

# DAB+ Labs สำหรับ Raspberry Pi

คู่มือการเรียนรู้ Digital Audio Broadcasting Plus


พร้อม RTL-SDR และ PyQt5

เวอร์ชัน 1.1 | กันยายน 2025

## วัตถุประสงค์โครงการ

### เรียนรู้เทคโนโลยี

- DAB+ จากพื้นฐานจนถึงขั้นสูง
- Python & PyQt5 GUI programming
- Software Defined Radio (SDR)
- RF Signal Processing

 เป้าหมาย: สร้างแอปที่ใช้งานได้จริงบน Raspberry Pi

### สร้างแอปพลิเคชัน

- DAB+ Station Scanner
- Program Recorder
- Signal Analyzer
- Touch-Friendly GUI

## ข้อกำหนดระบบ

### Hardware

- Raspberry Pi 4 (4GB+ RAM)
- RTL-SDR V4 Dongle
- หน้าจอสัมผัส 7" (HDMI)
- หูฟัง 3.5mm
- เส้าอากาศ DAB+

### Software

- Raspberry Pi OS Bookworm
- Python 3.11+
- PyQt5 GUI Framework
- welle.io DAB+ Decoder
- RTL-SDR Libraries

# ภาพรวมเล่มทั้งหมด

## Lab Series พื้นฐาน (เดิม - เน้น tools)


Lab	หัวข้อ	เวลา	ระดับ
1	การตั้งค่าเบื้องต้น RTL-SDR	20 นาที	★★
2	การรับสัญญาณ DAB+ พื้นฐาน	25 นาที	★★★
3	Command Line Tools สำหรับ DAB+	15 นาที	★★★
4	ETISnoop Analysis	15 นาที	★★★★
5	สรุปและวิเคราะห์ขั้นสูง	10 นาที	★★★★★

รวมเวลา: ~1.25 ชั่วโมง (75 นาที) |  เป้าหมาย: เรียนรู้การใช้งาน DAB+ tools

# ภาพรวมเล่มทั้งหมด (ต่อ)

## Lab Series ขั้นสูง (ใหม่ - เน้น development + trap exercises)

Lab	หัวข้อ	เวลา	ระดับ
1	RTL-SDR Setup + Hardware Detection Traps	2 ชั่วโมง	★★★
2	welle.io Integration + Audio Routing Traps	2 ชั่วโมง	★★★★★
3	pyrtlsdr Spectrum Analysis + IQ Processing Traps	2 ชั่วโมง	★★★★★
4	DAB+ Station Scanner + Database Traps	2 ชั่วโมง	★★★★★
5	Program Recorder + Scheduling Traps	2 ชั่วโมง	★★★★★
6	Signal Analyzer + OFDM Analysis Traps	2 ชั่วโมง	★★★★★

รวมเวลา: ~12 ชั่วโมง |  เป้าหมาย: สร้าง professional DAB+ applications

## LAB 0: Introduction to DAB+, Python, FRP และ PyQt5

 เวลารวม: 105 นาที (1 ชั่วโมง 45 นาที)

 ภาพรวมเนื้อหา

เป็นแล็บพื้นฐานสำหรับมือใหม่ ที่ยังไม่เคยใช้ Python หรือไม่รู้จัก DAB+

## ส่วนที่ 1: DAB+ Technology (15 นาที)

### DAB+ vs FM Radio

- เสียงดิจิทัล ไม่มี static หรือสัญญาณรบกวน
- คุณภาพคงที่ ไม่ขึ้นกับระยะทาง
- **Metadata** ชื่อเพลง, ศิลปิน แบบ real-time
- **MOT Slideshow** รูปภาพ album art
- **Multiplexing** หลายสถานีใช้ความถี่เดียว

### เทคโนโลยีที่ใช้

- **RTL-SDR**: ตัวรับสัญญาณ USB (~500-1500 บาท)
- **welle.io**: DAB+ decoder แบบ open source
- **Python**: สำหรับควบคุมและประมวลผล

### DAB+ ในประเทศไทย (Sep 2025)

สถานีทดลองออกอากาศ:

- **Block 9A**: 202.928 MHz Dharma Radio Station (Bangkok)
- **Block 6C**: 185.360 MHz Khon Kaen Station (Khon Kaen, Maha Sarakham)

see: <https://www.worlddab.org/countries/thailand>

## 🐍 ส่วนที่ 2: Python สำหรับมือใหม่ (30 นาที)

### Python Basics

```
# Variables และ Data Types
name = "สวัสดี"      # String
age = 25              # Integer
height = 175.5        # Float
is_student = True     # Boolean

# Lists และการใช้งาน
fruits = ["แอปเปิ้ล", "กล้วย", "ส้ม"]
fruits.append("มะม่วง")
print(len(fruits))    # แสดง: 4
```

### Control Flow

```
# Loops (การวนซ้ำ)
for fruit in fruits:
    print("ผลไม้:", fruit)

# Conditions (เงื่อนไข)
if age ≥ 18:
    print("เป็นผู้ใหญ่แล้ว")
else:
    print("เป็นเด็ก")

# Functions (ฟังก์ชัน)
def say_hello(name):
    return "สวัสดี " + name
```



# Python: Classes และ Hardware Integration

## Object-Oriented Programming

```
class DABStation:
    def __init__(self, name, frequency):
        self.name = name
        self.frequency = frequency
        self.is_playing = False

    def start_playing(self):
        self.is_playing = True
        print(f"เริ่มเล่น {self.name}")

    def stop_playing(self):
        self.is_playing = False
```

## Raspberry Pi GPIO

```
try:
    import RPi.GPIO as GPIO
    import time

    # ตั้งค่า GPIO pin 18
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(18, GPIO.OUT)

    # กระพริบ LED
    GPIO.output(18, GPIO.HIGH)
    time.sleep(1)
    GPIO.output(18, GPIO.LOW)

except ImportError:
    print("ทำงานบนคอมพิวเตอร์ทั่วไป")
```

## ส่วนที่ 3: FRP Client Setup (30 นาที)

### FRP คืออะไร?

**Fast Reverse Proxy** - เครื่องมือสำหรับเข้าถึง RPI จากภายนอก

### ปัญหาที่แก้:

- RPI อยู่หลัง NAT/Router
- IP บ้านเปลี่ยนแปลงบ่อย
- ไม่สามารถเปิด port forward
- ต้องการเชื่อมต่อจาก Colab

### การทำงาน:

```
[RPI:1234] → [FRP Client] → Internet  
→ [FRP Server:600x] ← [Colab/Client]
```

### ติดตั้ง FRP Client

```
# ตรวจสอบ architecture  
uname -m # aarch64 = ARM64  
  
# ดาวน์โหลด FRP  
wget https://github.com/fatedier/frp/\  
releases/download/v0.61.0/\  
frp_0.61.0_linux_arm64.tar.gz  
  
# แยกไฟล์และติดตั้ง  
tar -xzf frp_*.tar.gz  
cd frp_*  
sudo cp frpc /usr/local/bin/  
sudo chmod +x /usr/local/bin/frpc  
  
# ตรวจสอบ  
frpc --version
```

## FRP: การตั้งค่าและทดสอบ

### สร้าง Config File

```
sudo mkdir -p /etc/frp
sudo nano /etc/frp/frpc.toml
```

เนื้อหาไฟล์ (เปลี่ยน XX = เลขที่นั่ง):

```
serverAddr = "xxx.xxx.xxx.xxx"
serverPort = 7000
auth.method = "token"
auth.token = "YourToken"

[[proxies]]
name = "piXX-tcp-1234"
type = "tcp"
localIP = "127.0.0.1"
localPort = 1234
remotePort = 60XX
```

### ตั้งค่า Systemd Service

```
# สร้างไฟล์ service
sudo nano /etc/systemd/system/frpc.service
```

```
[Unit]
Description=FRP Client Service
After=network.target

[Service]
Type=simple
User=pi
Restart=on-failure
ExecStart=/usr/local/bin/frpc \
    -c /etc/frp/frpc.toml

[Install]
WantedBy=multi-user.target
```

# FRP: เปิดใช้งานและทดสอบ

## เริ่มต้น FRP Service

```
# โหลด config
sudo systemctl daemon-reload

# เปิดใช้งานอัตโนมัติ
sudo systemctl enable frpc

# เริ่มต้น service
sudo systemctl start frpc

# ตรวจสอบสถานะ
sudo systemctl status frpc
# ต้องเห็น: "start proxy success"

# ดู log
sudo journalctl -u frpc -f
```

## ทดสอบจาก Google Colab

```
import socket

# ใส่ข้อมูลของคุณ
FRP_SERVER = "xxx.xxx.xxx.xxx"
FRP_PORT = 60XX # เช่น 6001

sock = socket.socket(
    socket.AF_INET,
    socket.SOCK_STREAM
)
sock.settimeout(5)
result = sock.connect_ex(
    (FRP_SERVER, FRP_PORT)
)

if result == 0:
    print("✅ เชื่อมต่อสำเร็จ!")
else:
    print("❌ ไม่สามารถเชื่อมต่อ")
sock.close()
```

 ผลลัพธ์: เข้าถึง RPI จากภายนอกได้ผ่าน FRP tunnel

## ส่วนที่ 4: PyQt5 Hands-on (30 นาที)

### PyQt5 Components

```
from PyQt5.QtWidgets import *
import sys

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setup_ui()

    def setup_ui(self):
        # สร้าง central widget
        central_widget = QWidget()
        self.setCentralWidget(central_widget)

        # สร้าง layout
        layout = QVBoxLayout(central_widget)
```

### Touch-Friendly Design

```
# ปุ่มขนาดใหญ่สำหรับสัมผัส
button = QPushButton("กดที่นี่")
button.setMinimumSize(120, 60)

# Font ขนาดใหญ่
font = QFont()
font.setPointSize(14)
button.setFont(font)

# CSS Styling
button.setStyleSheet("""
    QPushButton {
        border-radius: 8px;
        background: #3498db;
        color: white;
    }
""")
```

## PyQt5: Signals & Slots

### Event Handling

```
class DABPlayerWidget(QWidget):
    def __init__(self):
        super().__init__()
        self.setup_ui()
        self.setup_connections()

    def setup_connections(self):
        # เชื่อม signals กับ slots
        self.play_button.clicked.connect(self.on_play)
        self.volume_slider.valueChanged.connect(self.on_volume_change)

    def on_play(self):
        print("เริ่มเล่นเพลง!")
```

### QTimer และ Updates

```
from PyQt5.QtCore import QTimer

class SignalMonitor(QWidget):
    def __init__(self):
        super().__init__()

        # Timer สำหรับ real-time update
        self.timer = QTimer()
        self.timer.timeout.connect(self.update_signal)
        self.timer.start(1000) # อัปเดตทุก 1 วินาที

    def update_signal(self):
        # อัปเดตค่าสัญญาณ
        signal_strength = self.get_signal_strength()
        self.signal_bar.setValue(signal_strength)
```

# LAB 0: Demo Applications

## Demo 1: Basic Widgets

- **QLabel** แสดงข้อความและรูปภาพ
- **QPushButton** ปุ่มกดขนาดใหญ่
- **QLineEdit** ช่องใส่ข้อความ
- **QTextEdit** พื้นที่ข้อความหลายบรรทัด
- **QSlider** แถบเลื่อนค่า
- **QProgressBar** แสดงความคืบหน้า

### สิ่งที่นักเรียนต้องเติม:

1. อ่านชื่อ จาก QLineEdit และแสดงใน QLabel
2. ควบคุม QProgressBar ด้วย QSlider
3. เริ่ม/หยุด QTimer และแสดงเวลาปัจจุบัน
4. เปลี่ยนสี ของปุ่มเมื่อกด (CSS styling)

## Demo 2: Touch Interface

- **ขนาดปุ่ม** อย่างน้อย 60x40 pixels
- **Font Size** 12-16pt สำหรับหน้าจอ 7"
- **Visual Feedback** เปลี่ยนสีเมื่อกด
- **Layout Management** responsive design
- **Error Handling** การจัดการข้อผิดพลาด

# LAB 0: ผลลัพธ์ที่คาดหวัง

## ความรู้ที่ได้รับ

### DAB+ Technology:

- เข้าใจความแตกต่างจาก FM
- รู้จักเทคโนโลยี RTL-SDR
- เข้าใจ DAB+ ในประเทศไทย

### Python Programming:

- Variables, functions, classes
- File handling และ modules
- GPIO programming พื้นฐาน

### FRP Remote Access:

- ติดตั้ง FRP Client
- ตั้งค่า systemd service
- Remote tunneling concepts

## Skills ที่พร้อมใช้

### PyQt5 GUI Development:

- Widget การใช้งานพื้นฐาน
- Signals & Slots system
- Touch-friendly UI design
- Real-time updates ด้วย QTimer

### Network & Remote:

- FRP tunnel management
- Remote access จาก Colab
- Service troubleshooting

เตรียมพร้อม สำหรับ Labs ขั้นสูง!



## LAB 1: การตั้งค่าเบื้องต้น RTL-SDR (20 นาที)

### การติดตั้ง RTL-SDR Drivers

```
# อัปเดตระบบ
sudo apt update && sudo apt upgrade -y

# ติดตั้ง dependencies
sudo apt install libusb-1.0-0-dev git cmake pkg-config build-essential

# ดาวน์โหลดและติดตั้ง RTL-SDR Blog drivers
git clone https://github.com/rtlsdrblog/rtl-sdr-blog
cd rtl-sdr-blog
mkdir build && cd build
cmake ../ -DINSTALL_UDEV_RULES=ON
make
sudo make install
sudo ldconfig

# Blacklist DVB-T drivers
echo 'blacklist dvb_usb_rtl28xxu' | sudo tee --append /etc/modprobe.d/blacklist-dvb_usb_rtl28xxu.conf
```

## ✓ การทดสอบ

```
# รีบูตเครื่อง  
sudo reboot  
  
# ทดสอบ RTL-SDR  
rtl_test -t
```

### สิ่งที่ควรตรวจสอบ:

- ☐ RTL-SDR ถูกตรวจพบโดยระบบ
- ☐ ไม่มี error messages
- ☐ Ready สำหรับ Lab 2

## Trap Exercises LAB 1:

### Trap 1.1: Hardware Detection Challenge

หลังรัน `lsusb` ให้วิเคราะห์ output และระบุว่า RTL-SDR อยู่ที่ port ไหน

### Trap 1.2: Driver Permission Investigation

หลัง blacklist DVB-T drivers ให้อธิบายทำไมต้อง blacklist และผลกระทบคืออะไร

### Trap 1.3: PPM Calibration Analysis

หลัง `rtl_test -t` ให้วิเคราะห์ PPM error และอธิบายความหมาย

## LAB 1 Extended: rtl\_tcp Server (15 นาที)

### การรัน rtl\_tcp Server

```
# เริ่มต้น rtl_tcp server
rtl_tcp -a 0.0.0.0 -p 1234 -d 0

# ตรวจสอบว่า server ทำงาน
netstat -an | grep 1234

# ควรเห็น: tcp 0.0.0.0:1234 LISTEN
```

#### Parameters:

- `-a 0.0.0.0` รับ connection จากทุก IP
- `-p 1234` ใช้ port 1234
- `-d 0` เลือก RTL-SDR device 0

### rtl\_tcp Protocol Commands

```
# Command format: 1 byte + 4 bytes (big endian)
import struct

# 0x01: Set frequency (Hz)
freq_cmd = struct.pack('>BI', 0x01, 185360000)

# 0x02: Set sample rate (Hz)
rate_cmd = struct.pack('>BI', 0x02, 2048000)

# 0x03: Set gain mode (0=auto, 1=manual)
mode_cmd = struct.pack('>BI', 0x03, 1)

# 0x04: Set gain (tenths of dB)
gain_cmd = struct.pack('>BI', 0x04, 200) # 20.0 dB

# 0x05: Set frequency correction (ppm)
ppm_cmd = struct.pack('>BI', 0x05, 0)
```

# LAB 1: Python rtl\_tcp Client Example

## Network Connection

```
import socket
import struct
import numpy as np

# เชื่อมต่อ rtl_tcp server
sock = socket.socket(
    socket.AF_INET,
    socket.SOCK_STREAM
)
sock.connect(('localhost', 1234))

# ตั้งค่าความถี่ DAB+ Thailand
freq = 185360000 # 185.360 MHz
cmd = struct.pack('>BI', 0x01, freq)
sock.send(cmd)

# ตั้งค่า sample rate
rate = 2048000 # 2.048 MHz
cmd = struct.pack('>BI', 0x02, rate)
sock.send(cmd)
```

## รับ I/Q Samples

```
# รับข้อมูล I/Q (8192 bytes = 4096 samples)
data = sock.recv(8192)

# แปลง uint8 → float → complex
iq_uint8 = np.frombuffer(data, dtype=np.uint8)
iq_float = (iq_uint8 - 127.5) / 127.5

# แยก I และ Q
i_samples = iq_float[::2]
q_samples = iq_float[1::2]
samples = i_samples + 1j * q_samples

print(f"Received {len(samples)} complex samples")

# ปิดการเชื่อมต่อ
sock.close()
```

## ข้อดี rtl\_tcp:

- Remote access ผ่าน network
- ใช้กับ FRP tunnel ได้
- เหมาะสำหรับ Google Colab

# LAB 1: ใช้งานผ่าน FRP Tunnel

## Architecture

```

[Colab/Client]
  ↓
[Internet]
  ↓
[FRP Server:600X]
  ↓
[FRP Client on RPI]
  ↓
[rtl_tcp:1234 on RPI]
  ↓
[RTL-SDR Hardware]

```

## การตั้งค่า FRP:

```

[[proxies]]
name = "piXX-tcp-1234"
type = "tcp"
localIP = "127.0.0.1"
localPort = 1234
remotePort = 60XX

```

## ✓ ทดสอบจาก Colab

```

# ใน Google Colab
import socket
import struct

FRP_SERVER = "xxx.xxx.xxx.xxx"
FRP_PORT = 60XX # remote port

# เชื่อมต่อผ่าน FRP
sock = socket.socket(
    socket.AF_INET,
    socket.SOCK_STREAM
)
sock.connect((FRP_SERVER, FRP_PORT))

# ตั้งค่า RTL-SDR ผ่าน network
freq = 185360000
cmd = struct.pack('>BI', 0x01, freq)
sock.send(cmd)

# รับ I/Q data จากกระยะไกล
data = sock.recv(8192)
print(f"Received {len(data)} bytes via FRP")

```

 **ผลลัพธ์:** สามารถควบคุม RTL-SDR จาก Colab ผ่าน FRP tunnel

## LAB 2: การรับสัญญาณ DAB+ พื้นฐาน (25 นาที)

### การติดตั้ง welle.io

```
# ติดตั้ง welle.io จาก package manager  
sudo apt install welle.io
```

### การตั้งค่า welle.io

- ☐ เปิดโปรแกรม welle.io
- ☐ เลือก Input Device: RTL-SDR
- ☐ ตั้งค่า Gain: Auto หรือ 20-30 dB

### ความถี่ DAB+ ในประเทศไทย (ตาม NBTC)

- ☐ **Channel 5C:** 178.352 MHz (Bangkok, Pattaya, Hua Hin)
- ☐ **Channel 6C:** 185.360 MHz (National Network)
- ☐ **Channel 7C:** 192.352 MHz (เชียงใหม่, ภาคใต้)
- ☐ **Channel 8C:** 199.360 MHz (Bangkok, Pattaya, Hua Hin)

### ขั้นตอนการ Scan

- ☐ เลือก "Band III" (174-230 MHz)
- ☐ กด "Automatic Scan"
- ☐ รอให้ scan เสร็จ (2-3 นาที)
- ☐ ตรวจสอบ Services ที่พบ

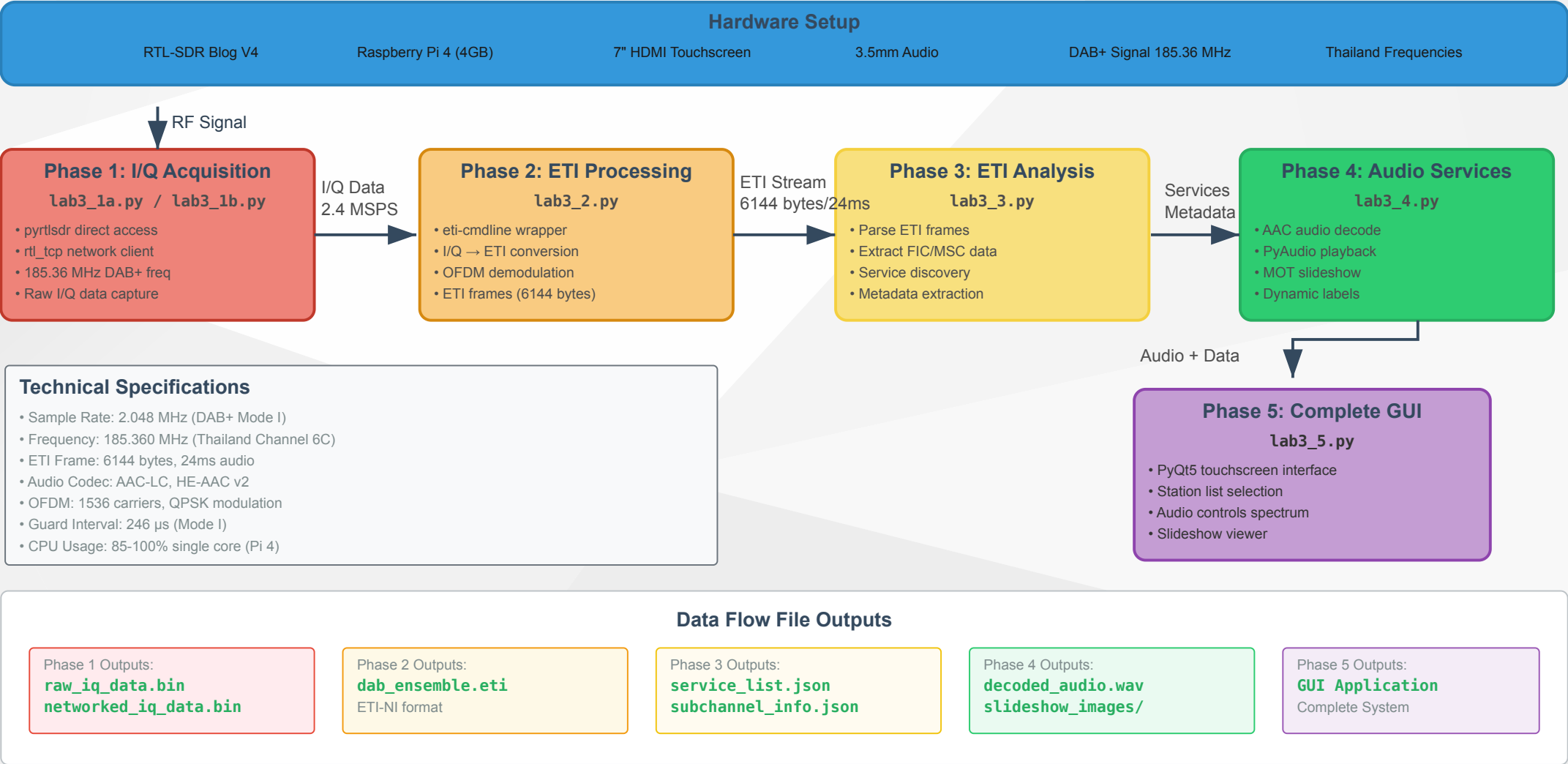
## LAB 3: Learning DAB+ with Raspberry Pi (Progressive Development)

 เวลารวม: 8-10 ชั่วโมง (แบ่งเป็น 5 phases)

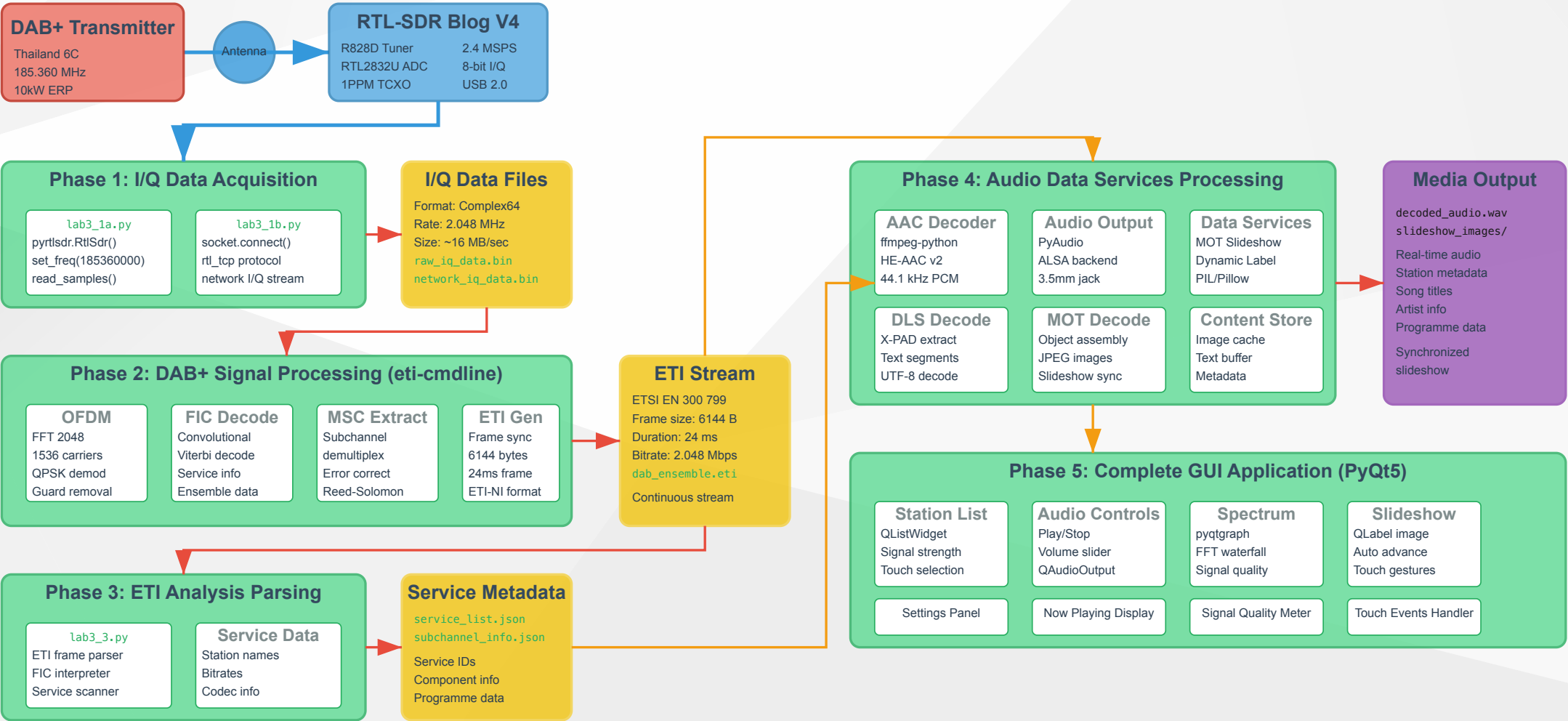
 ภาพรวม Lab 3: การพัฒนา DAB+ แบบ Step-by-Step

เป็นแล็บหลักที่สอนการสร้าง DAB+ receiver ตั้งแต่ต้นจนจบ





# Lab 3 รายละเอียด Technical Architecture



# Lab 3 Phase 1: RTL-SDR Data Acquisition

## Phase 1a: pyrtlsdr I/Q Capture

เรียนรู้: การใช้ pyrtlsdr library ควบคุม RTL-SDR เพื่อรับสัญญาณ I/Q จากความถี่ DAB+ ในประเทศไทย

```
# lab3_1a.py - RTL-SDR พื้นฐาน
import numpy as np
from rtlsdr import RtlSdr

class DABSignalCapture:
    def __init__(self):
        self.sdr = RtlSdr()
        self.dab_freq = 185.360e6 # Thailand
        self.sample_rate = 2.048e6

    def setup_sdr(self):
        # TODO: ตั้งค่า RTL-SDR parameters
        # TODO: center frequency, sample rate, gain
        pass

    def capture_iq_data(self, duration=5):
        # TODO: รับ I/Q samples
        # TODO: คำนวณ spectrum
        # TODO: บันทึก raw data
        pass
```

ผลลัพธ์: สามารถรับและวิเคราะห์ Spectrum ของสัญญาณ DAB+ ได้

## Phase 1b: rtl\_tcp Network Client

เรียนรู้: การสร้าง TCP client เชื่อมต่อ rtl\_tcp server เพื่อรับ I/Q data แบบ network streaming

```
# lab3_1b.py - Network รับ I/Q ผ่าน TCP
import socket
import struct
import numpy as np

class RTLTCPClient:
    def __init__(self, host='localhost', port=1234):
        self.host = host
        self.port = port
        self.socket = None

    def connect_to_server(self):
        # TODO: เชื่อมต่อ rtl_tcp server
        # TODO: ส่งคำสั่งตั้งค่า frequency
        pass

    def receive_iq_stream(self):
        # TODO: รับ I/Q data แบบ streaming
        # TODO: แปลง bytes เป็น complex samples
        pass
```

ผลลัพธ์: สามารถรับ I/Q data ผ่าน network และเปรียบเทียบ กับ USB direct connection 27

## Lab 3 Phase 2: ETI Stream Processing

### ETI-cmdline Integration

เรียนรู้: การใช้ eti-cmdline แปลงสัญญาณ I/Q จาก RTL-SDR เป็น ETI frames ขนาด 6144 bytes ต่อ frame

```
# lab3_2.py - DAB+ Signal → ETI Conversion
import subprocess
import threading
from queue import Queue

class ETIProcessor:
    def __init__(self):
        self.eti_cmdline_path = "/usr/local/bin/eti-cmdline"
        self.eti_queue = Queue()
        self.sync_status = False
    def start_eti_processing(self):
        # TODO: เริ่ม eti-cmdline subprocess
        # TODO: ส่ง I/Q data เข้า stdin
        # TODO: รับ ETI frames จาก stdout
        pass
    def parse_eti_frame(self, eti_data):
        # TODO: แยก ETI frame (6144 bytes)
        # TODO: ตรวจสอบ sync pattern
        # TODO: คำนวณ error rate
        pass
```


ผลลัพธ์ ได้ ETI stream ที่สามารถแยกเป็น FIC และ MSC data

### Signal Quality Monitoring

เรียนรู้: การติดตาม sync status และ error rate เพื่อประเมินคุณภาพสัญญาณ DAB+ แบบ real-time

```
class SignalQualityMonitor:
    def __init__(self):
        self.sync_rate = 0.0
        self.error_count = 0
        self.frame_count = 0
    def update_quality_metrics(self, frame):
        # TODO: คำนวณ sync success rate
        # TODO: ติดตาม error statistics และ TODO: แสดงผล real-time status
        pass
    def display_status(self):
        # TODO: แสดง sync status ทุก 1 วินาที
        # TODO: แสดง error rate percentage
        # TODO: แสดง signal strength
        pass
```

ผลลัพธ์: มี metrics แสดงคุณภาพสัญญาณและความเสถียรของการรับสัญญาณ

 **ผลลัพธ์ Phase 2:** ETI stream generation, sync monitoring, error tracking

## 🔍 Lab 3 Phase 3: ETI Analysis & Service Discovery

### FIC (Fast Information Channel) Parser

เรียนรู้: การถอดรหัส FIC data จาก ETI เพื่อดึงข้อมูล ensemble, services และ subchannels ผ่าน FIG decoding

```
# lab3_3.py - ETI Frame Analysis
class FICParser:
    def __init__(self):
        self.ensemble_info = {}
        self.services = {}
        self.subchannels = {}
    def parse_fic_data(self, fic_bytes):
        # TODO: แยก FIG (Fast Information Group)
        # TODO: ดึง ensemble information
        # TODO: แยก service list
        pass
    def decode_fig_types(self, fig_data):
        # TODO: FIG 0/0 - Basic ensemble info
        # TODO: FIG 0/1 - Basic subchannel info
        # TODO: FIG 0/2 - Basic service info
        # TODO: FIG 1/0 - Service labels
        pass
```

ผลลัพธ์: สามารถแยกข้อมูล ensemble และ service list จาก

DAB+ multiplex

Digital Audio Broadcasting Plus Learning Project

### Service Information Extraction

เรียนรู้: การจัดระเบียบข้อมูล services และ export เป็น JSON สำหรับใช้งานใน phases ต่อไป

```
class ServiceExtractor:
    def extract_services(self, fic_data):
        # TODO: สร้าง service database
        # TODO: จับคู่ service กับ subchannel
        # TODO: ดึง audio parameters
        pass
    def export_service_list(self):
        # TODO: บันทึกเป็น JSON format
        # TODO: ใส่ข้อมูล service labels
        # TODO: ระบุ audio bit rates
        return {
            "ensemble": "Thailand DAB+",
            "services": [
                {
                    "name": "Service 1",
                    "id": "0x1001",
                    "bitrate": 128,
                    "subchannel": 1
                }
            ]
        }
```

ผลลัพธ์: ไฟล์ service\_list.json และ subchannel\_info.json

## 🎵 Lab 3 Phase 4: Audio Processing & Playback

### AAC Audio Decoder

เรียนรู้: การแยก AAC audio frames จาก MSC data และ decode ด้วย ffmpeg + PyAudio สำหรับเล่นเสียงผ่าน 3.5mm jack

```
# lab3_4.py - DAB+ Audio Processing
import subprocess
import pyaudio
from queue import Queue
import threading

class DABAudioDecoder:
    def __init__(self):
        self.audio_queue = Queue()
        self.pyaudio_instance = pyaudio.PyAudio()
        self.stream = None
        self.current_service = None

    def extract_audio_frames(self, msc_data):
        # TODO: แยก audio super frames
        # TODO: ส่งข้อมูลไป ffmpeg decode
        # TODO: ได้ raw PCM audio
        pass

    def setup_audio_output(self):
        # TODO: ตั้งค่า PyAudio stream
        # TODO: รองรับ 3.5mm jack output
        # TODO: ความถี่, Volume Level
        pass
```

### Dynamic Label & Slideshow

เรียนรู้: การถอดรหัส DLS (song/artist info) และ MOT slideshow จาก PAD data เพื่อแสดงข้อมูลเพลงและรูปภาพ

```
class DABMetadataProcessor:
    def __init__(self):
        self.current_dls = ""
        self.slideshow_images = []

    def process_dls_data(self, dls_bytes):
        # TODO: แยก Dynamic Label Segment
        # TODO: รวม segments เป็น text
        # TODO: แสดง song title, artist
        pass

    def process_mot_slideshow(self, mot_data):
        # TODO: แยก MOT (Multimedia Object Transfer)
        # TODO: ประกอบ JPEG/PNG images
        # TODO: บันทึกรูปภาพ slideshow
        pass

    def display_now_playing(self):
        # TODO: แสดง current track info
        # TODO: แสดง slideshow image
        pass
```

ผลลัพธ์: แสดงชื่อเพลง, ศิลปิน และ slideshow images แบบ real-time



## Lab 3 Phase 5: Complete GUI Application

### Touch-Optimized GUI

เรียนรู้: การสร้าง PyQt5 GUI สำหรับหน้าจอสัมผัส 7" พร้อม spectrum analyzer และ dark theme

```
# lab3_5.py - PyQt5 Complete Application
from PyQt5.QtWidgets import *
from PyQt5.QtCore import *
import pyqtgraph as pg

class DABPlusGUI(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setFixedSize(800, 480) # 7" screen
        self.setup_dark_theme()
        self.setup_main_interface()

    def setup_main_interface(self):
        # TODO: สร้าง tabbed interface
        # TODO: Spectrum tab, Services tab, Player tab
        # TODO: Settings tab สำหรับ RTL-SDR
        pass

    def setup_spectrum_analyzer(self):
        # TODO: Real-time spectrum plot
        # TODO: Waterfall display
        # TODO: Signal quality indicators
        pass
```

### Integrated DAB+ Player

เรียนรู้: การรวม Phase 1-4 เข้าด้วยกันเป็น complete pipeline จาก I/Q → Audio พร้อม GUI controls

```
class DABPlayerController:
    def __init__(self, parent):
        self.parent = parent
        self.current_phase = 1
        self.all_components = {}

    def integrate_all_phases(self):
        # TODO: รวม Phase 1-4 เข้าด้วยกัน
        # TODO: I/Q → ETI → Services → Audio
        # TODO: Real-time processing pipeline
        pass

    def create_player_ui(self):
        # TODO: Service selection list
        # TODO: Audio controls (play/stop/volume)
        # TODO: DLS text display
        # TODO: Slideshow viewer
        # TODO: Signal quality meters
        pass
```

## Lab 3: Google Colab Version (สำหรับเรียนรู้ทางไกล)

### ทำไมต้องมี Colab Version?

#### ปัญหา:

- ไม่มี Raspberry Pi
- ไม่มี RTL-SDR
- อยากเรียนรู้ทางไกล
- ทดลองโค้ดก่อนติดตั้ง

#### วิธีแก้:

- ใช้ FRP tunnel เข้าถึง RPI
- เขียนโค้ดใน Google Colab
- ทดสอบ algorithm ได้ทันที
- Visualization พร้อมใช้

### Architecture

```
[Google Colab Notebook]
  ↓ Python code + visualization
[rtl_tcp client via FRP]
  ↓ network connection
[FRP Server:600X]
  ↓ internet tunnel
[RPI + FRP Client]
  ↓ local connection
[rtl_tcp:1234]
  ↓ USB
[RTL-SDR Hardware]
```

#### ข้อดี:

- เรียนได้ทุกที่
- ไม่ต้องติดตั้งอะไร
- Share notebook ง่าย



# Lab 3 Colab: Phase 1 - I/Q Acquisition

## Lab3\_Phase1\_IQ\_Acquisition\_Colab.ipynb

### Features:

- RTLTCPCClient class ครบถ้วน
- FRP connection testing
- I/Q sample acquisition
- Real-time monitoring

```
# Cell 1: Setup
!pip install numpy matplotlib scipy
```

```
# Cell 2: RTLTCPCClient
class RTLTCPCClient:
    def __init__(self, host, port):
        self.host = host
        self.port = port
        self.sock = None

    def connect(self):
        self.sock = socket.socket(
            socket.AF_INET,
            socket.SOCK_STREAM
        )
        self.sock.connect(
            (self.host, self.port)
        )
```

## Spectrum Analysis

```
# Cell 3: Acquire Samples
client = RTLTCPCClient(
    'frp_server_ip',
    60XX
)
client.connect()
client.set_frequency(185360000)
client.set_sample_rate(2048000)

samples = client.read_samples(1024*1024)

# Cell 4: FFT Analysis
fft_data = np.fft.fft(samples)
freqs = np.fft.fftfreq(
    len(samples),
    1/2048000
)
psd = 20*np.log10(np.abs(fft_data))

plt.plot(freqs/1e6, psd)
plt.xlabel('Frequency (MHz)')
plt.ylabel('Power (dB)')
plt.show()
```

ผลลัพธ์: spectrum plot, I/Q data

# Lab 3 Colab: Phase 2 - ETI Processing

## Lab3\_Phase2\_ETI\_Processing\_Colab.ipynb

### Features:

- ETIFrameParser class
- Simulated ETI frames
- Sync pattern detection
- FIC data extraction

```
# Cell 1: ETI Frame Parser
class ETIFrameParser:
    FRAME_SIZE = 6144
    FSYNC_PATTERN = 0x073AB6

    def parse_header(self, frame_bytes):
        # Parse ERR, FSYNC, LIDATA
        fsync = (frame_bytes[4] << 16) | \
                (frame_bytes[5] << 8) | \
                frame_bytes[6]

        return {
            'fsync_valid':
                fsync == self.FSYNC_PATTERN,
            'frame_bytes[7]'
        }
```

## Simulated ETI for Learning

```
# Cell 2: Generate Simulated ETI
def generate_simulated_eti():
    frame = bytearray(6144)

    # ERR (byte 0-3)
    frame[0:4] = b'\x00\x00\x00\x00'

    # FSYNC (byte 4-6)
    frame[4] = 0x07
    frame[5] = 0x3A
    frame[6] = 0xB6

    # FC (byte 7)
    frame[7] = 0x00

    # FIC (96 bytes)
    # MSC (rest of frame)

    return bytes(frame)

# Test parsing
parser = ETIFrameParser()
frame = generate_simulated_eti()
header = parser.parse_header(frame)
print(f"Valid: {header['fsync_valid']}")
```

## Lab 3 Colab: ข้อจำกัดและข้อดี

### ข้อจำกัด

#### Network:

- Latency จาก internet
- Bandwidth สำหรับ I/Q streaming
- FRP tunnel stability

#### Processing:

- ไม่มี etl-cmdline บน Colab
- ไม่มี native DAB+ tools
- ต้องใช้ simulated data บางส่วน

#### Hardware:

- ไม่สามารถเข้าถึง GPIO
- ไม่มี audio output ตรง
- ต้องพึ่งพา RPI ระยะไกล

### ข้อดี

#### Learning:

- เรียนรู้ concepts ได้ดี
- ทดสอบ algorithm ง่าย
- Visualization สวยงาม
- Share code ได้ทันที

#### Development:

- Prototype รวดเร็ว
- Debug ง่าย
- Version control ผ่าน Colab
- Collaborate ได้

#### Accessibility:

- เรียนได้ทุกที่ทุกเวลา

## Lab 3 Trap Exercises (แต่ละ Phase)

### Phase 1 Traps:

#### Trap 1.1: I/Q Data Analysis

วิเคราะห์ spectrum ที่ได้และระบุ DAB+ signal peaks

#### Trap 1.2: Network Protocol

อธิบายความแตกต่างระหว่าง USB และ TCP streaming

#### Trap 1.3: Sample Rate Optimization

ทดลองเปลี่ยน sample rate และประเมินผลกระทบ

### Phase 4-5 Traps:

#### Trap 4.1: Audio Pipeline

วิเคราะห์ audio processing chain และ latency

#### Trap 4.2: Metadata Parsing

### Phase 2-3 Traps:

#### Trap 2.1: ETI Frame Structure

วิเคราะห์ ETI frame header และอธิบาย fields

#### Trap 3.1: FIG Decoding

ถอดรหัส FIG manually และเปรียบเทียบกับ parser

#### Trap 3.2: Service Discovery

อธิบายขั้นตอนการค้นหา services ใน ensemble

### Integration Traps:

#### Trap 5.2: End-to-End Testing

ทดสอบ complete pipeline จาก RF → Audio

#### Trap 5.3: Error Recovery

## Trap Exercises LAB 3:

### Trap 3.1: Spectrum Data Analysis

วิเคราะห์ CSV file จาก rtl\_power และหา peak ที่แต่ละความถี่ DAB+

### Trap 3.2: Raw Data Capture Understanding

อธิบายขนาดไฟล์ที่ได้จาก rtl\_sdr และความสัมพันธ์กับ sample rate

### Trap 3.3: PPM Error Interpretation

อธิบายความหมายของ PPM error และผลกระทบต่อการรับสัญญาณ

## LAB 4: ETISnoop - การวิเคราะห์ DAB+ Stream (15 นาที)

### การติดตั้ง ETISnoop

#### 1. ติดตั้ง Dependencies

```
sudo apt install build-essential cmake libfftw3-dev librtlsdr-dev
```

#### 2. ดาวน์โหลดและ Compile ETISnoop

```
# Clone repository
git clone https://github.com/JvanKatwijk/eti-snoop
cd eti-snoop

# Build
mkdir build && cd build
cmake ..
make
```

#### 3. การใช้งาน ETISnoop

```
## รัน ETISnoop กับ RTL-SDR
./eti-snoop -D RTL_SDR -C 6C
```

## สิ่งที่ตรวจสอบใน ETISnoop

### 1. Ensemble Information

- ☐ Ensemble Label
- ☐ Country Code
- ☐ ECC (Extended Country Code)
- ☐ Ensemble ID

### 2. Service Information

- ☐ Service Labels
- ☐ Service IDs
- ☐ Program Types
- ☐ Bit Rates

### 3. Technical Parameters

- ☐ Frame Error Rate
- ☐ Signal Quality
- ☐ Frequency Offset
- ☐ Time/Date Information

## Trap Exercises LAB 4:

### Trap 4.1: Build Troubleshooting

ถ้า make ล้มเหลว ให้วิเคราะห์ error message และแก้ไข

### Trap 4.2: Log Data Interpretation

จาก log ให้ระบุ Service ID, Bit Rate และ Audio Codec ของแต่ละ service



# LAB 5: การตรวจสอบและวิเคราะห์สัญญาณชั้นสูง (10 นาที)

## รายงานผลการทดสอบรวม

ตาราง checklist รวม:

ความถี่ (MHz)	welle.io	Command Line	ETISnoop	Signal Quality
178.352 (5C)				
185.360 (6C)				
192.352 (7C)				
199.360 (8C)				

## การเปรียบเทียบ Tools

เครื่องมือ	ข้อดี	ข้อเสีย	เหมาะสำหรับ
welle.io	GUI ใช้งานง่าย, Real-time	Limited analysis	การฟังทั่วไป
Command Line	Fast, Scriptable	ต้องมีความรู้	การทดสอบเทคนิค
ETISnoop	Deep analysis	ซับซ้อน	การวิเคราะห์ขั้นสูง

## ข้อมูลที่ได้จากการทดสอบ

- [] Coverage Area: พื้นที่ที่รับสัญญาณได้
- [] Service Availability: บริการที่มีในแต่ละ ensemble
- [] Technical Quality: คุณภาพทางเทคนิค
- [] Comparison Data: เปรียบเทียบระหว่าง tools

## Automation Scripts

```
#!/bin/bash
echo "=== DAB+ Automated Test ==="
echo "1. Testing Hardware ..."
rtl_test -t

echo "2. Scanning Spectrum ..."
rtl_power -f 174M:230M:8k -g 30 -i 5 scan_$(date +%Y%m%d_%H%M).csv

echo "3. Testing Each Channel ..."
for freq in 178352000 185360000 192352000 199360000; do
    echo "Testing $freq Hz ..."
    timeout 30 rtl_sdr -f $freq -s 2048000 -n 1024000 test_$freq.raw
done

echo "Test Complete!"
```

## Trap Exercises LAB 5:

### Trap 5.1: Tool Performance Comparison

สร้างตารางเปรียบเทียบ CPU usage, Memory usage และ Accuracy ของแต่ละ tool

### Trap 5.2: Automation Script Development

ปรับปรุง automation script ให้เหมาะสมกับสภาพแวดล้อมของคุณ

# การแก้ไขปัญหาและ Troubleshooting

## ปัญหาที่พบบ่อยและการแก้ไข

### 1. RTL-SDR ไม่ทำงาน

- [] ตรวจสอบ USB connection
- [] ใช้คำสั่ง `lsusb` ดู device
- [] รีบูตระบบ
- [] ตรวจสอบ driver installation

### 2. Command Line Tools Error

- [] ตรวจสอบ PATH environment
- [] ใช้ `which rtl_test` หา location
- [] Re-install rtl-sdr package

### 3. ETISnoop ไม่ compile

- [] ติดตั้ง missing dependencies
- [] ใช้ `sudo apt install build-essential`
- [] ตรวจสอบ CMake version

### 4. ไม่พบสัญญาณ DAB+

- [] ตรวจสอบตำแหน่งเสาอากาศ (วางแนวตั้ง)
- [] ลองย้ายไปใกล้หน้าต่าง
- [] เปลี่ยน gain setting ใน tools ต่างๆ
- [] ทดสอบใน welle.io ก่อน

## 🚫 ปัญหาที่พบบ่อยและการแก้ไข

### 5. เสียงไม่ดี/กระตุก

- [ ] ตรวจสอบ CPU usage ( `top` command)
- [ ] ลด sample rate ถ้าจำเป็น
- [ ] ปิดโปรแกรมอื่นที่ไม่จำเป็น



### เคล็ดลับเพิ่มเติม

- เสออากาศ: ใช้แบบ vertical polarization
- **Gain Control:** เริ่มจาก Auto แล้วปรับตามต้องการ
- **Location:** ทดสอบในพื้นที่ที่ระบุใน NBTC frequency plan

### 🎯 ผลลัพธ์ที่คาดหวังหลังจบ Labs 1-5:

- [ ] ติดตั้งและใช้งาน RTL-SDR V4 ได้
- [ ] รับฟังสัญญาณ DAB+ ในประเทศไทยได้
- [ ] ใช้ Command Line Tools สำหรับการทดสอบ
- [ ] วิเคราะห์ DAB+ stream ด้วย ETISnoop
- [ ] เข้าใจพารามิเตอร์คุณภาพสัญญาณ
- [ ] แก้ไขปัญหาเบื้องต้นได้
- [ ] เปรียบเทียบประสิทธิภาพของ tools ต่างๆ


## LAB 6: สร้าง DAB+ Signal Analyzer (Advanced)

### Advanced Analysis

- OFDM Structure วิเคราะห์
- SNR, MER, BER คุณภาพสัญญาณ
- Constellation Diagram I/Q แสดงผล
- Waterfall Plot spectrum ตามเวลา

### Visualization & Reports

- Real-time Metrics LCD displays
- Professional Reports PDF generation
- Data Export CSV, JSON formats
- Advanced Matplotlib integration


 ผลลัพธ์: Professional DAB+ Signal Analyzer

## เทคโนโลยี DAB+ เบื้องลึก

### DAB+ Signal Structure

DAB+ Frame (96ms)

- Null Symbol (sync)
- PRS (Phase Reference)
- FIC (Fast Info Channel)
- MSC (Main Service Channel)
  - Audio Services
  - Data Services

 ความเข้าใจ: จากพื้นฐานไปถึงระดับ Professional RF Engineer

### OFDM Technology

- 2048 Carriers ใช้พร้อมกัน
- Guard Interval ป้องกัน multipath
- DQPSK Modulation ทนต่อ noise
- Error Correction Reed-Solomon

## การพัฒนาด้วย PyQt5

### Touch-Friendly GUI

```
# ปุ่มขนาดใหญ่
button.setMinimumSize(120, 60)

# Font สำหรับหน้าจอ 7"
font = QFont()
font.setPointSize(14)

# CSS Styling
button.setStyleSheet("""
    QPushButton {
        border-radius: 8px;
        background: #3498db;
        color: white;
        font-weight: bold;
    }
""")
```

### Signals & Slots

```
# Built-in signals
button.clicked.connect(self.on_click)
slider.valueChanged.connect(self.update_value)

# Custom signals
class MyWidget(QThread):
    data_ready = pyqtSignal(dict)

    def emit_data(self):
        self.data_ready.emit({'value': 42})
```





## การประมวลผลสัญญาณ (DSP)



### NumPy & SciPy

```
import numpy as np
from scipy import signal

# FFT Analysis
fft_result = np.fft.fft(iq_samples)
frequencies = np.fft.fftfreq(len(samples), 1/sample_rate)

# Power Spectrum
power_db = 20 * np.log10(np.abs(fft_result))

# Signal Quality
snr = signal_power / noise_power
```



### Real-time Visualization

```
import matplotlib.pyplot as plt
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgi

class SpectrumPlot(FigureCanvasQTAgi):
    def update_spectrum(self, freq, power):
        self.axes.clear()
        self.axes.plot(freq/1e6, power)
        self.draw()
```

## การแก้ไขปัญหาทั่วไป

### RTL-SDR Issues

```
# ตรวจสอบการเชื่อมต่อ
lsusb | grep RTL

# แก้ driver conflicts
sudo modprobe -r dvb_usb_rtl28xxu
lsmod | grep dvb

# Permissions
sudo usermod -a -G plugdev $USER
```

 **เคล็ดลับ:** อ่านคู่มือแต่ละ LAB ก่อนเริ่มได้ด

### Audio Issues

```
# ตั้งค่าเสียงออก 3.5mm
sudo raspi-config
# Advanced Options > Audio > Force 3.5mm

# ทดสอบเสียง
speaker-test -t wav -c 2

# PulseAudio restart
pulseaudio -k
```

## เส้นทางการเรียนรู้

### ระดับเริ่มต้น (Tools-based)

1. **Lab 0:** PyQt5 พื้นฐาน
2. **Lab 1:** RTL-SDR เบื้องต้น (20 นาที)
3. **Lab 2:** DAB+ รับฟัง (25 นาที)
4. **Lab 3:** Command Line Tools (15 นาที)
5. **Lab 4:** ETISnoop Analysis (15 นาที)

 เวลา: ~2 ชั่วโมง

 เป้าหมาย: เรียนรู้ DAB+ tools พื้นฐาน


### หลักสูตรหลัก + การบ้าน


หลักสูตรหลัก (ผู้สอน 14-15 ชั่วโมง)

1. **Lab 1:** RTL-SDR Setup (2-3 ชม.)
2. **Lab 2:** welle.io Integration (3-4 ชม.)
3. **Lab 3:** DAB+ RTL-SDR Pipeline (5 phases, 9-8 ชม.)

การบ้าน (นักเรียนทำเอง 9-12 ชั่วโมง)

4. **Lab 4:** Station Scanner (3-4 ชม.)
5. **Lab 5:** Program Recorder (3-4 ชม.)
6. **Lab 6:** Signal Analyzer (3-4 ชม.)

 เวลา: ~23-27 ชั่วโมง

 เป้าหมาย: Professional DAB+ Applications

## ผลลัพธ์ที่ได้รับ

### แอปพลิเคชันที่สร้างได้

#### เริ่มต้น (Tools-based):

- Basic DAB+ Reception
- Command Line Analysis

#### ขั้นสูง (Development-based):

- RTL-SDR Hardware Controller
- DAB+ Station Scanner
- Program Recorder with Scheduling
- Real-time Spectrum Analyzer
- Professional Signal Analyzer

### Trap Exercises: ทดสอบความเข้าใจลึก

### ความรู้ที่ได้รับ

#### เริ่มต้น:

- DAB+ Technology พื้นฐาน
- Command Line Tools

#### ขั้นสูง:

- Python & PyQt5 GUI Development
- RF & DSP Signal Processing
- Real-time Audio Processing
- Database & Threading
- OFDM & Machine Learning
- Professional RF Analysis

## แหล่งข้อมูลเพิ่มเติม

### Documentation

- [welle.io GitHub](#)
- [RTL-SDR.com](#)
- [PyQt5 Docs](#)
- [DAB+ Standard \(ETSI\)](#)

### Learning Resources

- GNU Radio สำหรับ SDR ขั้นสูง
- DSP Course Signal Processing
- RF Engineering คลื่นวิทยุ
- Embedded Linux สำหรับ IoT

## Next Steps - ขั้นตอนต่อไป

### พัฒนาเพิ่มเติม

- **Web Interface** ควบคุมผ่าน browser
- **Mobile App** Android/iOS remote
- **Cloud Integration** upload recordings
- **AI/ML** automatic classification

 จาก Hobby Project → Professional Career

### Career Paths

- **RF Engineer** วิศวกรคลื่นวิทยุ
- **SDR Developer** Software Defined Radio
- **IoT Developer** Internet of Things
- **Embedded Systems** ระบบฝังตัว




## ขอขอบคุณและสนับสนุน

### ขอขอบคุณที่เรียนรู้กับเรา!

**DAB+ Labs** เป็นโครงการ Open Source  
สำหรับการศึกษาและพัฒนาเทคโนโลยี เพื่อให้ความรู้เกี่ยวกับ  
ระบบวิทยุดิจิทัล DAB+ และการจัดทำเครื่องรับ DAB+ ด้วย  
ตนเอง จัดการอบรมโดย กสทช.

### ขอให้สนุกกับการเรียนรู้!

### ติดต่อและสนับสนุน

-  **Issues:** GitHub Issues
-  **Email:** project contact
-  **Community:** Forum discussion
-  **Star:** ถ้าชอบโครงการ


MIT License - ใช้ได้อย่างอิสระ

## สรุป: การเดินทาง DAB+ Learning

### สิ่งที่เราได้เรียนรู้:

- DAB+ Technology จาก 0 ถึง Hero
- Python & PyQt5 สำหรับ Professional GUI
- RTL-SDR & RF Engineering ด้วยมือ
- Project Development จาก Idea ถึง Working App

### Achievement Unlocked:

 DAB+ Expert |  Python GUI Master |  RF Engineer |  Maker



🎉 พร้อมสำหรับการผจญภัยใหม่แล้ว! 🎉