Department of Electrical and Computer Engineering

ECE 350 — Communications Theory and Systems I

Universal SDR Receiver: Decoding ACARS, IoT, and FT8 Signals with Raspberry Pi

Project No:	01
Report Submitted:	October 01, 2025
Section:	A01
Instructor:	Peter Driessen
Authors:	Colette Reimer (V00929031) David Emelu (V01025755 Gurshan Saran (V00993855) Yunu Choe (V01008760)

Project Report

Contents

List of Figures

List of Tables

1 Introduction

This project demonstrates the design and implementation of a standalone Software Defined Radio (SDR) receiver system built on a Raspberry Pi 4 with an RTL-SDR Blog V4 dongle. The goal is to reliably capture and decode three classes of real-world signals:

- ADS-B aircraft telemetry (1090 MHz)
- IoT/ISM band devices (315/390/433/868/915 MHz)
- Shortwave FT8 digital signals (via WSJT-X)

The project is aligned with the course requirement to build a receiver device capable of decoding aircraft, IoT, and shortwave digital modes, with data verification against external references.

2 Objectives

The main objectives of these projects are:

1. To set up a Raspberry Pi-based SDR platform with RTL-SDR Blog V4 hardware and appropriate software tools.
2. To configure and demonstrate ADS-B decoding using dump1090-fa and validate aircraft signals against online flight-tracking services.
3. To capture and decode IoT/ISM band device signals using rtl_433, log data in JSON format, and identify device protocols.
4. To implement FT8 shortwave decoding using WSJT-X (via WebSDR audio or local RF front-end) and validate decoded amateur radio callsigns via QRZ.com.
5. To document the hardware, software, and configuration process so that another student can replicate the receiver setup.

3 Team Contributions

Team Member	**Contributions**
David Emelu	Project coordination and documentation preparation.
Yunu Choe	Document review and quality assurance.
Colette Reimer	Voice-over narration for the project video.
Gurshan Saran	Video editing and post-production.

Table 1: Team member roles and contributions.

4 Hardware & Software Setup

4.1 Hardware Components

Component Name	**Used For**
Pi-Top 4 (with Raspberry Pi 4 inside)	Main control unit; runs the RTL-SDR and rtl_433 software, provides GUI and access point for users. ***See Figure 1***
RTL-SDR Blog V4 Dongle	Captures and receives RF signals for ADS-B, IoT/ISM, and FT8.
Dipole Antenna	General-purpose antenna used for capturing ADS-B, IoT, and shortwave signals.
32GB Micro-SD Card	Serves as the memory and boot drive for Raspberry Pi OS, storing software and logs.
5V, 3A Power Supply (USB-C)	Provides stable power to the Pi-Top 4 and attached SDR hardware.

Table 2: Hardware Components Used

(a): RTL-SDR Blog V4 dongle

(b): Exploded view of Pi-Top 4 with Raspberry Pi 4

(c): Closed Pi-Top 4 case

(d): Dipole antenna

Figure 1: Hardware components used in the SDR receiver project: RTL-SDR Blog V4 dongle, Pi-Top 4 (exploded and closed), and dipole antenna.

Software/Tool	**Description / Used For**
Raspberry Pi OS (64-bit)	The base operating system for the Raspberry Pi; provides Linux environment, package management, and supports all SDR applications.
RTL-SDR Blog V4 Drivers and Tools	Custom drivers required for the RTL-SDR Blog V4 dongle; resolves issues such as "PLL not locked" and enables stable signal reception. Includes utilities like `rtl_test` and `rtl_biast`.
dump1090-fa	Specialized software for decoding ADS-B signals (1090 MHz) transmitted by aircraft. Provides real-time aircraft positions, flight IDs, and can display results via a web interface.
rtl_433	Multi-protocol decoder for IoT/ISM devices operating at 315/390/433/868/915 MHz. Captures sensor data (e.g., weather stations, door sensors) and exports it in JSON format for logging or integration with smart systems.
GQRX + WSJT-X	**GQRX**: General-purpose SDR receiver with GUI for tuning, filtering, and demodulating signals. **WSJT-X**: Digital mode decoder used for amateur radio modes such as FT8, enabling weak-signal communication decoding and callsign extraction.
Chrony	Lightweight time synchronization tool. Essential for FT8 decoding since the protocol requires timing accuracy within ±1 second. Ensures Raspberry Pi clock stability when connected to the network.

Table 3: Software Environment

5 System Setup and Implementation

This section outlines the detailed steps followed to configure the Raspberry Pi 4 (inside the Pi-Top 4) with the RTL-SDR Blog V4 dongle for decoding ADS-B, IoT/ISM, and FT8 signals. Each step includes commands and configuration notes.

5.1 Step 1: OS Preparation

Update and upgrade the Raspberry Pi operating system:

```
sudo apt update
sudo apt full-upgrade -y
sudo reboot
```

Install required tools and libraries:

```
sudo apt install -y git cmake build-essential libusb-1.0-0-dev \
pkg-config rtl-sdr lighttpd netcat-openbsd sox pulseaudio pavucontrol
```

Note: The `rtl-sdr` package is installed for utilities like `rtl_test` and `rtl_biast`. It will be replaced later with the RTL-SDR Blog V4 build.

5.2 Step 2: Install RTL-SDR Blog V4 Drivers (Critical)

Prevent the kernel DVB driver from attaching to the dongle:

```
echo 'blacklist dvb_usb_rtl28xxu' | sudo tee /etc/modprobe.d/blacklist-rtl.conf
sudo reboot
```

After reboot, remove old libraries and build the RTL-SDR Blog fork:

```
sudo apt purge -y librtlsdr0 || true
sudo apt autoremove -y
cd ~
rm -rf rtl-sdr-blog
git clone https://github.com/rtlsdrblog/rtl-sdr-blog.git
cd rtl-sdr-blog && mkdir build && cd build
cmake -DDETACH_KERNEL_DRIVER=ON -DINSTALL_UDEV_RULES=ON ..
make -j"$(nproc)"
sudo make install
sudo ldconfig
sudo udevadm control --reload-rules && sudo udevadm trigger
```

Prioritize the new library:

```
echo '/usr/local/lib' | sudo tee /etc/ld.so.conf.d/00-local-rtlsdr.conf
sudo ldconfig
```

Verify installation:

```
ldconfig -p | grep -i rtlsdr
rtl_test -t
```

Expected: No "PLL not locked" errors. If issues persist, move the dongle to a USB 2.0 port, use a stable power supply, and shorten USB cables.

5.3 Step 3: ADS-B Decoding with dump1090-fa

Install dump1090-fa (FlightAware build)

```
wget https://www.flightaware.com/adsb/piaware/files/packages/pool/piaware/f/flightaware-apt-
    repository/flightaware-apt-repository_1.2_all.deb
sudo dpkg -i flightaware-apt-repository_1.2_all.deb
sudo apt update
sudo apt install -y dump1090-fa lighttpd
```

Minimal Configuration

Edit `/etc/default/dump1090-fa`:

```
RECEIVER_OPTIONS="--device-index 0 --gain -10"
DECODER_OPTIONS="--max-range 360"
NET_OPTIONS="--net"
JSON_OPTIONS="--json-location-accuracy 2"
```

Ensure systemd uses the Blog V4 library:

```
sudo mkdir -p /etc/systemd/system/dump1090-fa.service.d
printf "[Service]\nEnvironment=LD_LIBRARY_PATH=/usr/local/lib\n" | \
sudo tee /etc/systemd/system/dump1090-fa.service.d/override.conf
sudo systemctl daemon-reload
sudo systemctl enable --now dump1090-fa
sudo systemctl status dump1090-fa --no-pager
```

Add Coordinates

```
sudo sed -i 's/DECODER_OPTIONS="/DECODER_OPTIONS="--lat 48.428 --lon -123.365 /' \
/etc/default/dump1090-fa
sudo systemctl restart dump1090-fa
```

Viewing Options

- Interactive table:

```
sudo systemctl stop dump1090-fa
LD_LIBRARY_PATH=/usr/local/lib /usr/bin/dump1090-fa --device-index 0 --gain -10 --
    interactive
```

- Attach viewer:

```
view1090-fa --net --net-beast localhost:30005
```

- Text feed:

```
nc localhost 30003
\end{verbatim}
```

- Enable Bias-T (if powering LNA):

```
sudo systemctl stop dump1090-fa
rtl_biast -b 1
sudo systemctl start dump1090-fa
```

5.4 Step 4: IoT / ISM Band Capture with rtl_433

Install rtl_433:

```
sudo apt install -y rtl-433
```

Quick test (433.92 MHz):

```
rtl_433 -f 433.92M -M time:iso -M level -F json
```

Multi-band scan:

```
rtl_433 -f 315M -f 390M -f 433.92M -f 868.3M -f 915M -M time:iso -M level -F json
```

Log output to file:

```
rtl_433 -f 433.92M -M time:iso -M level -F json:/var/log/rtl_433.json
```

Add log rotation rule (`/etc/logrotate.d/rtl_433`):

```
/var/log/rtl_433.json {
  rotate 7
  daily
  compress
  missingok
  notifempty
}
```

Run as a systemd service:

```
sudo systemctl daemon-reload
sudo systemctl enable --now rtl_433
sudo journalctl -u rtl_433 -f
```

5.5 Step 5: FT8 Shortwave Decoding

Method A: WebSDR Audio to WSJT-X

Install WSJT-X:

```
sudo apt install -y wsjtx
```

Set up PulseAudio loopback:

```
pulseaudio --start 2>/dev/null || true
pactl load-module module-null-sink sink_name=LoopbackSink \
sink_properties=device.description=LoopbackSink
pactl load-module module-loopback source=alsa_input.platform-bcm2835_audio.analog-stereo \
sink=LoopbackSink latency_msec=1 2>/dev/null || true
```

Configure browser output to LoopbackSink and select **Monitor of LoopbackSink** as input in WSJT-X.

Method B: Local RF via GQRX and WSJT-X

Install GQRX:

```
sudo apt install -y gqrx-sdr
```

In GQRX:

- Select RTL-SDR device with HF front-end.
- Mode: USB, filter: 2.5–3 kHz, sample rate: 2 Msps.
- Route output to LoopbackSink using `pavucontrol`.

Time Synchronization (Critical for FT8)

```
sudo apt install -y chrony
sudo systemctl enable --now chrony
chronyc tracking
```

5.6 Step 6: Antenna, Gain, and RF Hygiene

- Gain: start with `-gain -10` (auto). Test fixed gains (e.g., 32.8, 38.6).
- Cables: use short USB/RF cables; avoid unpowered hubs; prefer USB 2.0 on Pi.
- Power: verify with `vcgencmd get_throttled` (expect `0x0`).
- Filtering/LNA: ADS-B benefits from 1090 SAW filter + LNA.
- Antenna placement: outdoors and elevated performs best; IoT often requires proximity.

5.7 Step 7: Troubleshooting

- **PLL not locked:** Wrong library or weak USB power. Reinstall Blog V4 driver, switch to USB2, or upgrade PSU.
- **Device busy:** DVB driver attached or another SDR app running. Blacklist DVB, stop services, rerun `rtl_test -t`.
- **dump1090-fa service fails:** Wrong library in systemd. Add `LD_LIBRARY_PATH` override.
- **FT8 not decoding:** Clock not synced, wrong audio device, or incorrect bandwidth. Enable Chrony, check WSJT-X input, set USB with 2.5–3 kHz bandwidth.

5.8 Step 8: Data and Service Hygiene

- Monitor logs with:

```
sudo journalctl -u dump1090-fa -f
sudo journalctl -u rtl_433 -f
```

- Store JSON logs under `/var/log` with logrotate.
- Secure web UI by limiting access to LAN or adding authentication.

5.9 Step 9: Quick Command Reference

Driver and sanity check:

```
echo 'blacklist dvb_usb_rtl28xxu' | sudo tee /etc/modprobe.d/blacklist-rtl.conf
sudo reboot
sudo apt purge -y librtlsdr0 || true
cd ~ && rm -rf rtl-sdr-blog && git clone https://github.com/rtlsdrblog/rtl-sdr-blog.git
cd rtl-sdr-blog && mkdir build && cd build
cmake -DDETACH_KERNEL_DRIVER=ON -DINSTALL_UDEV_RULES=ON .. && make -j"$(nproc)" && sudo make
    install
sudo ldconfig
ldconfig -p | grep -i rtlsdr
rtl_test -t
```

Install dump1090-fa:

```
wget https://www.flightaware.com/adsb/piaware/files/packages/pool/piaware/f/flightaware-apt-
    repository/flightaware-apt-repository_1.2_all.deb
sudo dpkg -i flightaware-apt-repository_1.2_all.deb
sudo apt update && sudo apt install -y dump1090-fa lighttpd
```

IoT quick scan:

```
sudo apt install -y rtl-433
rtl_433 -f 315M -f 390M -f 433.92M -f 868.3M -f 915M -M time:iso -M level -F json
```

FT8 essentials:

```
sudo apt install -y wsjtx gqrx-sdr chrony pavucontrol
pulseaudio --start || true
pavucontrol # route audio to 'LoopbackSink' and select it in WSJT-X
```

6 Results

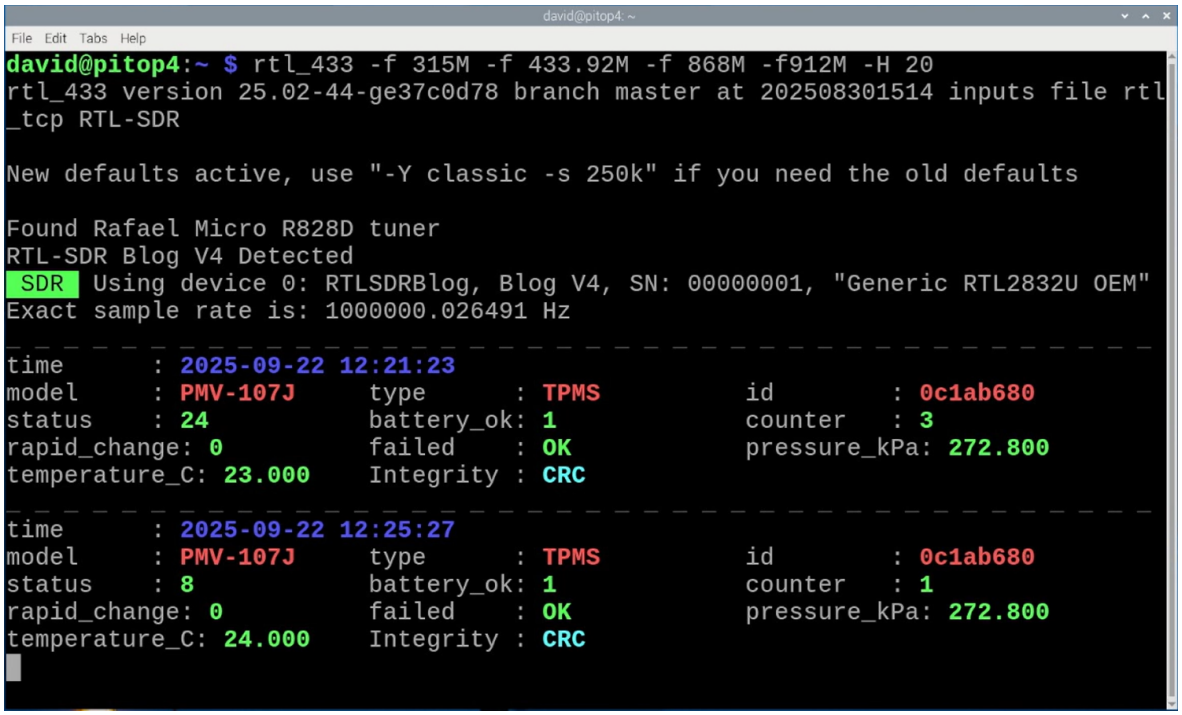

Figure 2: Output of rtl_433 decoding IoT/ISM band signals, showing TPMS sensor data including pressure and temperature.

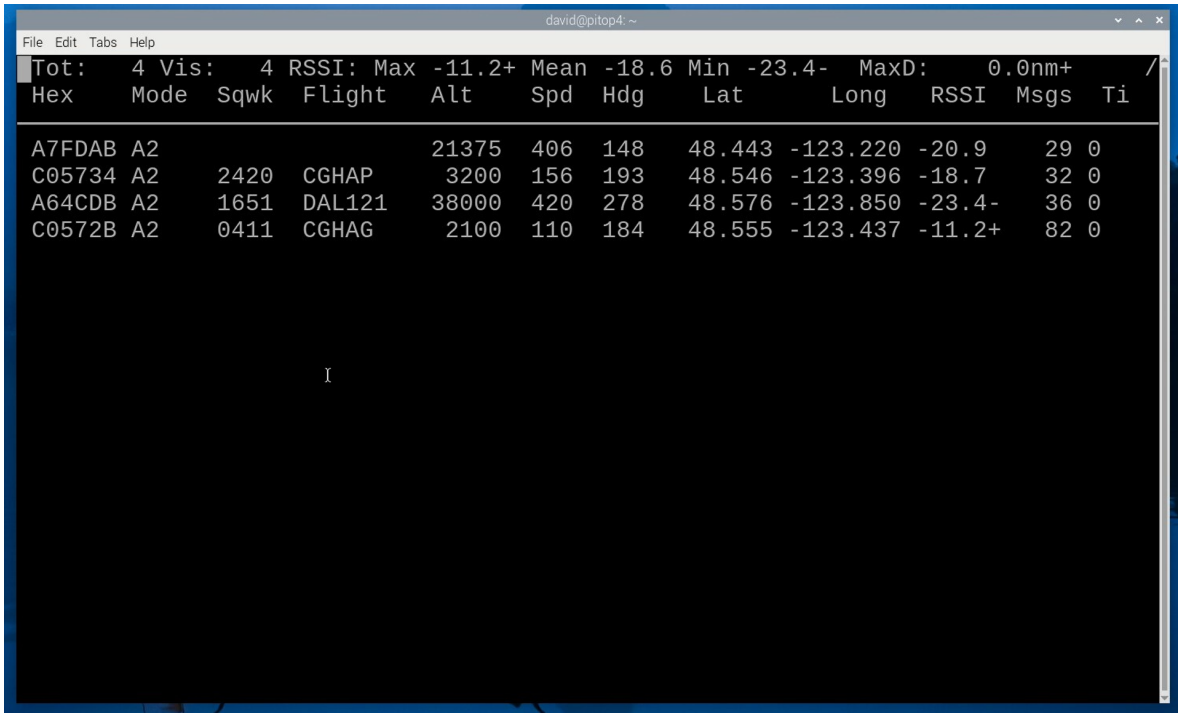

Figure 3: Terminal output from `dump1090-fa` showing decoded ADS-B aircraft data, including flight number, altitude, speed, and geographic coordinates.

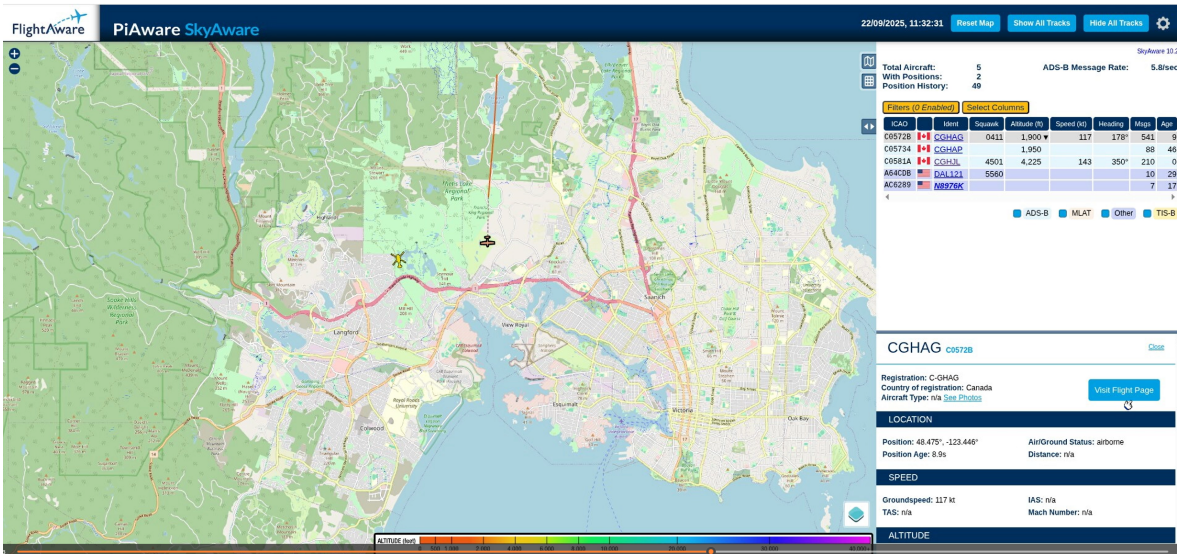

Figure 4: Verification of ADS-B aircraft signals using FlightAware SkyAware, displaying live aircraft positions over the Victoria area.

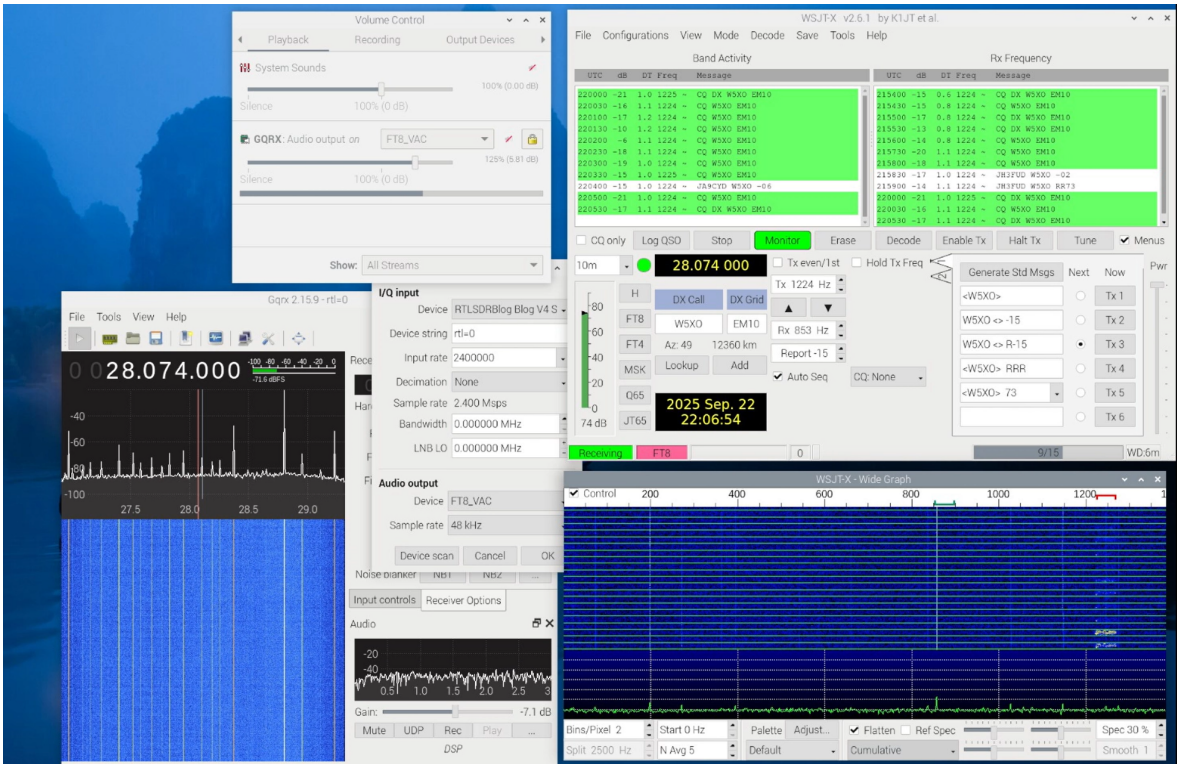

Figure 5: Decoding FT8 shortwave digital signals using GQRX for signal reception and WSJT-X for weak-signal digital mode decoding.

7 Conclusion

This project successfully demonstrated the design and implementation of a standalone Software Defined Radio (SDR) receiver system using a Raspberry Pi 4 and an RTL-SDR Blog V4 dongle. The system was able to capture and decode three distinct classes of real-world signals: ADS-B aircraft telemetry, IoT/ISM band devices, and FT8 shortwave digital modes. Each of these signals was validated against reliable external references, confirming the accuracy and robustness of the setup.

Through this project, we gained practical experience in SDR hardware and software integration, configuration of Linux-based signal processing tools, and validation of real-world communication signals. The team also strengthened skills in project coordination, documentation, and multimedia presentation, enabling the project to be completed within a short timeframe.

Acknowledgments

We would like to thank the following individuals for their support:

- **Mark Reimer** – For generously allowing us to use his private hangar during the ADS-B aircraft signal capture phase.