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.

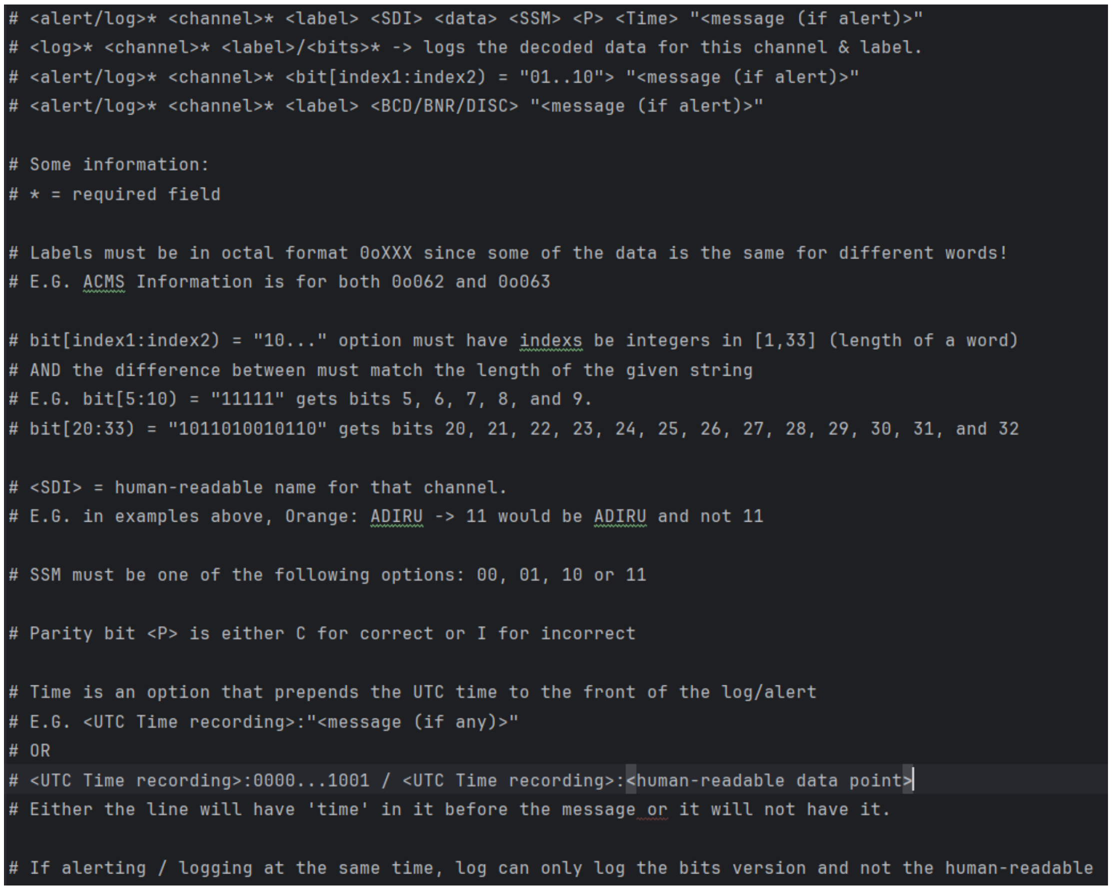
My rules-based IDS is geared towards alerting an aircraft crew of a cyber threat. If it flags a word or sequence of words as malicious I envision it alerting someone (i.e. a pilot, or trained flight attendant) to let them know of a cyber-attack. This would then allow for better situational awareness 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 Weeks):

Here's what I promised from the last progress report:

- Fix the label encoding problem
    - This was the first task I did last week, and the label encoding is now fixed.
- Simulate a transmitter LRU, the ADIRU, and test its hookup to the flight simulator to make sure it can input data correctly.
    - I make the python file LRU_ADIRU_Simulator.py, and various test cases in it to check its functionality and ability to encode words correctly, as well as decode words from the GPS. This took a long chunk of time, as like the other LRUs, it has a lot of words that it needed to encode.

- Create vulnerable program for FMC and a simple buffer overflow / rop hack for it to gain system access to the FMC.
  - I ran out of time this week to start the ROP attack, but I have already made a simple program from the last progress report that is vulnerable to a bounds overflow.
- Create the orange ARINC429 bus.
  - I created this in my main python file/function as it's own function with the GPS and ADIRU LRUs as their own defined objects.
- Create the blue ARINC429 bus.
  - I created this in my main python file/function as it's own function with the FMC and ADIRU LRUs as their own defined objects.
- Create final report outline.
  - This is in progress, but basically my report outline is:
    - Abstract
    - Model / Thread Model Explanation
    - Simulation Explanation
    - IDS Explanation
    - IDS Evaluation Metrics
- Create the purple/channel B ARINC 429 bus
  - I also added this to the code in the main python file/function.
- Create the green/channel A ARINC 429 bus
  - I also added this to the code in the main python file/function.
- 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 EECs that pitch the plane into a downward trajectory.
  - Not started since I ran out of time for the attack above. I will start this next week.
- Start the rules-based IDS system
  - I started this, see below two points for details.
- Implement the functionality to it: Create syntax for IDS rules
  - I created the following syntax: The IDS will read a rules file and parse that to create logical rules. This file will have: Location of outfiles for alerts/logs, Bus Channel Definitions, SDI Definitions, and Rules. Each section is declared by a new ! Character and aptly named: !Outfiles, !Channels, !SDI, and !Rules. They are all required.



```
# <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
```

    - The syntax for the Rules is above ^.
- Create functionality for logging bus traffic via the IDS:
    - I created a log function in the IDS that given a word writes it to the log file.
- Start logging ARINC 429 traffic for data collection.
    - In Progress → I am collecting words here with my IDS.

Additionally, I created a better defined evaluation test plan and added that to the evaluation section and timeline below.

It was noted on Ed Discussion that a draft of the product should be included in Progress Report 3. Since my code base as a whole is too large to fit into a progress report here, I will assume it is sufficient to show my main function file, please see figures 15 – 18 in the appendix.

Questions I have or Issues I'm running into:

1. First question, please let me know if this progress report is too sparse in the updates on the details. From feedback on the last progress report, I was too detailed.
2. Is there a particular product I missed that you would like to see in the next progress report?
3. One challenge that I ran into was time. I underestimate how long it will take me to do a certain task, and I end up taking more time to do everything than I plan for. This, coupled with some large commitments from work that popped up made me not complete the attack task from this schedule. I know this is probably a personal issue, and something I need to work on.

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.

Evaluation Plan:

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.

2. 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.

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.

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.

Timeline:

Week #	**Description of Task**	**Status**
Week 1	Research topic by reading articles. Focus the research on understanding how the ARINC 429	Completed

Monday, 13 May 2024 – Sunday 19 May 2024	protocol works.	
	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 bus, 1553 bus, and ARINC 629 to just ARINC 429.	Completed
	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	Completed

	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 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.	In Progress
	Create the orange ARINC429 bus.	Completed
	Create the blue ARINC429 bus.	Completed

Week 6 Monday, 17 June 2024 – Sunday 23 June 2024	Create final report outline.	In Progress
	Create the purple/channel B ARINC 429 bus	Completed
	Create the green/channel A ARINC 429 bus	Completed
	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.	In Progress
	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.	In Progress
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)	Not Started
	Implement Step 2 from Evaluation Plan	Not Started
Week 8 Monday, 1 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	Not Started

Sunday 7 July 2024	upon any of the alarms from IDS	
	Implement Evaluation Plan Step 3 and 4	Not Started
Week 9 Monday, 8 July 2024 – Sunday 14 July 2024	Create final demo presentation. It should include a demo of the simulation, attack, and defense implementations working with the attack.	Not Started
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:

To be delivered by progress report 4.

Report Outline:

To be delivered by progress report 5.

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

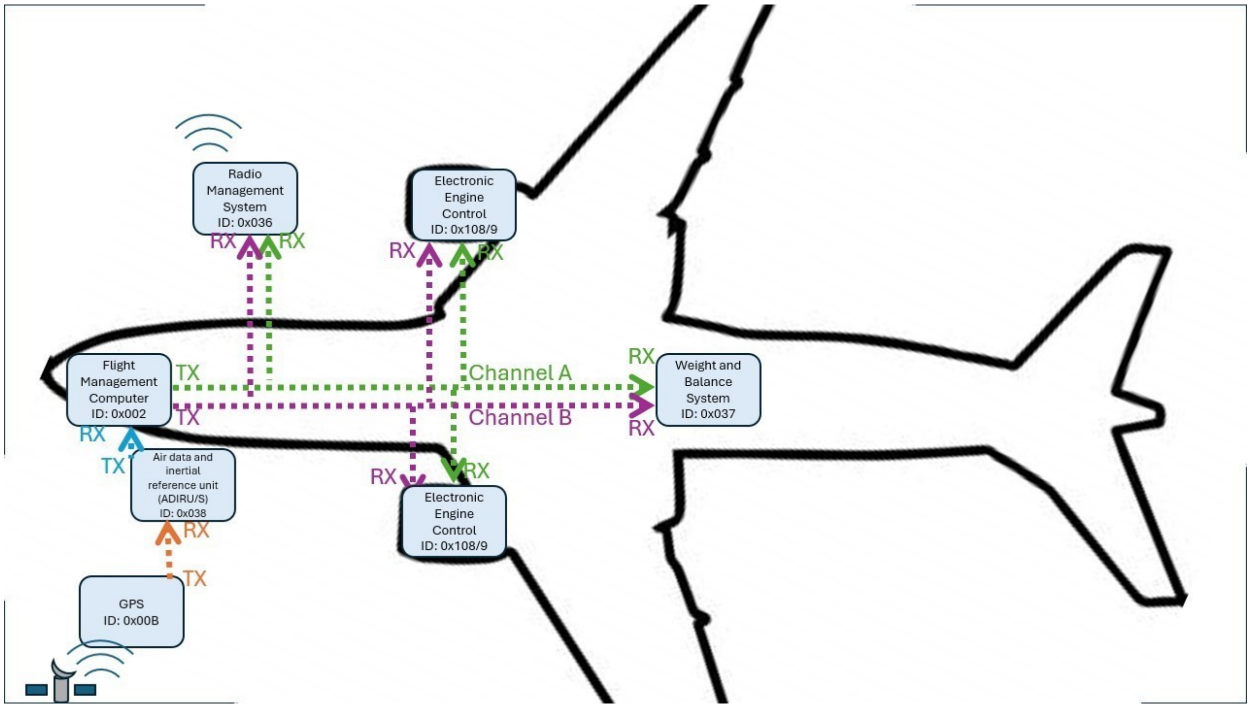
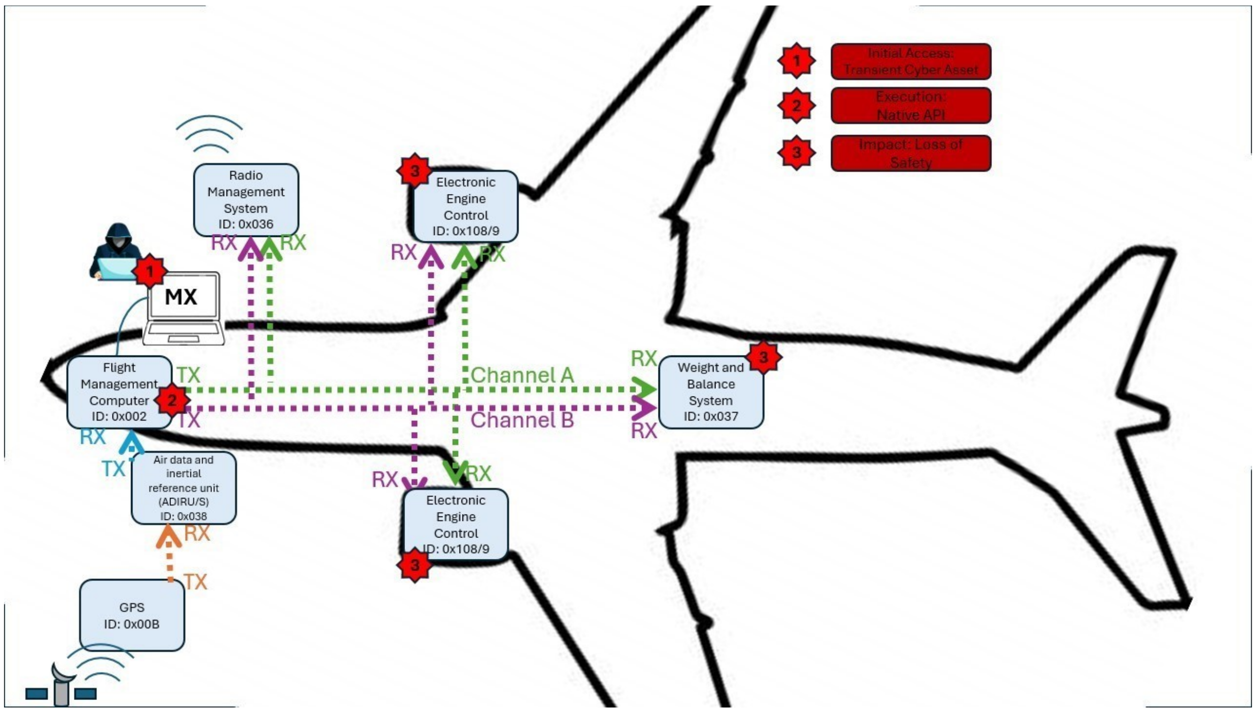

Radio Management System ID: 0x036
RX
RX
Electronic Engine Control ID: 0x108/9
RX
RX
Flight Management Computer ID: 0x002
TX
TX
Channel A
Channel B
RX
RX
Weight and Balance System ID: 0x037
RX
TX
Air data and inertial reference unit (ADIRU/S) ID: 0x038
RX
TX
GPS ID: 0x00B
RX
RX
Electronic Engine Control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

Figure 1: System Model



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

Figure 2: Threat Attack Model

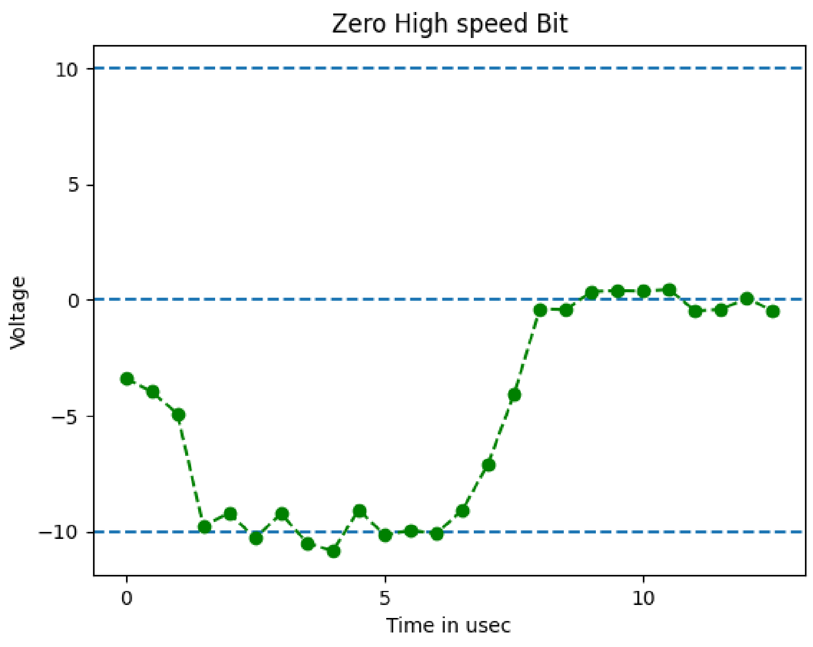


Figure 3: A set of voltages over time that represents a High Speed 0 bit.

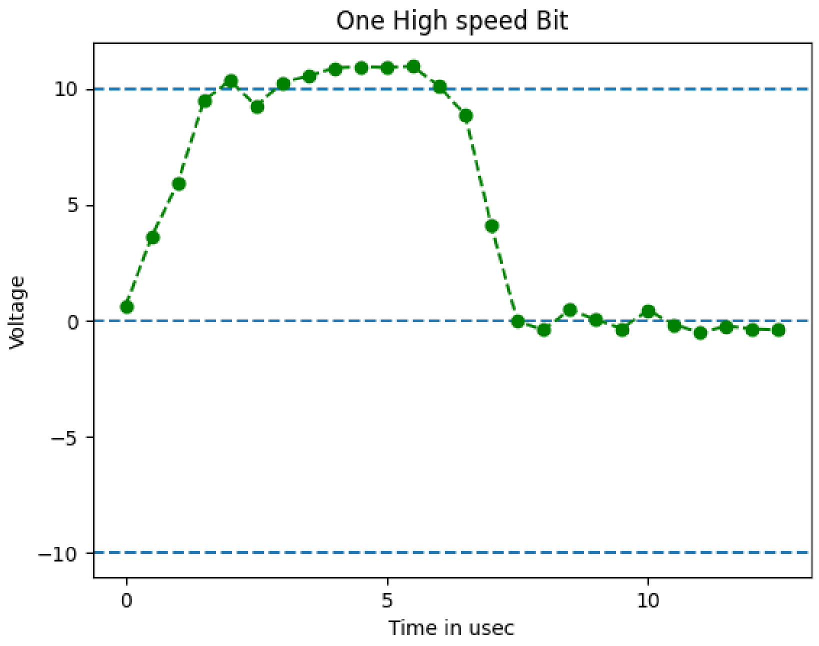


Figure 4: A set of voltages over time that represents a High Speed 1 bit.

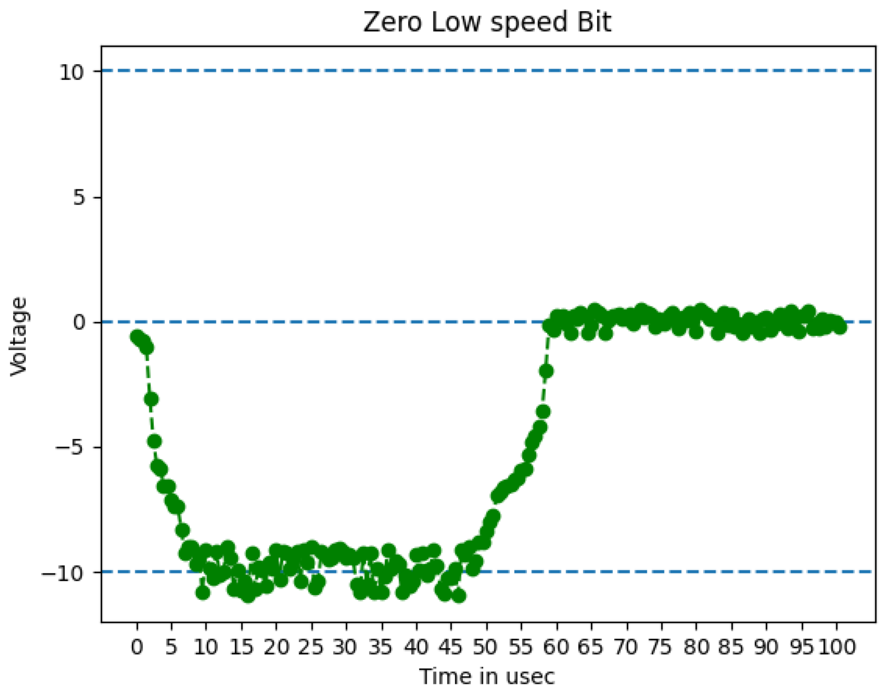


Figure 5: A set of voltages over time that represents a Low Speed 0 bit.

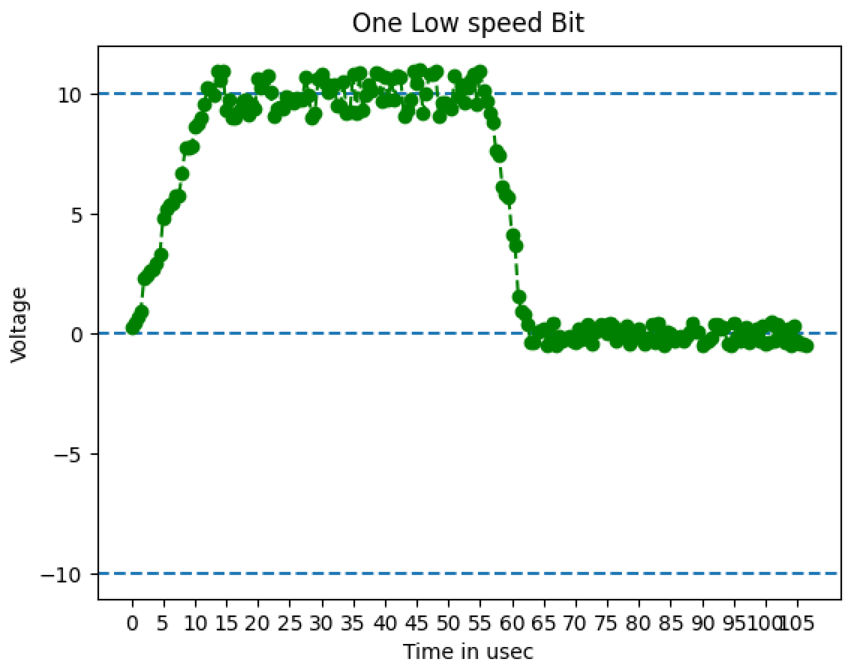


Figure 6: A set of voltages over time that represents a Low Speed 1 bit.

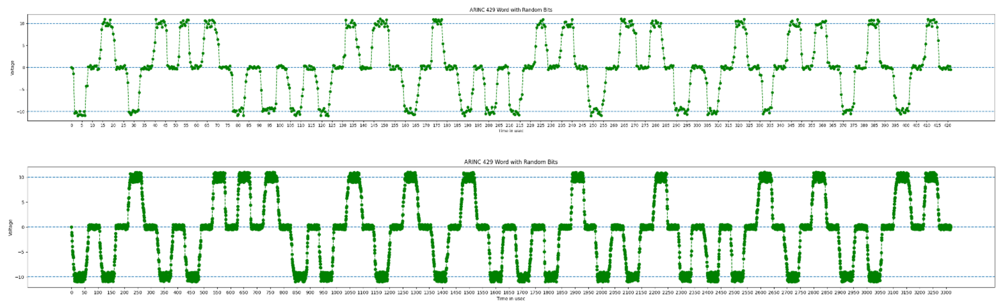


Figure 7 & 8: A set of voltages over times that represent a random word for respectively High and Low speed data buses.

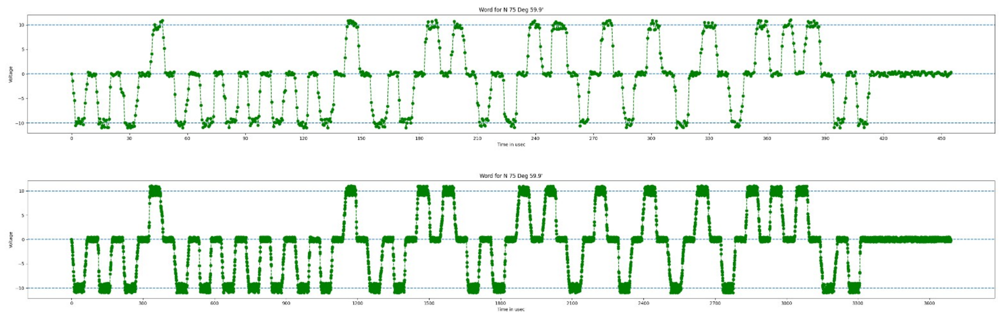


Figure 9 & 10: A set of voltages over time that represents a latitude information word at respectively High and Low Bus Speeds.

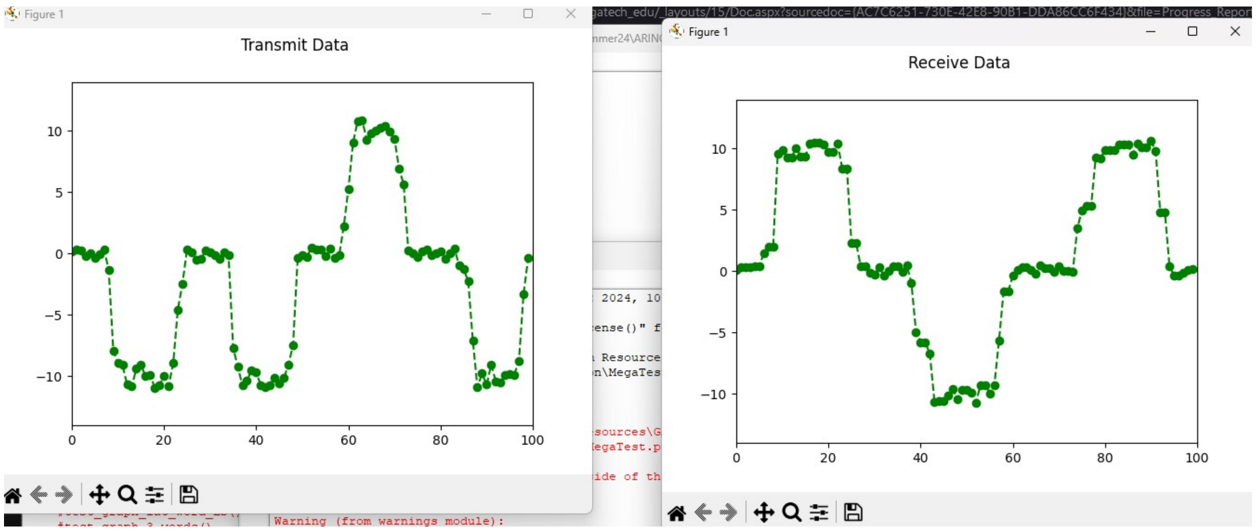
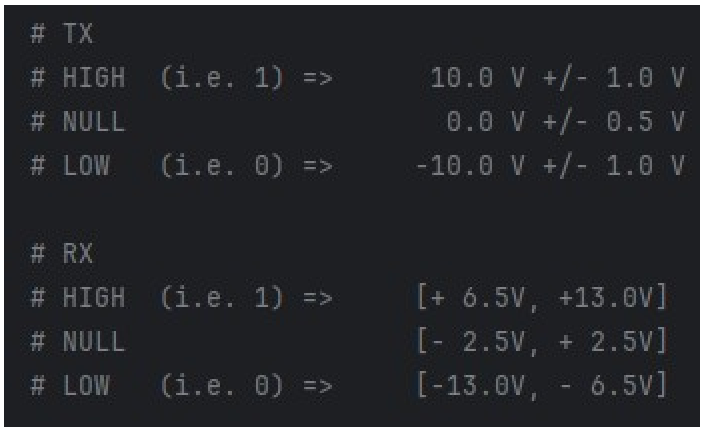


Figure 11: A screenshot of the bus TX and RX bus data over time. Notice the RX is delayed from the TX graph.



```
# 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.

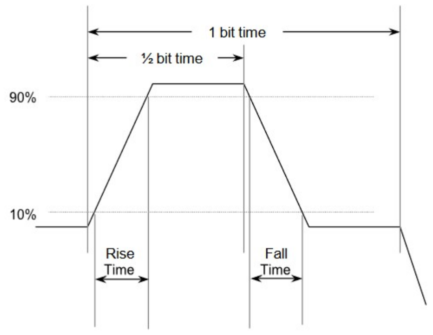


Figure 13: Real Voltage Specification from Progress Report 2.

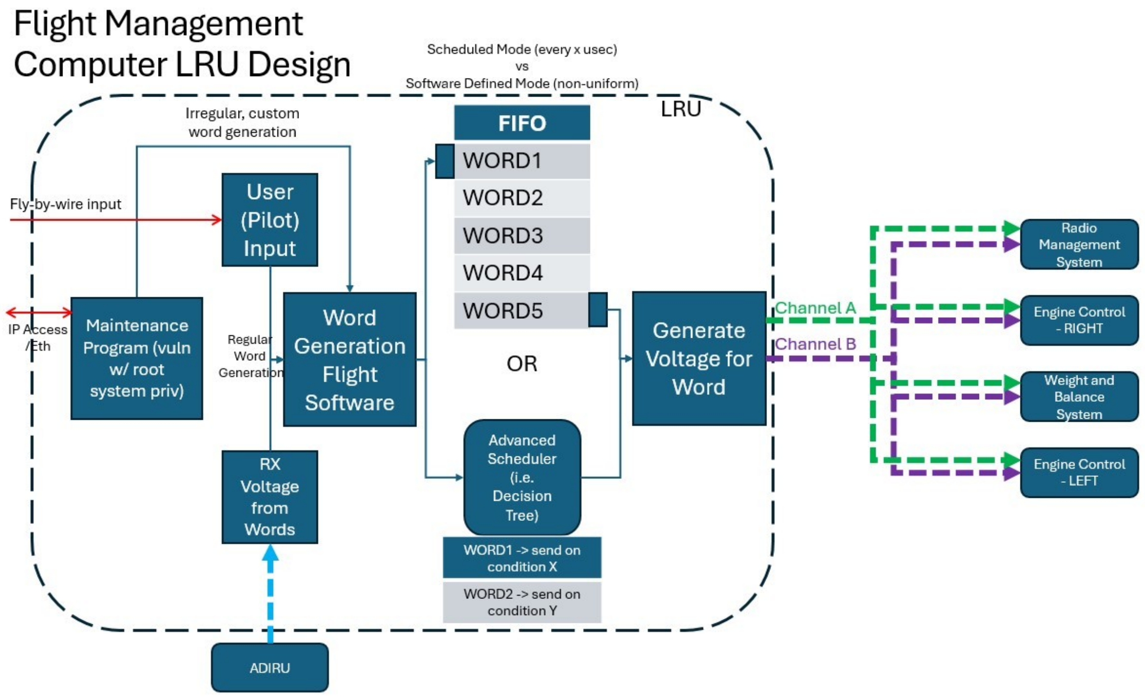
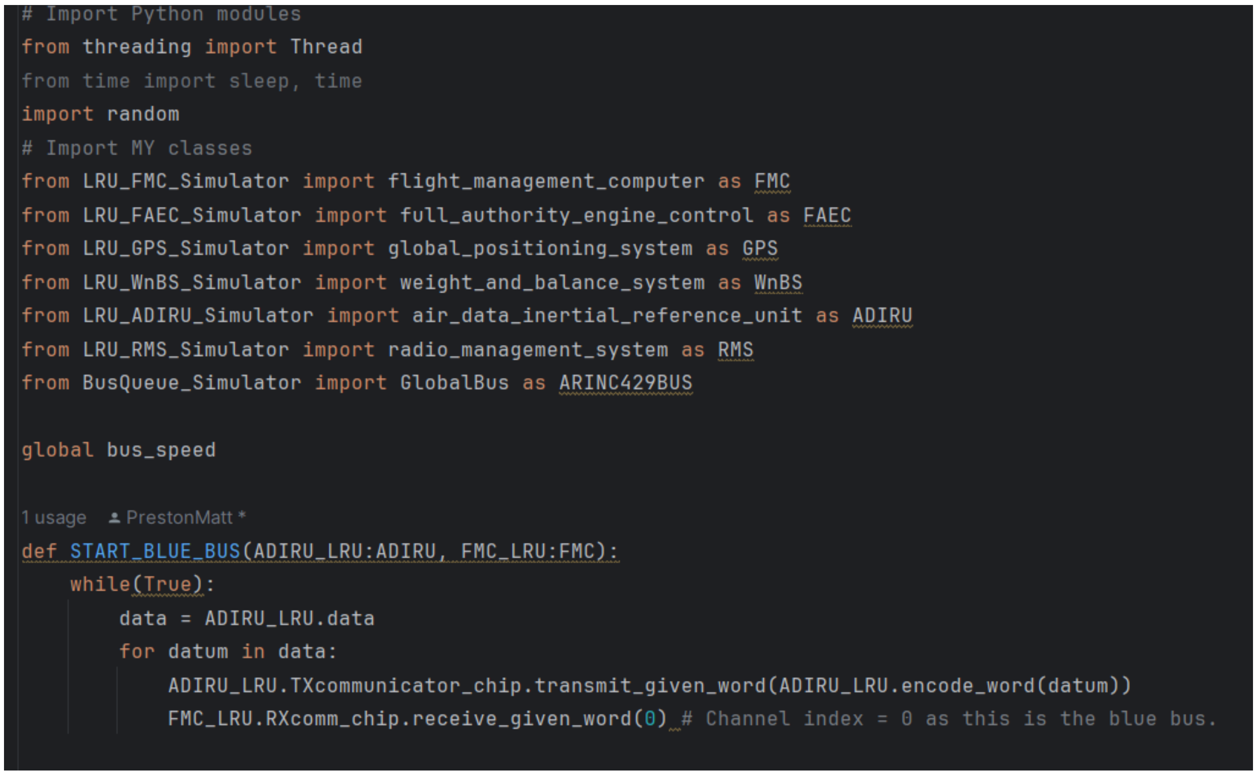
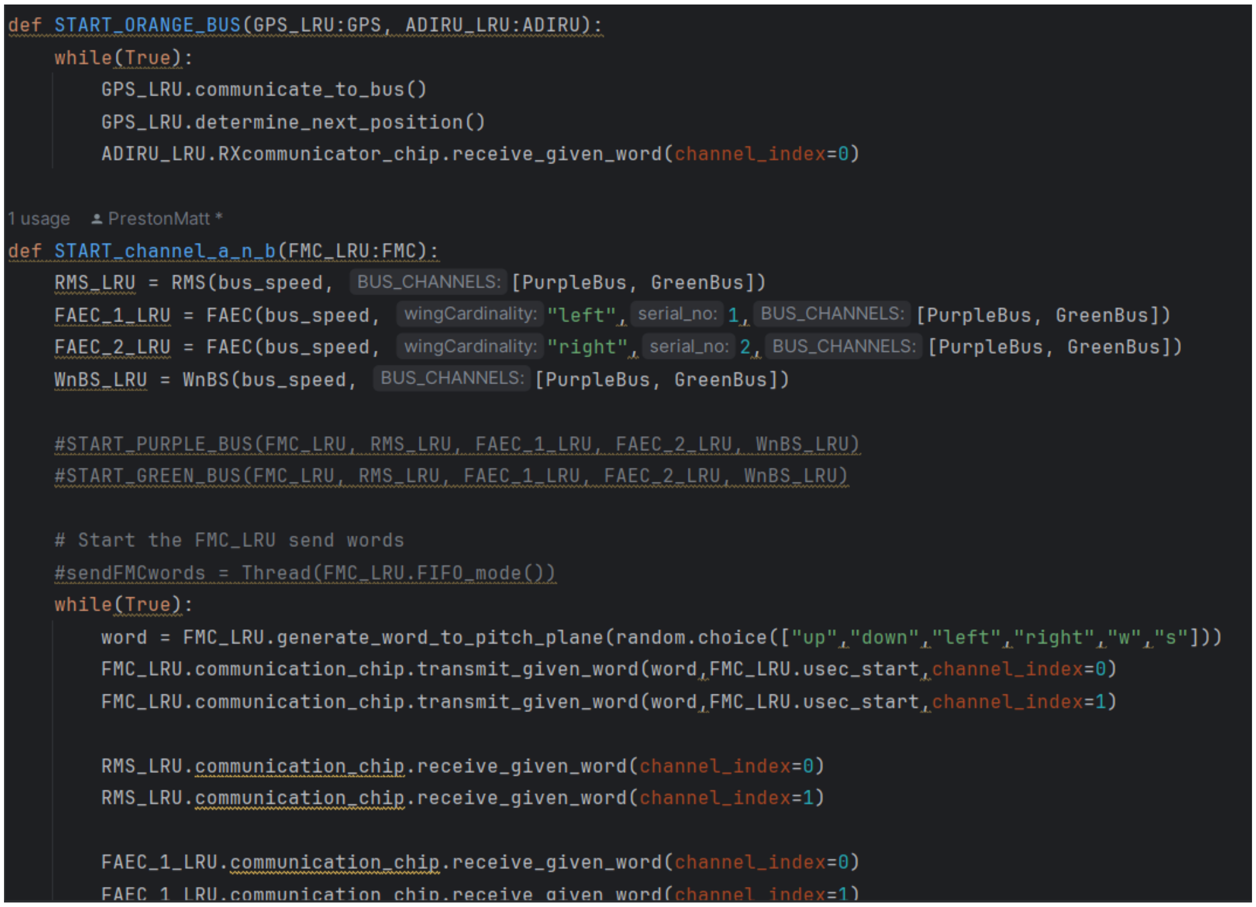

Flight Management Computer LRU Design
Scheduled Mode (every x usec)
vs
Software Defined Mode (non-uniform)
LRU
Irregular, custom word generation
FIFO
WORD1
WORD2
WORD3
WORD4
WORD5
OR
Fly-by-wire input
User (Pilot) Input
IP Access /Eth
Maintenance Program (vuln w/ root system priv)
Regular Word Generation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

Figure 14: FMC LRU Design.



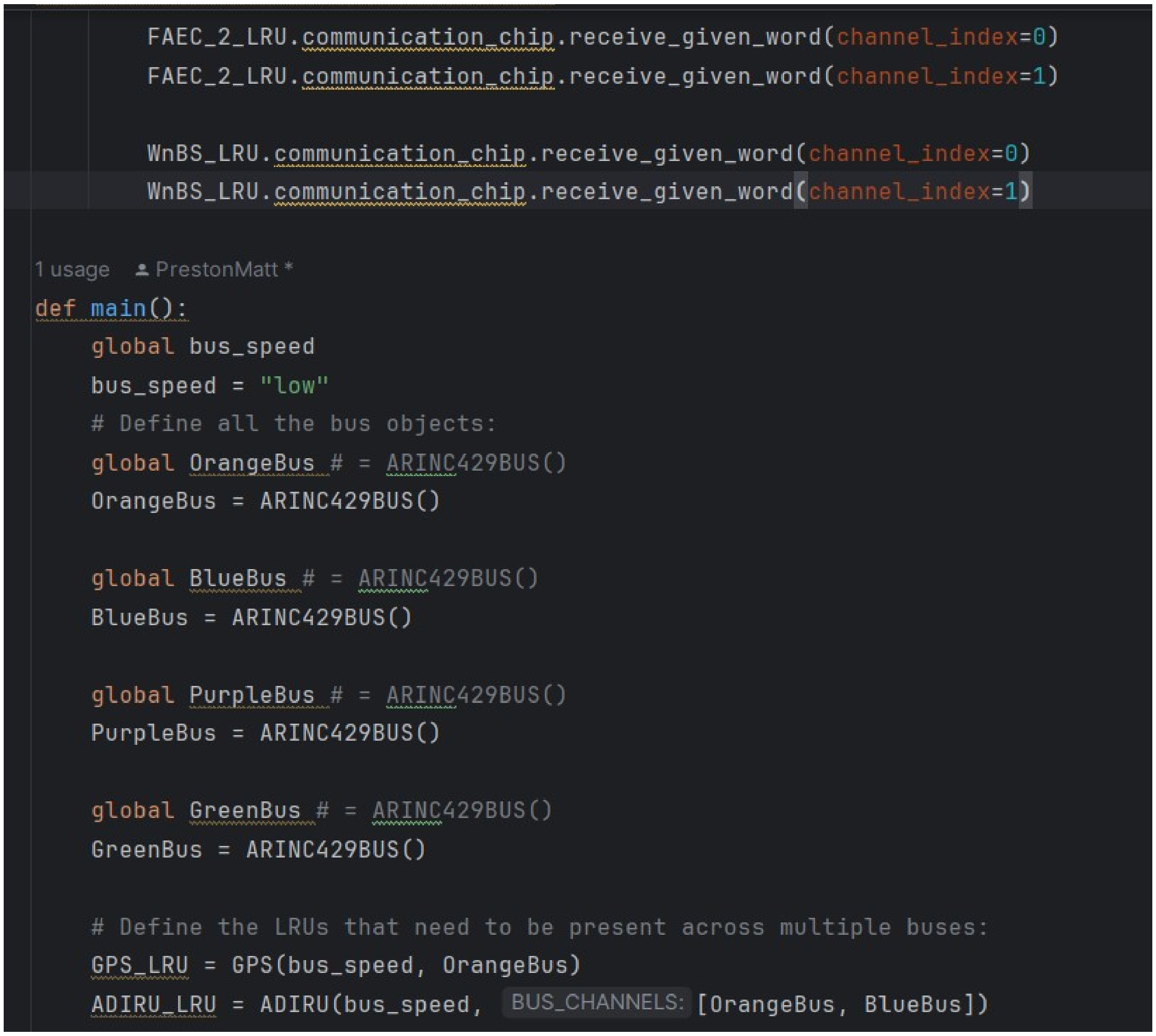
```
# Import Python modules
from threading import Thread
from time import sleep, time
import random
# Import MY classes
from LRU_FMC_Simulator import flight_management_computer as FMC
from LRU_FAEC_Simulator import full_authority_engine_control as FAEC
from LRU_GPS_Simulator import global_positioning_system 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.



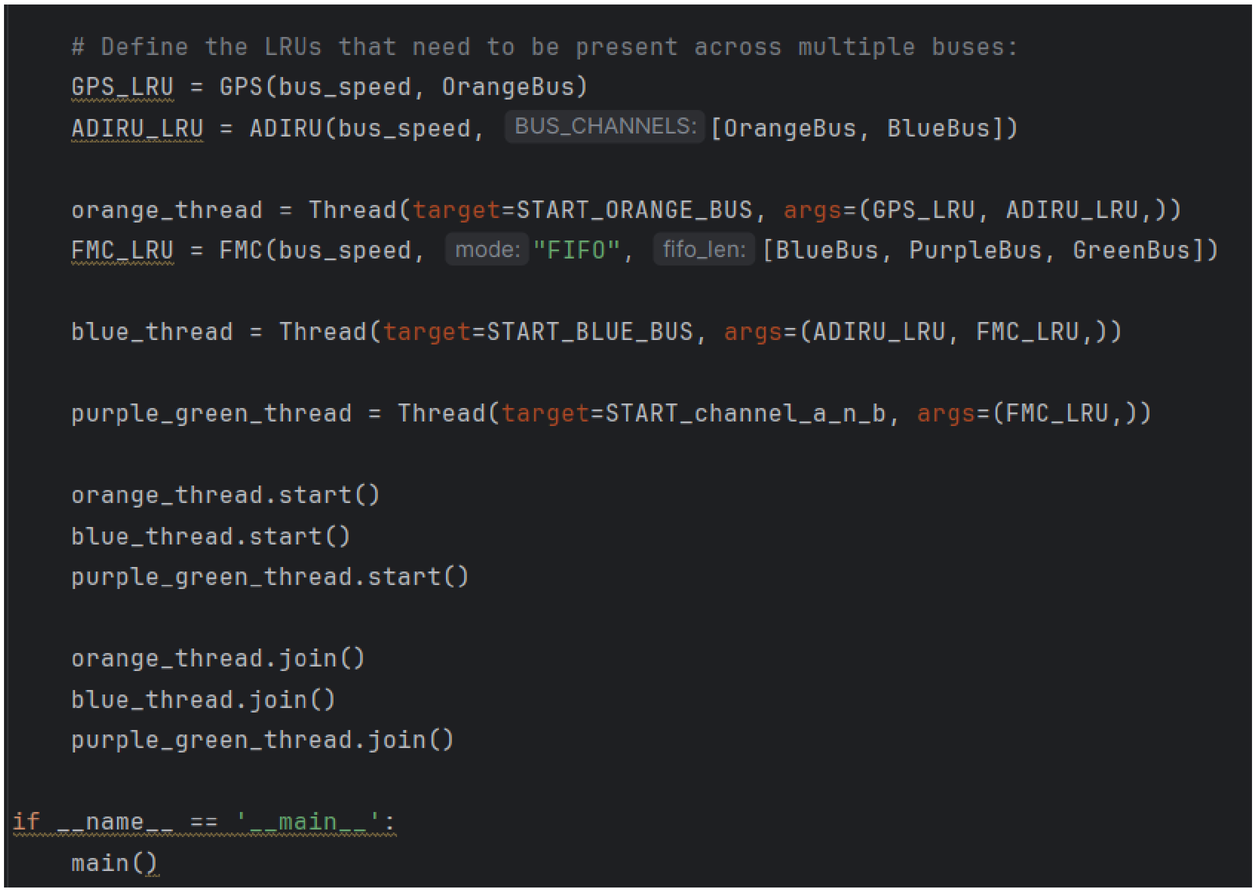
```
def START_ORANGE_BUS(GPS_LRU:GPS, ADIRU_LRU:ADIRU):
    while(True):
        GPS_LRU.communicate_to_bus()
        GPS_LRU.determine_next_position()
        ADIRU_LRU.RXcommunicator_chip.receive_given_word(channel_index=0)

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.



```python
    # 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.