Section: CS

Lowering Avionics Bus Trust: Moving ARINC 429 Bus Architecture Towards Zero Trust

Matthew Preston

Problem Statement:

ARINC 429 bus architecture, like most bus architectures, is insecure. Any command on the bus will be inherently trusted and executed by receiver LRUs. This means that compromising legacy systems and software can potentially cause catastrophic effects. Since 2022, the United States government has stated intent to securing its assets with Zero Trust principles. Therefore, moving ARINC 429 to zero trust without having to design and implement a whole new architecture or standard would help closer align the cybersecurity posture of critical infrastructure to this intent.

Solution Statement:

This research will simulate an ARINC 429 bus (as a digital twin) as well as implement an ARINC 429 traffic/rules-based intrusion detection system to engineer a defense system for ARINC 429 that helps move the ARINC 429 bus architecture towards zero trust. It will have the following deliverables:

- A simulation of an ARINC 429 bus architecture of and airplane
- A rules-based IDS that can log and flag ARINC 429 words.

My project will only move ARINC429 towards zero-trust and not achieve a 100% zero-trust architecture for ARINC429. If fully achieved for ARINC429 in the future, a Zero Trust architecture is the right approach for securing the ARINC429 bus protocol because it would shift the security posture from assuming words on the wire are trusted commands to double checking each command attempt. Given the critical nature of aviation systems and the increasing sophistication of cyber threats, Zero Trust would ensure that only authenticated and authorized devices can interact with the ARINC429 bus. By continuously monitoring and validating all interactions between LRUs, a full Zero Trust ARINC429 architecture would enhance the overall security posture of internal aviation communication systems.

My rules-based IDS is geared towards alerting an aircraft crew of a cyber threat. If it flags a word or sequence of words as malicious I envision it alerting someone (i.e. a pilot, or trained flight attendant) to let them know of a cyber-attack. This would then allow for better situational awareness and hopefully control of a pilot crew to then land in an emergency and also for investigators after the fact to learn what exactly happened.

Completed Tasks (Last 2 Week):

These past two weeks I:

- Completed all the functionality of the IDS to include:
  - Logging and Alerting words based on:
    - Bits
    - Data (i.e. 180 degs as lat/long, or 123 knots as airspeed etc)
    - Channel
    - Word Label (0o001 – 0o377).
    - Encoding type likelihood.
  - This took probably about 40-50 hours in total because of all the different word types and label edge cases. I also tested it before the evaluation plans for functionality. It is large and thus complicated.

- I also completed Evaluation Plans 1, 2, 3 and 4 from my last progress report. These took the better half of last week with the time off I had from the holiday weekend.

Tasks for the Next Project Report:

I will create the demo video, which may amount to some more coding of fitting things together and start the final report.

Questions I have or Issues I'm running into:

1. Mostly the largest issue I am running into is time. I am regularly spending over double the amount of time this course requires (20 hrs) to just get things working. This has been a week over week issue. I think now I should be done with everything I need for the demo and final report, so the problem has resolved itself. But it stemmed from taking on too much work at the beginning and overestimating my abilities.
2. One question is how would you like the evaluation results to be structured in the final report? Below I note some of the statistics and results in paragraph form, but I assume for the final report graphical displays are more ideal?

Methodology Paragraph Summary:

The process I will be employing is:

1. Plan structure of next component I need to build
2. Build component based on plan
3. Test component functionality by making test cases
4. Rework component based on test
5. Test component working with other components

Repeat until each component is finished.

Here's an example with the FMC.

- First I designed the component FMC, based on a description from UEI (similarly to Dnx-429-516). This is in the appendix as Figure 14.
- Then I coded each component in python to the "`LRU_FMC_Simulator.py`".
- Then I used my test file to test various functionality of it, i.e. the pilot input from my keys, the FIFO queue, the scheduler, the voltage generation, the word generation etc. From there I fixed what needed to be fixed.
- Finally, in my same test file, I will test interaction between it and other components, i.e. if the word sent can be received ok, and decoded ok by the other attached components in the picture.

Timeline:

Week #	**Description of Task**	**Status**
Week 1 Monday, 13 May 2024 – Sunday 19 May 2024	Research topic by reading articles. Focus the research on understanding how the ARINC 429 protocol works.	Completed
	Research topic by reading articles. Focus research on various bus architecture security solutions, to see what has been developed. This research has helped narrow down the architecture from the list of ARINC 429, CAN	Completed

	bus, 1553 bus, and ARINC 629 to just ARINC 429.	
	Research topic by reading articles. Focus the research on understanding zero trust cybersecurity, and if any zero trust implementation has been done for the above architectures. This helps inform a template for ARINC 429	Completed
	Draft Initial Proposal	Completed
Week 2 Monday, 20 May 2024 – Sunday 26 May 2024	Finalize Proposal	Completed
	Research topic by reading articles. Focus the research on any cybersecurity that has been done on ARINC 429, and if any of those help ARINC 429 implement zero trust principles/tenets.	Completed
	Simulate a receive LRU, the electronic engine control, that would be taking ARINC 429 commands and outputting actuations (for simulator).	Completed – Renamed to `full_authority_engine_control` as that was a more accurate LRU
	Start flight simulation software that should interface with LRU outputs. It should at this point just be an outline of the following functionality: - A tick/step-based time system that calculates the airplanes positional data from the last tick. Given the following attributes: altitude, x/y position, roll, yaw, pitch, forward velocity, and jet engine thrust, it should calculate the next x, y, altitude positional data. - Start at position 0, 0, 500 - Plot continuously the plan's positional data - A way to take in data from the actuators LRUs (jet engines, balancing LRUs for fins, etc) and translate that to the calculations. An outline here consists of creating the python file, putting in the math equations for steps/ticks for this, outlining a function that should take in	Completed

	data from an actuator and setting up the plotting given 3-d coordinates. This will be the codebase to test the ARINC429 simulation actuators from.	
Week 3 Monday, 27 May 2024 – Sunday 2 June 2024	Simulate a transmitter LRU, the Flight Management Computer. It should have the following features: - Multiple TX channels - Multiple RX channels - Basic flight functionality software that generates ARINC 429 words based on desired direction to go	Completed
	Finalize functionality for the flight simulator above.	Cancelled – based on peer feedback.
	Simulate a receiver LRU that will be the weight and balance system on the plane. It should output actuator data for the simulation.	Completed
Week 4 Monday, 3 June 2024 – Sunday 9 June 2024	Simulate a transmitter LRU, the GPS that reports actual positional x/y/altitude data. It should get this back from the simulator above.	Completed
	Simulate a receiver LRU, the radio management system.	Completed
	Test interaction and hookup between the electric engine control LRU, and weight and balance system to the simulator to make sure they can input and influence the motion of the plane.	Completed
Week 5 Monday, 10 June 2024 – Sunday 16 June 2024	Simulate a transmitter LRU, the ADIRU, and test its hookup to the flight simulator to make sure it can input data correctly.	Completed
	Create vulnerable program for FMC and a simple buffer overflow / rop hack for it to gain system access to the FMC.	Completed
	Create the orange ARINC429 bus.	Completed
	Create the blue ARINC429 bus.	Completed
Week 6 Monday, 17 June 2024 –	Create final report outline.	Completed
	Create the purple/channel B ARINC 429 bus	Completed
	Create the green/channel A ARINC 429 bus	Completed

Sunday 23 June 2024	Add functionality to the attack above that when system access is gained on the FMC, start transmitting ARINC 429 words from the FMC to the FAECs ~~EEC~~s that pitch the plane into a downward trajectory.	Completed
	Start the rules-based IDS system. Implement the functionality to it: - Create syntax for IDS rules - Create functionality for logging bus traffic	Completed
	Start logging ARINC 429 traffic for data collection.	Cancelled
Week 7 Monday, 24 June 2024 – Sunday 30 June 2024	Create first draft of final report	Not Started
	Add the following functionality to the IDS: - Ability to generate alarms based on transmission ID - Ability to generate alarms based on parity & parity correctness - Ability to generate alarms based on sign/statis matrix -> normal operation (N/S, E/W), functional test, failure warning, no computed data - Ability to generate alarms based on data (hopefully also granulate that based on flight directional data, etc. Combine with previous words to see if there is a suspicious word). - Ability to generate alarms based on source/destination field - Ability to generate alarms based on label (data type)	Completed
	Implement Step 2 from Evaluation Plan	Completed
Week 8 Monday, 1 July 2024 – Sunday 7 July 2024	Revise the first draft of the final report into second draft.	Not Started
	Create a mode in the IDS that can reset the bus upon any of the alarms from IDS	Cancelled
	Implement Evaluation Plan Step 3 and 4	Completed
Week 9 Monday, 8 July 2024 –	Create final demo presentation. It should include a demo of the simulation, attack, and defense implementations working with the attack.	Not Started

Sunday 14 July 2024		
Week 10 Monday, 15 July 2024 – Sunday 21 July 2024	Create / post final demo video.	Not Started
Week 11 Monday, 22 July 2024 – Thursday 25 July 2024	Finish project final report from second draft.	Not Started

Evaluation:

1. Correctness of the simulation: Ongoing – when creating a new LRU or python file, for each function that I write in that particular file, I will add 1 to 3 tests for that function. Since my code base is growing very large, I have to limit myself to a max of 3 tests per function for time management.
    a. Since my codebase is complete, vis-a-vis the final demo, this part of the evaulation testing is complete. The results is a large multi-thousand line python file called MegaTest.py that tests every function coded within my project. It uses pytest to assert that the test functions are passed.
2. Robustness of the IDS: These tests will be at receive 5 ARINC429 words from a given RX LRU (so FMC, GPS, or ADIRU). The speed of the bus will start slow – instead of ½ microsecond between voltage transmissions it will start at ½ second. Then the intervals will scale up logarithmic up in speed until it gets to ½ microsecond again. From there I will create a chart: the amount of words it's able to correctly receive and act upon (based on it's rules) per speed. Then I will repeat this test with various rules, from 1 to 10 rules.
    a. Unfortunately, unlike all the other tests, the IDS failed this evaluation. When slowed down to ½ to 0.05 second sampling rate, the IDS can 100% of the time receive the words correctly. At the 0.005-second sampling rate, it can get 2 or 3 out of the 5 words received correct. I also saw that adding rules also notably slow down the performance of the evaluation by a few microseconds, I haven't been able to measure how much it does yet in seconds. Let me know if that is a metric you want included in the final report. Any faster and the IDS doesn't keep up and doesn't register the words at all. I learned that choosing to write these programs in python, I had made a critical mistake. When TX'ing and RX'ing words, the program opens multiple threads. However, python is slow and multithreading is even slower. So this is a limitation (and lesson learned) on the software being used. I was afraid this may be the case since I saw the same results last progress report with my simulation having to slow it down, and thought I may have been able to fix it, but after banging my head against the wall these past two weeks trying to fix this and after some research I learned why the problem was so long. This will be included in my final report under lessons learned in more depth. For my other two tests instead of using the simulated wire class that I wrote (the BusQueue class from previous reports), I passed the word directly to the IDS "assuming" robustness so that I could have better evaluation results. For example, the next test is for checking if the IDS flags on the correct words, which would be incorrect if the IDS doesn't even get the correct word, etc.
3. IDS Correctness: I will use flight data found from NASA (https://c3.ndc.nasa.gov/dashlink/resources/664/) to plug into the ADIRU to simulate a snapshot

of a flight. This will start sending words to the FMC. Then I will use the IDS to log specific words from the FMC based on three rules.

   a. I was able to plug in data from the real word flight data and get the number of alerts and logs expected on that data. Specifically, I was able to stress test the IDS over 53,898,624 words sourced from the real word flight data such as latitude, longitude, airspeed, angle of attack, etc. I will put a copy of the python file below in the appendix. This test takes anywhere from 3 to 5 hours to complete.

4. IDS Detection of attackers: Repeat the test from above, but tailoring the word alerts based on if there is an attack (i.e. crazy weird angle of attack). Check that an attack I generate can be caught on the snapshot of flight data (i.e. message to be logged is "cyber-attack"). Repeat this for a few different flight datasets. Produce report based on 1: what the flight is, 2: if the IDS can correctly ID and attack.
   a. The IDS passed this test, by alerting on words where the indicated AoA was normal, but the FMC started going into a nosedive. I scraped data off of ADS-B exchange website for the flight data here to test on and created my attack scenario for this. The IDS was able to identify the attack.

Report Outline:

Please see other attached document submitted to Canvas, as I am using Latex.

References:

R. Vincent. "ARINC-429 RX Implementation in Labview FPGA." *Arinc-429 RX Implementation in LabVIEW FPGA*, NI Community, 28 Nov. 2023, https://forums.ni.com/t5/Example-Code/Arinc-429-Rx-Implementation-in-LabVIEW-FPGA/ta-p/3507624

aeroneous. "PyARINC429." *Discover PyARINC429, a simple Python module for encoding and decoding ARINC 429 digital information.* 17 Jul. 2018, https://github.com/aeroneous/PyARINC429

Peña, Lisa; and Shipman, Maggie. "Episode 64: Zero-Trust Cybersecurity for Vehicles." *Technology Today Podcast*, Southwest Research Institute, Feb. 2024, https://www.swri.org/podcast/ep64

"ARINC-429 with Cyber and Wirefault Protection" *ARINC-429 Solutions*. Sital Technology, https://sitaltech.com/arinc-429/

"Understanding Cyber Attacks on MIL-STD-1553 Buses" Sital Technology, https://sitaltech.com/understanding-cyber-attacks-on-mil-std-1553-buses/

"1553 Network and Cybersecurity Testing." Alta Data Technologies LLC, 19 Jan. 2021, https://www.altadt.com/wp-content/uploads/dlm_uploads/2020/10/1553-Network-and-Cybersecurity-Testing.pdf
Tilman, Bill. "Why You Need to Secure Your 1553 MIL-STD Bus and the Five Things You Must Have in Your Solution." Abaco Systems, 14 Dec. 2021, Original Link: https://abaco.com/blog/why-you-need-secure-your-1553-mil-std-bus-and-five-things-you-must-have-your-solution, Accessible Here: https://web.archive.org/web/20240223161240/https://abaco.com/blog/why-you-need-secure-your-1553-mil-std-bus-and-five-things-you-must-have-your-solution

Waldmann, B. "ARINC 429 Specification Tutorial." *Avionics Databus Solutions*, Version 2.2, AIM Worldwide, Jul. 2019, https://www.aim-online.com/wp-content/uploads/2019/07/aim-tutorial-oview429-190712-u.pdf, https://www.aim-online.com/products-overview/tutorials/arinc-429-tutorial/

"ARINC-429 tutorial: A Step-by-Step Guide." KIMDU Technologies, 26 Jun. 2023, https://kimdu.com/arinc-429-tutorial-a-step-by-step-guide/

"ARINC-429 Tutorial & Reference" *Understanding ARINC-429*, United Electronic Industries/AMETEK, https://www.ueidaq.com/arinc-429-tutorial-reference-guide

Biden, Joesph R. Jr. "Executive Order on Improving the Nation's Cybersecurity." *Briefing Room, Presidential Actions*, The White House, 12 May 2021, https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/

Rose, Scott; Borchert, Oliver; Mitchell, Stu; and Connelly, Sean. "Zero Trust Architecture." *NIST Special Publication 800-207*, National Institue of Standards and Technology, U.S. Department of Commerce, Aug. 2020, https://doi.org/10.6028/NIST.SP.800-207

Young, Shalanda D. "Moving the U.S. Government Toward Zero Trust Cybersecurity Principles" *MEMORANDUM FOR THE HEADS OF EXECUTIVE DEPARTMENTS AND AGENCIES*, Version M-22-09, Executive Office of the President; Office of Management and Budget, 26 Jan. 2022, https://whitehouse.gov/wp-content/uploads/2022/01/M-22-09.pdf

"Avionics Databus Tutorials." *Ballard Technology*, Astronics AES, https://www.astronics.com/avionics-databus-tutorials maewert. “Interfacing Electronic Circuits to Arduinos.” *Circuits; Arduino*, Autodesk Instructables, https://www.instructables.com/Interfacing-Electronic-Circuits-to-Arduinos/

Airlines Electronic Engineering Committee. “ARINC Specification 429 Part 1-17: Mark 33 – Digital Information Transfer System (DITS).” *ARINC Document*, Aeronautical Radio Inc. 17 May 2004, Original Link: https://read.pudn.com/downloads111/ebook/462196/429P1-17_Errata1.pdf , Accessible here: https://web.archive.org/web/20201013031536/https://read.pudn.com/downloads111/ebook/462196/429P1-17_Errata1.pdf

D. De Santo, C.S. Malavenda, S.P. Romano, C. Vecchio, “Exploiting the MIL-STD-1553 avionic data bus with an active cyber device.” *Computers & Security*, Volume 100, 2021, 102097, ISSN 0167-4048, https://doi.org/10.1016/j.cose.2020.102097. (https://www.sciencedirect.com/science/article/pii/S0167404820303709)

Gilboa-Markevich, N., Wool, A. (2020). “Hardware Fingerprinting for the ARINC 429 Avionic Bus.” In: Chen, L., Li, N., Liang, K., Schneider, S. (eds) Computer Security – ESORICS 2020. ESORICS 2020. Lecture Notes in Computer Science(), vol 12309. Springer, Cham. https://doi.org/10.1007/978-3-030-59013-0_3

Kiley, Patrick. “Investigating CAN Bus Network Integrity in Avionics Systems.” Rapid7, 30 Jul. 2019, https://www.rapid7.com/research/report/investigating-can-bus-network-integrity-in-avionics-systems/

Appendix

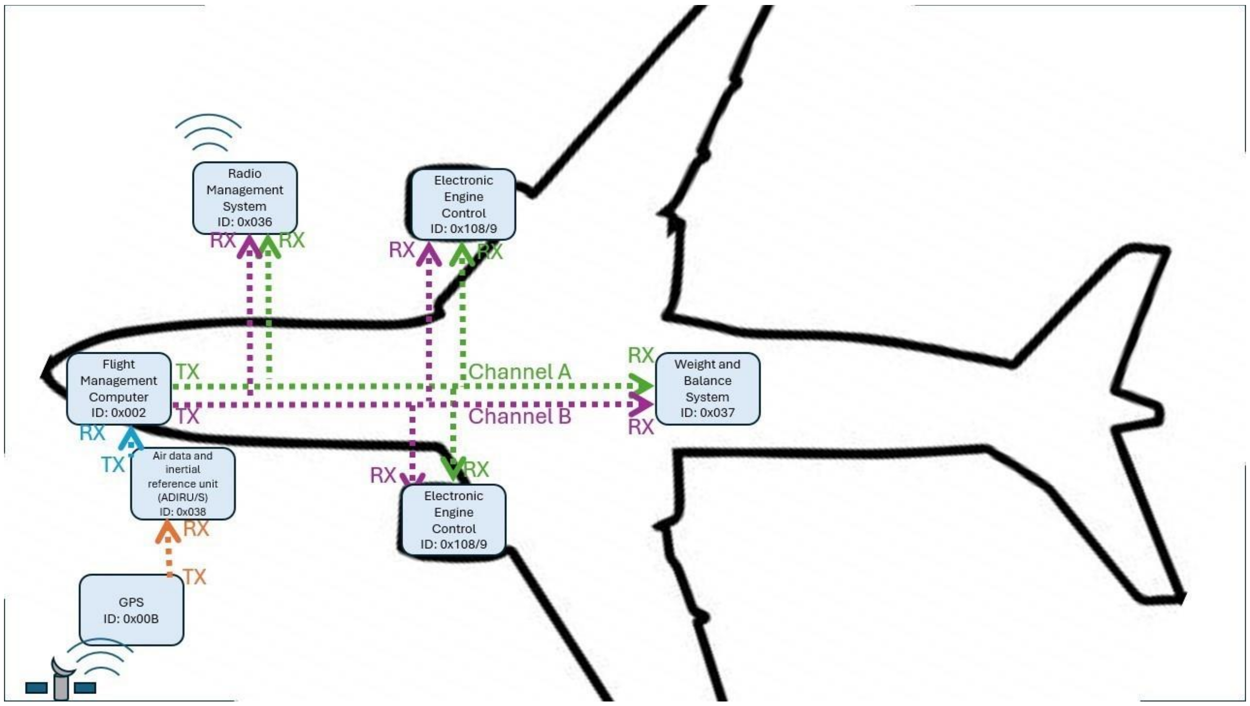

Radio Management System ID: 0x036
RX
RX
Electronic Engine Control ID: 0x108/9
RX
RX
Flight Management Computer ID: 0x002
TX
TX
Channel A
Channel B
RX
RX
Weight and Balance System ID: 0x037
RX
TX
Air data and inertial reference unit (ADIRU/S) ID: 0x038
RX
RX
Electronic Engine Control ID: 0x108/9
RX
TX
GPS ID: 0x00B

Figure 1: System Model

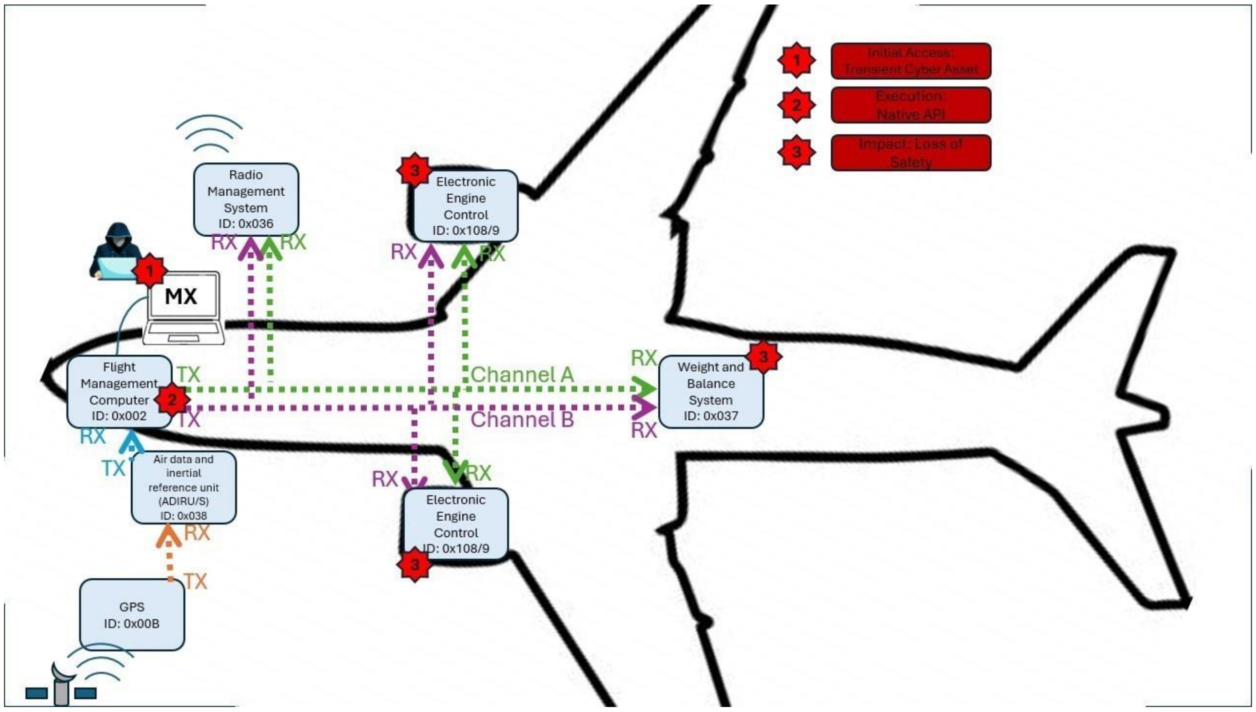

1
Initial Access: Transient Cyber Asset
2
Execution: Native API
3
Impact: Loss of Safety
MX
Radio Management System ID: 0x036
RX
RX
3
Electronic Engine Control ID: 0x108/9
RX
RX
1
Flight Management Computer ID: 0x002
2
TX
TX
Channel A
Channel B
RX
RX
Weight and Balance System ID: 0x037
3
RX
TX
Air data and inertial reference unit (ADIRU/S) ID: 0x038
RX
RX
Electronic Engine Control ID: 0x108/9
3
RX
TX
GPS ID: 0x00B

Figure 2: Threat Attack Model

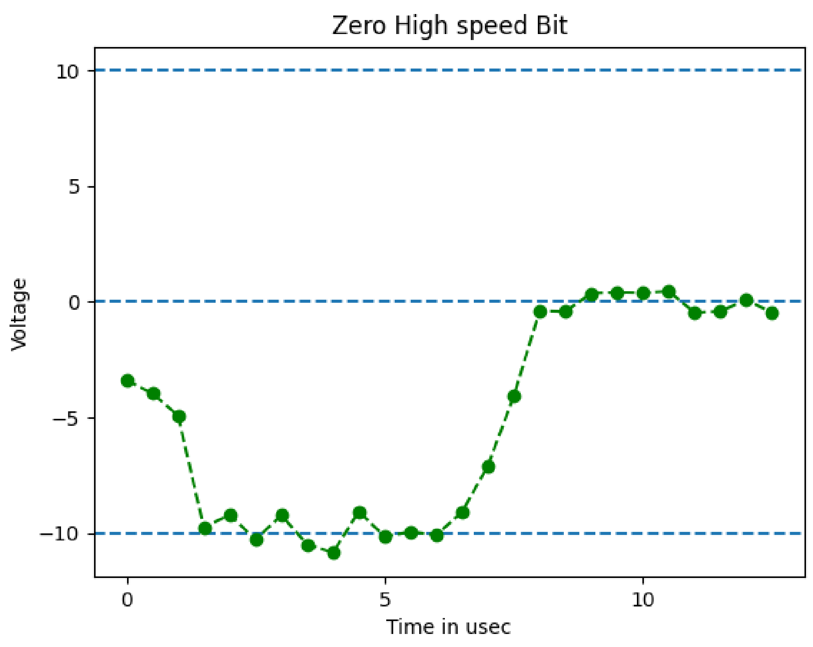


Figure 3: A set of voltages over time that represents a High Speed 0 bit.

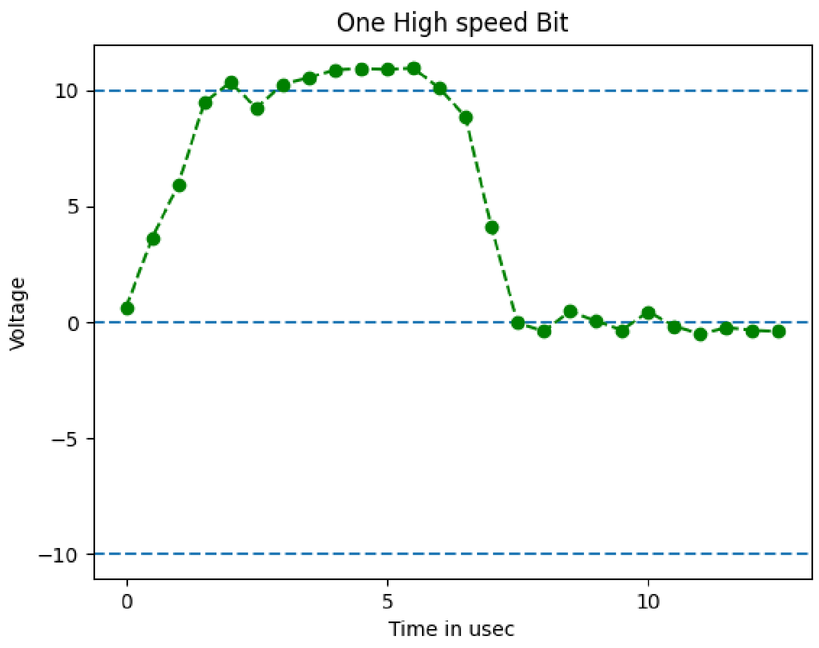


Figure 4: A set of voltages over time that represents a High Speed 1 bit.

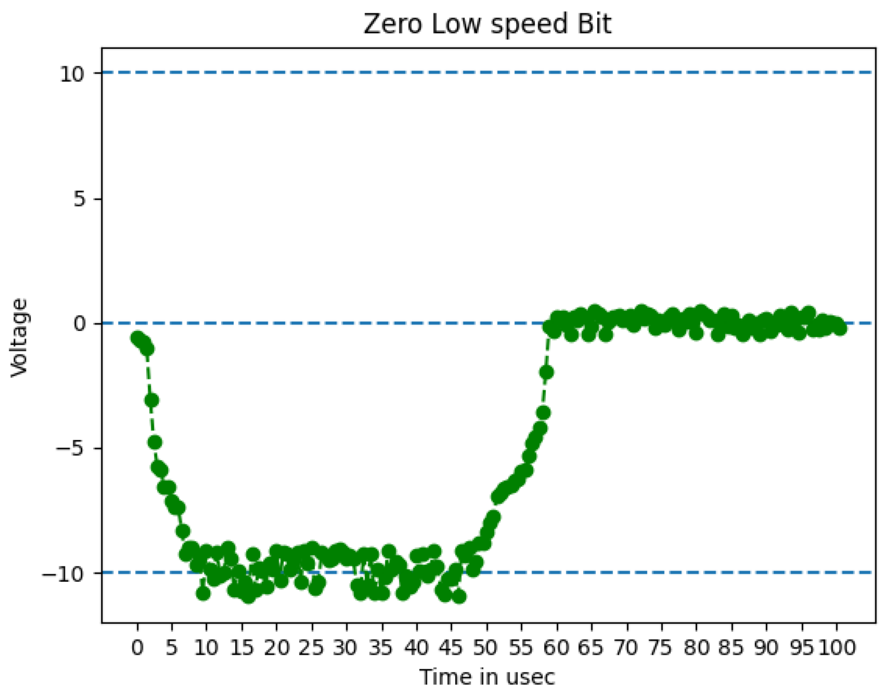


Figure 5: A set of voltages over time that represents a Low Speed 0 bit.

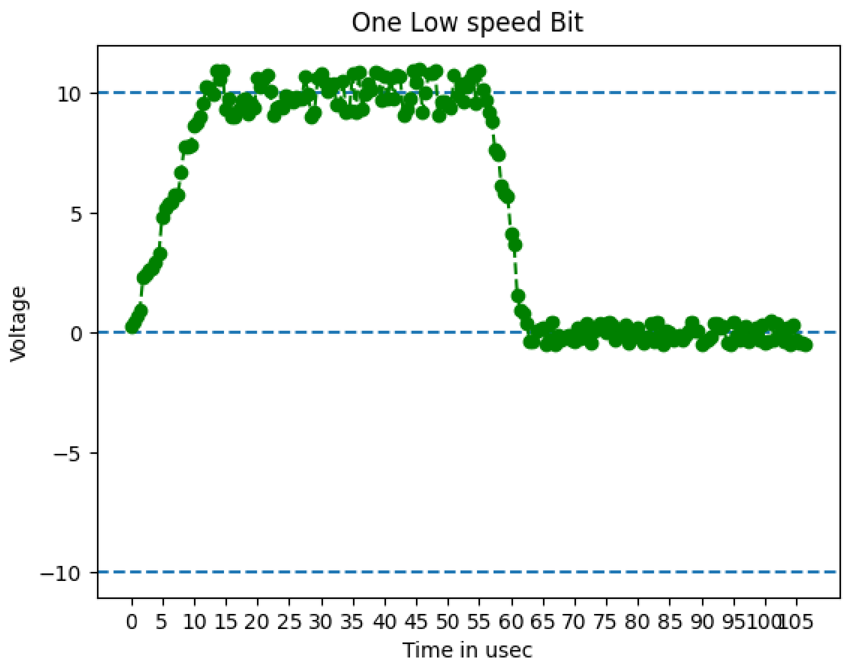


Figure 6: A set of voltages over time that represents a Low Speed 1 bit.

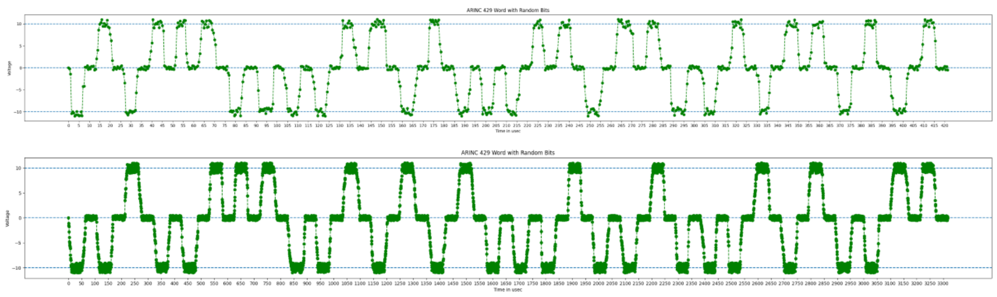


Figure 7 & 8: A set of voltages over times that represent a random word for respectively High and Low speed data buses.

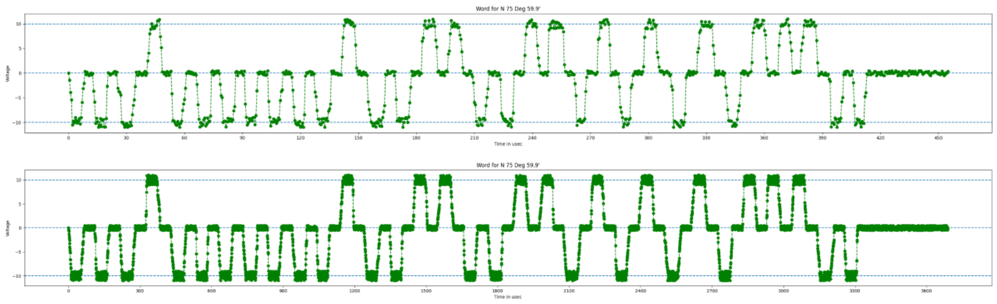
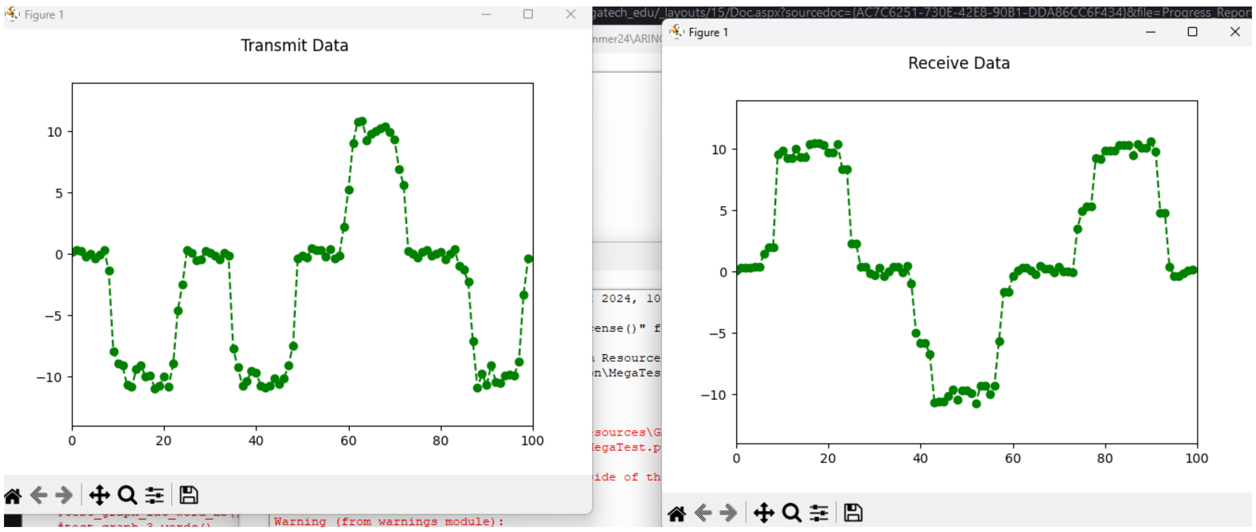
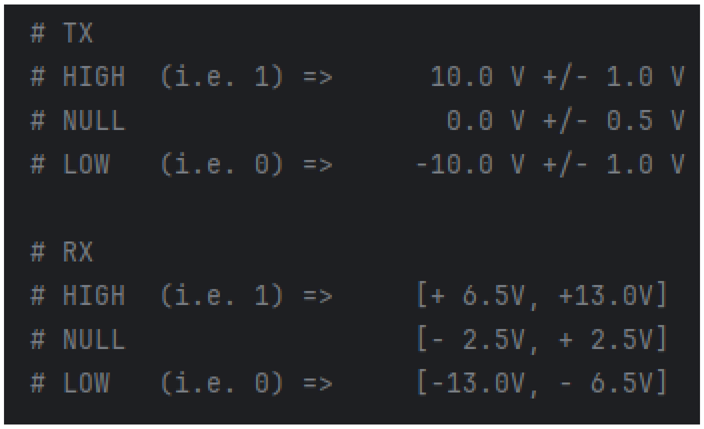


Figure 9 & 10: A set of voltages over time that represents a latitude information word at respectively High and Low Bus Speeds.



```
gatech_edu/_layouts/15/Doc.aspx?sourcedoc={AC7C6251-730E-42E8-90B1-DDA86CC6F434}&file=Progress_Repor
mmer24\ARIN
 2024, 10
ense()" f
 Resource
n\MegaTes
sources\G
egaTest.p
ide of th
Warning (from warnings module):
```

Figure 11: A screenshot of the bus TX and RX bus data over time. Notice the RX is delayed from the TX graph.



```
# TX
# HIGH  (i.e. 1) =>       10.0 V +/- 1.0 V
# NULL                     0.0 V +/- 0.5 V
# LOW   (i.e. 0) =>      -10.0 V +/- 1.0 V

# RX
# HIGH  (i.e. 1) =>      [+ 6.5V, +13.0V]
# NULL                   [- 2.5V, + 2.5V]
# LOW   (i.e. 0) =>      [-13.0V, - 6.5V]
```

Figure 12: Spec Voltage Notes from Progress Report 2.

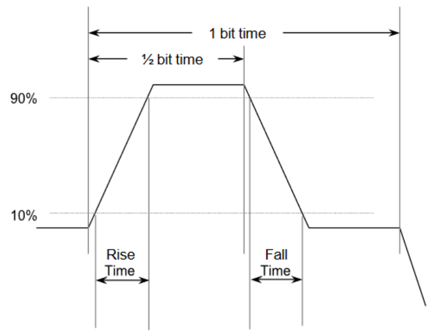
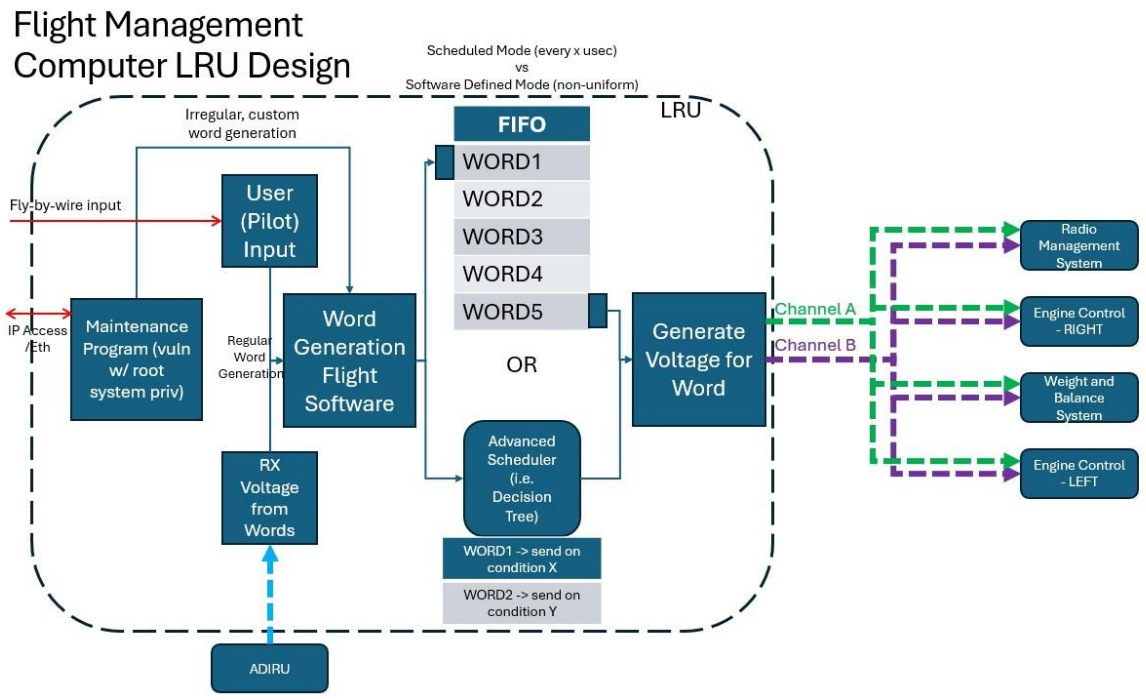
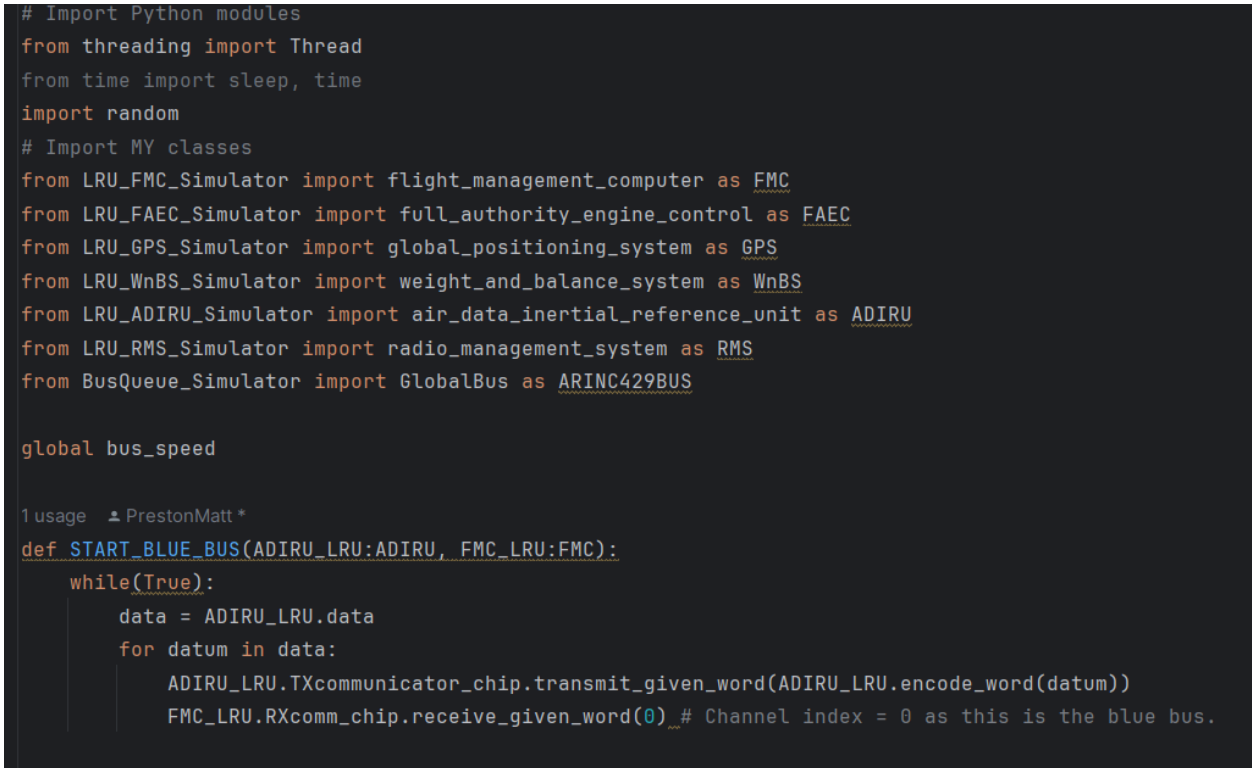


Figure 13: Real Voltage Specification from Progress Report 2.



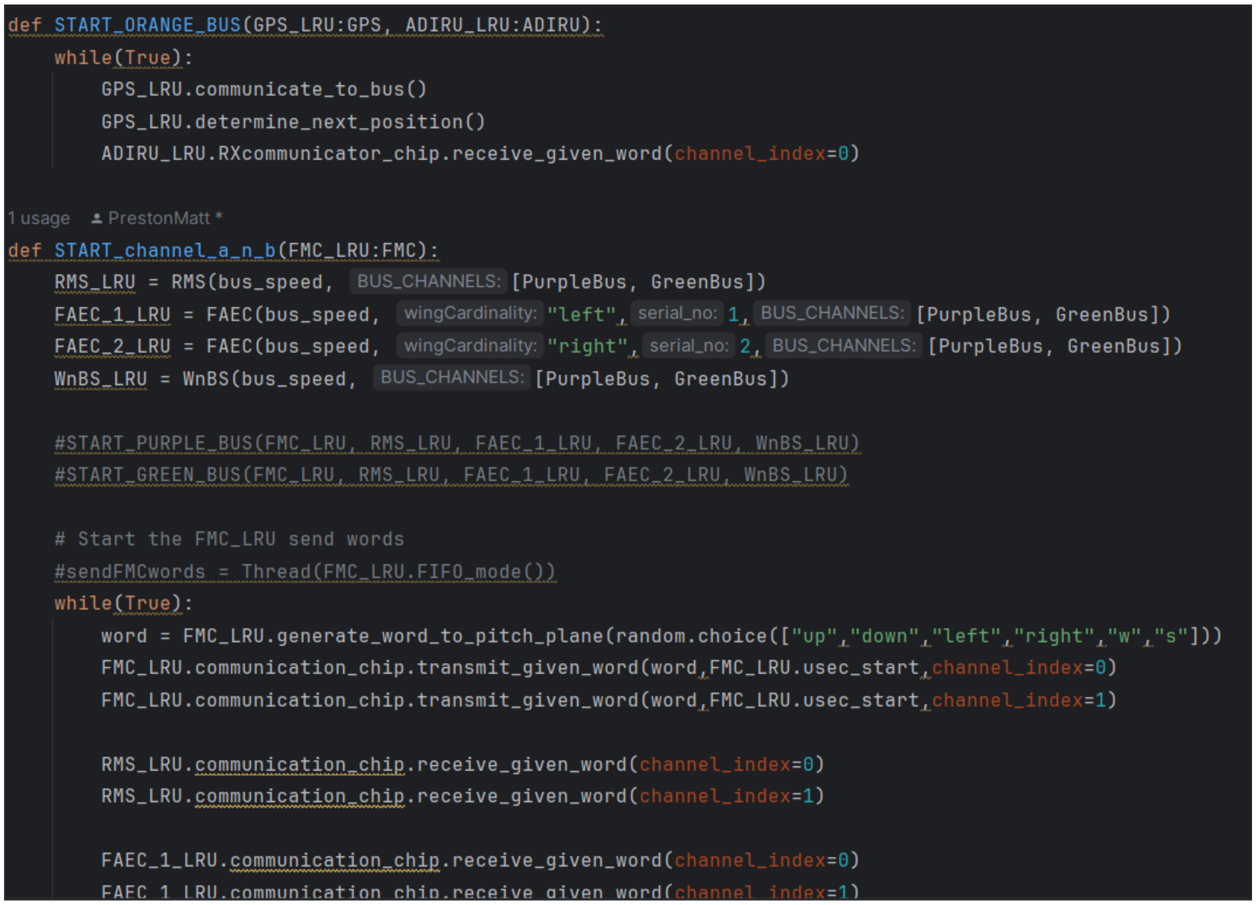
Flight Management Computer LRU Design
Scheduled Mode (every x usec) vs Software Defined Mode (non-uniform)
LRU
Irregular, custom word generation
Fly-by-wire input
User (Pilot) Input
IP Access /Eth
Maintenance Program (vuln w/ root system priv)
Regular Word Generation
Word Generation Flight Software
RX Voltage from Words
FIFO
WORD1
WORD2
WORD3
WORD4
WORD5
OR
Advanced Scheduler (i.e. Decision Tree)
WORD1 -> send on condition X
WORD2 -> send on condition Y
Generate Voltage for Word
Channel A
Channel B
Radio Management System
Engine Control - RIGHT
Weight and Balance System
Engine Control - LEFT
ADIRU

Figure 14: FMC LRU Design.



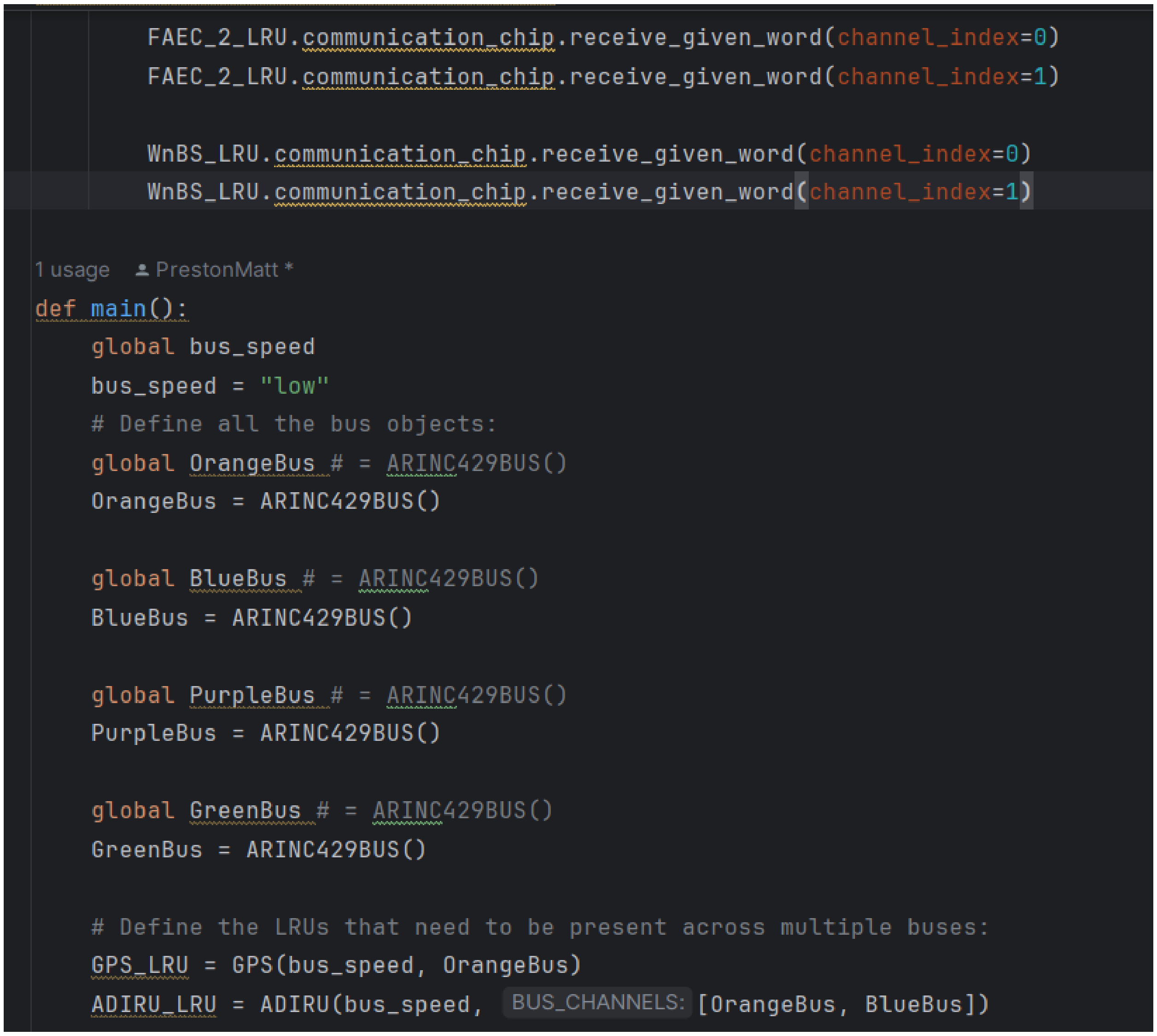
```
# Import Python modules
from threading import Thread
from time import sleep, time
import random
# Import MY classes
from LRU_FMC_Simulator import flight_management_computer as FMC
from LRU_FAEC_Simulator import full_authority_engine_control as FAEC
from LRU_GPS_Simulator import global_positioning_system as GPS
from LRU_WnBS_Simulator import weight_and_balance_system as WnBS
from LRU_ADIRU_Simulator import air_data_inertial_reference_unit as ADIRU
from LRU_RMS_Simulator import radio_management_system as RMS
from BusQueue_Simulator import GlobalBus as ARINC429BUS

global bus_speed

1 usage   PrestonMatt *
def START_BLUE_BUS(ADIRU_LRU:ADIRU, FMC_LRU:FMC):
    while(True):
        data = ADIRU_LRU.data
        for datum in data:
            ADIRU_LRU.TXcommunicator_chip.transmit_given_word(ADIRU_LRU.encode_word(datum))
            FMC_LRU.RXcomm_chip.receive_given_word(0) # Channel index = 0 as this is the blue bus.
```

Figure 15: Main File part 1.



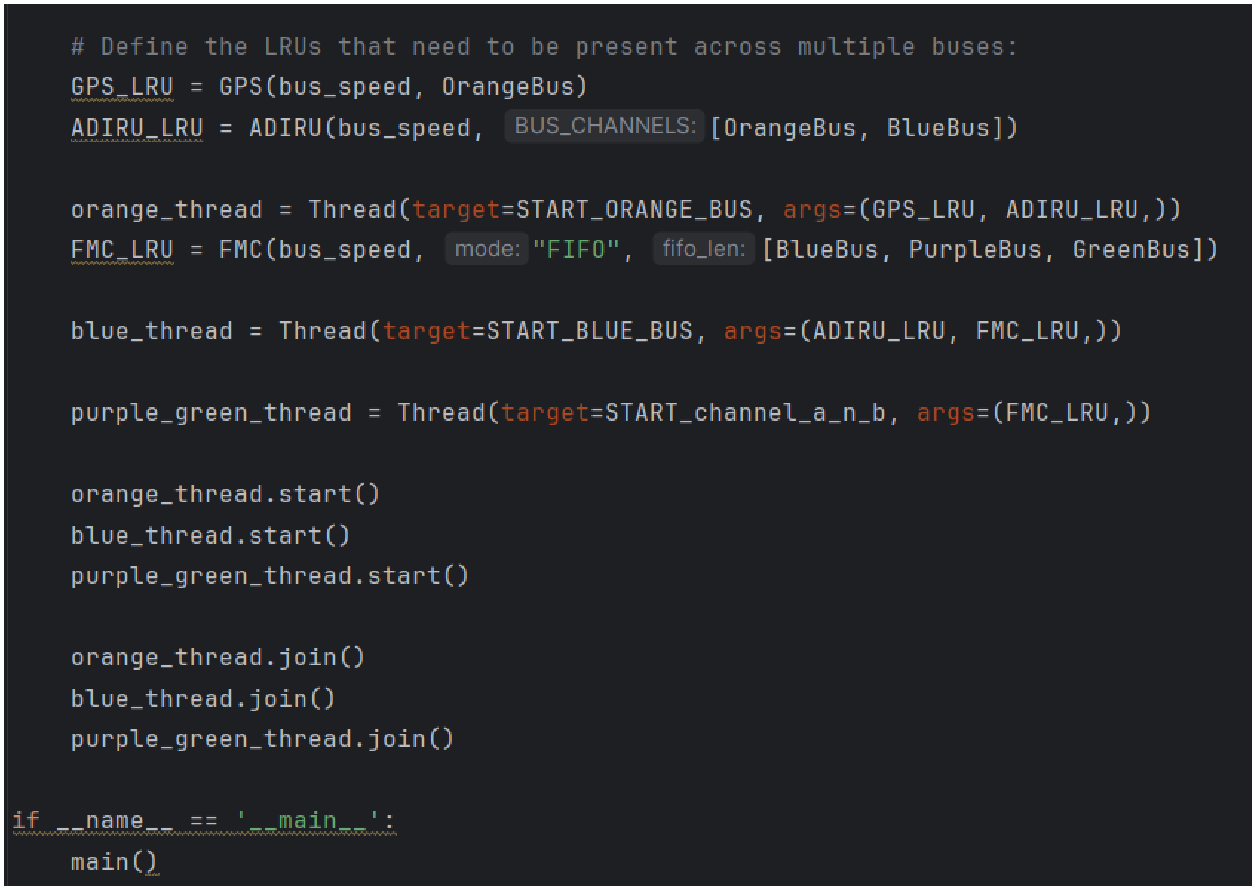
```
def START_ORANGE_BUS(GPS_LRU:GPS, ADIRU_LRU:ADIRU):
    while(True):
        GPS_LRU.communicate_to_bus()
        GPS_LRU.determine_next_position()
        ADIRU_LRU.RXcommunicator_chip.receive_given_word(channel_index=0)

1 usage  PrestonMatt *
def START_channel_a_n_b(FMC_LRU:FMC):
    RMS_LRU = RMS(bus_speed,  BUS_CHANNELS: [PurpleBus, GreenBus])
    FAEC_1_LRU = FAEC(bus_speed,  wingCardinality: "left",  serial_no: 1,  BUS_CHANNELS: [PurpleBus, GreenBus])
    FAEC_2_LRU = FAEC(bus_speed,  wingCardinality: "right",  serial_no: 2,  BUS_CHANNELS: [PurpleBus, GreenBus])
    WnBS_LRU = WnBS(bus_speed,  BUS_CHANNELS: [PurpleBus, GreenBus])

    #START_PURPLE_BUS(FMC_LRU, RMS_LRU, FAEC_1_LRU, FAEC_2_LRU, WnBS_LRU)
    #START_GREEN_BUS(FMC_LRU, RMS_LRU, FAEC_1_LRU, FAEC_2_LRU, WnBS_LRU)

    # Start the FMC_LRU send words
    #sendFMCwords = Thread(FMC_LRU.FIFO_mode())
    while(True):
        word = FMC_LRU.generate_word_to_pitch_plane(random.choice(["up","down","left","right","w","s"]))
        FMC_LRU.communication_chip.transmit_given_word(word,FMC_LRU.usec_start,channel_index=0)
        FMC_LRU.communication_chip.transmit_given_word(word,FMC_LRU.usec_start,channel_index=1)

        RMS_LRU.communication_chip.receive_given_word(channel_index=0)
        RMS_LRU.communication_chip.receive_given_word(channel_index=1)

        FAEC_1_LRU.communication_chip.receive_given_word(channel_index=0)
        FAEC_1_LRU.communication_chip.receive_given_word(channel_index=1)
```

Figure 16: Main File Part 2



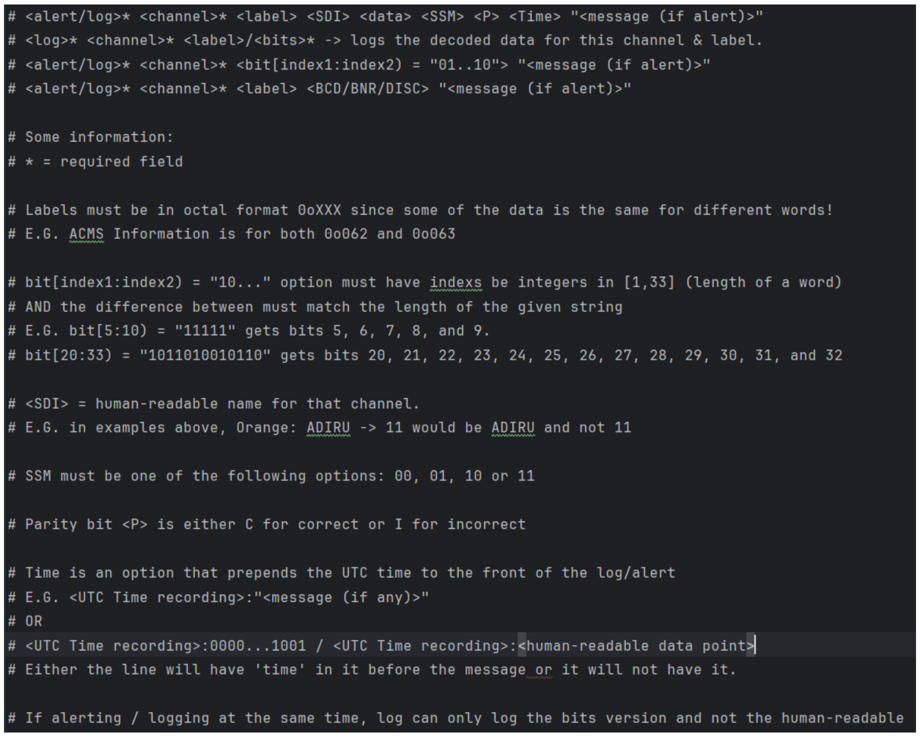
```
        FAEC_2_LRU.communication_chip.receive_given_word(channel_index=0)
        FAEC_2_LRU.communication_chip.receive_given_word(channel_index=1)

        WnBS_LRU.communication_chip.receive_given_word(channel_index=0)
        WnBS_LRU.communication_chip.receive_given_word(channel_index=1)

1 usage  PrestonMatt *
def main():
    global bus_speed
    bus_speed = "low"
    # Define all the bus objects:
    global OrangeBus # = ARINC429BUS()
    OrangeBus = ARINC429BUS()

    global BlueBus # = ARINC429BUS()
    BlueBus = ARINC429BUS()

    global PurpleBus # = ARINC429BUS()
    PurpleBus = ARINC429BUS()

    global GreenBus # = ARINC429BUS()
    GreenBus = ARINC429BUS()

    # Define the LRUs that need to be present across multiple buses:
    GPS_LRU = GPS(bus_speed, OrangeBus)
    ADIRU_LRU = ADIRU(bus_speed, BUS_CHANNELS: [OrangeBus, BlueBus])
```

Figure 17: Main File Part 3.



```
    # Define the LRUs that need to be present across multiple buses:
    GPS_LRU = GPS(bus_speed, OrangeBus)
    ADIRU_LRU = ADIRU(bus_speed, BUS_CHANNELS: [OrangeBus, BlueBus])

    orange_thread = Thread(target=START_ORANGE_BUS, args=(GPS_LRU, ADIRU_LRU,))
    FMC_LRU = FMC(bus_speed, mode: "FIFO", fifo_len: [BlueBus, PurpleBus, GreenBus])

    blue_thread = Thread(target=START_BLUE_BUS, args=(ADIRU_LRU, FMC_LRU,))

    purple_green_thread = Thread(target=START_channel_a_n_b, args=(FMC_LRU,))

    orange_thread.start()
    blue_thread.start()
    purple_green_thread.start()

    orange_thread.join()
    blue_thread.join()
    purple_green_thread.join()

if __name__ == '__main__':
    main()
```

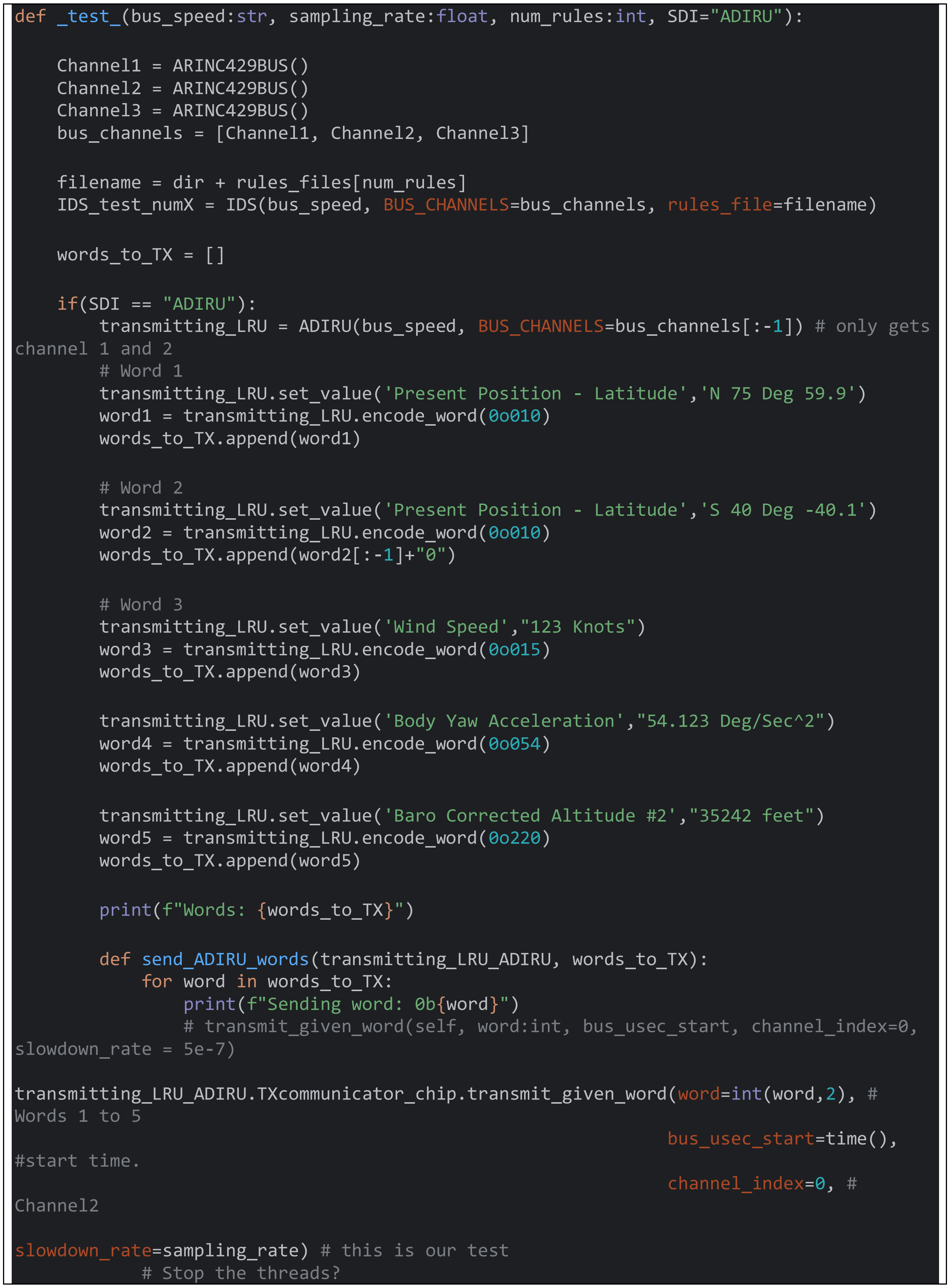
Figure 18: Main File Part 4.



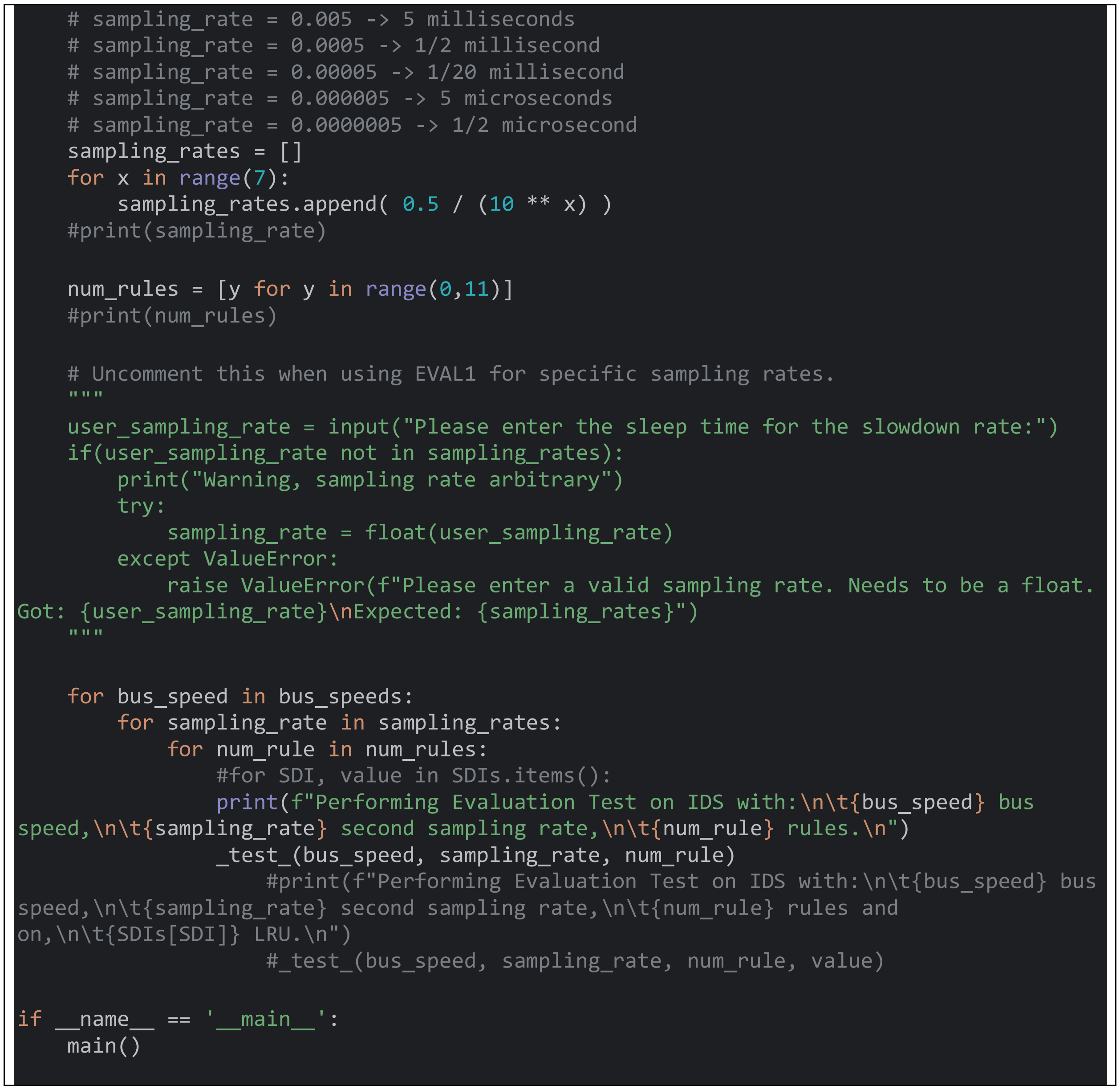
```
# <alert/log>* <channel>* <label> <SDI> <data> <SSM> <P> <Time> "<message (if alert)>"
# <log>* <channel>* <label>/<bits>* -> logs the decoded data for this channel & label.
# <alert/log>* <channel>* <bit[index1:index2) = "01..10"> "<message (if alert)>"
# <alert/log>* <channel>* <label> <BCD/BNR/DISC> "<message (if alert)>"

# Some information:
# * = required field

# Labels must be in octal format 0oXXX since some of the data is the same for different words!
# E.G. ACMS Information is for both 0o062 and 0o063

# bit[index1:index2) = "10..." option must have indexs be integers in [1,33] (length of a word)
# AND the difference between must match the length of the given string
# E.G. bit[5:10) = "11111" gets bits 5, 6, 7, 8, and 9.
# bit[20:33) = "1011010010110" gets bits 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, and 32

# <SDI> = human-readable name for that channel.
# E.G. in examples above, Orange: ADIRU -> 11 would be ADIRU and not 11

# SSM must be one of the following options: 00, 01, 10 or 11

# Parity bit <P> is either C for correct or I for incorrect

# Time is an option that prepends the UTC time to the front of the log/alert
# E.G. <UTC Time recording>:"<message (if any)>"
# OR
# <UTC Time recording>:0000...1001 / <UTC Time recording>:<human-readable data point>
# Either the line will have 'time' in it before the message or it will not have it.

# If alerting / logging at the same time, log can only log the bits version and not the human-readable
```

Figure 19: IDS Rules Template.

```
from ARINC429_IDS import arinc429_intrusion_detection_system as IDS
from LRU_FMC_Simulator import flight_management_computer as FMC
from LRU_GPS_Simulator import global_positioning_system as GPS
from LRU_ADIRU_Simulator import air_data_inertial_reference_unit as ADIRU
from BusQueue_Simulator import GlobalBus as ARINC429BUS
from LRU_TX_Helper import arinc429_TX_Helpers as lru_txr
from LRU_RX_Helper import arinc429_RX_Helpers as lru_rxr
from time import sleep, time
from threading import Thread

"""
Robustness of the IDS: Right now my bus code can transmit at the actual speed of an
ARINC429 bus. However, when
receiving words, especially with the IDS, I need to slow it down so that voltages
aren’t missed. This is an issue
with Python threading and dealing with microsecond timing. I will run tests to see how
robust my IDS is by testing
it’s receive function. These tests will be at receive 5 ARINC429 words from a given RX
LRU (so FMC, GPS, or ADIRU).
The speed of the bus will start slow – instead of ½ microsecond between voltage
transmissions it will start at ½ second.
Then the intervals will scale up logarithmic up in speed until it gets to ½ microsecond
again. From there I will create
a chart: the amount of words it’s able to correctly receive and act upon (based on it’s
rules) per speed. Then I will
repeat this test with various rules, from 1 to 10 rules.
"""

"""
    Given the sampling rate (decreasing logarithmically from 0.5 seconds to 0.5
microseconds)
    Run the IDS. Send it 5 words from:
        GPS
        FMC
        ADIRU
    Check accuracy percentage of RX'd words.
"""

dir = r"C:\Users\mspre\Desktop\Practicum
Resources\GATech_MS_Cybersecurity_Practicum_InfoSec_Summer24\ARINC429
Simulation\IDS_Rules_test_files\IDS_EVAL1_RULES_FILES"
rules_files = {
    0:r"\ZERO_RULES.txt",
    1:r"\ONE_RULE.txt",
    2:r"\TWO_RULES.txt",
    3:r"\THREE_RULES.txt",
    4:r"\FOUR_RULES.txt",
    5:r"\FIVE_RULES.txt",
    6:r"\SIX_RULES.txt",
    7:r"\SEVEN_RULES.txt",
    8:r"\EIGHT_RULES.txt",
    9:r"\NINE_RULES.txt",
    10:r"\TEN_RULES.txt"
}
```

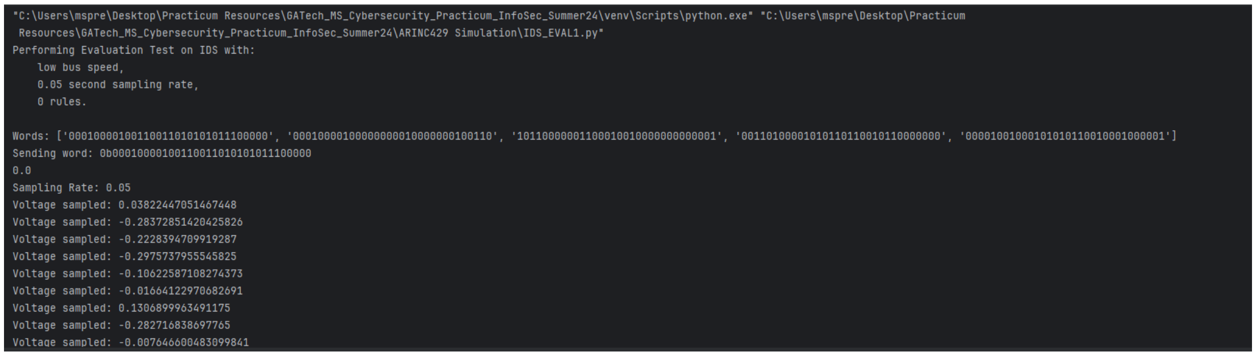


```
def _test_(bus_speed:str, sampling_rate:float, num_rules:int, SDI="ADIRU"):

    Channel1 = ARINC429BUS()
    Channel2 = ARINC429BUS()
    Channel3 = ARINC429BUS()
    bus_channels = [Channel1, Channel2, Channel3]

    filename = dir + rules_files[num_rules]
    IDS_test_numX = IDS(bus_speed, BUS_CHANNELS=bus_channels, rules_file=filename)

    words_to_TX = []

    if(SDI == "ADIRU"):
        transmitting_LRU = ADIRU(bus_speed, BUS_CHANNELS=bus_channels[:-1]) # only gets
channel 1 and 2
        # Word 1
        transmitting_LRU.set_value('Present Position - Latitude','N 75 Deg 59.9')
        word1 = transmitting_LRU.encode_word(0o010)
        words_to_TX.append(word1)

        # Word 2
        transmitting_LRU.set_value('Present Position - Latitude','S 40 Deg -40.1')
        word2 = transmitting_LRU.encode_word(0o010)
        words_to_TX.append(word2[:-1]+"0")

        # Word 3
        transmitting_LRU.set_value('Wind Speed',"123 Knots")
        word3 = transmitting_LRU.encode_word(0o015)
        words_to_TX.append(word3)

        transmitting_LRU.set_value('Body Yaw Acceleration',"54.123 Deg/Sec^2")
        word4 = transmitting_LRU.encode_word(0o054)
        words_to_TX.append(word4)

        transmitting_LRU.set_value('Baro Corrected Altitude #2',"35242 feet")
        word5 = transmitting_LRU.encode_word(0o220)
        words_to_TX.append(word5)

        print(f"Words: {words_to_TX}")

        def send_ADIRU_words(transmitting_LRU_ADIRU, words_to_TX):
            for word in words_to_TX:
                print(f"Sending word: 0b{word}")
                # transmit_given_word(self, word:int, bus_usec_start, channel_index=0,
slowdown_rate = 5e-7)

transmitting_LRU_ADIRU.TXcommunicator_chip.transmit_given_word(word=int(word,2), #
Words 1 to 5
                                                              bus_usec_start=time(),
#start time.
                                                              channel_index=0, #
Channel2

slowdown_rate=sampling_rate) # this is our test
            # Stop the threads?
        # Start the TXr transmission in thread
        transmitter_thread = Thread(target=send_ADIRU_words,
args=(transmitting_LRU,words_to_TX,))
        transmitter_thread.start()
        # Start the receiver in a separate thread
        receiver_thread = Thread(target=IDS_test_numX.receive_words, args=(0,
sampling_rate,))
        receiver_thread.start()
        # Start the real-time visualization of TX'd voltages in a separate thread
        visualization_threadTX = Thread(target=ARINC429BUS.queue_visual,
                                      args=(Channel1, sampling_rate, "Transmitted
Voltages for IDS Eval 1",))
        visualization_threadTX.start()
        # Start the real-time visualization of RX'd voltages in a final thread
        visualization_threadRX =
Thread(target=IDS_test_numX.communication_chip.visualize_LRU_receiveds_mother,
                                        args=(Channel1,"Received Voltages for Eval
1",sampling_rate),)
        visualization_threadRX.start()

        # Join threads to main thread keeping simulation running
        transmitter_thread.join()
        receiver_thread.join()
        visualization_threadTX.join()
        visualization_threadRX.join()

    """
    transmitting_LRU = None
    if(SDI == "00"):

       #transmitting_LRU = FMC(bus_speed, BUS_CHANNELS=bus_channels)
        #transmitting_LRU.generate_word_to_pitch_plane("up")
        #transmitting_LRU.generate_word_to_pitch_plane("down")
        #transmitting_LRU.generate_word_to_pitch_plane("left") # Transmits 3 words.
    elif(SDI == "01"):
        transmitting_LRU = GPS()
    elif(SDI == "02"):
        transmitting_LRU = ADIRU()
    """

def main():

    bus_speeds = [
        "low",
        "high"
    ]

    SDIs = {
        "00":"FMC",
        "01":"GPS",
        "10":"ADIRU"
    }

    # sampling_rate = 0.5 -> 1/2 second
    # sampling_rate = 0.05 -> 1/20th second
```



```
    # sampling_rate = 0.005 -> 5 milliseconds
    # sampling_rate = 0.0005 -> 1/2 millisecond
    # sampling_rate = 0.00005 -> 1/20 millisecond
    # sampling_rate = 0.000005 -> 5 microseconds
    # sampling_rate = 0.0000005 -> 1/2 microsecond
    sampling_rates = []
    for x in range(7):
        sampling_rates.append( 0.5 / (10 ** x) )
    #print(sampling_rate)

    num_rules = [y for y in range(0,11)]
    #print(num_rules)

    # Uncomment this when using EVAL1 for specific sampling rates.
    """
    user_sampling_rate = input("Please enter the sleep time for the slowdown rate:")
    if(user_sampling_rate not in sampling_rates):
        print("Warning, sampling rate arbitrary")
        try:
            sampling_rate = float(user_sampling_rate)
        except ValueError:
            raise ValueError(f"Please enter a valid sampling rate. Needs to be a float.
Got: {user_sampling_rate}\nExpected: {sampling_rates}")
    """

    for bus_speed in bus_speeds:
        for sampling_rate in sampling_rates:
            for num_rule in num_rules:
                #for SDI, value in SDIs.items():
                print(f"Performing Evaluation Test on IDS with:\n\t{bus_speed} bus
speed,\n\t{sampling_rate} second sampling rate,\n\t{num_rule} rules.\n")
                _test_(bus_speed, sampling_rate, num_rule)
                    #print(f"Performing Evaluation Test on IDS with:\n\t{bus_speed} bus
speed,\n\t{sampling_rate} second sampling rate,\n\t{num_rule} rules and
on,\n\t{SDIs[SDI]} LRU.\n")
                    #_test_(bus_speed, sampling_rate, num_rule, value)

if __name__ == '__main__':
    main()
```

Figure 20: Eval 2 Code (named IDS_EVAL1.py)



```
"C:\Users\mspre\Desktop\Practicum Resources\GATech_MS_Cybersecurity_Practicum_InfoSec_Summer24\venv\Scripts\python.exe" "C:\Users\mspre\Desktop\Practicum
 Resources\GATech_MS_Cybersecurity_Practicum_InfoSec_Summer24\ARINC429 Simulation\IDS_EVAL1.py"
Performing Evaluation Test on IDS with:
    low bus speed,
    0.05 second sampling rate,
    0 rules.

Words: ['00010000100110011010101011100000', '00010000100000000010000000100110', '10110000001100010010000000000001', '00110100001010110110010110000000', '00001001000101010110010001000001']
Sending word: 0b00010000100110011010101011100000
0.0
Sampling Rate: 0.05
Voltage sampled: 0.03822447051467448
Voltage sampled: -0.28372851420425826
Voltage sampled: -0.2228394709919287
Voltage sampled: -0.2975737955545825
Voltage sampled: -0.10622587108274373
Voltage sampled: -0.01664122970682691
Voltage sampled: 0.1306899963491175
Voltage sampled: -0.282716838697765
Voltage sampled: -0.007646600483099841
```

Figure 21: Eval 2 Running 1

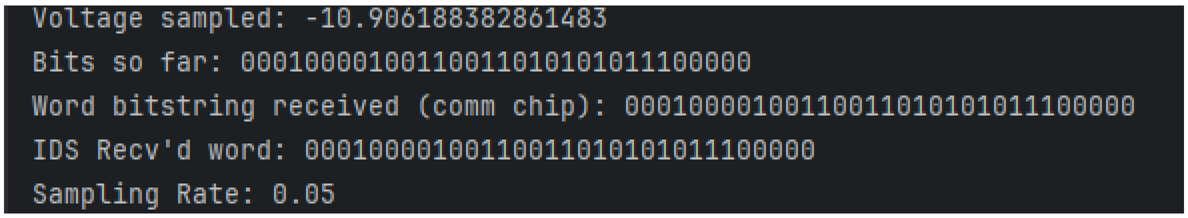
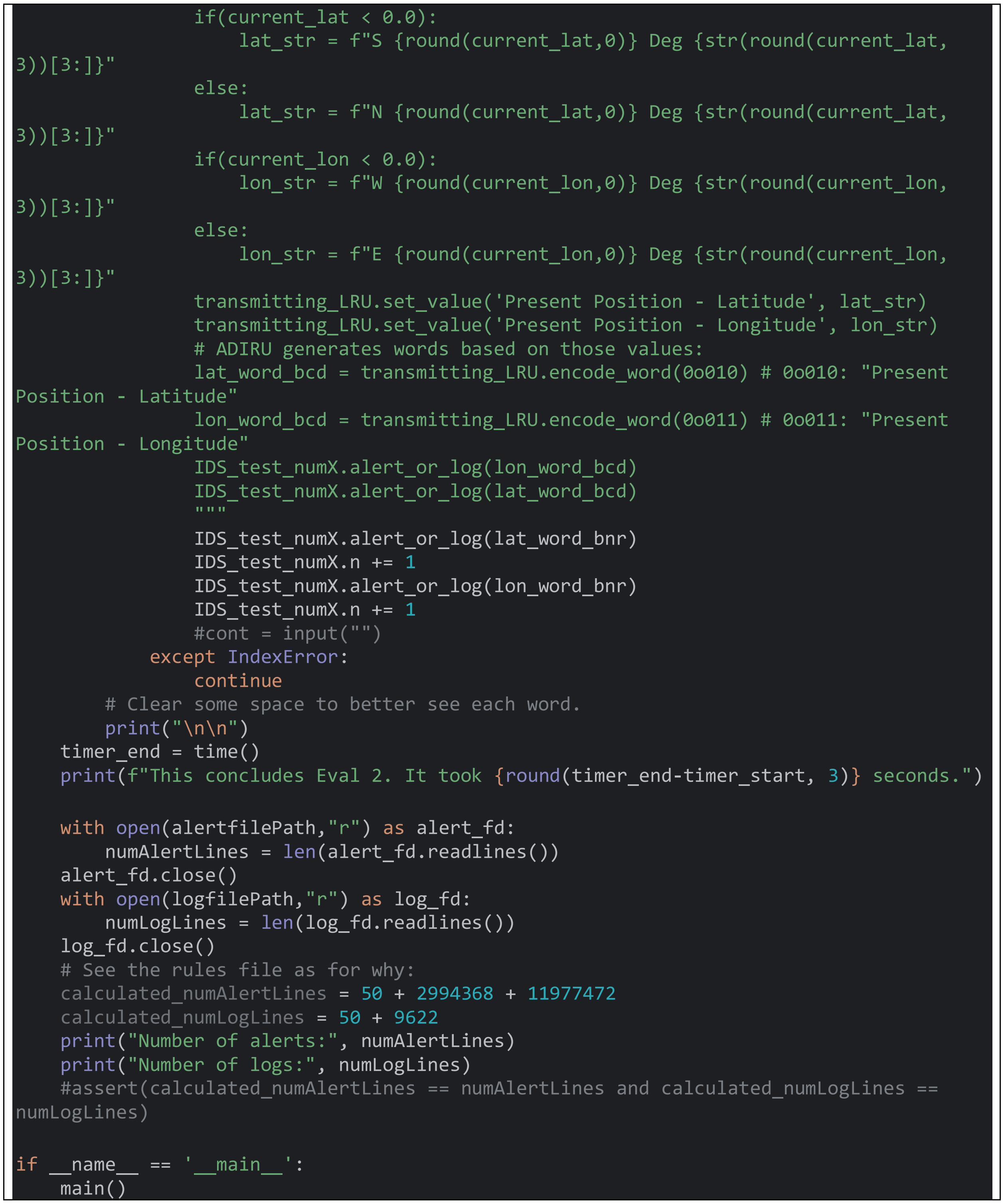

```
Voltage sampled: -10.906188382861483
Bits so far: 00010000100110011010101011100000
Word bitstring received (comm chip): 00010000100110011010101011100000
IDS Recv'd word: 00010000100110011010101011100000
Sampling Rate: 0.05
```

Figure 22: Eval 2 Running 2 – Got correct word #1

```
import os
import pytest

from ARINC429_IDS import arinc429_intrusion_detection_system as IDS
from LRU_ADIRU_Simulator import air_data_inertial_reference_unit as ADIRU
from BusQueue_Simulator import GlobalBus as ARINC429BUS
from time import sleep, time, ctime
def main():

    cont = input("ARE YOU SURE YOU WANT TO OVERWRITE THE ALERTS AND LOGS FILES AND 
START EVAL2?")

    rules_filename = os.getcwd() + 
r"\IDS_Rules_test_files\IDS_EVAL2_RULES_FILES\Eval2_Rules.txt"
    flightdata_filenames = os.getcwd() + r"\Flight_data\Tail_687_1"
    bus_speed = "low"

    Channel1 = ARINC429BUS()
    Channel2 = ARINC429BUS()
    channels = [Channel1, Channel2]

    IDS_test_numX = IDS(bus_speed, BUS_CHANNELS=channels, rules_file=rules_filename)
    transmitting_LRU = ADIRU(bus_speed, BUS_CHANNELS=channels)
    transmitting_LRU.set_sdi('00')

    # Check the output files:
    #print(IDS_test_numX.log_filepath)
    #print(IDS_test_numX.alert_filepath)
    alertfilePath = os.getcwd() + 
r"\IDS_Rules_test_files\IDS_EVAL2_RULES_FILES\Alerts_Logs\Alerts_EVAL2.txt"
    logfilePath = os.getcwd() + 
r"\IDS_Rules_test_files\IDS_EVAL2_RULES_FILES\Alerts_Logs\Logs_EVAL2.txt"
    # Reset the files in between runs:
    with open(alertfilePath,"w") as alert_fd:
        alert_fd.write(f"Starting EVAL2 test at {ctime()}\n")
    alert_fd.close()
    with open(logfilePath,"w") as log_fd:
        log_fd.write(f"Starting EVAL2 test at {ctime()}\n")
    log_fd.close()
    #print(alertfilePath)
    #print(logfilePath)
    # Some error handling:
    if(IDS_test_numX.alert_filepath != alertfilePath):
        IDS_test_numX.set_alertfile(alertfilePath)
    if(IDS_test_numX.log_filepath != logfilePath):
        IDS_test_numX.set_logfile(logfilePath)
```

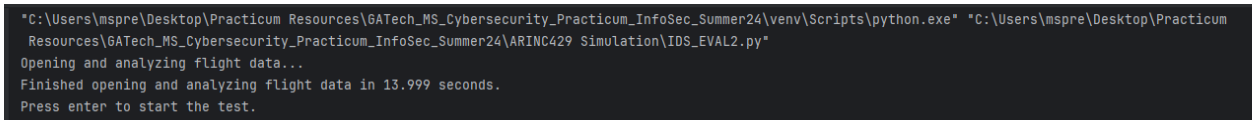
```python
    #print(IDS_test_numX.alert_filepath)
    #print(IDS_test_numX.log_filepath)
    #cont = input("")

    timer_start = time()
    print("Opening and analyzing flight data...")
    # Grab flight data to plug into ADIRU:
    # Remember to take off \n character in these files.
    # Dataset no. 1: Airspeed.
    with open(flightdata_filenames+r"\COMPUTED AIRSPEED LSP_Tail_687_1_data.txt", "r")
as airspeed_fd:
        airspeeds = airspeed_fd.readlines()
        #print(airspeeds)
    airspeed_fd.close()
    # Dataset no. 2: Corrected Angle of Attack.
    with open(flightdata_filenames+r"\CORRECTED ANGLE OF ATTACK_Tail_687_1_data.txt",
"r") as corrected_aoa_fd:
        corrected_aoas = corrected_aoa_fd.readlines()
        #print(corrected_aoas)
    corrected_aoa_fd.close()
    # Dataset no. 3: Indicated Angle of Attack.
    with open(flightdata_filenames+r"\INDICATED ANGLE OF ATTACK_Tail_687_1_data.txt",
"r") as indicated_aoa_fd:
        indicated_aoas = indicated_aoa_fd.readlines()
    indicated_aoa_fd.close()
    # Dataset no. 4: Latitude.
    with open(flightdata_filenames+r"\LATITUDE POSITION LSP_Tail_687_1_data.txt", "r")
as latitude_fd:
        lats = latitude_fd.readlines()
    latitude_fd.close()
    # Dataset no. 5: Longitude.
    with open(flightdata_filenames+r"\LATITUDE POSITION LSP_Tail_687_1_data.txt", "r")
as longitude_fd:
        lons = longitude_fd.readlines()
    longitude_fd.close()

    #print(len(airspeeds), len(corrected_aoas), len(indicated_aoas), len(lats),
len(lons))

    # Length of airspeeds == corrected_aoas == indicated_aoas
    for x in range(len(airspeeds)):
        airspeeds[x] = float(airspeeds[x].replace("\n",""))
        corrected_aoas[x] = float(corrected_aoas[x].replace("\n",""))
        indicated_aoas[x] = float(indicated_aoas[x].replace("\n",""))
    # Length of latitudes == longitudes
    for y in range(len(lats)):
        lats[y] = float(lats[y].replace("\n",""))
        lons[y] = float(lons[y].replace("\n",""))

    timer_end = time()
    print(f"Finished opening and analyzing flight data in {round(timer_end-
timer_start,3)} seconds.")
    cont = input("Press enter to start the test.")

    timer_start = time()
    print("Beginning flight data evaluation 2...")
    # Length of Airspeed, corrected_aoas, and indicated_aoas are 4 times that of length
```

```
of latitudes and longitudes
    for index in range(len(airspeeds)):
        # Get our data points:
        current_airspeed = airspeeds[index]
        current_corrected_aoa = corrected_aoas[index]
        current_indicated_aoa = indicated_aoas[index]
        # The ADIRU "collects" them:
        transmitting_LRU.set_value('True Airspeed', f"{current_airspeed} knots")
        transmitting_LRU.set_value('Corrected Angle of Attack',
f"{current_corrected_aoa} degrees")
        transmitting_LRU.set_value('Indicated Angle of Attack (Average)',
f"{current_indicated_aoa} degrees")
        # ADIRU generates words based on those values:
        airspeed_word_bcd = transmitting_LRU.encode_word(0o230) # 0o230: "True
Airspeed"
        airspeed_word_bnr = transmitting_LRU.encode_word(0o210) # 0o210: "True
Airspeed"
        corrected_aoa_word = transmitting_LRU.encode_word(0o241) # 0o241: "Corrected
Angle of Attack"
        indicated_aoa_word = transmitting_LRU.encode_word(0o221) # 0o221: "Indicated
Angle of Attack (Average)"
        # For progress visualization:
        print(f"Sending words:\nAirspeed BCD:\t\t\t\t\t0b{airspeed_word_bcd},\nAirspeed
BNR:\t\t\t\t\t0b{airspeed_word_bnr},")
        print(f"Corrected Angle of Attack:\t\t0b{corrected_aoa_word},\nIndicated Angle
of Attack:\t\t0b{indicated_aoa_word},")
        # See if it alerts/logs correctly:
        IDS_test_numX.alert_or_log(airspeed_word_bcd)
        IDS_test_numX.n += 1 # normally the RX func would update this but since we're
just feeding words straight we gotta help IDS out a bit.
        IDS_test_numX.alert_or_log(airspeed_word_bnr)
        IDS_test_numX.n += 1
        IDS_test_numX.alert_or_log(corrected_aoa_word)
        IDS_test_numX.n += 1
        IDS_test_numX.alert_or_log(indicated_aoa_word)
        IDS_test_numX.n += 1
        # Repeat this for lat and lon, except every 4 words because it tx's 1/4 as
much.
        if(index % 4 == 0):
            # Get the Lat/Lon
            try:
                current_lat = lats[index]
                current_lon = lons[index]
                # The ADIRU "collects" them
                transmitting_LRU.set_value('Present Position - Latitude',
f"{current_lat} degrees")
                transmitting_LRU.set_value('Present Position - Longitude',
f"{current_lon} degrees")
                # ADIRU generates words based on those values:
                lat_word_bnr = transmitting_LRU.encode_word(0o310) # 0o310: "Present
Position - Latitude"
                lon_word_bnr = transmitting_LRU.encode_word(0o311) # 0o311: "Present
Position - Longitude"
                print(f"Latitude BNR:\t\t\t\t\t0b{lat_word_bnr},\nLongitude
BNR:\t\t\t\t\t0b{lon_word_bnr}")
                # Redundant:
                """
                # The ADIRU "reformats" them again
```



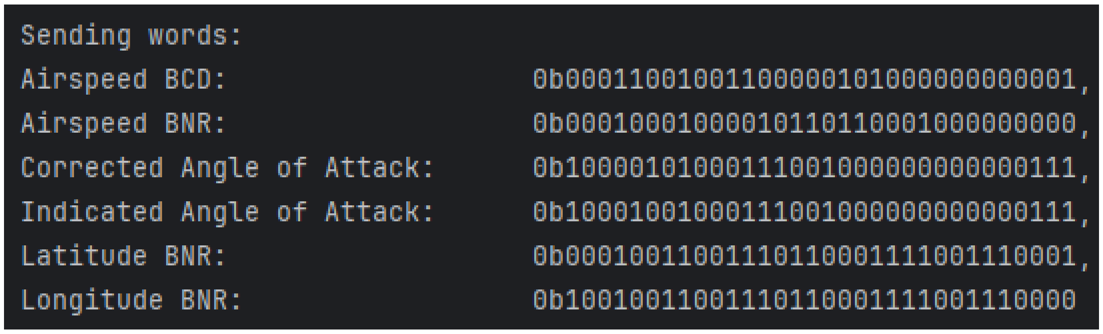
```
                if(current_lat < 0.0):
                    lat_str = f"S {round(current_lat,0)} Deg {str(round(current_lat,
3))[3:]}"
                else:
                    lat_str = f"N {round(current_lat,0)} Deg {str(round(current_lat,
3))[3:]}"
                if(current_lon < 0.0):
                    lon_str = f"W {round(current_lon,0)} Deg {str(round(current_lon,
3))[3:]}"
                else:
                    lon_str = f"E {round(current_lon,0)} Deg {str(round(current_lon,
3))[3:]}"
                transmitting_LRU.set_value('Present Position - Latitude', lat_str)
                transmitting_LRU.set_value('Present Position - Longitude', lon_str)
                # ADIRU generates words based on those values:
                lat_word_bcd = transmitting_LRU.encode_word(0o010) # 0o010: "Present
Position - Latitude"
                lon_word_bcd = transmitting_LRU.encode_word(0o011) # 0o011: "Present
Position - Longitude"
                IDS_test_numX.alert_or_log(lon_word_bcd)
                IDS_test_numX.alert_or_log(lat_word_bcd)
                """
                IDS_test_numX.alert_or_log(lat_word_bnr)
                IDS_test_numX.n += 1
                IDS_test_numX.alert_or_log(lon_word_bnr)
                IDS_test_numX.n += 1
                #cont = input("")
            except IndexError:
                continue
        # Clear some space to better see each word.
        print("\n\n")
    timer_end = time()
    print(f"This concludes Eval 2. It took {round(timer_end-timer_start, 3)} seconds.")

    with open(alertfilePath,"r") as alert_fd:
        numAlertLines = len(alert_fd.readlines())
    alert_fd.close()
    with open(logfilePath,"r") as log_fd:
        numLogLines = len(log_fd.readlines())
    log_fd.close()
    # See the rules file as for why:
    calculated_numAlertLines = 50 + 2994368 + 11977472
    calculated_numLogLines = 50 + 9622
    print("Number of alerts:", numAlertLines)
    print("Number of logs:", numLogLines)
    #assert(calculated_numAlertLines == numAlertLines and calculated_numLogLines ==
numLogLines)

if __name__ == '__main__':
    main()
```

Figure 23: Eval 3 (IDS_Eval2.py) code



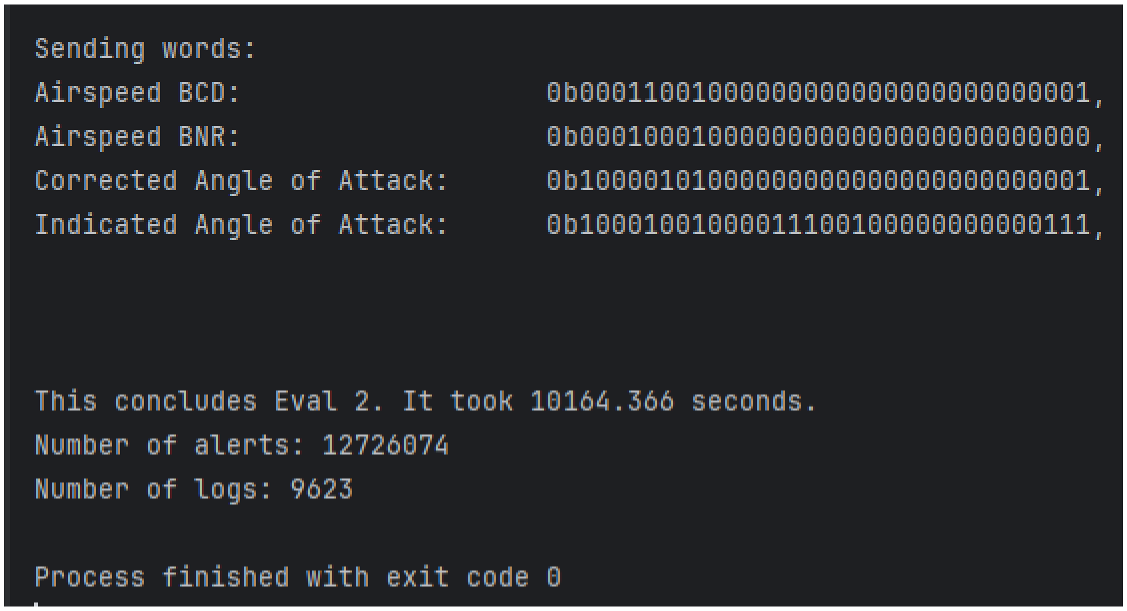
```
"C:\Users\mspre\Desktop\Practicum Resources\GATech_MS_Cybersecurity_Practicum_InfoSec_Summer24\venv\Scripts\python.exe" "C:\Users\mspre\Desktop\Practicum
 Resources\GATech_MS_Cybersecurity_Practicum_InfoSec_Summer24\ARINC429 Simulation\IDS_EVAL2.py"
Opening and analyzing flight data...
Finished opening and analyzing flight data in 13.999 seconds.
Press enter to start the test.
```

Figure 24: Eval 3 Starting



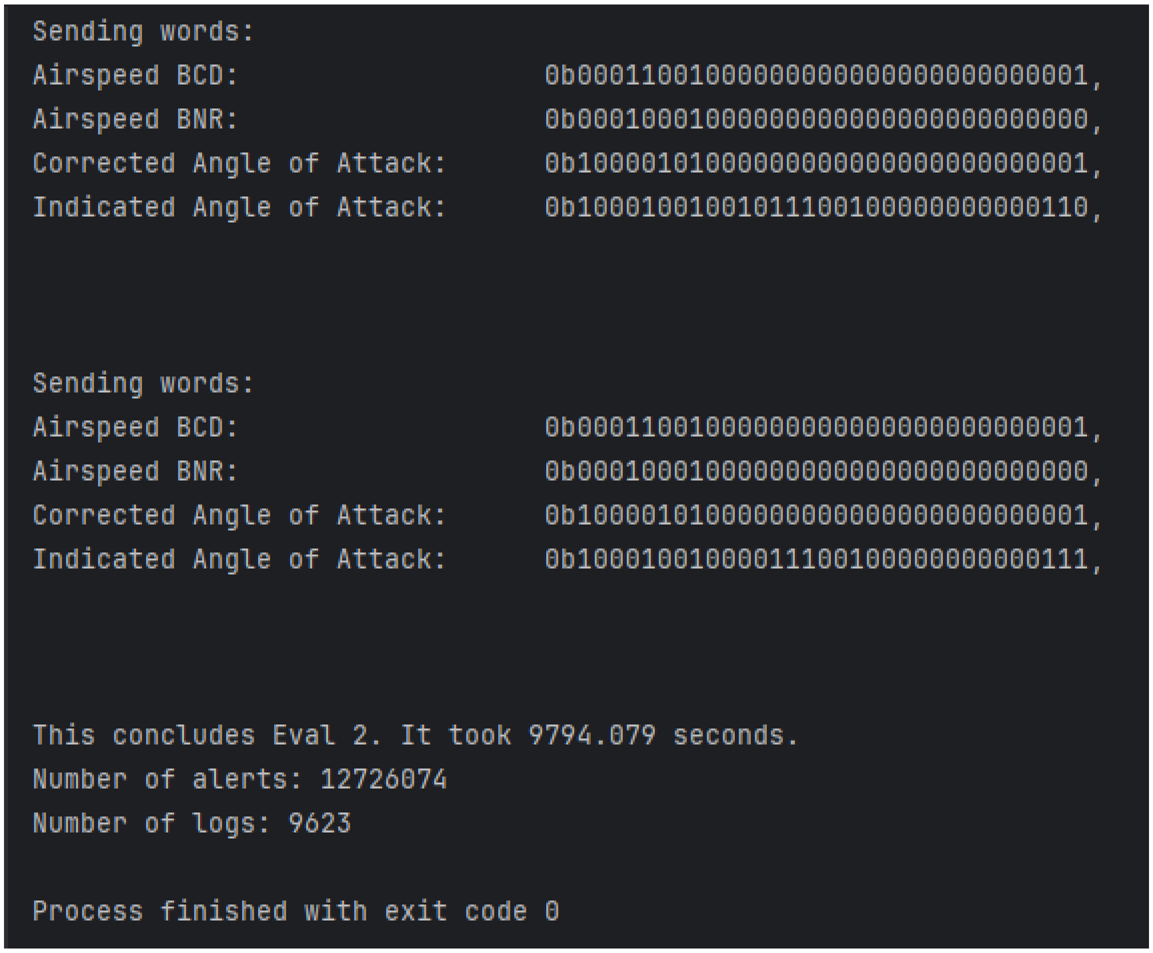
```
Sending words:
Airspeed BCD:                  0b00011001001100000101000000000001,
Airspeed BNR:                  0b00010001000010110110001000000000,
Corrected Angle of Attack:     0b10000101000111001000000000000111,
Indicated Angle of Attack:     0b10001001000011100100000000000111,
Latitude BNR:                  0b00010011001110110001111001110001,
Longitude BNR:                 0b10010011001110110001111001110000
```

Figure 25: Eval 3 Sample (words generated from the flight data plugged into the ADIRU)



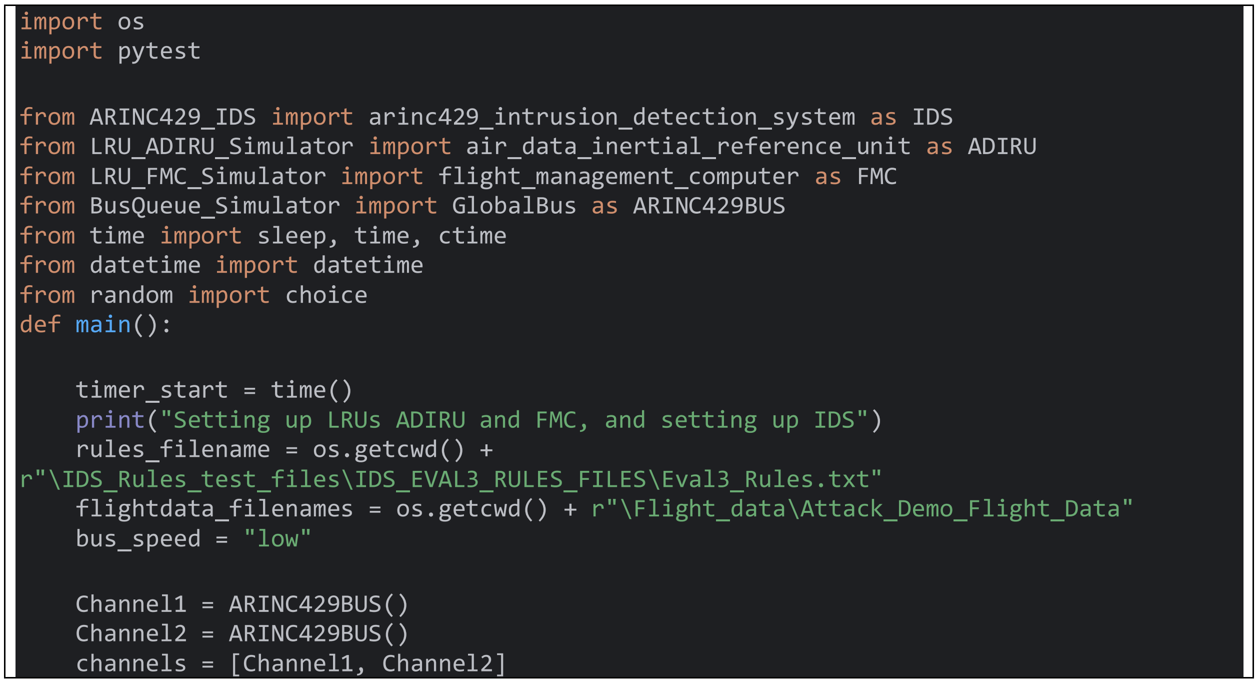
```
Sending words:
Airspeed BCD:                  0b00011001000000000000000000000001,
Airspeed BNR:                  0b00010001000000000000000000000000,
Corrected Angle of Attack:     0b10000101000000000000000000000001,
Indicated Angle of Attack:     0b10001001000011100100000000000111,




This concludes Eval 2. It took 10164.366 seconds.
Number of alerts: 12726074
Number of logs: 9623

Process finished with exit code 0
```

Figure 26: Eval 3 Completed showing the number of alerts and logs generated. Based on the rules file custom made for this test, this is expected.



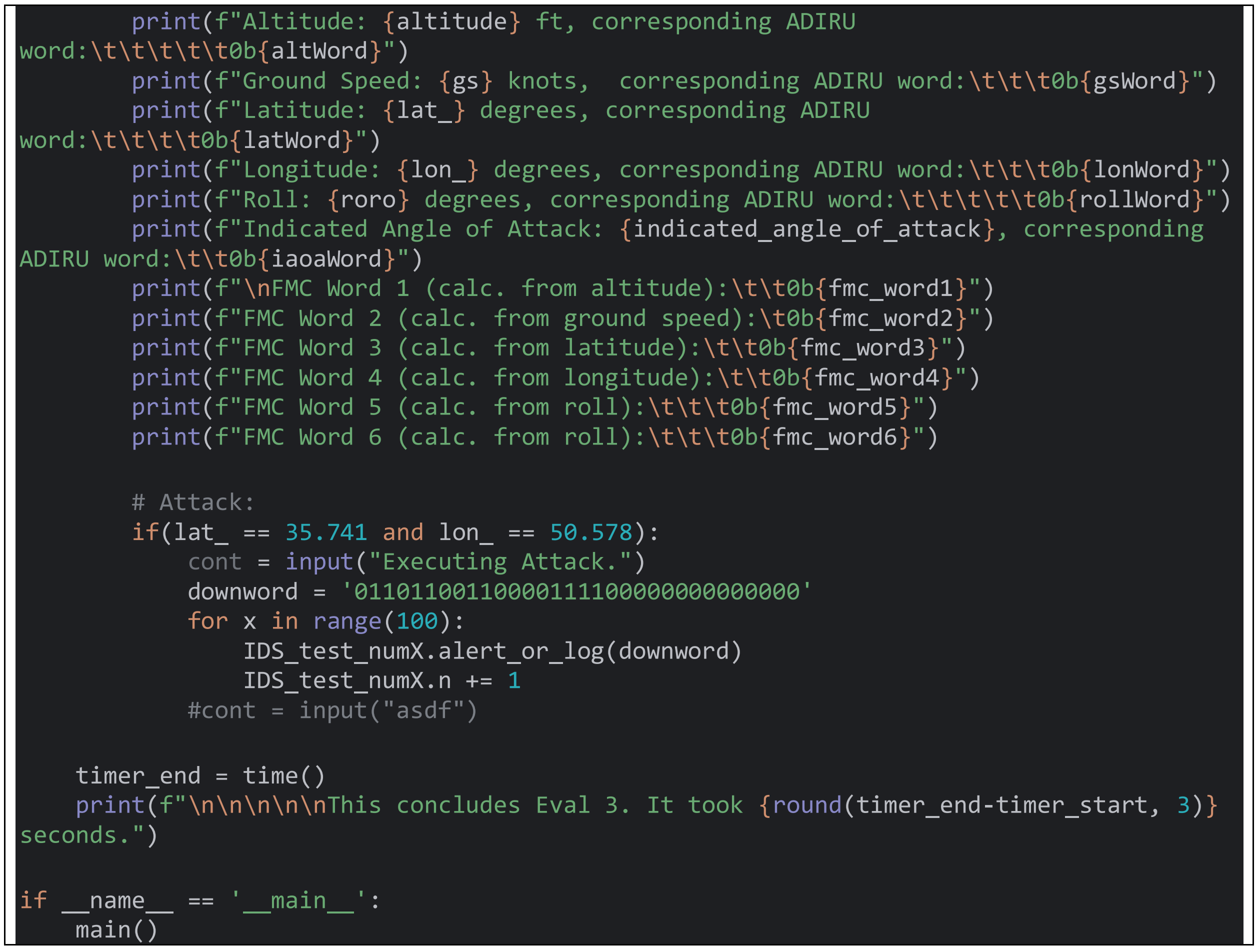
```
Sending words:
Airspeed BCD:                 0b00011001000000000000000000000001,
Airspeed BNR:                 0b00010001000000000000000000000000,
Corrected Angle of Attack:    0b10000101000000000000000000000001,
Indicated Angle of Attack:    0b10001001001011100100000000000110,



Sending words:
Airspeed BCD:                 0b00011001000000000000000000000001,
Airspeed BNR:                 0b00010001000000000000000000000000,
Corrected Angle of Attack:    0b10000101000000000000000000000001,
Indicated Angle of Attack:    0b10001001000001110010000000000111,



This concludes Eval 2. It took 9794.079 seconds.
Number of alerts: 12726074
Number of logs: 9623

Process finished with exit code 0
```

Figure 27: Another instance of Eval 3 Completed.



```python
import os
import pytest

from ARINC429_IDS import arinc429_intrusion_detection_system as IDS
from LRU_ADIRU_Simulator import air_data_inertial_reference_unit as ADIRU
from LRU_FMC_Simulator import flight_management_computer as FMC
from BusQueue_Simulator import GlobalBus as ARINC429BUS
from time import sleep, time, ctime
from datetime import datetime
from random import choice
def main():

    timer_start = time()
    print("Setting up LRUs ADIRU and FMC, and setting up IDS")
    rules_filename = os.getcwd() +
r"\IDS_Rules_test_files\IDS_EVAL3_RULES_FILES\Eval3_Rules.txt"
    flightdata_filenames = os.getcwd() + r"\Flight_data\Attack_Demo_Flight_Data"
    bus_speed = "low"

    Channel1 = ARINC429BUS()
    Channel2 = ARINC429BUS()
    channels = [Channel1, Channel2]
    IDS_test_numX = IDS(bus_speed, BUS_CHANNELS=channels, rules_file=rules_filename)
    transmitting_LRU = ADIRU(bus_speed, BUS_CHANNELS=channels)
    FMC_ = FMC(bus_speed, BUS_CHANNELS=channels)
    transmitting_LRU.set_sdi('00')

    # Check the output files:
    #print(IDS_test_numX.log_filepath)
    #print(IDS_test_numX.alert_filepath)
    alertfilePath = os.getcwd() +
r"\IDS_Rules_test_files\IDS_EVAL3_RULES_FILES\Alerts_Logs\Alerts_EVAL3.txt"
    logfilePath = os.getcwd() +
r"\IDS_Rules_test_files\IDS_EVAL3_RULES_FILES\Alerts_Logs\Logs_EVAL3.txt"
    # Reset the files in between runs:
    with open(alertfilePath,"w") as alert_fd:
        alert_fd.write(f"Starting EVAL3 test at {ctime()}\n")
    alert_fd.close()
    with open(logfilePath,"w") as log_fd:
        log_fd.write(f"Starting EVAL3 test at {ctime()}\n")
    log_fd.close()
    #print(alertfilePath)
    #print(logfilePath)
    # Some error handling:
    if(IDS_test_numX.alert_filepath != alertfilePath):
        IDS_test_numX.set_alertfile(alertfilePath)
    if(IDS_test_numX.log_filepath != logfilePath):
        IDS_test_numX.set_logfile(logfilePath)
    #print(IDS_test_numX.alert_filepath)
    #print(IDS_test_numX.log_filepath)
    #cont = input("")
    timer_end = time()
    print(f"Done setting up LRUs ADIRU, FMC, and IDS in {round(timer_end -
timer_start,10)} seconds")

    timer_start = time()
    print("Opening and analyzing flight data...")
    # Grab flight data to plug into ADIRU:
    # Remember to take off \n character in these files.
    # Dataset no. 1: Altitude.
    with open(flightdata_filenames+r"\Baro_Altitude.txt", "r") as altitude_fd:
        altitudes = altitude_fd.readlines()
    altitude_fd.close()
    # Dataset no. 2: Ground Speed.
    with open(flightdata_filenames+r"\Ground_Speed.txt", "r") as ground_speeds_fd:
        ground_speeds = ground_speeds_fd.readlines()
    ground_speeds_fd.close()
    # Dataset no. 3: Time as UTC Epoch.
    with open(flightdata_filenames+r"\Pos_Epoch.txt", "r") as times_fd:
        times_in_UNIX_epoch = times_fd.readlines()
    times_fd.close()
    # Dataset no. 4: Latitude and Longitudes.
    with open(flightdata_filenames+r"\Pos.txt", "r") as latlon_fd:
        lats_lons = latlon_fd.readlines()
    latlon_fd.close()
    # Dataset no. 5: Longitude.
    with open(flightdata_filenames+r"\Roll.txt", "r") as roll_fd:
        rolls = roll_fd.readlines()
    roll_fd.close()

    #print(len(altitudes), len(ground_speeds), len(times_in_UNIX_epoch),
len(lats_lons), len(rolls))

    # Data Handling:
    # Altitudes:
    prev_alt = 250.0 # There are some points where it cuts off, so I'll interpolate:
    for altIndex in range(len(altitudes)):
        if(altitudes[altIndex] == 'n/a\n'):
            altitudes[altIndex] = prev_alt
        else:
            alt = altitudes[altIndex].replace(' ft','').replace('\n','')
            try:
                alt = int(alt)
            except ValueError:
                #print(alt)
                alt = int(alt.split(' ')[1])
            altitudes[altIndex] = alt
        prev_alt = altitudes[altIndex]
    # Ground Speeds:
    # There are some points where it cuts off, so I'll interpolate:
    prev_gs = 181.0
    for gsIndex in range(len(ground_speeds)):
        if(ground_speeds[gsIndex] == 'n/a\n'):
            ground_speeds[gsIndex] = prev_gs
        else:
            ground_speeds[gsIndex] = float(ground_speeds[gsIndex].split(' ')[0])
        prev_gs = ground_speeds[gsIndex]
    # Time:
    for timeIndex in range(len(times_in_UNIX_epoch)):
        times_in_UNIX_epoch[timeIndex] =
int(times_in_UNIX_epoch[timeIndex].replace('\n',''))
    lats = []
    lons = []
    for posIndex in range(len(lats_lons)):
        line = lats_lons[posIndex].encode('utf-
8').replace(b'\n',b'').replace(b'\xc3\x82\xc2\xb0',b'').replace(b',',b'')
        line = line.decode('utf-8').split(' ')
        lat = float(line[0])
        lon = float(line[1])
        lats.append(lat)
        lons.append(lon)
    for rollIndex in range(len(rolls)):
        if(rolls[rollIndex] == 'n/a\n' or rolls[rollIndex] == 'n/a'):
            rolls[rollIndex] = 0.0 # in a runway so no roll obvs.
        else:
            rolls[rollIndex] = float(rolls[rollIndex].replace('\n',''))
    # Interprolate Indicated AoA
    with open(os.getcwd() + r"\Flight_data\Tail_687_1"+r"\INDICATED ANGLE OF
ATTACK_Tail_687_1_data.txt", "r") as indicated_aoa_fd:
        long_indicated_aoas = indicated_aoa_fd.readlines()
    indicated_aoa_fd.close()
    # Real world data set opened.
    # Starting interpolation.
    interprolated_aoa = []
    # Climbing:
```

```
    for aoaIndex1 in range(5981):
        iaoa = round(float(choice(long_indicated_aoas).replace('\n','')),3)
        while(iaoa < 0.0 or iaoa >= 3.0):
            iaoa = round(float(choice(long_indicated_aoas).replace('\n','')),3)
        interprolated_aoa.append(iaoa)
    interprolated_aoa = sorted(interprolated_aoa)
    # Stable:
    for aoaIndex2 in range(5981):
        iaoa = round(float(choice(long_indicated_aoas).replace('\n','')),3)
        while(iaoa < -2.0 or iaoa > 2.0):
            iaoa = round(float(choice(long_indicated_aoas).replace('\n','')),3)
        interprolated_aoa.append(iaoa)
    # Descending:
    desc_iaoa = []
    for aoaIndex2 in range(5982):
        iaoa = round(float(choice(long_indicated_aoas).replace('\n','')),3)
        while(iaoa < 0.0 or iaoa >= 3.0):
            iaoa = round(float(choice(long_indicated_aoas).replace('\n','')),3)
        desc_iaoa.append(iaoa)
    desc_iaoa = reversed(sorted(desc_iaoa))
    for desc_iaoa_ in desc_iaoa:
        interprolated_aoa.append(desc_iaoa_)
    #print(interprolated_aoa)
    #print(len(interprolated_aoa))


#print(f"Tables:\n{altitudes}\n{ground_speeds}\n{times_in_UNIX_epoch}\n{lats}\n{lons}\n
{rolls}")
    #print(len(altitudes), len(ground_speeds), len(times_in_UNIX_epoch), len(lats),
len(lons), len(rolls))

    timer_end = time()
    print(f"Finished opening and analyzing flight data in {round(timer_end-
timer_start,3)} seconds.")
    cont = input("Press enter to start the test.")
    # Start Eval 3:
    timer_start = time()
    print("Beginning flight data evaluation 3...")
    prev_altitude = altitudes[0]
    prev_gs = ground_speeds[0]
    prev_lat_ = lats[0]
    prev_lon_ = lons[0]
    prev_roro = rolls[0]
    prev_indicated_angle_of_attack = interprolated_aoa[0]
    # I have 17944 data points in each array:
    for index in range(17944):
        # Get all the data points:
        altitude = altitudes[index]
        gs = ground_speeds[index]
        t = times_in_UNIX_epoch[index]
        datetime_object = datetime.fromtimestamp(t)
        lat_ = lats[index]
        lon_ = lons[index]
        roro = rolls[index]
        indicated_angle_of_attack = interprolated_aoa[index]
        # Turn those into data points when applicable:
        transmitting_LRU.set_value("Baro Corrected Altitude #1", str(altitude) + "
knots")
        transmitting_LRU.set_value("Ground Speed", str(gs) + " knots")
        transmitting_LRU.set_value("Present Position - Latitude", str(lat_) + "
degrees")
        transmitting_LRU.set_value("Present Position - Longitude", str(lon_) + "
degrees")
        transmitting_LRU.set_value("Roll Angle", str(roro) + " deg")
        transmitting_LRU.set_value('Indicated Angle of Attack (Average)',
str(indicated_angle_of_attack) + ' deg')
        # Get the words from those values:
        altWord = transmitting_LRU.encode_word(0o204)
        gsWord = transmitting_LRU.encode_word(0o012)
        latWord = transmitting_LRU.encode_word(0o310)
        lonWord = transmitting_LRU.encode_word(0o311)
        rollWord = transmitting_LRU.encode_word(0o325)
        iaoaWord = transmitting_LRU.encode_word(0o221)
        # Send to FMC & IDS:
        # Handle Altitude Words:
        IDS_test_numX.alert_or_log(altWord)
        IDS_test_numX.n += 1
        fmc_word1 = FMC_.decodeADIRUword(altWord, prev_altitude)
        IDS_test_numX.alert_or_log(fmc_word1)
        # Handle Ground Speed Words:
        IDS_test_numX.alert_or_log(gsWord)
        IDS_test_numX.n += 1
        fmc_word2 = FMC_.decodeADIRUword(gsWord, prev_gs)
        IDS_test_numX.alert_or_log(fmc_word2)
        # Handle Latitude Words:
        IDS_test_numX.alert_or_log(latWord)
        IDS_test_numX.n += 1
        fmc_word3 = FMC_.decodeADIRUword(latWord, prev_lat_)
        IDS_test_numX.alert_or_log(fmc_word3)
        # Handle Longitude Words:
        IDS_test_numX.alert_or_log(lonWord)
        IDS_test_numX.n += 1
        fmc_word4 = FMC_.decodeADIRUword(lonWord, prev_lon_)
        IDS_test_numX.alert_or_log(fmc_word4)
        # Handle Roll Words:
        IDS_test_numX.alert_or_log(rollWord)
        IDS_test_numX.n += 1
        fmc_word5 = FMC_.decodeADIRUword(rollWord, prev_roro)
        IDS_test_numX.alert_or_log(fmc_word5)
        # Handle Indicated Angle of Attack Words:
        IDS_test_numX.alert_or_log(iaoaWord)
        IDS_test_numX.n += 1
        fmc_word6 = FMC_.decodeADIRUword(iaoaWord, prev_indicated_angle_of_attack)
        IDS_test_numX.alert_or_log(fmc_word6)
        # Record the next prev value for the FMC:
        prev_altitude = altitude
        prev_gs = gs
        prev_lat_ = lat_
        prev_lon_ = lon_
        prev_roro = roro
        prev_indicated_angle_of_attack = indicated_angle_of_attack
        # Visualization for the test:
        print('\n\n')
        print(f"Flight data points gathered for step #{index+1}:")
        print(f"Time: {datetime_object},")
```



```
        print(f"Altitude: {altitude} ft, corresponding ADIRU
word:\t\t\t\t\t0b{altWord}")
        print(f"Ground Speed: {gs} knots,  corresponding ADIRU word:\t\t\t0b{gsWord}")
        print(f"Latitude: {lat_} degrees, corresponding ADIRU
word:\t\t\t\t0b{latWord}")
        print(f"Longitude: {lon_} degrees, corresponding ADIRU word:\t\t\t0b{lonWord}")
        print(f"Roll: {roro} degrees, corresponding ADIRU word:\t\t\t\t\t0b{rollWord}")
        print(f"Indicated Angle of Attack: {indicated_angle_of_attack}, corresponding
ADIRU word:\t\t0b{iaoaWord}")
        print(f"\nFMC Word 1 (calc. from altitude):\t\t0b{fmc_word1}")
        print(f"FMC Word 2 (calc. from ground speed):\t0b{fmc_word2}")
        print(f"FMC Word 3 (calc. from latitude):\t\t0b{fmc_word3}")
        print(f"FMC Word 4 (calc. from longitude):\t\t0b{fmc_word4}")
        print(f"FMC Word 5 (calc. from roll):\t\t\t0b{fmc_word5}")
        print(f"FMC Word 6 (calc. from roll):\t\t\t0b{fmc_word6}")

        # Attack:
        if(lat_ == 35.741 and lon_ == 50.578):
            cont = input("Executing Attack.")
            downword = '01101100110000111100000000000000'
            for x in range(100):
                IDS_test_numX.alert_or_log(downword)
                IDS_test_numX.n += 1
            #cont = input("asdf")

    timer_end = time()
    print(f"\n\n\n\n\nThis concludes Eval 3. It took {round(timer_end-timer_start, 3)}
seconds.")

if __name__ == '__main__':
    main()
```

Figure 28: Eval 4 (IDS_Eval3.py) code.

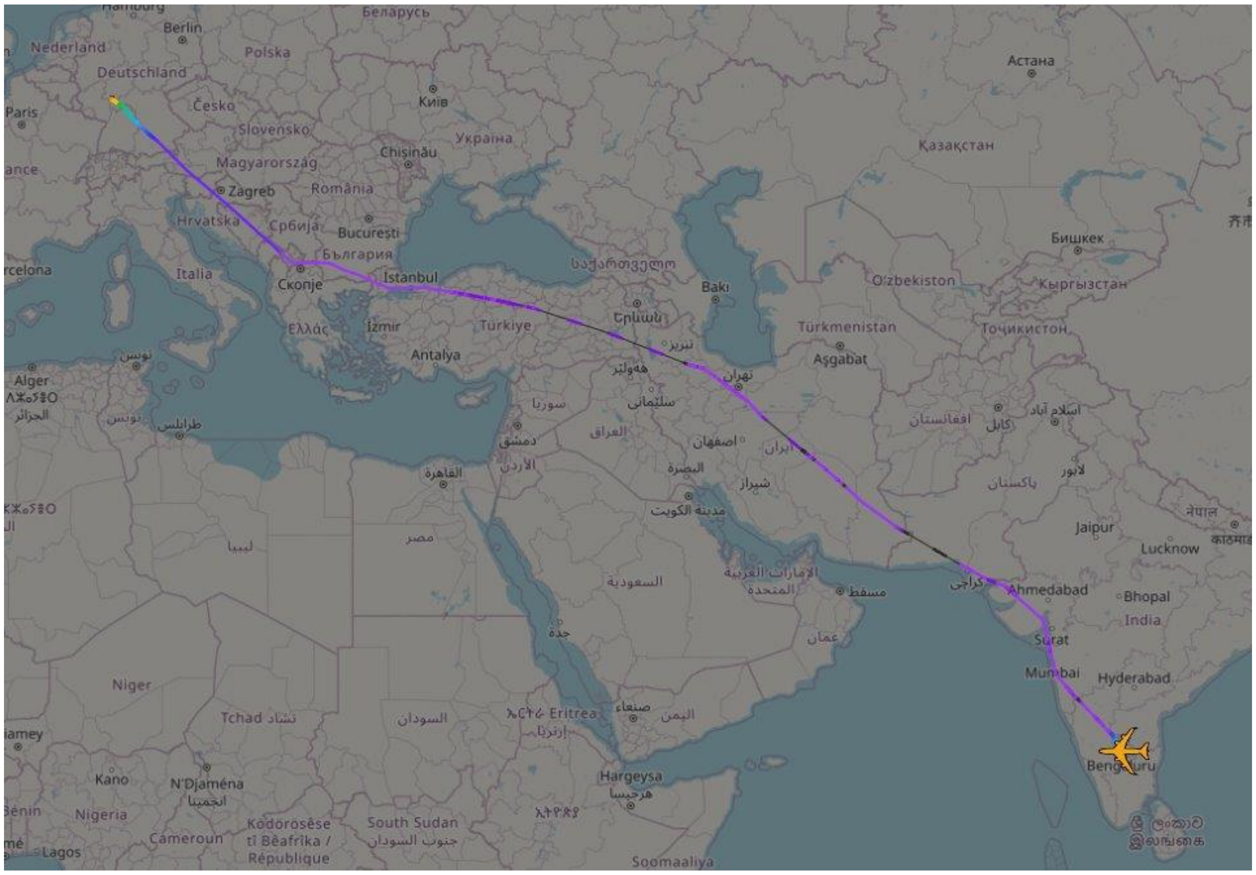
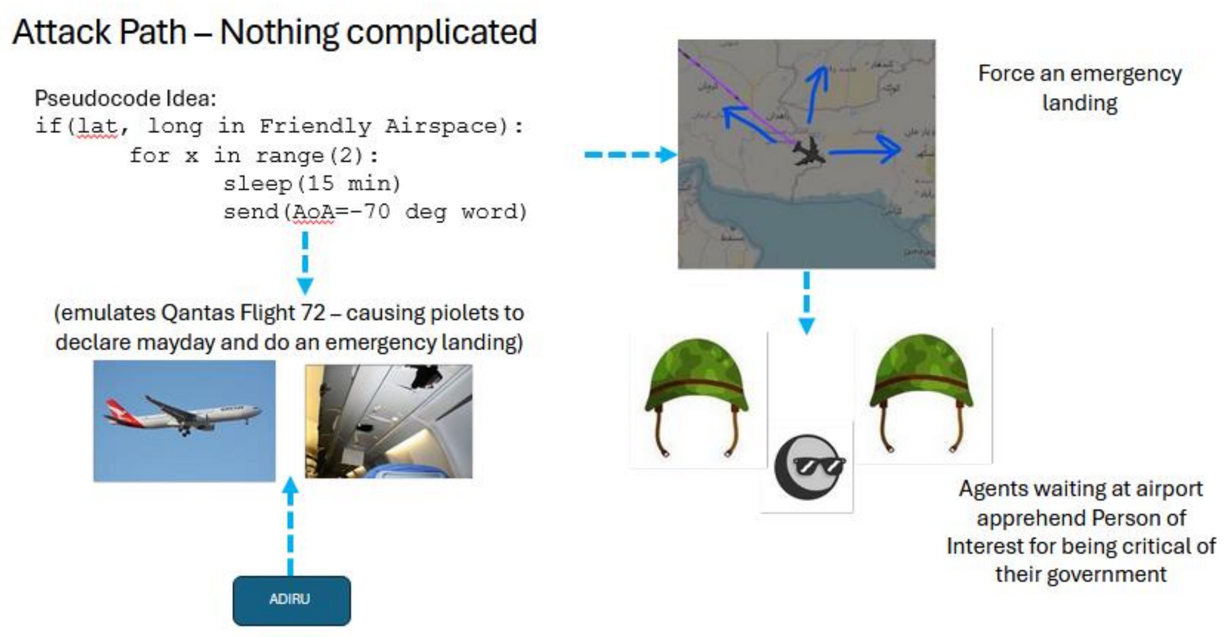


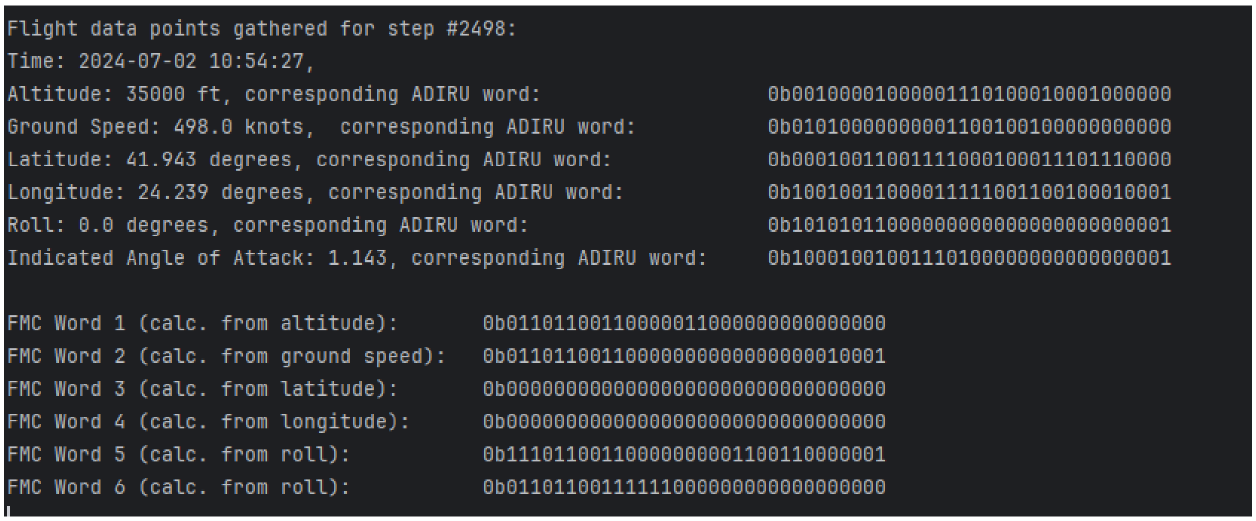
Figure 29: The flight data scraped from ADS-B exchange for Eval 3. The idea here is to force a mayday and emergency landing by making some components go haywire temporarily while over specific countries. Source: https://globe.adsbexchange.com/?icao=3c4b35&lat=40.621&lon=52.094&zoom=4.7&showTrace=2024-07-02&leg=2


Attack Path – Nothing complicated
Pseudocode Idea:

```
if(lat, long in Friendly Airspace):
        for x in range(2):
                sleep(15 min)
                send(AoA=-70 deg word)
```

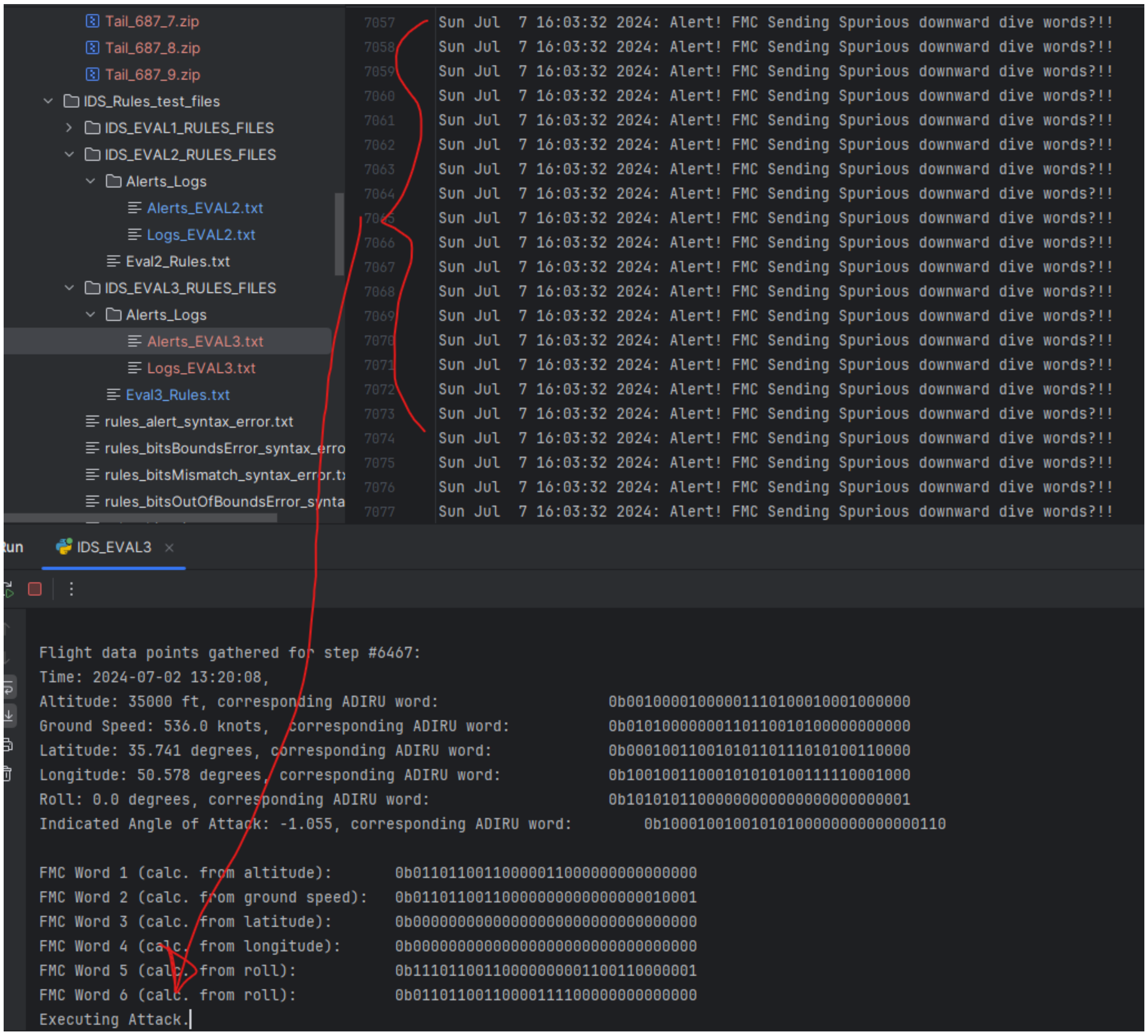
(emulates Qantas Flight 72 – causing piolets to declare mayday and do an emergency landing)
ADIRU
Force an emergency landing
Agents waiting at airport apprehend Person of Interest for being critical of their government

Figure 30: Visualized Attack Path where person of interest on airplane is forced to land in an unfriendly country.



```
Flight data points gathered for step #2498:
Time: 2024-07-02 10:54:27,
Altitude: 35000 ft, corresponding ADIRU word:                0b00100001000001110100010001000000
Ground Speed: 498.0 knots,  corresponding ADIRU word:        0b01010000000001100100100000000000
Latitude: 41.943 degrees, corresponding ADIRU word:          0b00010011001111000100011101110000
Longitude: 24.239 degrees, corresponding ADIRU word:         0b10010011000011111001100100010001
Roll: 0.0 degrees, corresponding ADIRU word:                 0b10101011000000000000000000000001
Indicated Angle of Attack: 1.143, corresponding ADIRU word:     0b10001001001110100000000000000001

FMC Word 1 (calc. from altitude):       0b01101100110000011000000000000000
FMC Word 2 (calc. from ground speed):   0b01101100110000000000000000010001
FMC Word 3 (calc. from latitude):       0b00000000000000000000000000000000
FMC Word 4 (calc. from longitude):      0b00000000000000000000000000000000
FMC Word 5 (calc. from roll):           0b11101100110000000001100110000001
FMC Word 6 (calc. from roll):           0b01101100111111000000000000000000
```

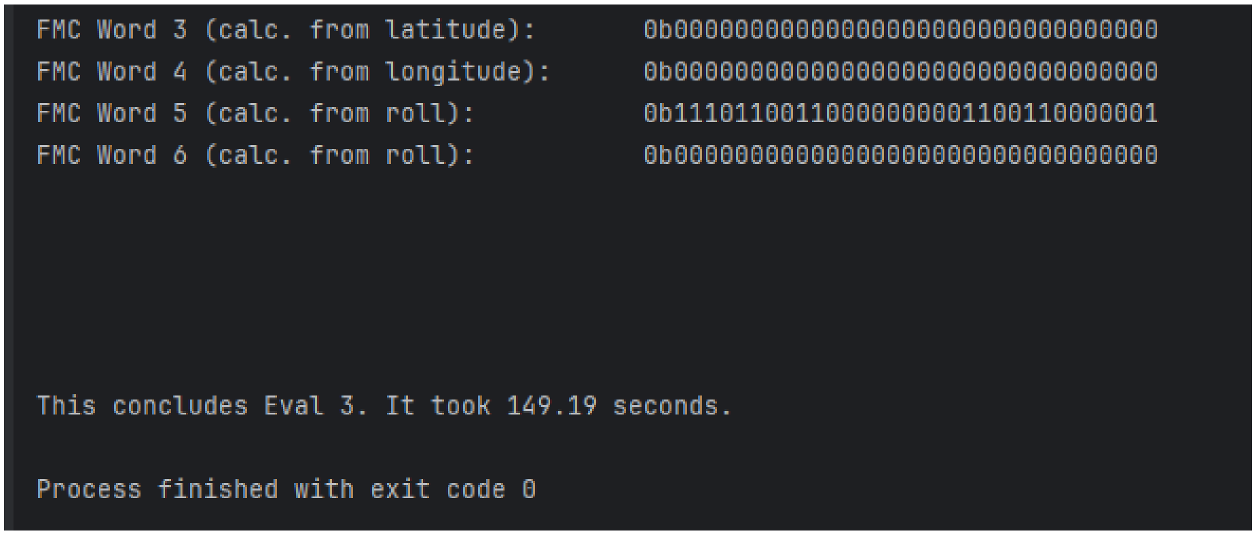
Figure 31: A sample from Eval 4 of words being sent to the IDS.



```
Sun Jul  7 16:03:32 2024: Alert! FMC Sending Spurious downward dive words?!!
Sun Jul  7 16:03:32 2024: Alert! FMC Sending Spurious downward dive words?!!
Sun Jul  7 16:03:32 2024: Alert! FMC Sending Spurious downward dive words?!!
Sun Jul  7 16:03:32 2024: Alert! FMC Sending Spurious downward dive words?!!
Sun Jul  7 16:03:32 2024: Alert! FMC Sending Spurious downward dive words?!!
Sun Jul  7 16:03:32 2024: Alert! FMC Sending Spurious downward dive words?!!
Sun Jul  7 16:03:32 2024: Alert! FMC Sending Spurious downward dive words?!!
Sun Jul  7 16:03:32 2024: Alert! FMC Sending Spurious downward dive words?!!
Sun Jul  7 16:03:32 2024: Alert! FMC Sending Spurious downward dive words?!!
Sun Jul  7 16:03:32 2024: Alert! FMC Sending Spurious downward dive words?!!
Sun Jul  7 16:03:32 2024: Alert! FMC Sending Spurious downward dive words?!!
Sun Jul  7 16:03:32 2024: Alert! FMC Sending Spurious downward dive words?!!
Sun Jul  7 16:03:32 2024: Alert! FMC Sending Spurious downward dive words?!!
Sun Jul  7 16:03:32 2024: Alert! FMC Sending Spurious downward dive words?!!
Sun Jul  7 16:03:32 2024: Alert! FMC Sending Spurious downward dive words?!!
Sun Jul  7 16:03:32 2024: Alert! FMC Sending Spurious downward dive words?!!
Sun Jul  7 16:03:32 2024: Alert! FMC Sending Spurious downward dive words?!!
Sun Jul  7 16:03:32 2024: Alert! FMC Sending Spurious downward dive words?!!
Sun Jul  7 16:03:32 2024: Alert! FMC Sending Spurious downward dive words?!!
Sun Jul  7 16:03:32 2024: Alert! FMC Sending Spurious downward dive words?!!
Sun Jul  7 16:03:32 2024: Alert! FMC Sending Spurious downward dive words?!!
```

```
Flight data points gathered for step #6467:
Time: 2024-07-02 13:20:08,
Altitude: 35000 ft, corresponding ADIRU word:                 0b00100001000001110100010001000000
Ground Speed: 536.0 knots,  corresponding ADIRU word:         0b01010000000110110010100000000000
Latitude: 35.741 degrees, corresponding ADIRU word:           0b00010011001010110111010100110000
Longitude: 50.578 degrees, corresponding ADIRU word:          0b10010011000101010100111110001000
Roll: 0.0 degrees, corresponding ADIRU word:                  0b10101011000000000000000000000001
Indicated Angle of Attack: -1.055, corresponding ADIRU word:      0b10001001001010100000000000000110

FMC Word 1 (calc. from altitude):      0b01101100110000011000000000000000
FMC Word 2 (calc. from ground speed):  0b01101100110000000000000000010001
FMC Word 3 (calc. from latitude):      0b00000000000000000000000000000000
FMC Word 4 (calc. from longitude):     0b00000000000000000000000000000000
FMC Word 5 (calc. from roll):          0b11101100110000000001100110000001
FMC Word 6 (calc. from roll):          0b01101100110000111100000000000000
Executing Attack.
```

Figure 32: Alerts generated based on attack where the FMC is sending downward dive words but the ADIRU’s indicated AoA does not match (middle of test).



```
FMC Word 3 (calc. from latitude):       0b00000000000000000000000000000000
FMC Word 4 (calc. from longitude):      0b00000000000000000000000000000000
FMC Word 5 (calc. from roll):           0b11101100110000000001100110000001
FMC Word 6 (calc. from roll):           0b00000000000000000000000000000000






This concludes Eval 3. It took 149.19 seconds.

Process finished with exit code 0
```

Figure 33: End of Eval 4 test.