lad_(Sweep_sinyali)...")
sdr.tx(tx_signal.astype(np.complex64))
time.sleep(0.05) # donan ma zaman tan
sdr.tx_destroy_buffer()
print("TX_bitti.")

# RX Al m
print("RX_ba l yor...")
samples = sdr.rx()
samples = samples - np.mean(samples) # DC offset d zeltmesi

# FFT Analizi
fft_data = fft(samples)
power = 20 * np.log10(np.abs(fftshift(fft_data)) + 1e-3)
power = np.clip(power, a_min=0, a_max=None)
freqs = fftshift(fftfreq(len(samples), 1/fs)) + sdr.rx_lo

#           FFT Grafi i
plt.figure(figsize=(12, 5))
plt.plot(freqs / 1e6, power, color="royalblue")
plt.title("435_MHz_AM_Sinyal_FFT")
plt.xlabel("Frekans_(MHz)")
plt.ylabel("G_(dB)")
plt.grid(True)
plt.tight_layout()
plt.show()

#           Spektrogram (Zaman-Frekans) Analizi
plt.figure(figsize=(12, 4))
plt.specgram(np.real(samples), Fs=fs, NFFT=1024, noverlap=512, cmap="
viridis")
plt.title("Zaman-Frekans_(Spektrum)_Analizi")
plt.xlabel("Zaman_(saniye)")
plt.ylabel("Frekans_(Hz)")
plt.tight_layout()
plt.show()

```

Listing 4: Frequency Selection

```

import os
import ctypes
import adi
import tkinter as tk
import numpy as np
import matplotlib.pyplot as plt
from scipy.fft import fft, fftfreq
from scipy.signal import find_peaks, firwin, lfilter, get_window
from sklearn.cluster import DBSCAN

```



```

# --- DLL ve SDR ayar ---
os.environ["PATH"] += os.pathsep + "C:/Program Files/II0 Oscilloscope/
bin"
ctypes.cdll.LoadLibrary("C:/Program Files/II0 Oscilloscope/bin/libiio.
dll")

sdr = adi.ad9361("ip:192.168.2.1")
sdr.sample_rate = int(2e6)
sdr.rx_enabled_channels = [0]
sdr.rx_buffer_size = 2048 # FFT          znrl          i in art r ld

# Filter bank parametreleri
n_bands = 8
band_width = (sdr.sample_rate // 2) // n_bands # Toplam bant
          geni li i fs/2
filter_order = 128
window_type = 'hann' # 'hamming', 'blackman' da deneyebilirsiniz

start_freq = 70
end_freq = 6000
region_count = 10
region_width = (end_freq - start_freq) // region_count
peak_frequencies_all = []

for i in range(region_count):
    region_start = start_freq + i * region_width
    region_end = region_start + region_width
    sweep_freqs = list(range(region_start, region_end, 5))

    print(f"\n      B lge {i+1}: {region_start} {region_end} MHz")
    plt.figure(figsize=(12, 6))

    for freq in sweep_freqs:
        sdr.rx_lo = int(freq * 1e6)
        samples = sdr.rx()
        samples = samples - np.mean(samples) # DC offset gider

        # Her alt band tek tek analiz et
        for b in range(n_bands):
            low = b * band_width
            high = low + band_width

            # D k s n r s f r olamaz, en az 1 Hz olsun
            if low == 0:
                low = 1

            # Y ksek s n r fs/2'yi a amaz
            if high >= sdr.sample_rate // 2:
                high = (sdr.sample_rate // 2) - 1

            # Hatal band atla
            if low >= high:
                continue

            low_hz = low
            high_hz = high

```

```

# FIR bandpass filter tasarla
taps = firwin(
    filter_order, [low_hz / (sdr.sample_rate/2), high_hz / (
        sdr.sample_rate/2)],
    pass_zero=False, window=window_type
)
filtered = lfilter(taps, 1.0, samples)

# Pencere uygula
window = get_window(window_type, len(filtered))
windowed = filtered * window

# FFT
N = len(windowed)
T = 1.0 / sdr.sample_rate
xf = fftfreq(N, T)
yf = fft(windowed)
xf_mhz = xf[:N//2] / 1e6 + freq + (low_hz / 1e6) # Frekans
        kaymas n da ekle
yf_mag = 2.0 / N * np.abs(yf[:N//2])

# ok d k genlikli bandlar atla
if np.max(yf_mag) < 0.01:
    continue

plt.plot(xf_mhz, yf_mag, alpha=0.18, label=f"Band_{b+1}" if
    freq == sweep_freqs[0] else None)

# Peak bul ve kaydet
peaks, properties = find_peaks(
    yf_mag,
    height=np.max(yf_mag) * 0.7,
    distance=10,
    prominence=1
)
peak_freqs = xf_mhz[peaks]
peak_heights = properties['peak_heights']
strong_peaks = peak_freqs[peak_heights > 0.01]
peak_frequencies_all.extend(strong_peaks)

plt.title(f"B lge_{i+1}:{region_start} {region_end}_MHz")
plt.xlabel("Frekans_(MHz)")
plt.ylabel("Genlik")
plt.grid(True)
plt.tight_layout()
plt.show()

# --- DBSCAN ile peak gruplama ---
print("\ n      _Toplam_ham_peak:", len(peak_frequencies_all))

if len(peak_frequencies_all) > 0:
    freqs_mhz = np.array(peak_frequencies_all).reshape(-1, 1)
    db = DBSCAN(eps=0.02, min_samples=2).fit(freqs_mhz) # 20 kHz, en az
        2 sinyal

    clusters = db.labels_
    grouped_peaks = [
        np.mean(freqs_mhz[clusters == i])

```

```

        for i in np.unique(clusters)
        if np.count_nonzero(clusters == i) > 1
    ]

    grouped_peaks = np.round(sorted(grouped_peaks), 6)

    print(f"\n   Se  ilen   Temiz   Peak   Frekanslar   ({len(grouped_peaks)}   adet):")
    for f in grouped_peaks:
        print(f"   -   {f}   MHz")
else:
    print("           Peak   bulunamad .")

from tkinter import ttk

def show_peaks_in_range():
    try:
        fmin = float(entry_min.get())
        fmax = float(entry_max.get())
        filtered = [f for f in grouped_peaks if fmin <= f <= fmax]
        # Tabloyu temizle
        for item in peak_table.get_children():
            peak_table.delete(item)
        # Listeyi tabloya ekle
        for f in filtered:
            peak_table.insert("", "end", values=(f"{f:.3f}   MHz",))
        if not filtered:
            result.set("No   peaks   found   in   this   range.")
        else:
            result.set(f"{len(filtered)}   peak(s)   found.")
    except Exception as e:
        result.set("Please   enter   valid   numbers.")

def clear_table():
    entry_min.delete(0, tk.END)
    entry_max.delete(0, tk.END)
    for item in peak_table.get_children():
        peak_table.delete(item)
    result.set("")

root = tk.Tk()
root.title("Peak   Frequency   Finder")
root.geometry("330x450")
root.resizable(False, False)

frm_top = tk.Frame(root, pady=10)
frm_top.pack()

tk.Label(frm_top, text="Min   Freq   (MHz):", font=("Segoe   UI", 11)).grid(
    row=0, column=0, sticky="e", padx=2)
entry_min = tk.Entry(frm_top, width=8, font=("Segoe   UI", 11))
entry_min.grid(row=0, column=1, sticky="w", padx=2)

tk.Label(frm_top, text="Max   Freq   (MHz):", font=("Segoe   UI", 11)).grid(
    row=1, column=0, sticky="e", padx=2)
entry_max = tk.Entry(frm_top, width=8, font=("Segoe   UI", 11))

```

```

entry_max.grid(row=1, column=1, sticky="w", padx=2)

frm_btn = tk.Frame(root)
frm_btn.pack(pady=5)

tk.Button(frm_btn, text="Show_Peaks", font=("Segoe_UI", 10), width=12,
          command=show_peaks_in_range).grid(row=0, column=0, padx=4)
tk.Button(frm_btn, text="Clear", font=("Segoe_UI", 10), width=8, command=
          clear_table).grid(row=0, column=1, padx=4)

result = tk.StringVar()
tk.Label(root, textvariable=result, font=("Segoe_UI", 11, "italic"), fg=
          "#333").pack(pady=4)

# --- Peak Table + Scrollbar ---
frm_table = tk.Frame(root)
frm_table.pack(fill="both", expand=True, padx=10, pady=2)

peak_table = ttk.Treeview(frm_table, columns=("Frequency",), show="
          headings", height=14)
peak_table.heading("Frequency", text="Frequency_(MHz)")
peak_table.column("Frequency", anchor="center", width=130)

scrollbar = ttk.Scrollbar(frm_table, orient="vertical", command=
          peak_table.yview)
peak_table.configure(yscroll=scrollbar.set)
peak_table.grid(row=0, column=0, sticky="nsew")
scrollbar.grid(row=0, column=1, sticky="ns")

frm_table.rowconfigure(0, weight=1)
frm_table.columnconfigure(0, weight=1)

root.mainloop()

```

Listing 5: Simulasyon

```

import os
import ctypes
import adi
import numpy as np
import matplotlib.pyplot as plt
from scipy.fft import fft, fftfreq
from scipy.signal import find_peaks
from sklearn.cluster import DBSCAN

# --- DLL ve ortam ayar ---
os.environ["PATH"] += os.pathsep + "C:/Program_Files/II0_Oscilloscope/
bin"
ctypes.cdll.LoadLibrary("C:/Program_Files/II0_Oscilloscope/bin/libiio.
dll")

# --- SDR ayarlar ---
sdr = adi.ad9361("ip:192.168.2.1")
sdr.sample_rate = int(2e6)
sdr.rx_enabled_channels = [0]
sdr.rx_buffer_size = 1024

# --- Sweep b lge ayarlar ---

```

```

start_freq = 70          # MHz
end_freq = 6000          # MHz
region_count = 10
region_width = (end_freq - start_freq) // region_count

peak_frequencies_all = []

# --- Sweep d n g s ---
for i in range(region_count):
    region_start = start_freq + i * region_width
    region_end = region_start + region_width
    sweep_freqs = list(range(region_start, region_end, 5))

    print(f"\n      B l g e {i+1}: {region_start} {region_end} MHz")
    plt.figure(figsize=(12, 4))

    for freq in sweep_freqs:
        sdr.rx_lo = int(freq * 1e6)
        samples = sdr.rx()

        N = len(samples)
        T = 1.0 / sdr.sample_rate
        xf = fftfreq(N, T)
        yf = fft(samples)

        xf_mhz = xf[:N//2] / 1e6
        yf_mag = 2.0/N * np.abs(yf[:N//2])
        true_freq = xf_mhz + freq

        # Zay f sinyalleri komple atla
        if np.max(yf_mag) < 0.01:
            continue

        # Grafik izimi (saydam)
        plt.plot(true_freq, yf_mag, alpha=0.4)

        # Daha az ve anlamlı peak bul
        peaks, properties = find_peaks(
            yf_mag,
            height=np.max(yf_mag) * 0.5, # %50 e ik
            distance=3                    # min rnek mesafesi
        )
        peak_freqs = true_freq[peaks]
        peak_frequencies_all.extend(peak_freqs)

    plt.title(f"B l g e {i+1}: {region_start} {region_end} MHz Spektrum")
    plt.xlabel("Frekans (MHz)")
    plt.ylabel("Genlik")
    plt.grid(True)
    plt.tight_layout()
    plt.show()

# --- 50 kHz i inde peak gruplama (daha s k ) ---
print("\n      T m T espit Edilen Peak Say s (filtrelenmeden):", len(
    peak_frequencies_all))

if len(peak_frequencies_all) > 0:

```

```

freqs_mhz = np.array(peak_frequencies_all).reshape(-1, 1)
db = DBSCAN(eps=0.05, min_samples=1).fit(freqs_mhz) # 50 kHz
clusters = db.labels_
grouped_peaks = [np.mean(freqs_mhz[clusters == i]) for i in np.
    unique(clusters)]

grouped_peaks = np.round(sorted(grouped_peaks), 6)
print("    Birle tirilmi Peak Frekanslar (50 kHz i inde
    gruplanm ):")
for f in grouped_peaks:
    print(f"-{f} MHz")
else:
    print("    Peak bulunamad .")

```