Simulink-Flight Software Bridge for Model Based Testing

Sherif Matta
NASA Johnson Space Center
2101 E NASA Pkwy, Houston, TX 77058
sherif.m.matta@nasa.gov

Marc Carbone
NASA Glenn Research Center
21000 Brookpark Rd, Cleveland, OH 44135
marc.a.carbone@nasa.gov

***Abstract*—Model-Based Testing (MBT) has become a powerful tool used for validating complex flight software systems, particularly in safety-critical space applications. This paper presents a novel MBT framework that leverages Simulink® with operational flight software used in space missions, including applications within NASA's Gateway program. The core of our approach is a Simulink–Flight Software Bridge. The bridge is designed to allow real-time sending and receiving of telemetry and commands with the flight software using the Space Packet Protocol (SPP).**

To facilitate rapid test development, the MBT bridge blockset is auto generated from standard interface definitions XTCE files. We developed a tool to parse the XTCE files and extract telemetry definitions and command arguments, and automatically configure the Simulink bridge accordingly. The resulting framework supports open and closed-loop simulations, Model-in-the-loop (MIL), Software-in-the-loop (SIL), Hardware-in-the-loop (HIL), and functional validation against interface specifications. Our implementation improves testing coverage and early bug detection in mission-critical spaceflight software. The solution can be leveraged to be used with other applications with similar interface architectures.

Keywords: Model-based testing, Flight software, Simulink, Space Packet Protocol, XTCE, NASA Gateway, Spacecraft testing

TABLE OF CONTENTS

1. INTRODUCTION 1
2. SIMULINK BRIDE BLOCKSET OVERVIEW 2
3. AUTOMATIC GENERATION OF THE BRIDGE BLOCKSET FROM A DATA DICTIONARY XML OR XTCE FILE 4
4. ELECTRICAL POWER SYSTEM USE CASE 4
5. SUMMARY 5
6. ABBREVIATION 5
7. ACKNOWLEDGEMENTS 6
8. REFERENCES 6
9. BIOGRAPHY 6

Figure 1 Simulink-FSW Testing Methods Block Diagram 2
Figure 2 Simulink-Flight Software Sender Block 2
Figure 3 Simulink-Flight Software Receiver Block 3
Figure 4 Simulink-Flight Software Parser Block 3
Figure 5 Port Test Block Layout 3
Figure 6 Simulink-Flight Software Event Viewer 4
Figure 7 EPS Testing Framework 5
Figure 8 MBT EPS Monte Carlo test results for different battery initial condition 5

1. INTRODUCTION

Developing and verifying flight software for space missions is challenging due to the complex requirements and distributed development of several interacting subsystems. In addition, the limited access to integrated hardware environments adds another challenge to the Verification and Validation (VnV) activity. Traditional verification methods often depend on hardware-in-the-loop (HIL) setups, which are costly, time-consuming, and difficult to scale for comprehensive testing [3] .

Model-Based Design (MBD) with tools like Simulink offer an alternative approach [5] [8] . Model Based Testing (MBT) enables early system validation and automated testing. However, directly integrating flight software with model-based testing frameworks remains a challenge especially when working with legacy C-code or systems using the Consultative Committee Space Data Systems (CCSDS) Space Packet Protocol.

To address this gap, we present a model-based testing approach that allows direct interaction between Simulink and flight software binaries through a custom-developed Flight Software–Simulink Bridge. This bridge supports real-time read/write access to telemetry and command data, complying with the Space Packet Protocol. The Bridge Interface can be automatically generated from standardized interface definition formats such as XML data dictionaries or XTCE files. The Auto generation and auto configuration of the Bridge reduces setup time and human error; it ensures integration free issues between suppliers' flight software. Auto-configuring the bridge expedites test case development.

This method is currently applied in testing the Electrical Power System (EPS) of the NASA Gateway project[7] [6] .

The Gateway project is a key component of NASA's Artemis program. Gateway is a planned space station in lunar orbit that will serve as a vital staging point for missions to the Moon and beyond. Ensuring the reliability and correctness of its subsystems, like the EPS, is critical. Our approach supports rapid and robust validation of flight software in a simulated environment, increasing test coverage while significantly reducing the need for hardware dependency**.**

To ensure error-free integration and high reliability of the flight software, a structured testing strategy is followed using Model-in-the-Loop (MIL), Software-in-the-Loop (SIL), and Hardware-in-the-Loop (HIL) methodologies[4][5] [8] .

In the MIL stage, both the control logic and subsystem models are developed and tested entirely within the Simulink environment. This allows early functional validation and rapid iteration of control algorithms.

The SIL stage introduces the actual compiled flight software running in a Linux-based emulated environment, while keeping the plant models in Simulink. Communication is handled via the Simulink–Flight Software Bridge, enabling verification of real software behavior and timing without deploying to hardware.

Finally, in the HIL stage, the same Simulink-based subsystem models interact with the flight software running on actual flight hardware or avionics targets. This stage validates hardware/software interfaces, real-time execution, and system-level interactions under operational conditions.

By progressing through MIL → SIL → HIL, The VnV team can identify and resolve integration issues early, minimize risks, and ensure a robust, validated control system before flight deployment.

Figure 1 Simulink-FSW Testing Methods Block Diagram illustrate the Model in the loop (MIL), Software in the loop (SIL) and the Hardware in the loop (HIL) methods and how the Simulink-Flight Software is necessary to perform the SIL and HIL.

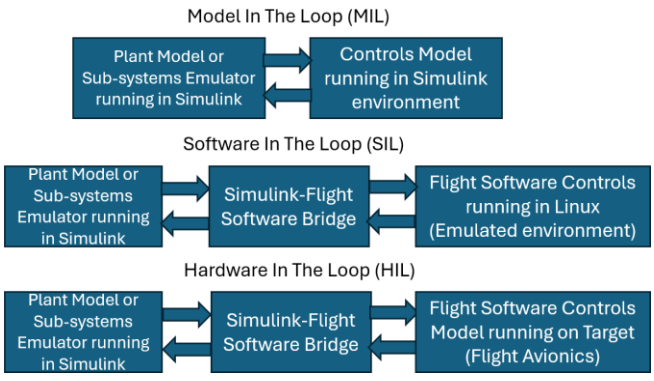

Figure 1 Simulink-FSW Testing Methods Block Diagram

2. SIMULINK BRIDE BLOCKSET OVERVIEW

To enable seamless communication between Simulink and flight software, we developed a custom Simulink Bridge Blockset that runs within the Simulink environment. This design ensures that testing can occur entirely within the simulation framework and reduces dependencies on external hardware setups while representing the flight software's data flows.

The blockset supports bidirectional telemetry and command exchange using the CCSDS Space Packet Protocol. This enables integration with flight software running on Linux within an emulated environment, or directly on target hardware. It enables the VnV engineer to build, simulate, and execute the test scenarios using model-based tools.

The blockset includes self-test mode, where each block can test the conductivity to the rest of subsystems before the test starts. This functionality ensures point to point connection. In addition, it allows testing the network layer, including SBNG and network configuration.

The blockset consists of 5 primary components**:**

A. Telemetry and Command Sender Block

This block handles the transmission of telemetry and command packets from the Simulink model to the flight software. Its functionalities include:

MCC Emulation: Emulates a Mission Control Center (MCC) by sending command packets to the flight software under test.

Subsystem Emulation: Simulates telemetry inputs from external subsystems such as HALO (Habitation and Logistics outpost) or PPE (Power and Propulsion Element), ensuring the test environment reflects the system's operational setup.

Automatically configurable via standardized metadata definitions (e.g., XTCE interface files), the Sender Block seamlessly integrates into any Simulink test harness.

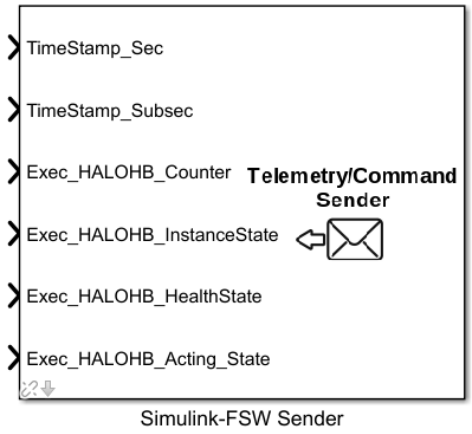

Figure 2 Simulink-Flight Software Sender Block

Figure 2 Simulink-Flight Software Sender Block shows the main block inputs including the time stamps inputs. When the block is set up to send additional parameters or command arguments, these extra parameters or command arguments will appear as additional inputs to the block.

B. Command and Telemetry Receiver Block

The Receiver Block captures incoming data streams—either telemetry or commands from the flight software or from external sources like the MCC or other command interfaces. With configurable settings for specific packet types or response codes, this block allows real-time observation of the flight software's behavior, all within the Simulink environment.

Together, these blocks enable developers to validate functional behavior, interface compliance, and system-level interactions entirely within Simulink. This not only speeds up the development and verification process but also significantly enhances test coverage by integrating directly into the model-based simulation framework.

Figure 3 Simulink-Flight Software Receiver Block shows the main block outputs. When the block is set up to receive additional telemetry parameters or command arguments, these extra parameters and arguments appear as additional outputs from the block. The block has standard outputs to allow cascading the parser blocks for wrapped telemetry or commands. The Trigger_out output will trigger the downstream blocks on message receive events while the Raw_msg output will output the payload to be parsed further by downstream Parsers. Finally, the Msg_Header, Count Received and Trigger Received outputs allow more control over the simulation model execution.

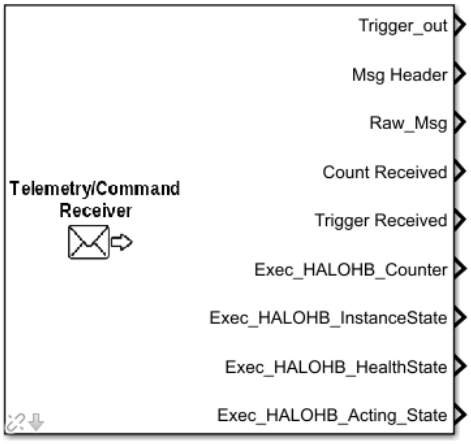

Figure 3 Simulink-Flight Software Receiver Block

C. Telemetry and Command Parser block

The parser block is developed to support parsing of the wrapped packet within another packet. The receiver block shall be used first, then the contents from the receiver block shall be parsed by the parser block. The parser block is also utilized when multiple telemetry or command messages are received on the same network port, which reduces processing overhead by eliminating the need to repeatedly open and close ports for each individual telemetry and command reception. In this latter case, a receiver block is used to capture the incoming data, and then the raw data is passed to a parser block for further processing.

Figure 4 Simulink-Flight Software Parser Block shows the main block inputs including the time stamps inputs. When the block is set up to parse additional parameters or command arguments, these extra parameters or command arguments will appear as additional outputs to the parser block. The *Trigger* and *Raw_Msg* inputs shall be connected from the receiver block.

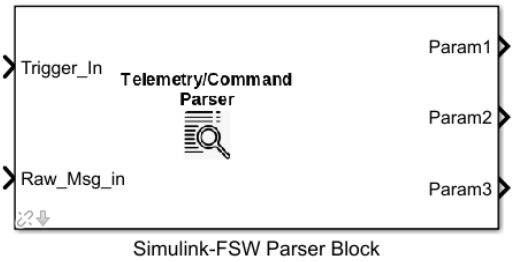

Figure 4 Simulink-Flight Software Parser Block

D. Port Test

The main purpose of the port test block is to test the point-to-point connection. The port test consists of two parts, the sender and the receiver part. The Port test sender allows sending known data patterns. The port test receiver checks if the received data is correct and confirms if the connection is successful. This setup allows testing of the SBNG and all network layers. This test is embedded in the interface generation, and it is transparent to the test case.

Figure 5 Port Test Block Layout shows the different options for the port test. For instance, sending a constant value will ensure static connection while sending a counter or sine wave allows for measuring latency between sender and receiver.

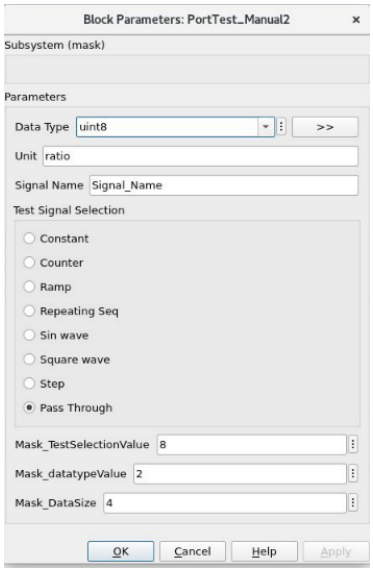

Figure 5 Port Test Block Layout

E. Events and Sequence viewer graph

The Event Viewer graph provides a timeline view for all received events from the Flight Software. This view is critical to analyze and understand the events sequence and decide if the system responded as expected or not.

Figure 6 Simulink-Flight Software Event Viewer shows an example of the event viewer.

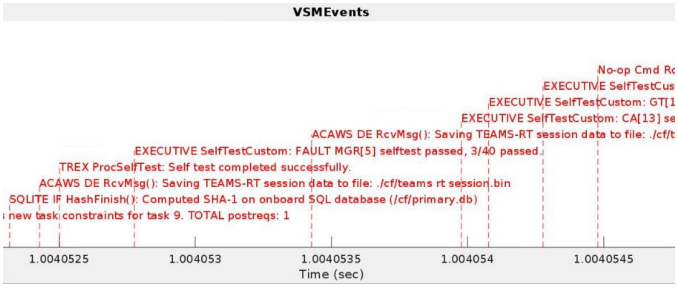

Figure 6 Simulink-Flight Software Event Viewer

3. AUTOMATIC GENERATION OF THE BRIDGE BLOCKSET FROM A DATA DICTIONARY XML OR XTCE FILE

The process of manually adding the Simulink to Flight software bridge can be very time consuming. A tool was developed to extract the telemetry and command structure from the XTCE or Data dictionary and automatically generate the required interfaces blockset and configure each block accordingly. Usually the data dictionary file (XTCE) doesn't contain the controller and network configuration which is required to fully configure the sender and receiver blocks. A producer-consumer relationship shall be defined along with all controllers and their prospective network configuration for the tool to generate the blocksets with the proper configurations. Each generated interface block will contain the port testing block which allow end-to-end connectivity testing between all subsystems.

Additional information can be provided through the Interface Control Document and the Flight Software Headers to enhance the generated blockset signal names and description.

The developed tool will auto-generate the interfaces for both the sender and the receiver simultaneously. This allows one to run both sender model and receiver model within the Simulink environment while message exchange is done via the network layer. The initial test ensures an error free network layer. Once the connectivity layer is verified, then we can switch to the functional test configuration where the system under test will run in the software environment while the testing model will run in the Simulink Environment. The MBT model will emulate the system interfaces to perform closed loop or open loop control testing.

In addition, the autogenerated models can be auto coded to generate executable or binary files. The binary executable files can run faster than the Simulink model. Finally, the executable models can be invoked from other NASA testing tools chain seamlessly.

4. ELECTRICAL POWER SYSTEM USE CASE

To demonstrate the usefulness of this approach, the power management control software was integrated with a Simulink-based model of the Electrical Power System (EPS) of NASA's Gateway vehicle. This flight application is part of the Vehicle System Manager software that provides controls and enables autonomous operation of the Gateway subsystems [2]. The power manager is responsible for the automatic control of the systems distributed batteries during insolation, eclipse, and other unplanned events such as disturbances and failures. Without access to physical hardware systems of the Gateway's direct current (DC) power distribution system, implementation and testing of the flight software presents a unique challenge.

To overcome this, a Simulink-based model of the EPS is developed using a DC space power library developed by PC Krause and Associates [1]. The model uses a Thevenin equivalent model of the DC power components including switchgear, power electronics, solar arrays and batteries, and controllable loads. The Simulink model runs at a fixed 200 μs timestep. The library blockset is then tailored to match the size, topology, and behavior of the Gateway EPS. More information about the model can be found in Section IV, Application to Spacecraft DC Microgrid in our previous work [1].

The model can generate many use cases to test and exercise the power management control software. To integrate and test NASA's flight software with Simulink, the bridge blockset proposed in this paper is used as described in the previous sections. Several tests are performed to evaluate the performance of the flight software. The first test of the integrated control platform consisted of the proportional discharge mode control. This method uses a weighted integral control scheme to provide load sharing on the vehicle's distributed batteries, while maintaining system constraints during both insolation and eclipse phases. To execute the test, pre-scripted load solar generation schedules are designed to emulate the orbit phase of the vehicles mission. The simulation then executes the EPS dynamics in a faster than real-time execution. During each controller update period, telemetry is collected and sent through the FSW bridge blockset, where the FSW controller executes the control scheme to determine the EPS power electronic set points to be sent back into the simulation through the same bridge blockset. There the target points are set in the EPS model until the next controller update occurs. The EPS model supports varying loads, inserting faults, and injecting disturbances to ensure maximum testing coverage.

Figure 7 EPS Testing Framework shows the implementation of the model-based testing using the Simulink-Flight software bridge coupled with the Gateway EPS running on the flight hardware.

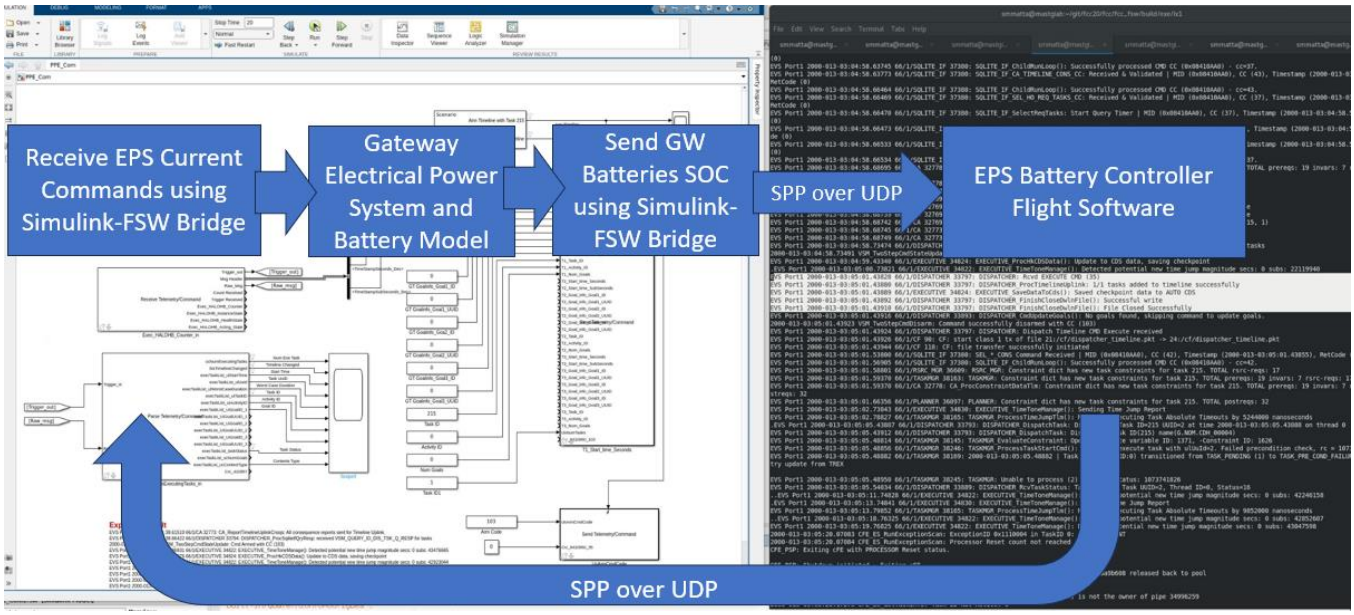

Figure 7 EPS Testing Framework

Figure 8 MBT EPS Monte Carlo test resultPresents an example of the MBT capability of displaying the results and compares MIL with SIL and HIL. It also allows the running of Monte Carlo simulations. The presented simulation shows different battery initial SOC impacts on the controller performance.

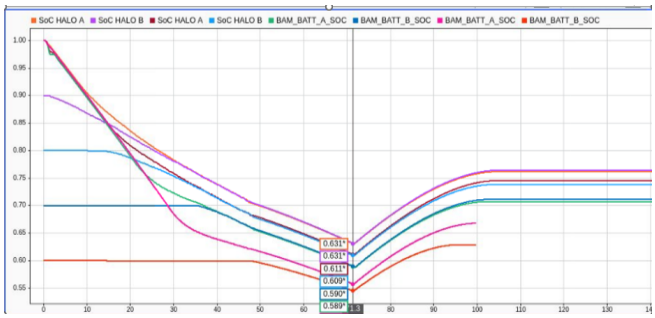

Figure 8 MBT EPS Monte Carlo test results for different battery initial conditions

5. SUMMARY

This paper presents a model-based testing framework that integrates flight software with Simulink using a custom-developed Simulink Bridge Blockset. The blockset runs within the Simulink environment and supports real-time and bidirectional communication using the CCSDS Space Packet Protocol (SPP). It consists of configurable sender, receiver and Parser blocks capable of emulating both Mission Control Center commands (MCC) and subsystem telemetry. The Blockset and the methodology allow for complete software validation without depending on hardware-in-the-loop setups.

The Simulink Bridge is automatically configurable through a standard interface definition such as XTCE. The framework supports unit, integration, and system-level testing of flight software components.

This approach has been successfully applied to test the battery controller of the Electrical Power System (EPS) of NASA's Gateway project, demonstrating its effectiveness in real mission-critical applications and validating its potential as a scalable testing solution for future spaceflight software development.

6. ABBREVIATIONS

FSW	Flight Software
SBNG	Software Bus Network Gateway
VnV	Verification and Validation
MBD	Model Based Design
MBT	Model Based Testing
SPP	Space Packet Protocol
XTCE	XML Telemetric and Command Exchange
MCC	Mission Control Center
CCSDS	Consultative Committee for Space Data Systems
EPS	Electrical Power System
GW	NASA Gateway project
VSM	Vehicle System Manager
ICD	Interface Control Document
PPE	Power and Propulsion Element Module
HALO	Habitation and Logistics outpost
MIL	Model in the loop testing methodology
SIL	Software in the loop testing methodology
HIL	Hardware in the loop testing methodology
SOC	State of Charge
UDP	User Datagram Protocol

7. ACKNOWLEDGEMENTS

This work was supported by the NASA Gateway project under the ER4 division. The authors gratefully acknowledge the leadership and technical guidance provided by Jonathan Rogers, Dr. Julia M. Badger, Dr. Michael Whitzer and Pavan Rajagopal.

8. REFERENCES

[1] Carbone, M. A., & Loparo, K. A. (2023). Fault detection and diagnosis in spacecraft electrical power systems. *Journal of Aerospace Information Systems*, *20*(6), 308-318.

[2] Anderson, M., & Badger, J. (2024, March). Gateway autonomy for enabling deep space exploration. In *2024 IEEE Aerospace Conference* (pp. 1-7). IEEE.

[3] Cardoso, R. C., Kourtis, G., Dennis, L. A., Dixon, C., Farrell, M., Fisher, M., & Webster, M. (2021). A review of verification and validation for space autonomous systems. *Current Robotics Reports*, *2*(3), 273-283.

[4] P. Hecker, R. Kennel, and M. Dirscherl, "Testing of Embedded Systems with Software-in-the-Loop Simulation," in Proceedings of the IEEE International Conference on Industrial Technology, 2006, pp. 2877–2882.

[5] Sherif Matta, Nabil Chalhoub, "Following A V-Model Software Development Cycle Using Model Based Design to Streamline Development of Complex Automotive Systems", Embedded Software Integrity for Automotive May23-25, 2016.

[6] Badger, J.; Strawser, P.; and Claunch, C. 2019. A Distributed Hierarchical Framework for Autonomous Spacecraft Control. In Proceedings of the IEEE Aerospace Conference.

[7] "Gateway Overview," IEEE Gateway Overview, NASA, 2022

[8] MathWorks, "What are MIL, SIL, PIL, and HIL, and how do they integrate with the Model-Based Design approach?" MathWorks Documentation.

9. BIOGRAPHY

Sherif Matta *is a senior engineer and technology leader with over 25 years of experience in robotics, automotive systems, and space technology. He holds a Bachelor of Science in Electrical Engineering from Ain Shams University and a Master of Science in Embedded Systems from Wayne State University.*

Sherif is the founder of a U.S.-based technology company specializing in autonomous systems and robotics. He has deep expertise in model-based development, embedded software integration, and system verification, with a strong focus on tools like Simulink and real-time testing frameworks.

Currently, Sherif leads the Tools and Methods development for the Vehicle System Manager (VSM) of NASA's Gateway project, where he is responsible for building and deploying testing infrastructures that ensure the reliability of critical subsystems.

Marc A. Carbone *(S'19--M'20--SM'24) received a B.S. degree in engineering physics from John Carroll University, Cleveland, OH, in 2014 and M.S. and Ph.D. degrees in systems and control engineering from Case Western Reserve University, Cleveland, OH in 2016 and 2021 respectively. Currently, he is a controls engineer in the power management and distribution branch at NASA Glenn Research Center in Cleveland, OH. His research focus is on model-based fault detection and diagnosis in electrical power systems, where he serves as the autonomous power control team lead and portfolio lead of the Earth Independent Operations Mission Management portfolio in the Mars Campaign Office. His other research interests include modeling and control of electrical power systems, autonomous systems, and state estimation.*