

Development of signal interface for band-switchable transceiver IC

Summer Internship Report

Raunaq Ray

Department of Electrical and Computer Engineering

University of Washington, Tacoma

2024

Under the supervision of

Dr. Debasis Dawn

Professor

Electrical and Computer Engineering

University of Washington, Tacoma

Suman K Saripalli

Vice President

KalScott Engineering Inc.

Kansas, USA

Contents

| | |
|---|-----|
| Abstract..... | iii |
| 1. Introduction..... | 1 |
| 2. Aircraft Surveillance Technologies..... | 3 |
| 2.1. Primary Surveillance Radar (PSR)..... | 3 |
| 2.2. Secondary Surveillance Radar (SSR)..... | 5 |
| 3. ADS-B..... | 11 |
| 3.1. Message Structure | 11 |
| 3.2. Availability and Transmission Rate | 13 |
| 3.3. ADS-B Versions..... | 15 |
| 4. ADS-B message receiver | 18 |
| 4.1. Receiver Range:..... | 18 |
| 4.2. Antenna Setup | 19 |
| 4.3. Receiver Setup..... | 20 |
| 5. Rationale for ADS-B as Starting Point | 24 |
| 5.1. Remote ID Signals | 24 |
| 5.2. 900 MHz ISM Signals | 24 |
| 6. ADS-B Message Types and Creation | 25 |
| 6.1. Aircraft identification and Category | 25 |
| 6.2. Aircraft Airborne Position | 32 |
| 6.3. Aircraft Surface Position | 45 |
| 7. Exporting created Signal..... | 51 |
| 7.1. PPM Parameters used in ADS-B..... | 51 |
| 7.2. Implementation in MATLAB | 52 |
| 8. Interfacing with Cadence Virtuoso | 54 |
| 9. ADS-B Signal Processing in Multi-Band Transceiver Chain..... | 56 |
| 9.1. Input Baseband Signal | 56 |
| 9.2. Frequency Upconversion..... | 56 |
| 9.3. Amplification and Transmission | 56 |

| | | |
|-------|---|----|
| 9.4. | Signal Reception | 56 |
| 10. | Results and Simulation Constraints | 57 |
| 10.1. | Simulation Constraints | 57 |
| 10.2. | Signal Decoding | 58 |
| 10.3. | Signal Integrity..... | 58 |
| 11. | Future Work | 60 |
| 11.1. | ADS-B Signal Testing..... | 60 |
| 11.2. | Transceiver Integration | 60 |
| 11.3. | Validation and Refinement..... | 60 |
| 11.4. | Signal Expansion Research..... | 61 |
| 12. | Additional Work – AES-ECB Encryption for ADS-B messages | 62 |
| 12.1. | Context and Motivation | 62 |
| 12.2. | AES-ECB Encryption..... | 62 |
| 12.3. | Java AES Function Implementation | 63 |
| | References..... | 65 |

Abstract

This work addresses the need for a comprehensive signal generation and processing framework to enable rigorous testing, validation, and simulation of various scenarios for a reconfigurable transceiver IC in aviation communication applications. The report presents the development of an Automatic Dependent Surveillance-Broadcast (ADS-B) signal generation and processing system, focusing on integration with a multi-band transceiver IC being developed at University of Washington Tacoma.

The work is primarily aimed at creating a robust framework for generating, simulating, and analyzing signals that can be used to test and utilize the full potential of the reconfigurable transceiver IC within the context of modern aviation surveillance technologies. By developing MATLAB-based algorithms for signal creation and modulation, this project provides a versatile platform for implementing various test scenarios, enabling comprehensive validation and error checking of the transceiver's performance across multiple configurations.

This work demonstrates successful generation of ADS-B messages using MATLAB-based scripts through user-defined inputs. The accuracy of these generated messages was verified using pyModeS, an open-source tool whose core functions were utilized to validate the message generation process. The verified messages were then converted into Pulse Position Modulation (PPM) encoded signals for transmission. The results confirm that these PPM-encoded baseband signals, containing the ADS-B messages, were successfully transmitted through the transceiver chain and recovered at the output, maintaining signal integrity throughout the process. This was achieved despite computational constraints limiting full-duration simulations to 3-4 μ s segments.

The integration with Cadence Virtuoso for hardware simulation validates the practical applicability of the designed framework. Additionally, an experimental approach to ADS-B message encryption using AES-ECB has been explored, identifying potential avenues for testing the transceiver's ability to handle secure communications. The study concludes by outlining future scopes of work, including the development of more comprehensive signal testing frameworks and expanded research into RemoteID and 900 MHz ISM signal processing, building upon the successful implementation and validation of the ADS-B signal generation and processing system.

1. Introduction

The rapid evolution of aviation surveillance technologies has necessitated the development of sophisticated signal generation and processing frameworks to ensure the reliability and security of air traffic management systems. This work presents a comprehensive study on modern aviation communication technologies, with a particular focus on Automatic Dependent Surveillance-Broadcast (ADS-B) systems. The primary objective of this study is to design a versatile signal generation and processing framework that enables users to create custom signals tailored to various simulation use-case constraints, specifically within the context of aviation surveillance.

Recent studies have highlighted significant challenges in implementing next-generation air traffic management systems. Strohmeier *et al.* [1] identified issues such as message loss due to increasing traffic on the 1090MHz channel and security vulnerabilities associated with software-defined radios. These findings underscore the critical need for robust testing frameworks. The project directly addresses these concerns by proposing a user-defined signal processing framework that facilitates rigorous testing and validation.

The importance of open-source tools in air transportation research has been emphasized by Sun *et al.* [2] through their development of PyModeS, a Python library for decoding Mode-S and ADS-B signals. Building upon this work, our project incorporates PyModeS for verifying generated ADS-B messages, ensuring the accuracy and reliability of our signal generation process. Additionally, we have leveraged the OpenSky network, a large-scale ADS-B sensor network described by Schafer *et al.* [3], to gain insights into real-world ADS-B data and refine our understanding of aircraft surveillance technologies.

The significance of this project lies in its provision of a comprehensive testbed for simulating various scenarios to validate transceiver systems and other aviation communication components. By offering a user-defined signal framework, the project presents a versatile solution applicable in both research and industrial settings. The project builds upon the innovative work of Hamidi and Dawn in developing band-switchable CMOS power amplifiers and low noise amplifiers [4] [5], which form the foundation of the multi-band transceiver IC that this signal generation framework will interface with.

The scope of this work encompasses an in-depth analysis of current aviation standards and the development of a system that incorporates key elements of aviation wireless communication. The study begins with an examination of primary and secondary surveillance techniques employed in aircraft communications. It then delves into a detailed breakdown of ADS-B messages and their modulation techniques. The project proposes a MATLAB-based framework for generating key types of ADS-B messages, including aircraft identification and position encoding for both airborne and surface operations. These messages are then encoded using Pulse Position Modulation (PPM) with parameters set by the FAA for ADS-B transmission. Finally, the generated signals are interfaced with Cadence Virtuoso and simulated in a band-reconfigurable transceiver IC model for verification and validation, paving the way for future expansions into RemoteID and 900MHz ISM band applications.

2. Aircraft Surveillance Technologies

2.1. Primary Surveillance Radar (PSR)

Aircraft Surveillance started with the concept of capturing reflections from a rotating radio transponder with an omni-directional antenna, now known as the Primary Surveillance Radar (PSR). The radar transmits a $1\mu\text{s}$ pulse for every 1ms and listens to the reflections from the airplanes. Concepts of phase filters and Doppler filters are used to filter out moving targets (aircrafts/drones) and remove static objects (mountains, buildings and other obstacles). The PSR determines the aircraft's position using 2 key measurements –

1. Range – Calculated from the time difference between when the pulse was emitted and when the reflection was received.
2. Bearing – Determined by the antenna's azimuth at the moment the reflection was received.

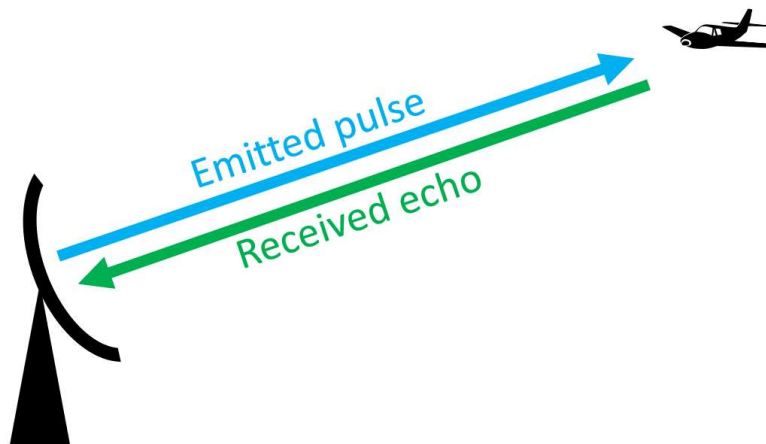


Fig.1: Principle of operation for PSR ¹

The PSR outputs the data in a polar coordinate system, providing range and bearing of detected targets relative to the antenna's position. The range provided is the slant distance from the antenna to the aircraft. This creates a 2D representation of aircraft positions in the PSR's coverage area.

¹ Image Source: https://skybrary.aero/sites/default/files/PSR_1_0.png

However, the PSR has several significant disadvantages for aircraft position mapping –

1. Coverage Limitations –

PSR often has inadequate coverage, especially for airspace directly above the antenna due to its radiation pattern. This creates blind spots that require multiple radar installations to fully cover the area.

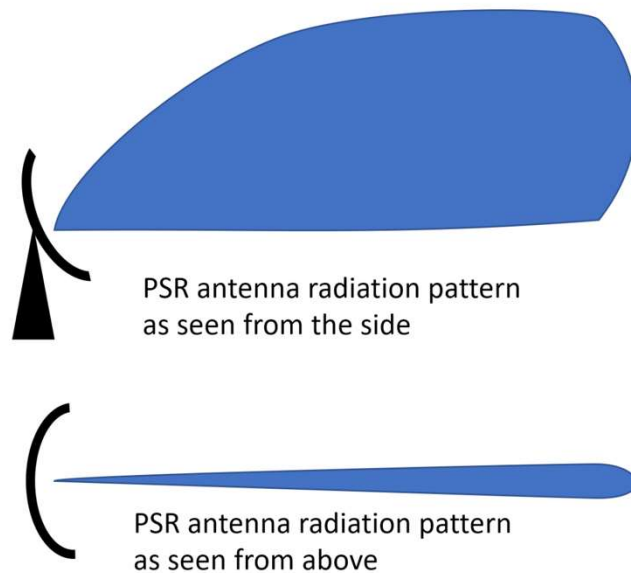


Fig.2: PSR Antenna Radiation Pattern ²

2. Data Limitations –

It does not provide any information related to the identity of the aircraft. It also fails to capture accurate altitude information due to the nature of its tracking.

3. Detection Issues –

It cannot distinguish between multiple aircraft at the same slant range but different altitudes. It may also detect false targets from ground vehicles, weather, birds, etc. Additionally, performance degrades in the presence of ground clutter and adverse weather conditions.

² Image Source: https://skybrary.aero/sites/default/files/PSR_2.png

4. Resource Limitations –

PSRs need optimal sites with unobstructed views and minimal ground clutter. They often require high transmitter power for long range performance and is expensive to install and maintain as compared to other surveillance technologies.

These disadvantages have led to the addition of more technologies on top of the PSR to compensate for the shortcomings. The Secondary Surveillance Radar (SSR), also known as the Air Traffic Control Beacon System (ATCBS), was designed to provide air traffic controllers with more information. The SSR can be installed separately or on top of a primary radar.

2.2. Secondary Surveillance Radar (SSR)

2.2.1. *Mode A/C*

The SSR transmits interrogation messages at 1030MHz, and the aircraft transponder transmits replies at 1090MHz. Initial implementations of SSR worked on Mode A and Mode C (civilian communication protocols) to continuously interrogate the squad code (aircraft ID) and the altitude of the aircraft.

Mode A was designed to provide aircraft identification. An aircraft equipped with a transponder would respond to SSR interrogations by transmitting a 4-digit octal code, which is manually set by the pilot. This code, known as a squawk code, helps air traffic controllers identify the aircraft located by the PSR. Mode A was limited to 4096 unique codes (8484), which posed challenges in areas with high air traffic density.

Mode C added altitude reporting to the capabilities of Mode A. It provided the aircraft's pressure altitude data, allowing air traffic controllers to determine the aircraft's vertical position in addition to its identity. Mode C transponders responded to SSR interrogations by transmitting altitude information, typically with a resolution of 100 feet.

The SSR initiates Mode A and Mode C interrogations with 2 different pulse patterns, which are shown in the figure below –

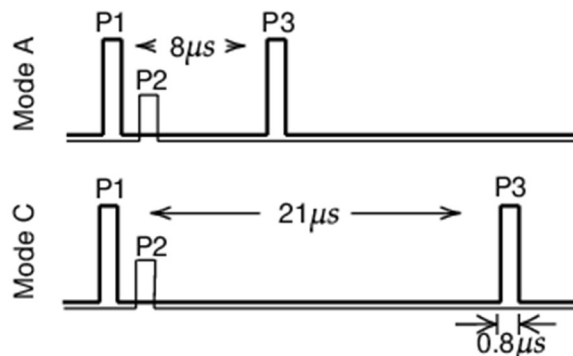


Fig.3: Mode A and Mode C pulse patterns³

The pulses are about $0.8 \mu s$ wide. P1 and P3 are the two main pulses sent by the directional antenna. They are separated by $8 \mu s$ and $21 \mu s$, respectively for Mode A and Mode C. P2 is a pulse emitted through the omnidirectional antenna right after P1. Pulse P2 is introduced for side-lobe suppression. When the power of P2 is higher than P1, the interrogation is likely from the side lobes of the directional antenna and should be ignored by the aircraft. This can happen when the aircraft is close to the radar.

The figure below shows an example of Mode A/C reply –

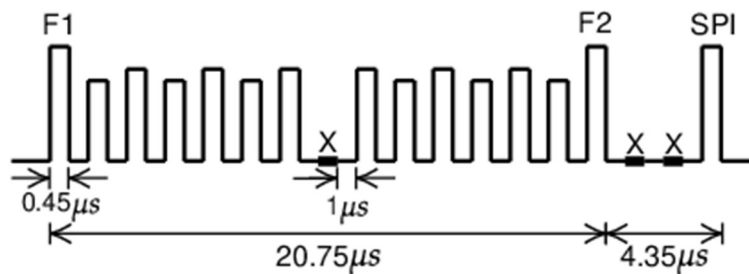


Fig.4: Mode A and Mode C reply pulse pattern⁴

Each reply consists of two persistent pulses, F1 and F2, separated by $20.3 \mu s$. Within this period, either the identity code or the altitude code is encoded using 13 pulses of $0.45 \mu s$. The pulses are separated by gaps of $1 \mu s$. The pulse at the center serves as a verification pulse and is always absent. The presence or absence of any of the other 12 pulses represents

³ Image Source: https://mode-s.org/decode/figures/intro/mode_ac_uplink_pulses.png

⁴ Image Source: https://mode-s.org/decode/figures/intro/mode_ac_downlink_pulses.png

a 1 or 0 bit. When required by air traffic controllers for identification purposes, a special purpose identification (SPI) may follow F2 after two absent pulses.

This design of ATCRBS works sufficiently well with low-density air traffic, but it cannot efficiently cope with higher flight densities since all aircraft replies are transmitted on the same frequency. When several aircraft are in the same direction of the radar beam, reply signals can overlap and introduce errors for decoding. This is known as *synchronous garbling*.

When there are multiple secondary radars in the vicinity, replies originated by other radars may be considered as valid responses of one radar, which in turn causes errors and confusion. This syndrome is called *FRUIT (False Replies Unsynchronized In Time)*.

2.2.2. Mode S

Mode S (Mode Select Beacon System) introduces the capability of selective interrogations. This allows the SSR to interrogate different information from different aircraft separately. By using selective interrogation, it largely mitigated the problem of garbling in Mode A/C and thus greatly improved the capacity of the communication channel.

Unlike the limited number (4096) of unique identification codes in Mode A communication, the Mode S transponder is identified by a 24-bit transponder code, which can support up to 16,777,216 (2^{24}) unique addresses. In addition to these two major advantages, the Mode S protocol introduced many different types of information that could be interrogated and downlinked.

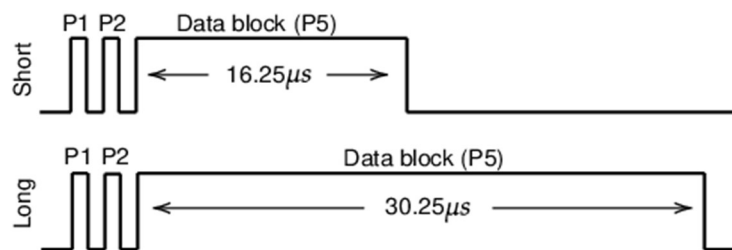


Fig.5: Mode-S uplink pulses ⁵

⁵ Image Source: https://mode-s.org/decode/figures/intro/mode_s_uplink_pulses.png

The Mode S uplink signal contains parameters that indicate which information is desired by the air traffic controller. There are two types of Mode S interrogations, shown in the figure below. The short interrogation has 56 bits of information contained in the data block, while the long interrogation contains 112 bits of information. The P2 pulse acts as the side-lobe suppression for Mode A/C transponders so that they will ignore the rest of the interrogation pulses. Information in the Mode S interrogation data block uses the Differential Phase-Shift Keying (DPSK) modulation, which is a type of Phase Modulation.

There are two types of Mode S downlink signals, the short reply and the long reply, which correspond to the short and long interrogations from the SSR. For each microsecond, one bit is transmitted. All Mode S replies start with an 8 μ s fixed preamble and continue with 56 μ s or 112 μ s data block. The structure of the downlink message is shown in the figure below.

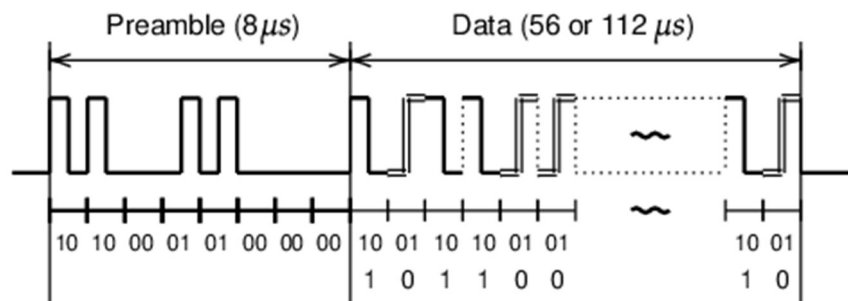


Fig. 6: Mode-S reply ⁶

The information contained in the data block is modulated using the Pulse Position Modulation (PPM), which is a type of amplitude modulation. In PPM, the 1 bit is represented by a 0.5 μ s of pulse followed by a 0.5 μ s flat signal. The 0 bit is reversed compared to the 1 bit, which is represented by a 0.5 μ s flat signal and followed by a 0.5 μ s pulse.

The Mode S communication protocol is designed to handle different types of uplink and downlink message formats. The first five bits of the message define the uplink format (UF) or downlink format (DF) number of the message. Based on the UF/DF number, different

⁶ Image Source: https://mode-s.org/decode/figures/intro/mode_s_downlink_pulses.png

structures of the data block are presented. The table below shows all available Mode-S formats. Numbers not on this table are currently reserved for future use.

| UF/DF | Bits | Uplink type | Downlink type |
|-------|------|-----------------------------------|-----------------------------------|
| 0 | 56 | Short air-air surveillance (ACAS) | Short air-air surveillance (ACAS) |
| 4 | 56 | Surveillance, altitude request | Surveillance, altitude reply |
| 5 | 56 | Surveillance, identity request | Surveillance, identity reply |
| 11 | 56 | Mode S All-Call | All-Call reply |
| 16 | 112 | Long air-air surveillance (ACAS) | Long air-air surveillance (ACAS) |
| 17 | 112 | - | Extended squitter |
| 18 | 112 | - | Extended squitter/non transponder |
| 19 | 112 | - | Military extended squitter |
| 20 | 112 | Comm-A, altitude request | Comm-B, altitude reply |
| 21 | 112 | Comm-A, identity request | Comm-B, identity reply |
| 24 | 112 | Comm-C (ELM) | Comm-D (ELM) |

Among all uplink formats, UF 17, 18, and 19 are not used. This is because the corresponding downlink messages (extended squitter messages) are designed to be broadcast automatically without the need for SSR interrogations. One of the most common applications for the extended squitter is the Automatic Dependent Surveillance-Broadcast service, which is commonly known as ADS-B. Figure below shows Mode S services and their relationships with different Mode S downlink formats.

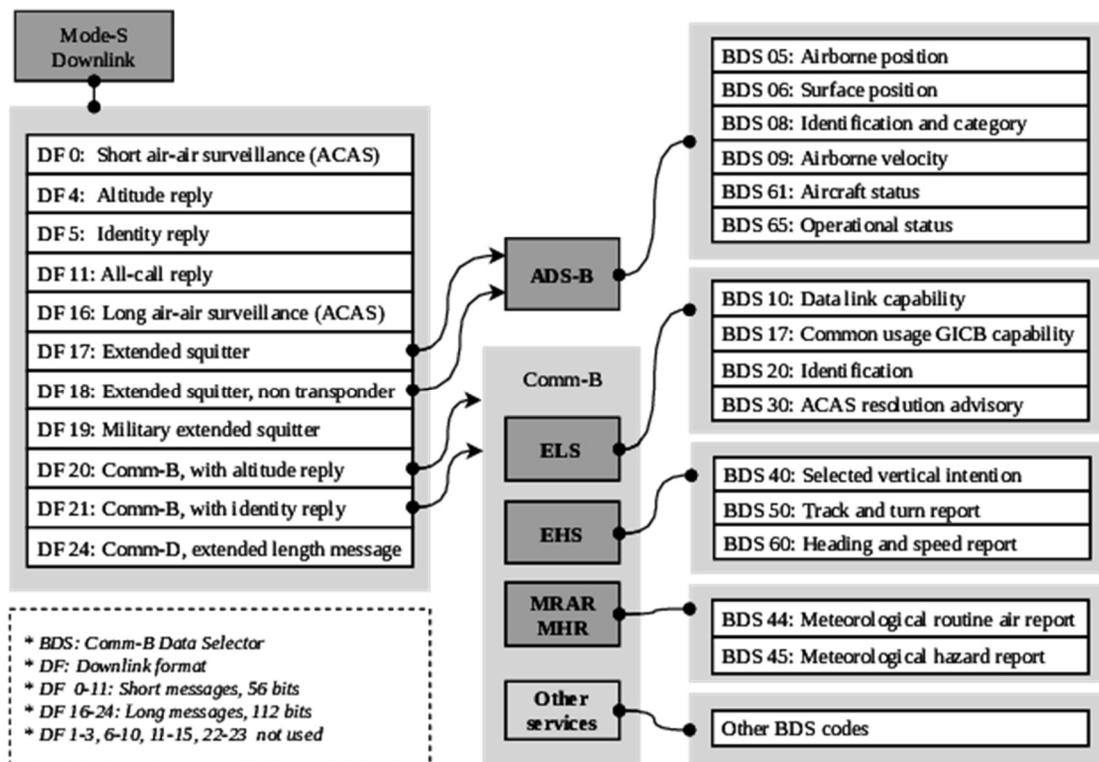


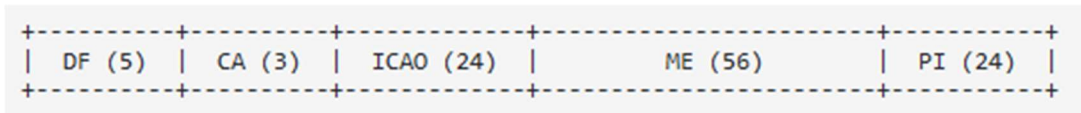
Fig.7: Relationship between Mode S downlink formats and different services ⁷

⁷ Image Source: https://mode-s.org/decode/figures/intro/mode_s_services.png

3. ADS-B

Automatic Dependent Surveillance-Broadcast (ADS-B) is a surveillance technology designed to allow aircraft to broadcast their flight state periodically without the need for interrogation. The word *automatic* refers to the fact that no input from controllers or pilots are required. The word *dependent* indicates this technology depends on information from other onboard systems, such as air data systems and navigation systems.

3.1. Message Structure



An ADS-B frame is 112 bits long and consists of the following main parts -

| Bit | No. bits | Abbreviation | Information |
|---------|----------|--------------|----------------------------|
| 1–5 | 5 | DF | Downlink Format |
| 6–8 | 3 | CA | Transponder capability |
| 9–32 | 24 | ICAO | ICAO aircraft address |
| 33–88 | 56 | ME | Message, extended squitter |
| (33–37) | (5) | (TC) | (Type code) |
| 89–112 | 24 | PI | Parity/Interrogator ID |

1. **DF (Downlink Format):** The first 5 bits indicate the format of the message. For ADS-B messages, this is set to 10001 (binary), which corresponds to Downlink Format 17.
2. **CA (Capability):** The next 3 bits provide information about the transponder's capabilities, which can vary based on the type of ADS-B message being sent. The capability value can be a decimal value between 0 and 7.

| CA | Definition |
|-----|---------------------|
| 0 | Level 1 transponder |
| 1–3 | Reserved |

| | |
|---|--|
| 4 | Level 2+ transponder, with ability to set CA to 7 when on-ground |
| 5 | Level 2+ transponder, with ability to set CA to 7 when airborne |
| 6 | Level 2+ transponder, with ability to set CA to 7 when either on-ground or airborne |
| 7 | Indicates that the Downlink Request value is 0, or that the Flight Status is 2, 3, 4, or 5, either airborne or on the ground |

A Level 1 transponder is the most basic type of transponder. It primarily provides fundamental identification capabilities. This level typically transmits a unique identification code (squawk code) when interrogated by air traffic control (ATC) but does not provide altitude information or advanced data capabilities. Level 1 transponders may be used in less congested airspace or for operations that do not require detailed surveillance data. A Level 2+ transponder represents a more advanced category with enhanced capabilities beyond basic identification. Level 2+ transponders can provide altitude information and operational status. Level 2+ transponders are required in controlled airspace and are essential for commercial aviation, where precise tracking and altitude reporting are critical for safety.

3. **ICAO Address:** The ICAO address is located from 9 to 32 bits in binary (or 3 to 8 in hexadecimal positions). A unique ICAO address is assigned to each Mode S transponder of an aircraft and serves as the unique identifier for each aircraft, according to ICAO regulations [6]. In principle, this code does not change over the lifetime of the aircraft. However, it is possible to reprogram a transponder so that the messages contain a different address. This has been observed for some military aircraft, as well as some private airplanes opting-in for the FAA Privacy ICAO Address System [7].
4. **ME (Message Elements):** The 56 bits in this section contain various types of data, including position, velocity, and other operational parameters. The specific content depends on the Type-Code defined in this field.

| Type Code | Data frame content |
|-----------|-------------------------------------|
| 1–4 | Aircraft identification |
| 5–8 | Surface position |
| 9–18 | Airborne position (w/Baro Altitude) |
| 19 | Airborne velocities |
| 20–22 | Airborne position (w/GNSS Height) |
| 23–27 | Reserved |
| 28 | Aircraft status |
| 29 | Target state and status information |
| 31 | Aircraft operation status |

5. **PI (Parity/Interrogator ID):** The last 24 bits are used for error checking and known as the CRC (Cyclic Redundancy Check) remainder. The primary purpose of the CRC remainder is to detect errors that may occur during the transmission of ADS-B messages.

3.2. Availability and Transmission Rate

| Messages | TC | Ground (still) | Ground (moving) | Airborne |
|--------------------------|-------------|---|--|----------|
| Aircraft identification | 1–4 | 0.1 Hz | 0.2 Hz | 0.2 Hz |
| Surface position | 5–8 | 0.2 Hz | 2 Hz | - |
| Airborne position | 9–18, 20–22 | - | - | 2 Hz |
| Airborne velocity | 19 | - | - | 2 Hz |
| Aircraft status | 28 | 0.2 Hz (<i>no TCAS RA and Squawk Code change</i>) | | |
| | | 1.25 Hz (<i>change in TCAS RA or Squawk Code</i>) | | |
| Target states and status | 29 | - | - | 0.8 Hz |
| Operational status | 31 | 0.2 Hz | 0.4 Hz (<i>no NIC/NAC/SIL change</i>) | |
| | | | 1.25 Hz (<i>change in NIC/NAC/SIL</i>) | |

ADS-B messages are available to any receiver equipped with ADS-B IN capabilities, allowing for widespread situational awareness among pilots and air traffic controllers. Different ADS-B messages have different transmission rates. The update frequency also differs depending on

whether the aircraft is on-ground or airborne, as well as whether the aircraft is still or moving when on the ground.

For Operational Status messages (Type Code 31), when there is a change in NIC, NAC, or SIL, the transmission rate **increases to 1.25 Hz**. This means that if any of these integrity indicators change, an operational status message will be sent more frequently to ensure that air traffic control and other aircraft are aware of the updated reliability of navigation data. If there are no changes in these categories, the transmission rate **remains at 0.4 Hz**.

- **NIC (Navigation Integrity Category):** NIC indicates the integrity of the navigation data being transmitted. It assesses how accurately the aircraft's position can be trusted based on the navigation system's performance. A higher NIC value means greater confidence in the accuracy of the position information.
- **NAC (Navigation Accuracy Category):** NAC represents the accuracy of the position information provided by the aircraft's navigation system. It quantifies how close the reported position is to the actual position. This helps air traffic controllers and other aircraft gauge how reliable the positional data is.
- **SIL (Source Integrity Level):** SIL indicates the reliability of the source providing the position information. It assesses whether the data is trustworthy and meets operational requirements.

For Aircraft Status Messages (Type Code 28), if there is a change in TCAS RA or squawk code (the unique identifier assigned to each aircraft), the transmission rate **increases to 1.25 Hz**. This ensures that any critical changes affecting collision avoidance are communicated promptly. If there are no changes in TCAS RA or squawk code, these messages are sent at a rate of **0.2 Hz**.

- **TCAS RA (Traffic Collision Avoidance System Resolution Advisory):** TCAS RA refers to alerts generated by a Traffic Collision Avoidance System when an aircraft is at risk of collision with another aircraft. The system advises pilots on necessary maneuvers to avoid potential collisions.

3.3.ADS-B Versions

ADS-B has had 3 versions of implementations throughout its lifespan. This was done to include more information in ADS-B messages. These have been put together from the official ICAO 9871 document [8].

1. Version 0: DO-260/ED-102

This was the initial standard for ADS-B, providing foundational guidelines for technology. Introduced the concept of broadcasting aircraft position, velocity, and identification based on GPS data. The **Navigation Uncertainty Category for Position (NUCp)** was the only means to indicate the accuracy or integrity of horizontal position data. The reliance solely on NUCp for integrity reporting limited the system's ability to provide detailed information about navigation accuracy and integrity.

2. Version 1: DO-260A

Introduced distinct parameters for reporting accuracy and integrity, including:

- **Navigation Accuracy Category for Position (NACp)**: Indicates the accuracy of position data.
- **Navigation Integrity Category (NIC)**: Reflects the integrity of navigation data.
- **Surveillance Integrity Level (SIL)**: Assesses the reliability of surveillance data.
- Added new status parameters and message types, including those for **Traffic Information Service – Broadcast (TIS-B)** and **Automatic Dependent Surveillance – Rebroadcast (ADS-R)**.

TIS-B is a service that provides information about non-ADS-B equipped aircraft to those that are equipped with ADS-B. It broadcasts traffic information derived from ground-based surveillance systems, such as radar, to enhance situational awareness in the cockpit. **ADS-R** is a method used to rebroadcast ADS-B information received on one frequency to another frequency, allowing different types of equipped aircraft to see each other. For example, it enables communication between aircraft using

the 1090 MHz Extended Squitter (for commercial aviation) and those using the Universal Access Transceiver (UAT) at 978 MHz (primarily for general aviation).

3. Version 2: DO-260B/ED-102A

This version was developed based on operational experience gained from earlier versions.

- **Enhanced Integrity Reporting:** Further separated position source and system integrity reporting, allowing for more detailed assessments of navigation data.
- **Additional NIC Levels:** Introduced more levels of NIC to better support both airborne and surface applications.
- **Mode A Code Broadcasting:** Incorporated the ability to broadcast Mode A code in emergency/priority messages, increasing transmission rates after a Mode A code change.
- Expanded parameters in Target State and Status Messages to provide more comprehensive traffic information.

4. Version 3: DO-260C

This version focuses on optimizing data transmission and enhancing functionality for modern air traffic management needs.

- **Interval Management Support:** Enhanced capabilities for managing flight intervals, allowing for better spacing between aircraft during approach and landing.
- **Weather Data Broadcasting:** Introduced optional messages that can report additional weather parameters such as wind speed, temperature, and icing status.
- **Improved Receiver Performance:** Enhanced capabilities for track initiation and maintenance, reducing response times for both airborne and surface traffic tracking.
- **Autonomous Distress Tracking (ADT):** Added functionality to automatically transmit an aircraft's position at least once per minute when in distress, improving search and rescue operations.
- Support for new applications such as wake turbulence avoidance and hazardous weather detection.

| Version | Key Features & Changes |
|------------------|---|
| Version 0 | Initial standard; relied solely on NUCp for integrity. |
| Version 1 | Introduced NACp, NIC, SIL; added TIS-B/ADS-R messages. |
| Version 2 | Enhanced integrity reporting; more NIC levels; Mode A broadcasting. |
| Version 3 | Interval management support; weather data broadcasting; autonomous distress tracking. |

Steps to Verify the ADS-B Version

1. **Step 1:** Determine if the aircraft is transmitting ADS-B messages with Type Code (TC) 31.
If no such messages are detected, it can be concluded that the aircraft is operating on Version 0.
2. **Step 2:** If messages with TC=31 are present, examine the version numbers found in bits 41–43 of the Message Element (ME) or bits 73–75 of the data packet.

Once the ADS-B version for an aircraft is identified (which typically remains constant), you can proceed to decode the associated TC=28 and TC=31 messages accordingly.

4. ADS-B message receiver

This section outlines the hardware and software setup for an ADS-B receiver using an RTL-SDR (Software Defined Radio) connected to a Raspberry Pi. The setup is designed to receive and decode raw ADS-B messages transmitted at 1090MHz. The receiver setup has been made with low-cost off-the-shelf components along with open-source software tools to support the extraction and decoding of data from Mode S signals.



Fig. 8: ADS-B receiver using RaspberryPi 3B+ and RTL-SDR

4.1.Receiver Range:

Mode S uses L band signals that follow the line-of-sight propagation, any obstructions between the transmitter and receiver can cause a significant number of signals to be blocked. If no obstacle exists between the aircraft and the receiver, and the transmitter has enough power, the maximum range of the receiver is determined by the curvature of the earth.

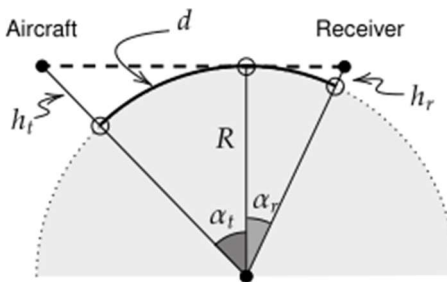


Fig.9: Receiver range calculations

The maximum distance (d) of a Mode S receiver can be obtained by knowing the altitude of the receiver antenna:

$$d = (\alpha_r + \alpha_t)R$$
$$d = \left(\cos^{-1} \frac{R}{R + h_r} + \cos^{-1} \frac{R}{R + h_t} \right) R$$

where R is the radius of the earth, while h_t and h_r are the height of the aircraft and receiver above sea level. Using this equation, we can calculate the maximum receiving range for aircraft flying at different altitudes. However, in real-life applications, Mode S signal follows the Friis transmission model, which states that the maximum distance also depends on the power of the transmitter, as well as the directivities of the transmission and receiving antennas. When considering these factors, the actual radio range for the receiver is typically lower than the theoretical values calculated from the equations mentioned.

4.2. Antenna Setup

In principle, any antenna designed for radio frequency around 1GHz can be used for receiving Mode S signals. The carrier frequency of Mode S is 1090MHz, which corresponds to the wavelength of 27.5cm. To have an antenna that is tuned to this specific frequency, a simple metal wire (conductor) and a coaxial feeder cable was used to design a dipole antenna. A monopole antenna can also be used for the same purpose. The monopole antenna and the dipole antenna both are half-wavelength ($\lambda/2$) antennas with a total conductor length of 13.75cm.



Fig. 10: Antenna Setup

4.3.Receiver Setup

The receiver is built using a RTL-SDR. SDRs allow the user to select the center frequency of the receiver and the sampling rate, as well as define the bandwidth of the onboard low-pass filter if it is supported. An RTL-SDR Blog V4 R828D RTL2832U 1PPM TCXO HF Bias Tee SMA Dongle was used with a Raspberry Pi 3 to capture the Mode S signals. It can work with radio frequencies from 24 to 1766 MHz. The maximum sampling rate is 2.8 million samples per second (MSPS). This SDR is only capable of listening in the frequency ranges (can receive radio signals but cannot transmit radio signals). The dipole antenna was mounted at an elevated position to maximize reception range. An active antenna with a Low-Noise Amplifier (LNA) can be used to further reduce the signal-to-noise ratio.

The Raspberry Pi runs a basic copy of the Raspbian Operating System. Several software tools are available to work with some of these SDR devices and decode Mode S and ADS-B signals directly.

4.3.1. *dump1090*

dump1090 [9] is one of the most frequently used open-source Mode S decoder. It also offers embedded HTTP server that displays the currently detected aircrafts on Google Map, along with TCP server streaming and receiving raw data to/from connected clients (using `--net`). Additionally, it provides an interactive command-line-interface mode where aircrafts currently detected are shown as a list refreshing as more data arrives. It also supports CPR coordinates decoding and track calculation from velocity.

1. The directory was cloned (downloaded) to a local source as shown:

```
$ git clone https://github.com/antirez/dump1090.git
```

2. Installing the dependencies:

```
sudo apt-get install build-essential debhelper librtlsdr-dev pkg-config dh-systemd libncurses5-dev libbladerf-dev
```

3. Compile the directory

```
$ cd dump1090
```



```
$ make
```

The successful compilation marks the complete installation of dump1090. Once it is compiled and the RTL-SDR receiver is connected, the following command starts receiving and decoding the signals:

```
$ ./dump1090
```

With the default option, Mode S messages and decoded information are displayed in the terminal.

```
1. *8d451dbd9905b5018004005979c5;  
2. CRC: 000000  
3. RSSI: -20.6 dBFS  
4. Score: 1800  
5. Time: 9214131.08us  
6. DF:17 AA:451DBD CA:5 ME:9905B501800400  
7. Extended Squitter Airborne velocity over ground, subsonic (19/1)  
8. ICAO Address: 451DBD (Mode~S / ADS-B)  
10. Air/Ground: airborne  
12. Ground track 271.4  
14. Groundspeed: 436.1 kt  
16. Geom rate: 0 ft/min  
18. NACv: 0
```

To view Raw Mode S messages, the `–raw` option can be used

```
$ ./dump1090 –raw
```

The terminal output will only have the raw ADS-B messages.

```
*5d4074358ad00c;  
*8d407435990dbd01900484f66c3c;  
*8d40743558af828cd326fe0c2fe9;  
*a80011b1e0da112fe0140060939f;  
*a00015b8c2680030a80000318667;  
*5d4074358ad030;  
*5d4074358ad030;
```

dump1090 can also provide a live view of all aircraft seen by the receiver using the `–interactive` option:

```
$ ./dump1090 --interactive
```

| Hex | Mode | Sqwk | Flight | Alt | Spd | Hdg | Lat | Long | RSSI | Msgs | Ti |
|--------|------|------|---------|-------|-----|-----|--------|-------|-------|------|----|
| 40750D | S | 3573 | WUK2SH | 38000 | 435 | 285 | 51.707 | 4.833 | -27.6 | 452 | 0 |
| 471EA8 | S | 5205 | WZZ9CS | 37025 | 446 | 105 | 51.650 | 4.404 | -32.6 | 2928 | 1 |
| 471F33 | S | 2246 | WZZ1751 | 37025 | 435 | 106 | | | -34.3 | 7279 | 37 |
| 0CA56D | S | 1347 | RYR74PQ | 40000 | 436 | 266 | 52.029 | 4.579 | -29.7 | 7400 | 0 |

4.3.2. *pyModeS*

PyModeS is an open-source Python library developed by Junzi Sun and contributors from TU Delft's Aerospace Engineering Faculty for decoding Mode-S and ADS-B signals [10]. The generated sample ADS-B messages was verified using pyModeS. The stable version of pyModeS can be installed as:

```
pip install --upgrade pyModeS
```

pyModeS provides the possibility to view live traffic through the modeslive command.

```
$ modeslive [-h] --source SOURCE [--connect SERVER PORT DATATYPE]
[--latlon LAT LON] [--show-uncertainty] [--dumpton DUMPTO]
arguments:
-h, --help
Show this help message and exit
--source SOURCE
Choose data source, "rtlsdr" or "net"
--connect SERVER PORT DATATYPE
Define server, port and data type. Supported data
types are: ['raw', 'beast', 'skysense']
--latlon LAT LON
Receiver latitude and longitude, needed for the surface
position, default none
--show-uncertainty
Display uncertainty values, default off
--dumpton DUMPTO
Folder to dump decoded output, default none
```

For this project, the RTL-SDR was used to fetch traffic data from dump1090. The dump1090 data source was then supplied through a TCP output stream using these commands –

```
$ dump1090 --net --quiet
$ modeslive --source net --connect localhost 30002 raw
```

The functionalities of this are further broken down to use the low-level decoding functionalities to decode and verify the generated ADS-B messages.

Core functions of pyModeS can be used to decode Downlink Format, ICAO address, ADS-B Type Code, as well as to perform parity check:

```
import pyModeS as pms

pms.df(msg)      # Downlink Format
pms.icao(msg)     # Infer the ICAO address from the message
pms.crc(msg)     # Perform parity check
pms.typecode(msg) # Obtain ADS-B message Type Code
```

ADS-B related functions allow information such as identity, position, and velocities to be decoded.

```
# position messages
pms.adsb.position(msg_even, msg_odd, t_even, t_odd)
pms.adsb.altitude(msg)

# velocity messages
pms.adsb.velocity(msg)
```

There are also several functions designed to infer and decode Mode S downlink messages.

```
pms.common.altcode(msg) # Mode S altitude code (DF=4/20)
pms.common.idcode(msg) # Mode S squawk code (DF=5/21)
pms.bds.infer(msg)      # Infer Modes S BDS code
```

Once a Mode S message type is identified, type specific functions can be used to decode corresponding parameters.

```
# BDS 4,0
pms.commb.selalt40mcp(msg) # MCP/FCU selected altitude (ft)
pms.commb.selalt40fms(msg) # FMS selected altitude (ft)
pms.commb.p40baro(msg)    # Barometric pressure setting (mb)

# BDS 5,0
pms.commb.roll50(msg)     # Roll angle (deg)
pms.commb.trk50(msg)      # True track angle (deg)
pms.commb.gs50(msg)       # Ground speed (kt)
pms.commb.rtrk50(msg)     # Track angle rate (deg/sec)
pms.commb.tas50(msg)      # True airspeed (kt)

# BDS 6,0
pms.commb.hdg60(msg)      # Magnetic heading (deg)
pms.commb.ias60(msg)      # Indicated airspeed (kt)
pms.commb.mach60(msg)     # Mach number (-)
pms.commb.vr60baro(msg)   # Barometric altitude rate (ft/min)
pms.commb.vr60ins(msg)    # Inertial vertical speed (ft/min)
```

These are some commonly used functions related to Mode S decoding. A complete list of the APIs can be found in the pyModeS library API documentation.

5. Rationale for ADS-B as Starting Point

Our decision to begin with ADS-B signal generation and processing was strategic, considering the challenges associated with other relevant signal types:

5.1. Remote ID Signals

- The standards for RemoteID are still evolving, with ongoing discussions and potential changes in regulatory requirements.
- Academic resources and established protocols for RemoteID are currently limited, making initial research and development challenging.
- ADS-B, in contrast, offers well-established protocols and abundant documentation, providing a solid foundation for our initial work.

5.2. 900 MHz ISM Signals

- The 900 MHz ISM band encompasses a wide range of applications, each with its own signal characteristics and requirements.
- FCC regulations governing this band are complex and vary depending on the specific application and power levels.
- Starting with ADS-B allowed us to develop our methodologies using a more standardized signal before tackling the diversity of 900 MHz ISM applications.

6. ADS-B Message Types and Creation

ADS-B messages can be broadly categorized into several types, each serving a specific purpose in transmitting aircraft information.

6.1. Aircraft identification and Category

ADS-B messages include a specific type known as Aircraft Identification and Category message. This message serves two primary purposes: it transmits the aircraft's identification, commonly referred to as the callsign, and communicates the aircraft's wake vortex category. By broadcasting this information, the message enables air traffic control systems and other aircraft to accurately identify and categorize the transmitting aircraft. In this message, the Type Code can be from 1 to 4. The 56-bit ME field consists of 10 parts and is structured as follows:

```
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| TC,5 | CA,3 | C1,6 | C2,6 | C3,6 | C4,6 | C5,6 | C6,6 | C7,6 | C8,6 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+

TC: Type code
CA: Aircraft category
C*: A character
```

6.1.1. Identification (Callsign)

The aircraft identification included in the message is the callsign. A callsign is not a unique identifier of an aircraft, since different aircraft flying the same route at different times would share the same callsign.

The last eight fields (C1 to C8) in the previous structure diagram represent the callsign characters. The characters are mapped based on a lookup table which maps the corresponding decimal number (represented in binary code) to each character. The character mapping is as follows -

```
#ABCDEFGHIJKLMNOPQRSTUVWXYZ#### #0123456789####
```

The # symbols represent characters that are not used. The character and their decimal representations are as follows –

| | |
|---------|---------|
| A - Z : | 1 - 26 |
| 0 - 9 : | 48 - 57 |
| ␣ : | 32 |

The ␣ symbol refers to a space character. It is worth noting that it is easy to identify that a callsign character is encoded using the lower six bits of the same character in ASCII (American Standard Code for Information Interchange) code.

6.1.2. *Wake Vortex Category*

The CA value in combination with TC value defines the wake vortex category of the aircraft.

| TC | CA | Category |
|-----|-----|---|
| 1 | ANY | Reserved |
| ANY | 0 | No category information |
| 2 | 1 | Surface emergency vehicle |
| 2 | 3 | Surface service vehicle |
| 2 | 4-7 | Ground obstruction |
| 3 | 1 | Glider, sailplane |
| 3 | 2 | Lighter-than-air |
| 3 | 3 | Parachutist, skydiver |
| 3 | 4 | Ultralight, hang-glider, paraglider |
| 3 | 5 | Reserved |
| 3 | 6 | Unmanned aerial vehicle |
| 3 | 7 | Space or trans-atmospheric vehicle |
| 4 | 1 | Light (less than 7000 kg) |
| 4 | 2 | Medium 1 (between 7000 kg and 34000 kg) |

| | | |
|---|---|---|
| 4 | 3 | Medium 2 (between 34000 kg to 136000 kg) |
| 4 | 4 | High vortex aircraft |
| 4 | 5 | Heavy (larger than 136000 kg) |
| 4 | 6 | High performance (>5 g acceleration) and high speed (>400 kt) |
| 4 | 7 | Rotorcraft |

ADS-B has its own definition of wake categories, which is different from the ICAO wake turbulence category definition commonly used in aviation. The relationships of ICAO wake turbulence category (WTC) and ADS-B wake vortex category are:

- ICAO WTC L (Light) is equivalent to ADS-B (TC=4, CA=1).
- ICAO WTC M (Medium) is equivalent to ADS-B (TC=4, CA=2 or CA=3).
- ICAO WTC H (Heavy) or J (Super) is equivalent to ADS-B (TC=4, CA=5).

6.1.3. ADS-B Message Creation

A MATLAB function has been defined that generates an ADS-B message for aircraft identification and category. It takes inputs related to the aircraft and constructs a complete ADS-B message, including error checking bits (CRC). The code then converts this message into a Pulse Position Modulation (PPM) encoded signal, which is how ADS-B messages are typically transmitted.

Main Function:

ADSB_aircraftID_category.m

Input Parameters:

- DF: Downlink Format (5 bits)
- CA: Capability (3 bits)
- ICAO_hex: ICAO address in hexadecimal (24 bits)
- type_code: Message type code (5 bits)
- category: Aircraft category (3 bits)
- aircraft_id: Aircraft identification/callsign (up to 8 characters)

```

DF = 17;
CA = 5;
ICAO_hex = '4840D6';
type_code = 4;
category = 0;
aircraft_id = 'KLM1023';

```

Processing Stages:

1. Binary Conversion:

- Converts DF and CA to binary strings.
- Converts ICAO address from hex to binary.
- Converts type_code and category to binary.

```

DF_bin = dec2bin(DF, 5);
CA_bin = dec2bin(CA, 3);
ADS_B_message = [DF_bin CA_bin];
ICAO_bin = dec2bin(hex2dec(ICAO_hex), 24);
type_code_bin = dec2bin(type_code, 5);
category_bin = dec2bin(category, 3);

```

2. Aircraft ID Encoding:

- Pads aircraft_id to 8 characters.
- Converts each character to a 6-bit binary representation using charToBinary6bit.

```

aircraft_id_bin = '';
aircraft_id = pad(aircraft_id, 8, 'right', ' ');
for i = 1:length(aircraft_id)
    aircraft_id_bin = [aircraft_id_bin
    charToBinary6bit(aircraft_id(i))];
end

```

3. Message Assembly:

- Concatenates all binary parts to form the complete ADS-B message.

```

payload_bin = [type_code_bin category_bin aircraft_id_bin];
ADS_B_complete = [ADS_B_message ICAO_bin payload_bin];

```

4. CRC Calculation:

- Calculates CRC (Cyclic Redundancy Check) parity bits using ADSB_CRC.

```

ADS_B_hex = binaryToHexManual(ADS_B_complete);
[parity_bin, parity_hex] = ADSB_CRC(ADS_B_hex);
ADS_B_with_parity = [ADS_B_complete parity_bin];

```

5. Final Message Formation:

- Appends CRC bits to the message.

```

disp('ADS-B Message without Parity (Hexadecimal):');
disp(ADS_B_hex);
disp('Parity Bits (Hexadecimal):');
disp(parity_hex);

```



```
ADS_B_hex_final = binaryToHexManual(ADS_B_with_parity);
disp('Final ADS-B Message with Parity (Hexadecimal):');
disp(ADS_B_hex_final);
```

6. PPM Signal Generation:

- Generates a PPM encoded signal using generatePPM.

```
[ppm_signal, time_axis] = generatePPM(ADS_B_with_parity);
figure;
plot(time_axis * 1e6, ppm_signal);
ylim([-0.5, 1.5]);
grid on;
title('PPM Encoded ADS-B Message');
xlabel('Time ( μs)');
ylabel('Amplitude');
```

7. Saving PPM Signal to File

```
outputPath = 'C:\Users\rauna\OneDrive - UW\Study\Project\Summer_Internship\ADS-B\ADS-
    B_WaveGen\ADSB_Encode\CSV\ppm_signal.txt';
writematrix([time_axis', ppm_signal'], outputPath, 'Delimiter', 'tab');
disp(['PPM signal saved to: ', outputPath]);
```

Supporting Functions

1. charToBinary6bit

- Convert a character to its 6-bit binary representation.

```
function char_bin = charToBinary6bit(char)
if char >= 'A' && char <= 'Z'
    char_dec = double(char) - double('A') + 1;
elseif char >= '0' && char <= '9'
    char_dec = double(char) - double('0') + 48;
elseif char == ' '
    char_dec = 32;
else
    error('Invalid character for aircraft ID');
end
char_bin = dec2bin(char_dec, 6);
end
```

2. binaryToHexManual

```
function hex_str = binaryToHexManual(bin_str)
hex_str = '';
for i = 1:4:length(bin_str)
    nibble = bin_str(i:min(i+3, length(bin_str)));
    if length(nibble) < 4
        nibble = [nibble, repmat('0', 1, 4-length(nibble))];
    end
    dec_val = sum(2.^(3:-1:0) .* (nibble == '1'));
    if dec_val < 10
        hex_str = [hex_str, char(dec_val + '0')];
    else
        hex_str = [hex_str, char(dec_val - 10 + 'A')];
    end
end
end
```

3. ADSB_CRC

- Calculate CRC parity bits for the ADS-B message.

```
function [remainder_bin, remainder_hex] = ADSB_CRC(data_hex)
    generator_bin = '111111111111010000001001';
    generator = double(generator_bin) - '0';
    data_bin = hexToBinaryVector(data_hex, 88);
    data_bin = [data_bin, zeros(1, 24)];
    for i = 1:(length(data_bin) - length(generator) + 1)
        if data_bin(i) == 1
            data_bin(i:i+length(generator)-1) = xor(data_bin(i:i+length(generator)-1), generator);
        end
    end
    remainder = data_bin(end-(length(generator)-2):end);
    remainder_bin = num2str(remainder);
    remainder_bin = strrep(remainder_bin, ' ', '');
    remainder_hex = dec2hex(bin2dec(remainder_bin), 6);
end
```

4. generatePPM

- Generate a Pulse Position Modulation signal from the binary message.

```
function [ppm_signal, time_axis] = generatePPM(binary_message)
    bit_rate = 1000000;
    samples_per_second = 20000000;
    samples_per_bit = samples_per_second / bit_rate;
    pulse_width_samples = round(0.5 * samples_per_bit);
    ppm_signal = zeros(1, length(binary_message) * samples_per_bit);
    for i = 1:length(binary_message)
        if binary_message(i) == '0'
            start_index = round((i-1) * samples_per_bit) + 1;
        else
            start_index = round((i-1) * samples_per_bit + samples_per_bit/2) + 1;
        end
        end_index = min(start_index + pulse_width_samples - 1, length(ppm_signal));
        ppm_signal(start_index:end_index) = 1;
    end
    time_axis = (0:length(ppm_signal)-1) / samples_per_second;
end
```

Output

The output would look something like this:

```
ADSB-B Message without Parity (Hexadecimal):
8D4840D6202CC371C32CE0
Parity Bits (Hexadecimal):
576098
Final ADSB-B Message with Parity (Hexadecimal):
8D4840D6202CC371C32CE0576098
PPM signal saved to: C:\Users\rauna\OneDrive - UW\Study\Project\Summer_Internship\ADSB-B\ADSB-
B_WaveGen\ADSB_Encode\CSV\ppm_signal.txt
```

- **ADSB-B Message without Parity (Hexadecimal):**

This is the hexadecimal representation of the ADS-B message before the parity bits are added. It includes the Downlink Format, Capability, ICAO address, and the encoded aircraft identification.

- **Parity Bits (Hexadecimal):**

These are the CRC parity bits calculated for error detection. They are appended to the message to ensure integrity during transmission.

- **Final ADS-B Message with Parity (Hexadecimal):**

This is the complete ADS-B message including the parity bits, ready for transmission.

- **PPM signal saved notification:**

This confirms that the Pulse Position Modulation (PPM) encoded signal has been saved to a file at the specified location.

- **PPM Signal Plot:**

The graph shows the PPM encoded signal over time. Each pulse represents a bit in the ADS-B message, with the position of the pulse indicating whether it's a '0' or '1'.

- Additionally, the code will generate a plot of the PPM encoded signal, which would look something like this:

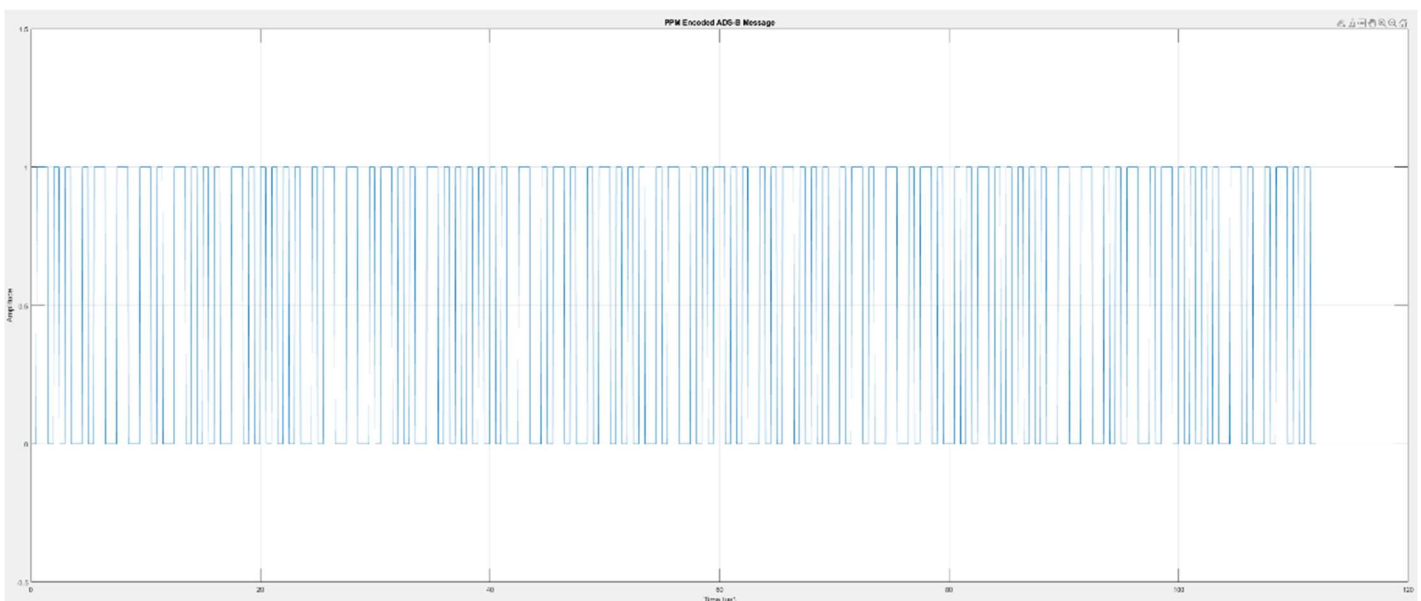


Fig. 11: ADSB_aircraftID_category message PPM waveform (MATLAB Output)

6.2. Aircraft Airborne Position

This type of message broadcasts position and altitude information of the aircraft. These messages can have Type Codes from 9 to 18 (encoded altitude represents barometric altitude) and Type Codes 20 to 22 (encoded altitude represents the GNSS altitude).

The specific TC value within this range indicates the Navigation Integrity Category (NIC), which represents the integrity of the horizontal position information. The structure of airborne messages is as follows –

```
+-----+-----+-----+-----+-----+-----+-----+
| TC, 5 | SS, 2 | SAF, 1 | ALT, 12 | T, 1 | F, 1 | LAT-CPR, 17 | LON-CPR, 17 |
+-----+-----+-----+-----+-----+-----+-----+
```

| FIELD | | MSG | ME | BITS |
|--|------------|-------|-------|------|
| Type Code <ul style="list-style-type: none"> 9–18: with barometric altitude. 20–22: with GNSS altitude | TC | 33-37 | 1-5 | 5 |
| Surveillance status <ul style="list-style-type: none"> 0: No condition 1: Permanent alert 2: Temporary alert 3: SPI condition | SS | 38-39 | 6-7 | 2 |
| Single Antenna Flag | SAF | 40 | 8 | 1 |
| Encoded altitude | ALT | 41-52 | 9-20 | 12 |
| Time | T | 53 | 21 | 1 |
| CPR Format <ul style="list-style-type: none"> 0: even frame 1: odd frame | F | 54 | 22 | 1 |
| Encoded Latitude | LAT | 55-71 | 23-39 | 17 |
| Encoded Longitude | LON | 72-88 | 40-56 | 17 |

6.2.1. *Surveillance Status (SS)*

Surveillance Status field provides crucial information about the aircraft's current operational state:

- Normal Operation (0): The aircraft is flying under routine conditions.
- Permanent Alert (1): This indicates an emergency, such as hijacking or severe technical failure. It's equivalent to setting the transponder to 7500 (hijacking) or 7700 (general emergency).
- Temporary Alert (2): This might be used when an aircraft changes its flight parameters suddenly, like a rapid descent or course change due to weather or traffic avoidance.
- SPI Condition (3): This is typically activated momentarily by the pilot to help air traffic controllers identify the aircraft on their radar screens.

6.2.2. *Single Antenna Flag (SAF)*

This flag is important for assessing the reliability of the position data:

- When set to 1 (single antenna), it indicates that the aircraft is using only one antenna for ADS-B transmission. This could potentially result in brief signal losses during maneuvers due to antenna shadowing.
- When set to 0 (dual antennas), it suggests more reliable signal transmission as the aircraft can switch between antennas to maintain constant coverage.

6.2.3. *Altitude (ALT)*

The altitude field is critical for air traffic management and collision avoidance:

- For TC 9-18, it represents the aircraft's barometric altitude, which is used by air traffic control for vertical separation.
- For TC 20-22, it represents GNSS altitude, which can be more accurate and is especially useful over oceans or in areas without reliable pressure readings.

6.2.4. *Time (T)*

The Time bit is used for synchronization purposes:

- It toggles between 0 and 1 each second, allowing receivers to detect if they've missed any messages and to synchronize their local clocks with the aircraft's clock.
- This is crucial for accurate velocity calculations and trajectory predictions.

6.2.5. *CPR Format (F)*

The CPR Format bit alternates between even (0) and odd (1) frames:

- This alternation is key to the Compact Position Reporting system, allowing high-precision position reporting within the limited bandwidth.

6.2.6. *Latitude (LAT) and Longitude (LON)*

These fields contain the encoded position of the aircraft:

- The actual values are not direct latitude and longitude coordinates, but rather encoded values that represent the aircraft's position within a global grid.
- When decoded, these provide the aircraft's precise location, which is essential for air traffic management, collision avoidance, and search and rescue operations in case of emergencies.

In practice, these fields work together to provide a comprehensive picture of an aircraft's status and position. For example, an air traffic controller might see an aircraft suddenly change its Surveillance Status to "Permanent Alert" while its altitude starts to decrease rapidly. This combination would immediately signal an emergency situation requiring immediate attention and possibly the initiation of search and rescue procedures.

6.2.7. *Compact Position Reporting (CPR)*

CPR is an encoding method that efficiently represents aircraft positions (latitude and longitude) within the constrained bandwidth of ADS-B messages. It balances between global position ambiguity and local position precision. The system alternates between two types of position messages, distinguished by the odd and even frame bit. Based on these messages, there are two methods to decode an airborne position:

1. Globally Unambiguous Position Decoding –

This method uses both odd and even ADS-B messages to determine an aircraft's position without any prior knowledge of its location.

- Requires two messages: one odd frame and one even frame.
- Messages should be received within 10 seconds of each other.
- Can decode positions up to ~1500 nautical miles from the receiver.

2. Locally Unambiguous Position Decoding –

This method uses a single ADS-B message (either odd or even) along with a known reference position to determine the aircraft's location.

- Requires only one message (odd or even).
- Needs a reliable reference position (usually from a previous global decode or known airport location).
- Faster and more efficient than global decoding.
- Suitable for continuous tracking once an initial position is established.
- Effective up to ~45 nautical miles from the reference position.

CPR is designed to transmit aircraft position data in a compact form, reducing the number of bits required while maintaining high precision. It achieves this by dividing the Earth's surface into grids and using two different message types (odd and even) to encode positions.

1. **Latitude Zones:** The Earth is divided into 15 latitude zones between the equator and each pole. The solid and dashed lines represent the latitude zone size of even and odd messages.

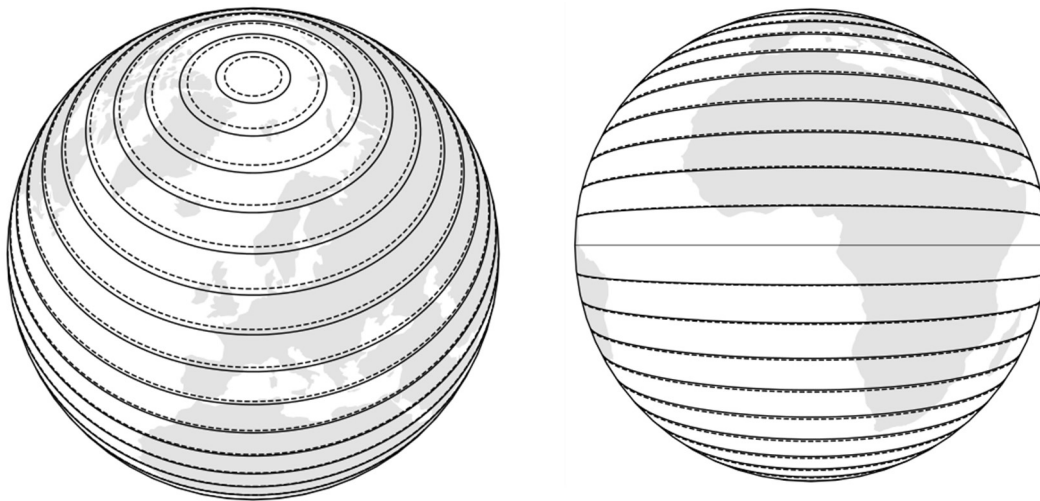


Fig.12: Latitude zones in compact position reporting, from different viewpoints.^{8,9}

⁸ Image Source: https://mode-s.org/decode/figures/adsb/cpr_lat_zone_high.png

⁹ Image Source: https://mode-s.org/decode/figures/adsb/cpr_lat_zone_low.png

2. **Longitude Zones:** The number of longitude zones varies with latitude. At the equator, there are more zones, which decrease in number as you move towards the poles due to convergence of meridians.

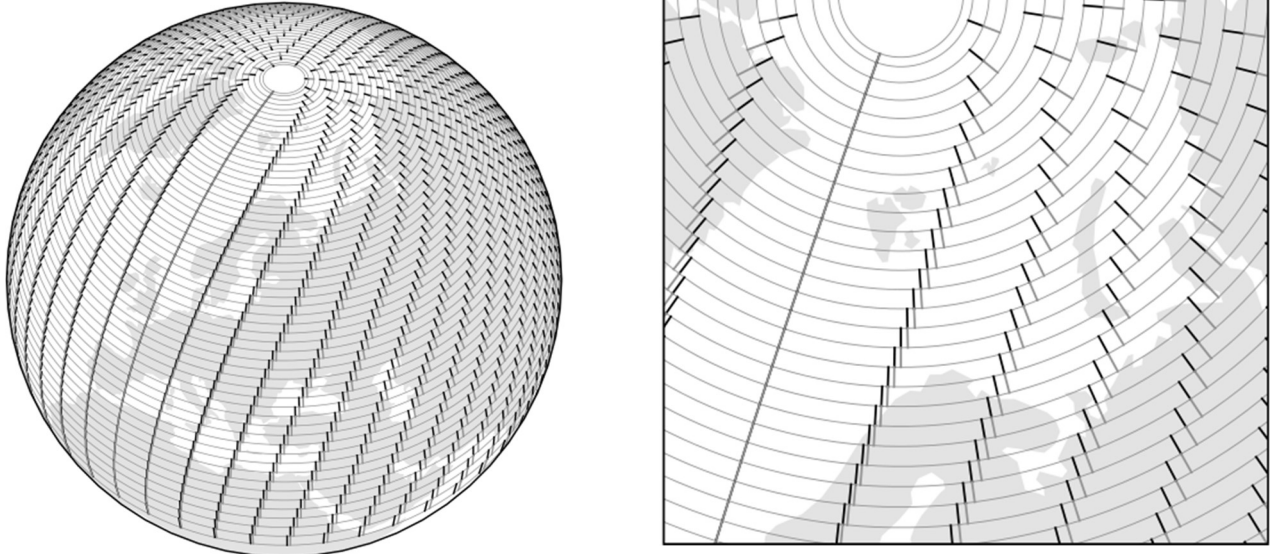


Fig.13: Longitude zones in compact position reporting, from different viewpoints.^{10, 11}

6.2.8. Globally Unambiguous Position Decoding –

Latitude Encoding

In each position message, bit 54 specifies if the message is odd or even frame. A value of 0 signifies an even message, whereas a value of 1 signifies an odd message. The fractions of the latitude and longitude within their respective grids are represented by bits 55–71 and bits 72–88, referred to as lat_{cpr} and lon_{cpr} .

$$lat_{cpr} = \frac{N_{cpr,lat}}{2^{17}} \qquad lon_{cpr} = \frac{N_{cpr,lon}}{2^{17}}$$

For even and odd messages, the **latitude zone sizes** are defined as follows:

¹⁰ Image Source: https://mode-s.org/decode/figures/adsb/cpr_lon_zone_full.png

¹¹ Image Source: https://mode-s.org/decode/figures/adsb/cpr_lon_zone_zoom.png

$$dLat_{\text{even}} = \frac{360^\circ}{4N_z} \quad dLat_{\text{odd}} = \frac{360^\circ}{4N_z - 1}$$

Here, N_z represents the number of latitude zones between the equator and a pole. $N_z = 15$ in Mode S communication.

Calculate **Latitude Zone Index**:

$$j = \text{floor} \left(59 \cdot \text{lat}_{\text{cpr},\text{even}} - 60 \cdot \text{lat}_{\text{cpr},\text{odd}} + \frac{1}{2} \right)$$

Here, *floor* function $\text{floor}(x)$ returns the greatest integer value k , where $k \leq x$. For example:

$$\begin{aligned} \text{floor}(4.6) &= 4 \\ \text{floor}(-7.6) &= -8 \end{aligned}$$

Based on both even and odd frames, two **latitudes** are as follows:

$$\begin{aligned} \text{lat}_{\text{even}} &= dLat_{\text{even}}(\text{mod}(j, 60) + \text{lat}_{\text{cpr},\text{even}}) \\ \text{lat}_{\text{odd}} &= dLat_{\text{odd}}(\text{mod}(j, 59) + \text{lat}_{\text{cpr},\text{odd}}) \end{aligned}$$

In the southern hemisphere, the results from the earlier calculations fall between 270 and 360 degrees. Therefore, it is necessary to adjust the latitude to ensure it remains within the range of $[-90, +90]$ using these equations:

$$\begin{aligned} \text{lat}_{\text{even}} &= \text{lat}_{\text{even}} - 360, \text{ if } \text{lat}_{\text{even}} \geq 270 \\ \text{lat}_{\text{odd}} &= \text{lat}_{\text{odd}} - 360, \text{ if } \text{lat}_{\text{odd}} \geq 270 \end{aligned}$$

The final latitude is chosen according to the time stamps of the messages as:

$$\text{lat} = \begin{cases} \text{lat}_{\text{even}} & \text{if } T_{\text{even}} \geq T_{\text{odd}} \\ \text{lat}_{\text{odd}} & \text{otherwise} \end{cases}$$

The current latitude is the most recent of these two latitudes.

Note - Before calculating the longitude, it's important to determine $NL(\text{lat}_{\text{even}})$ and $NL(\text{lat}_{\text{odd}})$ and verify that they match. If they differ, it indicates that the messages originate from different longitude zones, making it impossible to accurately compute the global position. In such cases, decoding should be halted until a pair of messages from the same latitude zone is received. This scenario typically occurs when an aircraft is crossing the boundaries between longitude zones. Calculation of NL values are discussed on the next page.

The encoding process in the ADS-B system essentially reverses the decoding process to create messages that efficiently convey an aircraft's position.

Longitude Encoding

Calculate **Longitude Index, m** :

$$m = \text{floor} \left(\text{lon}_{\text{cpr},\text{even}} \cdot [NL(lat) - 1] - \text{lon}_{\text{cpr},\text{odd}} \cdot NL(lat) + \frac{1}{2} \right)$$

Here, $NL(lat)$ is the Longitude zone number, which is defined as:

$$NL(lat) = \text{floor} \left\{ \frac{2\pi}{\cos^{-1} \left[1 - \frac{1 - \cos \left(\frac{\pi}{2N_z} \right)}{\cos^2 \left(\frac{\pi}{180} \cdot lat \right)} \right]} \right\}$$

For latitudes that are close to the equator or the poles, the following values are used:

| | | |
|-----------|----|---------|
| lat = 0 | -> | NL = 59 |
| lat = +87 | -> | NL = 2 |
| lat = -87 | -> | NL = 2 |
| lat > +87 | -> | NL = 1 |
| lat < -87 | -> | NL = 1 |

Calculate the **longitude zone size**, which is dependent on the latitude.

For even and odd messages, the **number of longitude zones** is defined as:

$$n_{\text{even}} = \max[NL(lat), 1] \quad n_{\text{odd}} = \max[NL(lat - 1), 1]$$

The **longitude zone sizes** are defined as:

$$d\text{Lon}_{\text{even}} = \frac{360^\circ}{n_{\text{even}}} \quad d\text{Lon}_{\text{odd}} = \frac{360^\circ}{n_{\text{even}}}$$

The **longitude** is calculated as:

$$\text{lon}_{\text{odd}} = d\text{Lon}_{\text{odd}} (\text{mod}(m, n_{\text{odd}}) + \text{lon}_{\text{cpr},\text{odd}})$$

$$\text{lon}_{\text{even}} = d\text{Lon}_{\text{even}} (\text{mod}(m, n_{\text{even}}) + \text{lon}_{\text{cpr},\text{even}})$$

Position messages are often converted from $[0^\circ, 360^\circ]$ to $[-180^\circ, +180^\circ]$

$$lon = lon - 360, \quad \text{if } lon \geq 180$$

Altitude Encoding

The aircraft's altitude can be decoded from a single position message, whether it is an even or odd type. However, the decoding method varies based on the message's type code:

- For TC=9–18: The message contains the airborne position with barometric altitude encoded in feet.

For barometric altitude, the 8th bit in the 12-bit altitude field is known as the Q bit. This bit specifies whether the altitude is encoded in increments of 25 feet or 100 feet. When $Q = 1$, the altitude is encoded in 25-foot increments. By excluding the Q bit, the altitude can be calculated as:

$$h = (25 N - 1000) \text{ feet}$$

- For TC=20–22: The message includes the airborne position with GNSS height encoded in meters.

The GNSS height is encoded using a 12-bit field. This height is determined from global positioning satellites, and the decimal value of the entire 12-bit field represents the aircraft's altitude in meters.

6.2.9. ADS-B Message Creation

A MATLAB function has been defined that generates an ADS-B message for encoding airborne position. It takes inputs related to the aircraft position and constructs a complete ADS-B message, including error checking bits (CRC). The code provides the odd and even message frame, marking the most recent one.

Main Function

ADSB_encode_airbornePosition.m

Input Parameters:

- DF: Downlink Format (5 bits)
- CA: Capability (3 bits)

- ICAO_hex: ICAO address in hexadecimal (24 bits)
- type_code: Message type code (5 bits)
- Latitude
- Longitude
- Altitude (in feet)
- Timestamps (t0 and t1)
- surveillanceStatus
- singleAntennaFlag

```

1. latitude = 52.2572;
2. longitude = 3.91937;
3. altitude = 38000;
4. t0 = 1457996402;
5. t1 = 1457996400;
6. typeCode = 19;
7. surveillanceStatus = 0;
8. singleAntennaFlag = 1;
9. DF = 17;
10. CA = 5;
11. ICAO = '40621D';

```

Processing Stages in the Main function:

1. Constants and Initial Calculations:

- **NZ Constant:** The constant NZ is set to 15, which is used in calculations related to latitude zones.
- **Latitude Zone Sizes:** dlat0 and dlat1 are calculated to define the size of latitude zones for even and odd CPR frames, respectively.

```

% Constants
NZ = 15;

% Calculate dlat values
dlat0 = 360 / (4 * NZ);
dlat1 = 360 / (4 * NZ - 1);

```

2. CPR Latitude and Longitude Calculation

- **Latitude CPR:** The function calculates lat_cpr0 and lat_cpr1 by normalizing the latitude within its respective zone size for even and odd frames.
- **Longitude Zones:** Calls calculateNL to determine the number of longitude zones (NL) based on latitude. Ensures NL is at least 1.
- **Longitude CPR:** Computes lon_cpr0 and lon_cpr1 similarly to latitudes, using the calculated longitude zone size (dlon).

```
% Calculate CPR latitudes
lat_cpr0 = mod(latitude, dlat0) / dlat0;
lat_cpr1 = mod(latitude, dlat1) / dlat1;

% Calculate NL and dlon
NL = calculateNL(latitude);
if NL < 1
    NL = 1;
end
dlon = 360 / NL;

% Calculate CPR longitudes
lon_cpr0 = mod(longitude, dlon) / dlon;
lon_cpr1 = mod(longitude, dlon) / dlon;
```

3. Altitude Encoding

- The function calls encodeAltitude, which converts the altitude from feet into a 12-bit binary string suitable for ADS-B messages.

```
% Encode altitude (assuming altitude is in feet)
alt_enc = encodeAltitude(altitude);
```

4. Binary Conversion of CPR Values

- Convert the normalized CPR values into 17-bit binary strings by scaling them with 217217 and rounding to the nearest integer.

```
% Convert CPR to binary
lat_cpr0_bin = dec2bin(round(lat_cpr0 * 2^17), 17);
lon_cpr0_bin = dec2bin(round(lon_cpr0 * 2^17), 17);
lat_cpr1_bin = dec2bin(round(lat_cpr1 * 2^17), 17);
lon_cpr1_bin = dec2bin(round(lon_cpr1 * 2^17), 17);
```

5. Message Field Construction and Assembly

- Constructs two ME fields (me0, me1) by concatenating binary representations of type code, surveillance status, single antenna flag, encoded altitude, and CPR coordinates.
- Converts ME fields to hexadecimal using bin2hex.
- Combines DF (Downlink Format) and CA (Capability) into a single byte in hexadecimal format.
- Constructs full messages (message0, message1) by concatenating DFCA hex value, ICAO address, and ME hex values.

```
% Construct ME fields (56 bits each) with type code, surveillance status, and single antenna flag
me0 = [dec2bin(typeCode, 5) dec2bin(surveillanceStatus, 2) dec2bin(singleAntennaFlag, 1) alt_enc lat_cpr0_bin lon_cpr0_bin];
me1 = [dec2bin(typeCode, 5) dec2bin(surveillanceStatus, 2) dec2bin(singleAntennaFlag, 1) alt_enc lat_cpr1_bin lon_cpr1_bin];
% Convert ME fields to hexadecimal
me0_hex = bin2hex(me0);
```

```

me1_hex = bin2hex(me1);
% Combine DF and CA into a single byte and convert to hex
DFCA_bin = [dec2bin(DF, 5) dec2bin(CA, 3)];
DFCA_hex = dec2hex(bin2dec(DFCA_bin), 2); % Convert to 2-digit hex
% Construct the full message with DF, CA, and ICAO
message0 = [DFCA_hex ICAO me0_hex];
message1 = [DFCA_hex ICAO me1_hex];

```

6. Determine Most Recent Message

- Compares timestamps (t_0, t_1) to determine which message (ME0 or ME1) was generated more recently. This helps in identifying which message should be prioritized or considered current in real-time applications.

```

% Determine the most recent message
if t0 >= t1
    mostRecent = 'ME0 is the most recent message';
else
    mostRecent = 'ME1 is the most recent message';
end

```

Supporting Functions

1. calculateNL

- Extreme Latitudes:** If the latitude is near the poles (≥ 87 degrees), it sets NL to 2 because longitude zones become very narrow and less significant.
- Equator Handling:** At the equator, where latitude is 0, it sets NL to 59, reflecting the maximum number of zones.
- General Calculation:** For other latitudes, it uses a trigonometric formula to calculate NL, which accounts for the varying width of longitude zones as latitude changes.

```

function NL = calculateNL(lat)
% Calculate the number of longitude zones
if abs(lat) > 87
    NL = 1;
elseif abs(lat) < -87
    NL = 1;
elseif abs(lat) == -87
    NL = 2;
elseif abs(lat) == 87
    NL = 2;
elseif lat == 0
    NL = 59;
else
    NZ = 15;
    NL = floor(2*pi / (acos(1 - (1-cos(pi/(2*NZ)))) / (cos(pi/180*abs(lat))^2)));
end
end

```

2. encodeAltitude

- Range Limiting: Limits altitude to between -1000 and 50175 feet, which are typical bounds for ADS-B altitude encoding.
- Quantization: Converts altitude into discrete steps of 25 feet by adding an offset and dividing by 25.
- Binary Encoding: Converts the quantized value into a 12-bit binary string using dec2bin.

```
function alt_enc = encodeAltitude(altitude_ft)
    % Encode altitude according to ADS-B format
    altitude_ft = max(min(altitude_ft, 50175), -1000); % Limit range
    N = floor((altitude_ft + 1000) / 25);

    alt_enc = dec2bin(N, 12);
end
```

Outputs

The output would look something like this:

```
message0: '8D40621D265862D690C8ACF9EA27'
message1: '8D40621D26586241ECC8ACDF67B5'
mostRecent: 'ME0 is the most recent message'
```

1. message0:

- This is the first ADS-B message encoded in hexadecimal format. It represents the airborne position information using Compact Position Reporting (CPR) for an even frame.
- The message includes several components: Downlink Format (DF), Capability (CA), ICAO address, ME field (which contains type code, surveillance status, single antenna flag, encoded altitude, and CPR-encoded latitude and longitude), and a CRC checksum to ensure data integrity.

2. message1:

- This is the second ADS-B message, also in hexadecimal format. It represents the airborne position information using CPR for an odd frame.

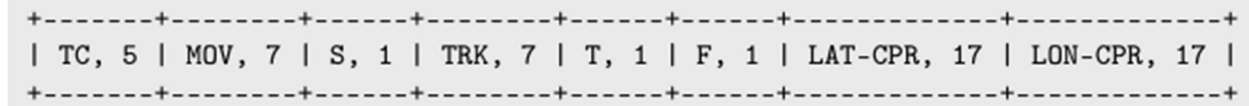
- Similar to message0, it contains DF, CA, ICAO address, ME field with the same components but with CPR values calculated differently for odd frames, and a CRC checksum.

3. mostRecent:

- This is a string indicating which of the two messages (ME0 or ME1) is more recent based on the provided timestamps (t0 and t1).
- If t0 (timestamp for message0) is greater than or equal to t1 (timestamp for message1), it returns 'ME0 is the most recent message'. Otherwise, it returns 'ME1 is the most recent message'.

6.3. Aircraft Surface Position

Surface position messages in ADS-B use Type Codes (TC) 5 through 8. The 56-bit Message Element (ME) field of an ADS-B message contains the following information for surface position.



The Type Code (TC) indicates the level of position uncertainty. TCs 5 through 8 are used for surface messages, with lower numbers representing higher accuracy. TC 5 signifies the highest precision with a Horizontal Containment Radius Limit (RC) < 7.5 meters, while TC 8 represents the lowest precision with $RC \geq 185.2$ meters.

| FIELD | | MSG | ME | BITS |
|--|-----|-------|-------|------|
| Type Code | TC | 33-37 | 1-5 | 5 |
| Movement | MOV | 38-44 | 6-12 | 7 |
| Status for ground track <ul style="list-style-type: none"> 0: Invalid 1: Valid | S | 45 | 13 | 1 |
| Ground Track | TRK | 14 | 9-20 | 7 |
| Time | T | 53 | 21 | 1 |
| CPR Format <ul style="list-style-type: none"> 0: even frame 1: odd frame | F | 54 | 22 | 1 |
| Encoded Latitude | LAT | 55-71 | 23-39 | 17 |
| Encoded Longitude | LON | 72-88 | 40-56 | 17 |

6.3.1. Movement (MOV)

The movement field in ADS-B surface position messages represent the aircraft's ground speed using a non-linear encoding scheme. This encoding method employs varying levels of quantization, allowing for more precise representation of slower speeds compared to higher speeds.

| Encoded Speed | Ground Speed Range | Quantization |
|---------------|---------------------------|----------------|
| 0 | Speed not available | |
| 1 | Stopped ($v < 0.125$ kt) | |
| 2-8 | $0.125 \leq v < 1$ kt | 0.125 kt steps |

| | | |
|---------|--|----------------------|
| 9-12 | $1 \text{ kt} \leq v < 2 \text{ kt}$ | 0.25 kt steps |
| 13-38 | $2 \text{ kt} \leq v < 15 \text{ kt}$ | 0.5 kt steps |
| 39-93 | $15 \text{ kt} \leq v < 70 \text{ kt}$ | 1 kt steps |
| 94-108 | $70 \text{ kt} \leq v < 100 \text{ kt}$ | 2 kt steps |
| 109-123 | $100 \text{ kt} \leq v < 175 \text{ kt}$ | 5 kt steps |
| 124 | $v \geq 175 \text{ kt}$ | |
| 125-127 | Reserved | |

6.3.2. Ground Track

The ground track information in ADS-B surface position messages is encoded using a 7-bit field, with its validity indicated by a status bit. When this status bit is set to 1, the ground track is considered valid and is encoded with a resolution of 2.8125 degrees (360/128).

The reference point for the ground track is true north, represented by 0 degrees.

Ground track (χ) is calculated as:

$$\chi = \frac{360n}{128} \text{ degrees}$$

If the status bit is set to 0, any data in the ground track field should be considered not valid.

6.3.3. Position

ADS-B surface position messages employ the Compact Position Reporting (CPR) format for encoding latitude and longitude, similar to airborne messages. However, the decoding process for surface positions has some unique characteristics.

The latitude zone size for surface positions is reduced to one-quarter of that used in airborne messages.

$$dLat_{even} = \frac{90^\circ}{4N_z}$$

$$dLat_{odd} = \frac{90^\circ}{4N_z - 1}$$

Where N_z represents the number of latitude zones. $N_z = 15$ in Mode S communication.

The longitude zone size is also smaller for surface positions.

$$dLon_{even} = \frac{90^\circ}{n_{even}}$$

$$dLon_{odd} = \frac{90^\circ}{n_{odd}}$$

6.3.4. ADS-B Message Creation

Like airborne position messages, A MATLAB function has been defined that generates an ADS-B message for encoding surface position. It takes inputs related to the aircraft position on the surface and constructs a complete ADS-B message, including error checking bits (CRC). The code provides an odd and even message frame, marking the most recent one.

Main Function

ADSB_encode_surfacePosition.m

Input Parameters:

- DF: Downlink Format (5 bits)
- CA: Capability (3 bits)
- ICAO_hex: ICAO address in hexadecimal (24 bits)
- type_code: Message type code (5 bits)
- Latitude
- Longitude
- groundTrackStatus
- movementSpeed (in knots)
- trackAngle (in degrees)
- refLat (of nearest airport)
- refLon (of nearest airport)
- Timestamps (t0 and t1)

```
% typeCode = 7;
% groundTrackStatus = 1;
% movementSpeed = 17;
% trackAngle = 92.8125;
% latitude = 4.73473;
% longitude = 4.375;
% refLat = 51.990;
% refLon = -122.3331;
% t0 = 1457996410;

% Type code for surface position
% Ground track status (1 for valid, 0 for invalid)
% Speed in knots
% Track angle in degrees
% Latitude in degrees (example: Seattle, WA)
% Longitude in degrees (example: Seattle, WA)
% Reference latitude (example: SeaTac, WA)
% Reference longitude (example: SeaTac, WA)
```

```

% t1 = 1457996412;
% DF = 17;
% CA = 4;
% ICAO = '484175';
% Downlink Format
% Capability
% ICAO address (hexadecimal)

```

Processing Stages in the Main function:

1. CPR Encoding –

- Calculating latitude zone sizes for even and odd frames
- Computes CPR latitudes and longitudes

```

dLatEven = 90 / (4 * NZ);
dLatOdd = 90 / (4 * NZ - 1);
latCPREven = mod(latitude, dLatEven) / dLatEven;
latCPROdd = mod(latitude, dLatOdd) / dLatOdd;

```

2. Movement Speed Encoding –

- Converts speed to a 7-bit encoded value
- Uses different encoding rules for various speed ranges

```

function movEnc = encodeMovementSpeed(movementSpeed)
% Encode movement speed according to ADS-B format
if movementSpeed < 0.125
    movEnc = dec2bin(1, 7);
elseif movementSpeed < 1
    movEnc = dec2bin(2, 7);
elseif movementSpeed < 7
    movEnc = dec2bin(3 + floor(movementSpeed - 1), 7);
elseif movementSpeed < 15
    movEnc = dec2bin(9 + floor((movementSpeed - 7) / 2), 7);
elseif movementSpeed < 70
    movEnc = dec2bin(13 + floor((movementSpeed - 15) / 5), 7);
elseif movementSpeed < 130
    movEnc = dec2bin(39 + floor(movementSpeed - 70), 7);
elseif movementSpeed < 180
    movEnc = dec2bin(94 + floor((movementSpeed - 130) / 2), 7);
elseif movementSpeed < 310
    movEnc = dec2bin(109 + floor((movementSpeed - 180) / 5), 7);
else
    movEnc = dec2bin(124, 7);
end
end

```

3. Track Angle Encoding –

- Converts the track angle to a 7-bit representation

```

trackEnc = encodeTrackAngle(trackAngle);

function trackEnc = encodeTrackAngle(trackAngle)
    trackEnc = dec2bin(round(trackAngle * 128 / 360), 7);
end

```

4. Message Construction –

- Creates 56-bit ME fields for even and odd messages
- Includes type code, movement, ground track status, track angle, time flag, and CPR coordinates

```
meEven = [dec2bin(typeCode, 5) movEnc num2str(groundTrackStatus) trackEnc num2str(timeFlag) '0' latCPREvenBin
lonCPREvenBin];
meOdd = [dec2bin(typeCode, 5) movEnc num2str(groundTrackStatus) trackEnc num2str(timeFlag) '1' latCPROddBin
lonCPROddBin];
```

5. Full Message Assembly –

- Combines DF, CA, ICAO address with ME field
- Converts the entire message to hexadecimal format and performs CRC

```
% Combine DF and CA into a single byte and convert to hex DFCA_bin = [dec2bin(DF, 5) dec2bin(CA, 3)];
DFCA_hex = dec2hex(bin2dec(DFCA_bin), 2);
% Construct the full message with DF, CA, and ICAO msg0 = [DFCA_hex ICAO meEvenHex];
msg1 = [DFCA_hex ICAO meOddHex];
% Append CRC to the messages
[~, crcEvenHex] = ADSB_CRC(msg0);
[~, crcOddHex] = ADSB_CRC(msg1);
msg0 = [msg0 crcEvenHex];
msg1 = [msg1 crcOddHex];
```

6. Timestamp Comparison –

- Determines the most recent message (ME0 or ME1) based on timestamps

```
if t0 >= t1
    mostRecent = 'msg0 is the most recent message';
else
    mostRecent = 'msg1 is the most recent message';
end
```

The supporting functions used in this implementation (calculateNL, bin2hex and ADSB_CRC) are like those used for encoding airborne position messages. These functions handle tasks such as calculating the number of longitude zones, converting between binary and hexadecimal formats, performing CRC calculations, and encoding specific message components.

Output

The code generates two ADS-B messages (even and odd) for surface position reporting.

```
msg0: 8C48417538DA13858A126CABE85A
msg1: 8C48417538DA15323E11E81C4BB2
msg1 is the most recent message
mostRecent =
    'msg1 is the most recent message'
```

1. Message 0 (Even Frame): 8C48417538DA13858A126CABE85A

- DF (Downlink Format): 17 (10001)
- CA (Capability): 4 (100)
- ICAO: 484175
- ME (Message, Extended squitter): 38DA13858A126C
- Movement: DA
- CPR Latitude (Even): '58A12'
- CPR Longitude (Even): '6C'
- CRC: ABE85A

2. Message 1 (Odd Frame): 8C48417538DA15323E11E81C4BB2

- DF+CA: '8C' (same as message0)
- ICAO: 484175 (same as message0)
- ME: 38DA15323E11E8
- Movement: DA (same as message0)
- CPR Latitude (Odd): 23E11
- CPR Longitude (Odd): E8
- CRC: 1C4BB2

3. mostRecent: 'msg1 is the most recent message'

- This is determined by comparing timestamps t0 and t1

7. Exporting created Signal

The information contained in the ADS-B data blocks is modulated using the Pulse Position Modulation (PPM). PPM is a form of signal modulation in which the message information is encoded in the timing of the pulses in a signal. The amplitude and width of the pulse are kept constant, while the position of the pulse within a time frame is varied according to the modulating signal.

ADS-B (Automatic Dependent Surveillance-Broadcast) uses PPM for several reasons:

1. **Compatibility:** PPM is compatible with existing Mode S transponder systems, allowing for easier integration into current aviation infrastructure.
2. **Robustness:** PPM is relatively resistant to amplitude variations and noise, making it suitable for long-range transmissions in varying atmospheric conditions.
3. **Bandwidth Utilization:** PPM allows for efficient use of the available bandwidth in the 1090 MHz frequency used by ADS-B.

7.1. PPM Parameters used in ADS-B

ADS-B uses a specific implementation of PPM for encoding data at the physical layer.

1. **Bit Rate:** 1 Mbps (1 microsecond per bit)
2. **Pulse Width:** 0.5 microseconds
3. **Bit Encoding:**
 - '1' bit: Pulse in the first half of the 1 μ s bit period
 - '0' bit: Pulse in the second half of the 1 μ s bit period
4. **Preamble:** 8 μ s long, consisting of four pulses at specific positions (0 μ s, 1 μ s, 3.5 μ s, and 4.5 μ s)
5. **Message Length:** 112 bits (120 μ s including preamble) for extended squitter

7.2. Implementation in MATLAB

A MATLAB function has been developed to incorporate the PPM standards and encode the generated ADS-B Hex message into baseband signal.

7.2.1. Initialization and Parameter Setting

This section sets up the key parameters:

- The bit rate is set to 1 Mbps, matching ADS-B specifications.
- A 20 MHz sampling rate is used, providing 20 samples per bit for smooth signal representation.
- The pulse width is set to 0.5 μ s, half of the bit period.

```
bit_rate = 1000000; % 1 Mbps
samples_per_second = 20000000; % 20 MHz sampling rate
samples_per_bit = samples_per_second / bit_rate;
pulse_width_samples = round(0.5 * samples_per_bit); % 0.5  $\mu$ s pulse width
```

7.2.2. PPM Signal Generation

For each bit in the binary message:

- If the bit is '1', a pulse is placed in the first half of the bit period.
- If the bit is '0', a pulse is placed in the second half of the bit period.

```
ppm_signal = zeros(1, length(binary_message) * samples_per_bit);

for i = 1:length(binary_message)
    start_index = round((i-1) * samples_per_bit) + 1;
    mid_index = start_index + pulse_width_samples;
    end_index = start_index + samples_per_bit - 1;

    if binary_message(i) == 1
        % 1 bit: 0.5  $\mu$ s pulse followed by 0.5  $\mu$ s flat signal
        ppm_signal(start_index:mid_index-1) = 1;
    else
        % 0 bit: 0.5  $\mu$ s flat signal followed by 0.5  $\mu$ s pulse
        ppm_signal(mid_index:end_index) = 1;
    end
end
```

```
time_axis = (0:length(ppm_signal)-1) / samples_per_second;
```

7.2.3. Saving PPM Signal as Time-Value Pairs

This process involves:

- Data Preparation: The time axis and PPM signal are combined into a matrix.
- File Writing: The matrix is written to a tab-delimited text file.

- File Format: Each row in the file represents a sample, with the first column being the time and the second column being the signal value (0 or 1).
- Precision: The high sampling rate (20 MHz) ensures that the time-value pairs accurately represent the PPM signal, including the precise timing of pulse transitions.

```
outputPath = 'C:\Users\rauna\OneDrive - UW\Study\Project\Summer_Internship\ADS-B\ADS-  
B_WaveGen\PPM\CSV\ppm_signal.txt';  
writematrix([time_axis', ppm_signal'], outputPath, 'Delimiter', 'tab');
```

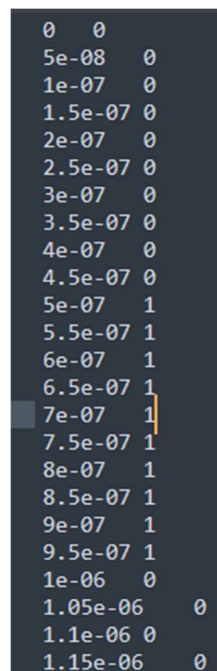
8. Interfacing with Cadence Virtuoso

This process involves generating a custom waveform in MATLAB, exporting it as a text file, and then using it as an input source in Cadence Virtuoso.

The first step in the interfacing process involves exporting the MATLAB-generated ADS-B waveform as a text file containing time-amplitude pairs. This format is chosen for its compatibility with Cadence Virtuoso's Piece-Wise Linear (PWL) voltage source.

```
% Save PPM signal to text file
outputPath = 'C:\Users\rauna\OneDrive - UW\Study\Project\Summer_Internship\ADS-B\ADS-B_WaveGen\PPM\CSV\ppm_signal.txt';
writematrix([time_axis', ppm_signal'], outputPath, 'Delimiter', 'tab');
disp(['PPM signal saved to: ', outputPath]);
```

This code creates a text file named “ppm_signal.txt” with two columns: time and signal amplitude. Each row represents a single point in the waveform.

A screenshot of a text file named 'ppm_signal.txt' showing time-amplitude pairs. The file contains 24 rows of data. The first 16 rows show a signal rising from 0 to 1 over time, with the amplitude staying at 1 for the last two rows of that segment. The next three rows show the signal returning to 0. The final row shows the signal rising slightly again.

| | |
|----------|---|
| 0 | 0 |
| 5e-08 | 0 |
| 1e-07 | 0 |
| 1.5e-07 | 0 |
| 2e-07 | 0 |
| 2.5e-07 | 0 |
| 3e-07 | 0 |
| 3.5e-07 | 0 |
| 4e-07 | 0 |
| 4.5e-07 | 0 |
| 5e-07 | 1 |
| 5.5e-07 | 1 |
| 6e-07 | 1 |
| 6.5e-07 | 1 |
| 7e-07 | 1 |
| 7.5e-07 | 1 |
| 8e-07 | 1 |
| 8.5e-07 | 1 |
| 9e-07 | 1 |
| 9.5e-07 | 1 |
| 1e-06 | 0 |
| 1.05e-06 | 0 |
| 1.1e-06 | 0 |
| 1.15e-06 | 0 |

Fig. 14: Screenshot of the exported text file showing time-amplitude pairs

In Cadence Virtuoso, the ADS-B waveform is imported using a PWL voltage source. A voltage source (vsource) is used from the analogLib library and configured as a PWL source by changing the “Type” to ‘pwl’ or ‘pwlF’ (PWL from file). The file path of the txt file is specified in the ‘File’ field.

Apply To: ☒ only current ☒ instance

Show: ☐ system ☒ user ☒ CDF

Browse Reset Instance Labels Display

| Property | Value | Display |
|---------------|-----------|---------|
| Library Name | analogLib | off |
| Cell Name | vsource | off |
| View Name | symbol | off |
| Instance Name | v3 | off |

Add Delete Modify

| User Property | Master Value | Local Value | Display |
|---------------|--------------|-------------|---------|
| lvignore | TRUE | | off |

CDF Parameter Value Display

DC voltage: off

Source type: ☒ pwl ☐ off

Frequency name 1: off

Waveform Entry Method: ☒ File ☐ Voltage/Time points off

Browse and select file: off

PWL File as Design Var? ☐ off

File name: mloads/ppm_signal(1).txt off

Delay time: off

DC offset: off

Time scale factor: off

Desired rms value: off

Breakpoints: off

Period: off

Period start time: off

Transition width: off

Type of rising & falling edge: off

Amplitude scale factor: off

Cosine Filter: off

Display small signal params: ☐ off

Fig. 15: Screenshot of PWL voltage source properties dialog in Cadence Virtuoso

To ensure the ADS-B waveform has been correctly imported, a simple transient analysis plots this input waveform and enables us to compare this with the MATLAB generated waveform.

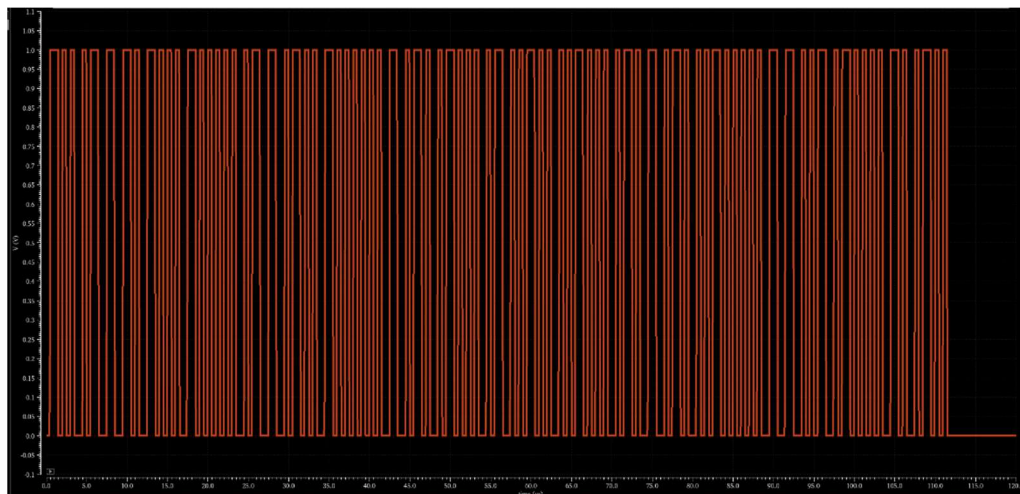


Fig. 16: Cadence-simulated waveform

9. ADS-B Signal Processing in Multi-Band Transceiver Chain

9.1. Input Baseband Signal

The ADS-B message, generated in MATLAB, is imported into Cadence Virtuoso as I+ and I- components. These differential signals serve as the input baseband for the transceiver chain.

The PWL files contain time-amplitude pairs with a sampling rate of at least 20 MHz to accurately represent the 1 Mbps data rate.

9.2. Frequency Upconversion

The baseband signals undergo frequency upconversion to the ADS-B standard frequency of 1090MHz. This process involves mixing the baseband signal with a local oscillator (LO) signal.

9.3. Amplification and Transmission

The upconverted signal is amplified using a Power Amplifier (PA) to ensure adequate transmission power. The simulation is conducted under ideal conditions to assess performance without external interference.

9.4. Signal Reception

The transmitted signal is received by an ADS-B receiver, typically starting with a Low Noise Amplifier (LNA). The received signal is downconverted back to baseband for analysis.

10. Results and Simulation Constraints

The ADS-B signal generated in MATLAB was successfully imported into Cadence Virtuoso and simulated through the transceiver chain. The simulation results demonstrate that the PPM-encoded signal can be recovered at the output of the transceiver chain, validating the design's functionality.

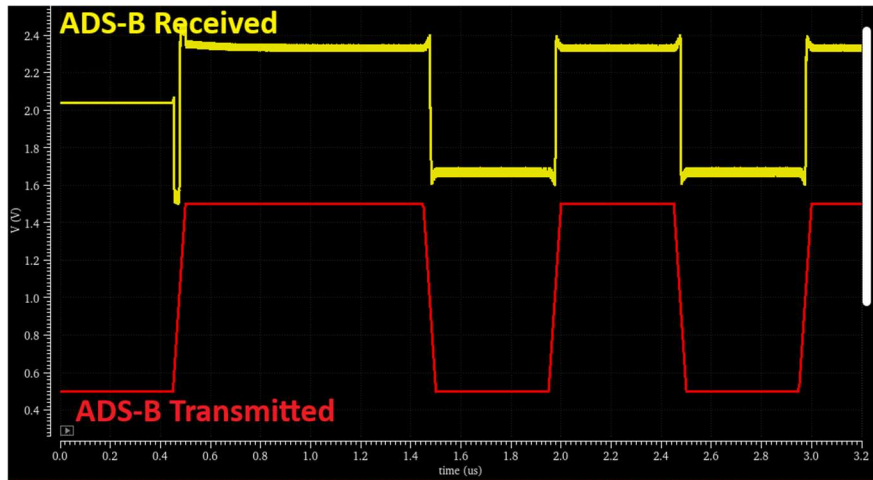


Fig. 17: Simulation Results

10.1. Simulation Constraints

While the complete ADS-B message spans 112 μs , significant computational limitations were encountered during simulation:

1. **Simulation Duration:** Only 3-4 μs of the signal could be simulated due to extensive processing time.
2. **Computational Intensity:** Each 1 μs of simulation required approximately 3 hours of processing time.
3. **Local Oscillator Synchronization:** The Local Oscillator (LO) requires 0.3-0.4 μs to synchronize, resulting in no meaningful output during this initial period.

10.2. Signal Decoding

Post-simulation, the recovered PPM signal is decoded to extract the baseband information. A MATLAB function was developed for this purpose:

```
function [decoded_hex, decoded_binary] = decodePPM(file_path)
% Read and plot PPM signal
data = readmatrix(file_path);
time = data(:, 1);
ppm_signal = data(:, 2);

% Decoding parameters
bit_duration = 1e-6; % 1 μs per bit
samples_per_bit = 20; % 20 MHz sampling rate
threshold = 0.5;

% Decode each bit
for bit_index = 1:length(decoded_binary)
    start_time = (bit_index - 1) * bit_duration;
    end_time = start_time + bit_duration;
    % ... [Bit decoding logic]
end

% Convert to hexadecimal
decoded_hex = binaryVectorToHex(decoded_binary);
end
```

This function performs the following key steps:

1. Reads the PPM signal from a text file.
2. Analyses pulse positions within each bit period.
3. Decodes the binary sequence based on pulse timing.
4. Converts the decoded binary to hexadecimal.

This decoder successfully extracts the binary and hexadecimal representations of the simulated signal, confirming that the essential information in our generated ADS-B waveform is preserved throughout the simulation process.

10.3. Signal Integrity

1. **Waveform Fidelity:** The imported signal maintains its Pulse Position Modulation (PPM) characteristics in the Cadence environment, confirming successful data transfer between MATLAB and Cadence.
2. **Time-Domain Accuracy:** The precise timing of pulses, crucial for PPM encoding, is preserved in the simulation, validating our export and import methodology.

The successful simulation and decoding of the initial 3-4 μs of the ADS-B signal provide strong evidence of the signal generation quality:

1. **Signal Consistency:** The PPM-encoded signal maintains its characteristics throughout the simulated duration.
2. **Precision:** The decoder's ability to accurately extract bit information validates the precision of our pulse positioning in the generated waveform.
3. **Sampling Rate Adequacy:** The successful decoding confirms that our chosen sampling rate (20 MHz) in the MATLAB generation process is sufficient for representing the 1 Mbps ADS-B data rate.

In conclusion, while computational constraints limited the Cadence simulation to 3-4 μs , the results provide strong confidence in the quality and accuracy of our MATLAB-generated ADS-B signal over its entire 112 μs duration. The successful import, simulation, and decoding of the initial portion validate our signal generation methodology.

11. Future Work

The successful generation, simulation, and initial testing of ADS-B signals lay a strong foundation for future research and development. Our next steps will focus on expanding the scope of our work and integrating it with broader avionics and communication systems. The following outlines our key objectives and rationale for future work:

11.1. ADS-B Signal Testing

We will conduct comprehensive transmission and reception tests of ADS-B signals through the integrated transceiver system. This phase will include:

1. Transmitting our generated ADS-B signals through the transceiver under various operational conditions.
2. Receiving and decoding ADS-B signals processed by the transceiver.
3. Analyzing signal integrity, bit error rates, and overall system performance across different environmental scenarios.

11.2. Transceiver Integration

Our primary objective is to interface our ADS-B signal generation and processing system with the multi-band transceiver being developed by Babak. This integration will involve:

1. Adapting our MATLAB-generated signals to meet the specific input requirements of the transceiver.
2. Developing robust interfaces between our signal processing algorithms and the transceiver's hardware.
3. Implementing real-time signal generation and processing capabilities to match the transceiver's operational parameters.

11.3. Validation and Refinement

A critical aspect of our future work will be to compare our experimental findings with the simulated results obtained from Cadence Virtuoso. This comparison will serve to:

1. Validate the accuracy of our simulation models and assumptions.
2. Identify any discrepancies between simulated and real-world performance.

3. Refine our signal generation algorithms and simulation parameters based on empirical data.
4. Optimize the transceiver design for improved ADS-B signal handling.

11.4. Signal Expansion Research

Building on our experience with ADS-B, we will expand our research to include RemoteID and 900 MHz ISM signal generation and processing:

1. Develop MATLAB models for generating RemoteID signals, adhering to the latest standards and protocols.
2. Create signal processing algorithms for 900 MHz ISM band communications, considering the diverse applications in this spectrum.
3. Conduct transmission and reception tests for both RemoteID and 900 MHz ISM signals through the multi-band transceiver.
4. Analyze the performance and compatibility of these signals within our integrated system.

12. Additional Work – AES-ECB Encryption for ADS-B messages

As an extension of our work on ADS-B signal generation and processing, we have explored an experimental approach to encrypt ADS-B messages before transmission. This section outlines a proof-of-concept implementation of AES encryption for ADS-B data, which could potentially enhance the security of ADS-B communications in sensitive situations.

12.1. Context and Motivation

ADS-B is an essential component of modern air traffic management systems. However, the open nature of ADS-B transmissions raises security concerns, as the unencrypted data can be intercepted and potentially exploited by malicious actors. To address these vulnerabilities, we propose an experimental method to encrypt ADS-B messages using the Advanced Encryption Standard (AES) algorithm. In this approach, the standard 112 μ s ADS-B message is encrypted and expanded to a 128 μ s encrypted data block before transmission. This encrypted data is then PPM (Pulse Position Modulation) encoded and exported as time-value pairs in a text file, ready for interfacing with Cadence Virtuoso for further simulation and analysis.

12.2. AES-ECB Encryption

The Advanced Encryption Standard (AES) is a symmetric block cipher widely used for secure data transmission. It encrypts data in fixed-size blocks of 128 bits using keys of 128, 192, or 256 bits. We specifically employ AES in Electronic Codebook (ECB) mode for this proof-of-concept. Key features of AES include:

1. Block size: 128 bits (16 bytes)
2. Key sizes: 128, 192, or 256 bits (we use 128-bit keys in this implementation)
3. Number of rounds: 10, 12, or 14 (depending on key size)
4. Symmetric Key Algorithm: Uses the same key for both encryption and decryption.

AES operates on a 4x4 array of bytes called the state, performing several transformations in each round:

1. **SubBytes**: A non-linear substitution step where each byte is replaced using an S-box to provide confusion.
2. **ShiftRows**: A transposition step where rows are cyclically shifted to provide diffusion.
3. **MixColumns**: A mixing operation that combines bytes within each column to further diffuse the plaintext.
4. **AddRoundKey**: Each byte of the state is combined with a byte from the round key using XOR.

ECB is one of the simplest modes of operation for block ciphers like AES. In ECB mode, each block of plaintext is encrypted independently using the same key, resulting in identical ciphertext blocks for identical plaintext blocks. Each block is processed independently without chaining or feedback mechanisms.

12.3. Java AES Function Implementation

We utilize Java's cryptography libraries for AES implementation due to their robustness and wide support. The `javax.crypto` package provides a high-level interface for encryption operations, ensuring secure and efficient implementation.

```
import javax.crypto.*
import javax.crypto.spec.*
cipher = javax.crypto.Cipher.getInstance('AES/ECB/PKCS5Padding');
secretKey = javax.crypto.spec.SecretKeySpec(key, 'AES');
cipher.init(javax.crypto.Cipher.ENCRYPT_MODE, secretKey);
```

This code snippet initializes the AES cipher in ECB mode with PKCS5 padding, using a 128-bit key.

The `AES_Encryption` function takes a hexadecimal string as input and returns the encrypted data along with encryption statistics. Here's a breakdown of its key components:

12.3.1. Input Processing

This code removes whitespace, ensures uppercase, and converts the hex input to a byte array.

```
hex_input = upper(strrep(hex_input, ' ', ''));  
input_bytes = uint8(hex2dec(reshape(hex_input, 2, [])));
```

12.3.2. Encryption

The entire input is encrypted in one operation.

```
encrypted_bytes = reshape(dec2hex(encrypted_bytes), 1, []);
```

12.3.3. Output Formatting

The encrypted bytes are converted back to a hexadecimal string.

```
encrypted_hex = reshape(dec2hex(encrypted_bytes), 1, []);
```

12.3.4. Statistics Calculation

The function calculates and reports various statistics about the input and encrypted data, including byte and hex lengths.

```
% Calculate statistics  
input_hex_length = length(hex_input);  
input_byte_length = length(input_bytes);  
encrypted_byte_length = length(encrypted_bytes);  
encrypted_hex_length = length(encrypted_hex);  
  
% Prepare the output string  
output_string = sprintf('Input Statistics:\n');  
output_string = [output_string sprintf(' Hex length: %d characters\n', input_hex_length)];  
output_string = [output_string sprintf(' Byte length: %d bytes\n', input_byte_length)];  
output_string = [output_string sprintf('Encryption Output:\n')];  
output_string = [output_string sprintf(' Encrypted byte length: %d bytes\n', encrypted_byte_length)];  
output_string = [output_string sprintf(' Encrypted hex length: %d characters\n', encrypted_hex_length)];  
output_string = [output_string sprintf('\nUnencrypted (HEX): %s\n', hex_input)];  
output_string = [output_string sprintf('Encrypted (HEX): %s', encrypted_hex)];
```

This experimental approach to ADS-B message encryption demonstrates the potential for enhancing the security of ADS-B communications.

References

- [1] M. S. M. L. V. & M. I. Strohmeier, "Realities and challenges of nextgen air traffic management: The case of ADS-B.," *IEEE Communications Magazine*, 52(5), 111-118., 2014.
- [2] J. V. H. E. J. & H. J. M. Sun, "PyModeS: Decoding Mode-S surveillance data for open air transportation research.," *IEEE Transactions on Intelligent Transportation Systems*, 21(7), 2777-2786, 2019.
- [3] M. S. M. L. V. M. I. & W. M. Schafer, "OpenSky: A large-scale ADS-B sensor network for research," *International symposium on information processing in sensor networks*, 2014.
- [4] S. B. H. a. D. Dawn, "A New Pathway Toward Implementing a Fully Integrated Band-Switchable CMOS Power Amplifier Utilizing Bit Optimized Reconfigurable Network (BORN)," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 30, no. 9, pp. 1294-1305, 2022.
- [5] S. B. H. a. D. Dawn, "A Fully-Integrated Band-Switchable CMOS Low Noise Amplifier," *IEEE Wireless and Microwave Technology Conference (WAMICON), Melbourne, FL, USA, 2023*, pp. 61-64, 2023.
- [6] W. A. H. a. K. N. Blythe, "Ads-b implementation and operations guidance document," *International Civil Aviation Organization*. , 2011.
- [7] D. a. B. C. Gray, "Privacy impact assessment (PIA) - federal aviation administration (FAA) privacy ICAO address system.," *U.S. Department of Transportation*. , 2019.
- [8] ICAO, "Technical provisions for mode s services and extended squitter.," *International Civil Aviation Organization*. , 2008.
- [9] antirez, "github/dump1090," [Online]. Available: <https://github.com/antirez/dump1090>.
- [10] J. Sun, "github.com/junzis/pyModeS," [Online]. Available: <https://github.com/junzis/pyModeS>.