2020

Implementation of a Hierarchical Embedded Cyber-Attack Detection System in Unmanned Aerial Systems

Kevin Yang
Virginia Commonwealth University

Implementation of a Hierarchical Embedded Cyber-Attack Detection System in Unmanned Aerial Systems

by

Kevin Yang

Submitted to the Department of Electrical and Computer Engineering
in partial fulfillment of the requirements for the degree of

Master of Science in Electrical and Computer Engineering

at the

VIRGINIA COMMONWEALTH UNIVERSITY

May 2020

Author .....
Department of Electrical and Computer Engineering
May 18, 2020

Certified by .....
Robert H. Klenke, Ph.D
Professor, Department of Electrical and Computer Engineering
Thesis Supervisor

Implementation of a Hierarchical Embedded Cyber-Attack Detection System in Unmanned Aerial Systems

by

Kevin Yang

Submitted to the Department of Electrical and Computer Engineering
on May 18, 2020, in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical and Computer Engineering

Abstract

With the development of technology revolving around Unmanned Aerial Systems (UAS), UAS are becoming more widely used across a variety of applications. In every scenario, the system in question needs to have very distinct requirements and specifications, many of which revolve around safety and reliability. As such, these systems need to provide mechanisms in order to handle different attacks by adversaries on the outside. Since these systems have the ability to operate in many different flight modes according to their mission, detection and mitigation of the attacks become increasingly difficult over time. Cyber-attacks and their focuses are often evolving, and as a result, existing mitigation solutions slowly become obsolete. Many solutions that exist involve an intrusive solution or embedded software, which provide another attack surface for adversaries to and gain entry. Here, a hierarchical embedded cyber-attack detection system is explored and implemented, providing different methodologies and strategies for handling both cyber-attacks and hardware faults and failures on a hardware and information level.

Thesis Supervisor: Robert H. Klenke, Ph.D
Title: Professor, Department of Electrical and Computer Engineering

Acknowledgments

I would to express my gratitude Dr. Robert H. Klenke, for his time and guidance throughout this project. Over the development and implementation of this paper, Dr. Klenke provided much insight and explanation with the decisions made with designing critical sections of this project. With the help of Dr. Klenke, the development and implementation of this project was extremely rewarding, and much was learned and accomplished.

Secondly, I would like to thank Dr. Matthew L. Leccaditto, Peter Truslow, and Andy Fabian, as without their guidance, many of the barriers that popped up during the research and development process would not have been broken. Over the course of the project, a lot of information regarding the programming and setup of the existing flight control system were advised, and without knowledge of these changes, the development process would be slowed tenfold.

Third, I would like to thank Dr. Carl R Elks, Dr. Sherif Abdelwahed and Dr. Ruixin Niu, for their guidance on the higher level approaches to identifying attacks and testing of the HECAD on UAS systems. Many of the design decisions made in the project revolved around the high level overview at the higher levels of abstraction.

Finally, I would like to express my gratitude to my fellow researchers, friends, and family for providing much support in my decision to pursue this degree. Without their insight, a large part of education and experiences would not be possible. Thank you for providing much support and guidance through this experience.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With the evolving technology surrounding unmanned aerial systems over the years, these systems have become increasing versatile. As such, the applications of these systems have expanded over several fields in industry, including civilian, medical and military. In these respective fields, the concept of safety and reliability are heavily emphasized and extremely important. Over time, the research of UAS has shifted focus to improving safety and preserving mission-critical functionality. In safety-critical systems and applications, a failure of any kind will often result in catastrophic consequences. The same applies for mission-critical systems. In order to prevent these situations, these systems need to provide different mechanisms and solutions to detect hardware faults and cyber-attacks and minimize the catastrophic failures. As the capabilities of UAS and cyber-attacks targeting unmanned systems evolve, the existing solutions safeguarding legacy unmanned systems become obsolete. As a result, more intensive solutions requiring quicker interventions are needed, especially for systems that can potentially cause harm to humans and the environment. Methodologies for monitoring a system in real-time is explored and implemented.

An unmanned aerial system is often best described as a cyber-physical system. This idea involves the integration of physical components of a system (sensors and actuators) and the cyber components of a system (algorithms and computation flows). On any cyber-physical system, the "correct and functional" operation of the system depends greatly on its interface and interconnections with the many on-board modules.

For unmanned aerial systems, sensors and actuators provide data to the main flight controller as well as their respective communication buses. A general architecture for a UAS is shown in figure 1-1. In systems that perform missions autonomously, orientation and geolocation data is critical. Sensors that provide orientation data, such as a gyroscope or an accelerometer, or sometimes any combination of the two to form an inertial measurement unit (IMU) or a motion processing unit (MPU), will need to function correctly to ensure false data is not sent to the controller. Without them, the flight controller will have no reference to determine its direction or orientation for its mission. The UAS will also need to continuously know its location precisely, so the presence of a functional GPS module will be needed. Without these components, the UAS will be rendered useless due to its inability to determine its orientation and location to carry out its mission. Each individual sensor will communicate with the flight control system through different protocols to perform its function.

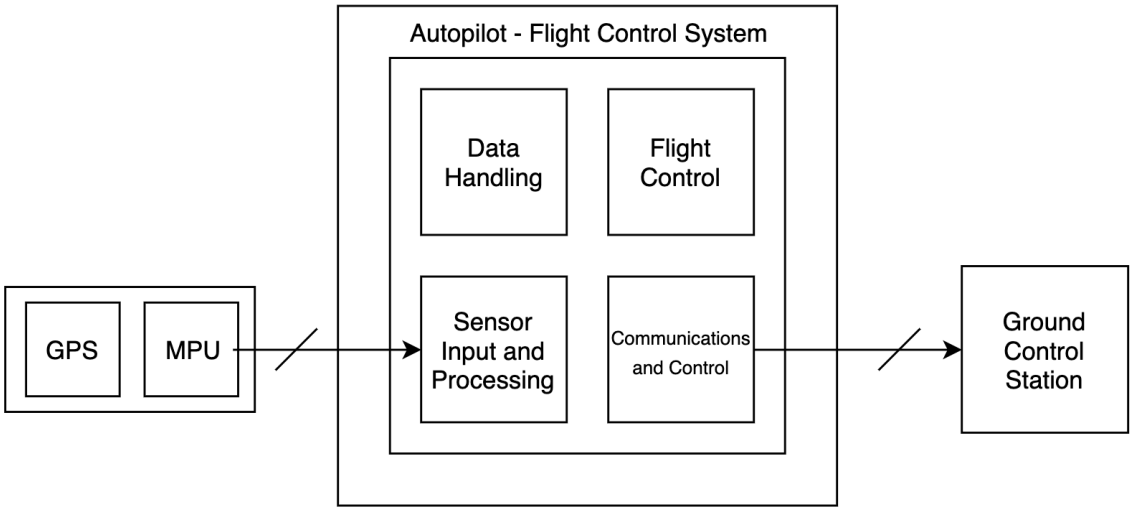

Figure 1-1: General Architecture of a Unmanned Aerial System

A UAS can be represented as a cyber-physical system. The functional requirements can be broken down into required subcomponents, each of which need to function correctly and independently. When modeling and designing an unmanned system, each individual underlying subcomponent that the main flight controller depends

on is identified. This allows for proper design and implementation in the main flight control system to ensure that the most accurate and up-to-date data is always obtained. Every dependent subcomponent must be highlighted. A similar method can be used to identify the vulnerabilities associated with the system.

In general, an unmanned system comprises of a series of subcomponents: communication networks, sensors, and actuators. These subcomponents work together to be able to obtain data from each other and play a critical role in correct functionality. However, the inclusion of each subcomponent introduces an additional vulnerability. With communication networks, adversaries have the ability to perform data injection via packet sniffing and false data injection. For example, in systems that communicate live flight data with the ground control station, adversaries can perform a man-in-the-middle attack, packet sniffing the data going back and forth and manipulating the data before sending it back out.

The sensor itself can also act as a gateway for other cyber-attacks. With a given sensor, there can be multiple points in the supply chain process that allows for injection of malicious software. Although these sensors can be simplistic, the firmware or circuitry can be modified (with or without malicious intent) to output data that is not true to its expected behavior. As a result, supply chain attacks can be harder to detect without further knowledge aside from how the sensor is behaving over the communication bus and the information that is being returned. This would require a higher level of processing to be able to detect valid anomalies.

In Figure 1-2, there are two main general vulnerabilities that can be identified from each main subcomponent. The functionality of an unmanned system depends on its ability to obtain reliable data on its vehicle such as airspeed, geolocation, altitude or orientation, so it is critical that the flight controller is able to (1) obtain *valid* data from each individual subcomponent and (2) obtain *reasonable* data from from the subcomponent. There is a difference between valid data and reasonable data. With valid data, the goal is to verify whether or not the data being returned is actually what is being reported and follows the constraints set by the manufacturer, while reasonable data is data that makes sense when it taken into context. If data

being returned indicates activity on the sensor, but in reality no action is being taken, then the data received is perceived as valid but unreasonable, so long as the data is within range and specification. However, if the data received is is out of range, then this shows invalid data. This is important, especially when dealing with sensors that require further processing upon receiving data. For conversion operations requiring calculations based on values returned, valid data is especially important to account for specific data types and ranges. If invalid data is reported, results from calculations depending on this data will be incorrect.

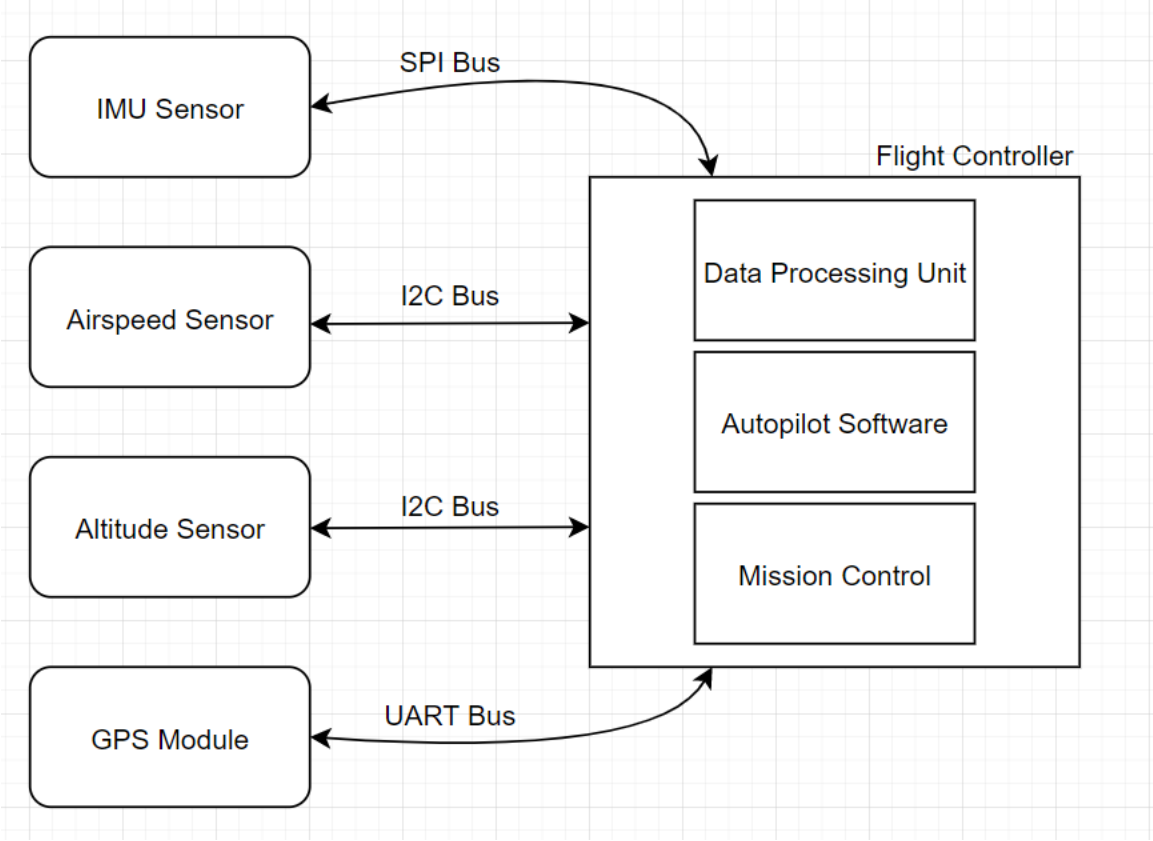

Figure 1-2: Devices and Their Communication Buses

The requirement that the main flight controller needs to be able to obtain correct data from its subcomponents identifies two big potential vulnerabilities: the failure of a communication bus and the failure of the sensor itself. The failure of a bus can be as simple as an issue with the physical connection: a wire could be incorrectly connected or perhaps the master device can request data from an unknown address,

forcing the slave to hang the bus. Timing issues can arise, and cause problems getting correct data across. In any case, the failure of the bus is detrimental, as it can also hang the flight controller itself.

Previously, a hierarchical embedded cyber-attack detection (HECAD) system was outlined that can detect attacks and faults at multiple levels of abstraction [1]. HECAD contains four levels of detection: the hardware resource integrity level, the information integrity level, the functional integrity level, and the execution integrity level.

The first level involves detection at the physical hardware level. This allows for validation that the bus protocol is working as intended, and any malfunctions such as timing issues or glitches and unexpected behavior are detected and alerted. At this level of abstraction, the hardware validation knows nothing about the device on either end of the bus. It knows only about the configuration and operation of the bus (baud rate, transmission speeds, addresses, etc).

At the information level, another detection module takes into account the functions of the devices on the ends of the communication bus. Here, information received to and from either device are validated against what is expected. As a result, this abstraction layer is very tightly coupled with its respective device. Addresses, data ranges, and commands are checked, making sure that they fall within the range of what is expected. If there are custom packet protocols developed to communicate with the ground control system, all data coming across the communication and data link is verified to match the information packet described.

Since the hardware and information level checking levels are designed as data and configuration verification for each sensor or other devices, a higher level detection module is required to detect the less obvious and more complex system-level attacks and faults. At the functional integrity level, more complex processing is required within the HECAD processing system on the validated data to ensure proper system-level functionality. More computationally intensive and in-depth algorithms that are less feasible to implement in hardware or easily implemented in software are implemented here, as well as any other possible implementations such as artificial

intelligence solutions.

The combination of these multiple levels of detection in hardware and software provides a global, minimally intrusive and tightly-coupled fault and cyber-attack detection system that detects anomalies on a hardware, information, and functional level of operation. By isolating HECAD from the main system, the chances that an adversary is able to attack the HECAD system itself is minimized. The goal is for the HECAD system to detect attacks from a bottom-up approach as well as from a higher level point of view. HECAD is designed so that integration into the flight controller is minimal, and would prevent additional attack surfaces. HECAD operates independently to do cross-checking, verification and validation of functionality across subsystems in real-time.

Contributions of this thesis

The work completed in this thesis contributes to implementing a portion of a hierarchical embedded attack detection system for Flight Control Systems (FCS) of small Unmanned Aerial Systems (sUAS). While some techniques exist for detecting cyber-attacks from a high-level perspective (such as neural network detection as explored in [2] and [3], malware detection using embedded decoy processes [4], monitoring of power and heat characteristics [5] or control-flow analysis [6]), additional techniques are needed for detecting different types of cyber-attacks at different levels in an sUAS. While well-coordinated attacks often occur at a higher level by targeting multiple parts of a system, this work describes methods for detecting attacks at the hardware and information level using a bottom-up approach. These methods lay the groundwork for breaking cyber-attacks down into smaller steps at different levels in the system for detection and mitigation. These lower-level hardware and information-level detection techniques were implemented using a field-programmable-gate-array (FPGA) along with an FCS and its respective on-board sensor components that communicate using the universal asynchronous receiver transmitter (UART) and inter-integrated circuit (I2C) communication buses.

Chapter 2

Background and Related Works

2.1 Unmanned Aerial System - General Architecture

An unmanned aerial system (UAS) or an unmanned aerial vehicle (UAV) is an example of a cyber-physical system whose operation highly depends on the correct operation of the on-board components such as sensors and actuators. When UAS were introduced to the industry, a commercial-off-the-shelf (COTS) UAS used for everyday recreational purposes generally did not have many self-piloting features that allowed them to fly without user intervention. Over the years, as size, weight and performance capabilities evolved, more and more features involving automated flight modes became available. Hovering, flips, and object-following are a few examples. Systems such as the DJI Mavic or the Skydio 2 are able to perform tasks such as following a subject in video through mountains and forests or taking pictures upon a voice command. In order to be able to successfully perform these tasks without entering a failed state of the aircraft, the on-board components need to be dependable in the presence of failures and cyber-attacks by providing correct data and feedback on the status of the vehicle at all times. These components generally involve image processing units, graphics processing units, camera sensors, GPS units, inertial movement units, central processing units, and other sensors and actuators, many of which form subsystems. By integrating these physical components that provide data constantly and correctly, unmanned systems now have a highly-reliable, highly-available

system of sensors and actuators. The flight controller, which generally has a main microcontroller or processor and various necessary peripherals, manages all of the data processing and storage as well as communication between subsystems and other parts of the unmanned system.

2.2 Vulnerabilities of UAS

During the planning and development of an unmanned system, the most important aspects of the system are considered: safety and security. The following defines several terms related to the security of a UAS.

1. Vulnerability - The presence of a component that can be used by attackers to gain entry into the system
2. Safety - The state of the UAS being uncompromised and safeguarded against external and internal threats
3. Security - The presence of several safeguards designed to protect the operational state of the UAS
4. Mitigation technique - Actions taken in response to finding a threat or compromised safety / security
5. Spoofing - The process of tricking a sensor or device by attacking the sensor's or device's point of reference
6. Attack - The process of introducing errors and unfamiliar information and commands into the system through exploitation of different vulnerabilities
7. Injection - The process of introducing different types of malware / unfamiliar information, commands, or operations into the system intentionally under a controlled testing environment
8. Threat - The existence of unfamiliar information, commands, or operations seen from the system's point of view

9. Attack Surface - The different ways that the attacker can gain access to the system [7]

With aerial systems, there are several requirements put into place regarding operation and flight. The biggest concern is the ability for the UAS to not lose control and fall out of the sky. While this may not be the case for many industrial commercial off the shelf (COTS) drones, sometimes the security of the systems is overlooked. Aside from the very basic requirements of an unmanned aerial system (vehicles need to have security mechanisms in place to prevent UAS from falling out of the sky and other requirements to prevent injury), a more in-depth analysis is often not considered. Traditionally, the development of a UAS involves building the drone to operation, then tackling the problems involved with security from a top-down overview [8]. This approach is often not enough, considering the constant evolving cyber attacks that involve targeting different parts of an unmanned system. Depending on the goal of the adversary, attacks can either bring the entire vehicle down, or perform enough manipulation to throw the state of the UAS into chaos (such as manipulation orientation or positional data either though sensor spoofing or false data injection).

The concept of security within many systems and subsystems can be classified into four broad areas: Confidentiality, Integrity, Availability, and Authentication.

1. Confidentiality - The confidentiality is the measure that that determines who has access to what information. Generally, rules are put into place to determine ownership of specific data, along with evaluating the importance of data and the detrimental effects of it being compromised.

2. Integrity - Integrity is a measure used to verify the integrity or authenticity of data from a source. For data that is transmitted between devices, there is generally a communication protocol along with specific data constraints provided by manufacturers. Often times, there are built-in mechanisms to verify integrity of data during transmission to ensure that the data has not been manipulated or changed.

3. Availability - Availability is a measure that is used to ensure that all systems communicating between devices are reliably able to access data at any point in time. Placing measures that ensures reliable communication for data at any point in time, downtime between communication devices are minimized.

4. Authentication - Authentication is used to verify that the devices attempting to communicate with each other are really the devices they claim to be. This is usually done with addressing or identification bytes.

2.2.1 Naturally Occurring Faults vs. Cyber-Attacks

In systems such as a UAS, naturally occurring faults can arise. For example, single-event upsets (SEU) may occur, especially in systems that spend time in high altitudes where cosmic rays can induce naturally occurring faults. When this happens, a bit or multiple bits within a device may be affected, changing the data to be reported to the requesting device. In this case, data that is then reported appears to be manipulated. However, the issue here is that it is difficult to differentiate naturally occurring faults from direct cyber-attacks. For example, assuming that an adversary has successfully injected malware that is intended to scramble the data being reported into a sensor by randomly flipping bits, then this data that is being passed through a checker has no way of verifying that the data is indeed from an attacked sensor as opposed to environmental noise such as an SEU, since the data will appear to be scrambled. Unless there is a method to identify patterns that are associated with cyber-attacks as opposed to natural faults, it is extremely difficult to differentiate between the two. If the user had information regarding how data is being scrambled by a compromised sensor, then more information will be available for this. However, the chances are very low for the checker to know attack information, especially with well-coordinated attacks performed with the intention of hijacking sensors and systems where the attacker may be able to inject malicious information. Well-coordinated attacks will not follow the assumptions that the system will know information about outside attacks, injected malware, or that the attacks target a single sensor. As a result, a

detection scheme attempting to differentiate between natural faults and cyber-attacks will be more successful in determining any kind of discrepancy, rather than attempting the classify the type of discrepancy.

2.2.2 System Security and Attack Surfaces

The security of a UAS is often described by several attributes, including how easily attackable the system is (susceptibility), as well as how the system can be exploited (accessibility). These two attributes describe the different actions on different attack surfaces. Often times, attacks that are well coordinated and have extensive attack surfaces are more successful against systems. In order to combat the different types of attacks, the general operations of the attack surfaces need to be analyzed. Generally, attacks involve some form of extraction, manipulation, and then reinsertion. This is especially true for attacks on wireless communication links or communication buses if access has been gained through either a man-in-the-middle attack or malicious malware injection into the system / sensor.

In Figures 1-1 and 1-2 in the previous chapter, there are multiple attack surfaces on a UAS. The first involves attacking the sensors and actuators themselves. Since unmanned systems rely heavily on components that provide data on orientation and position, the downtime of these components can result in fatal operation of the aerial system. With many different sensors, their correct operation depends on the correct functionality of a reference point. For example, GPS modules receive signals from four or more satellites, completing a calculation that informs the receiver module where they are geographically located. Satellites constantly transmit the time in which a signal is sent, and based on the difference between the time the signal is received by the GPS module and the time the signal is sent by the satellite, the receiver can figure out the distance to each GPS satellite using trilateration. A spoofing attack can work when the attacker can successfully replicate these satellite signals, and transmit them locally. Since UAS are typically flying closer to the ground, it is possible that GPS signals from the attackers can be picked up and override the GPS signals from the satellites. In theory, this is possible. However, because unmanned systems in

the air are often moving and constantly changing, an active spoofing attack will be very difficult, especially at higher altitudes and speeds. In section 4, a mitigation technique for this is issue explored.

In many mobile phones, there are often software applications that allow for changing the device location, primarily through offering an alternative set of coordinates to the operating system rather than relying on the data received from the GPS modules. On an unmanned system, a similar situation may occur. Attackers can attack by sneaking in malicious software or firmware to the on-board to processor that either manipulates the sensor to transmit incorrect data or overrides the sensor's data and inserts its own set of malicious data. Frequently, the firmware gets into the system through an external source. Typically, these involve an attacker cloning an original source of firmware and modifying the firmware to be re-distributed, the source comes from the developers themselves (either with malicious intent or simply lack of testing and verification), or source becomes corrupted through distribution through the many points of contact (manufacturer, distributor, assembly or during programming) [1]. This situation would be more often found in situations where a sensor does not rely on an external reference point, such as an airspeed sensor. These sensors rely on factory-written data in the calibration registers to be able to provide accurate data. Malware within the sensor is able to manipulate the reported data or change the calibration data.

In any generic unmanned system, there will typically be a communication path between the on-board flight controller and the ground control station. Frequently, when there are missions that need to be completed by the UAS, the ground controller will send mission data (coordinates, way-points, geolocation data, etc) to the UAS, and the UAS may send additional data back. Generally, communication between the UAS and the ground controller will have a very specific protocol and data format. Because of this, attackers may sit in between the two points and listen in on the protocol. If the data being sent is not encrypted, it will only make it easier for the attacker to figure out the protocol. Once the protocol has been figured out, the attacker can either replicate or manipulate the data and resend the data back out,

performing a man-in-the-middle network attack on the flight controller.

Additionally, because the ground controller generally consists of a program that is provided and maintained by third parties, there are no guarantees that the software used in the ground control station is not compromised. In the event that an attacker is able to trick the user into downloading a malicious version of the ground controller, the attacker can use the ground controller to send malicious commands and data to the flight controller. If the ground controller is developed by a user, any insecure connection to the internet provides a point of entry for the attacker to inject malicious data into the ground control station. A known instance of a compromised ground control station is described in [9] and [10], where an attacker successfully implanted keylogger malware into a ground station to be able to steal key information. A similar process can be done here, especially if information is stored unsecured.

The use of an encryption method for communication will only provide further security between the points of communication. For an XBee communication transceiver, there exists a built-in Advanced Encryption Standard (AES) encryption that can be enabled. However, this requires that the user still provide an encryption key on either end of the transmission link. If this information is stored on the ground control station and the flight controller, the attacker can access this information if there is an unsecured connection to the internet from the ground control station. AES is a symmetric encryption algorithm, meaning the same key is used to decrypt and encrypt the block of data. As a result, if an attacker gains access to the key, they can successfully encrypt and decrypt the data going both ways. With this information, the attacker now has full control of the communication link, given that they are able to determine the protocol and packet formation standard.

2.2.3 Types of Attacks on Unmanned Aerial Systems

In general, the failure of a UAS can be classified into two broad domains: UAS mission-related failures and UAS safety-related failures. Mission-related failures are related to the objectives or mission revolving around the user or autonomous missions. On the other hand, safety-related failures are related to the failure of constraints put

in place to ensure safety to people, environment and infrastructure. Of these two domains, there is potential for overlap. Mission-related failures can often be caused by a failure of a component that does not provide correct data at any given point in flight time. For a UAS, this means that a component or components have failed, either through a hardware fault or a direct cyber-attack. On unmanned systems, an example of a fault can result from single event upsets caused by cosmic rays. Most commonly seen in systems in space, these events can happen more often to aerial systems than ground systems due to their constant high altitude. When these events happen, information provided by the component can be incorrect and will result in a failed operation of the UAS. For direct cyber attacks, the manipulation of the information coming off of the sensor are often controlled attacks. If the goal is to divert the aircraft away from the area of operation (AO), then an attacker may spoof a combination of sensors by manipulating the reference points to divert it in a certain direction. However, this is very unlikely since manipulation of sensors without external reference points will need to be from the supply chain level. With this approach, an unmanned system with multiple sensors from the same distributor or manufacturer will provide an advantage to the attacker. In either case, the presence of a faulty sensor or a direct cyber-attack will place limits into the performance requirements of the aircraft in order to complete its mission.

On the other hand, failure of safety constraints can result from faulty sensors. With the failure of orientation and positional sensors such as the IMU or a GPS module, the main points of reference for the UAS to maintain its position have been lost. The UAS then has no way of knowing how to stay upright. The same effect can be achieved by attacks on sensors by manipulating the reference points of the sensors. For sensors such as GPS, attacks can be performed via spoofing or jamming. As explained previously, the data reported by the sensor can be ignored and data from a secondary source can be utilized. In the event that an attack is able to insert malicious code prior to the flight, attacks can replace the incoming data with malicious data, performing a false data injection via malware infection or a supply chain attack.

Furthermore, a successful attempt to insert malicious code allows for attacks on the source of the controller, or rather the processing platform itself. The presence of malicious code can manipulate the behavior of the software and hardware modules on-board. These involve forcing the flight controller to behave in unexpected ways (running malicious or alternative code or hanging communication buses). Once the firmware has successfully entered the system, there are several approaches that the firmware can take. The firmware can run malicious code, forcing resources to be held up or causing the battery to drain (resource locking and sleep deprivation), insert malicious data into the local database or memory (database and false data injection), or force malicious code to manipulate data processing, actuator control algorithms, and memory management.

Outside of the flight controller, there is the possibility of attack between two communication points. As most unmanned systems communicate over some form of a network, there are many different types of attacks through a wireless entry point. Between the flight controller and the ground control station, attackers are able to perform a man-in-the-middle attack, where the attacker listens in on the communication protocol. If the series of data is encrypted as described above, attackers may perform a brute-force attack on encryption algorithms to determine the key used to encrypt the data. With any knowledge of a communication standard, attackers know to look for a few sync bytes and a message length. The data can then be captured and the communication standard can be broken down, allowing attackers to perform replay and relay attacks. Furthermore, once the attackers know the protocol, they are able to perform command injection or false data injection to manipulate the tasks of the UAS. Alternatively, attacks can be carried out to perform denial-of-service or communication jamming attacks by either dropping parts, if not all, of the data packets going back and forth. Figure 2-1 summarizes a wide variety of attacks on cyber-physical systems.

CPS Exploits	
Method	**Description**
Network Attacks	
Black Hole/Gray Hole	Compromising the network, dropping all packets (black hole) or selectively dropping packets (gray hole).
Command Injection	Accessing a target control unit or network to execute a command with malicious intent.
Communication Jamming	Intentional physical interference with the reception of a required signal; the adversary needs to be in vicinity of nodes to use a strong enough signal to jam the wireless channel.
Denial of Service	Cyber equivalent of jamming; bombarding a network with large volumes of meaningless traffic.
False Data Injection	Communication-Based: Compromising the communication channel, blocking data measurements and inserting false data [5].
Fuzzing	Gaining network access and bombarding the target with messages to observe which one has a physical effect.
Man-in-the-Middle	Connecting independently to two computers that are part of the system with the purpose of eavesdropping and injecting manipulated messages [6], [7].
Network Isolation	Attacks aimed at isolating a particular physical geographical area from a network [8].
Packet Sniffing	Gaining network access to eavesdrop on messages.
Password Cracking	Guessing or otherwise determining a password in documentation or brute force attack.
Relay Attack	Capturing a communications signal and relaying it through a longer-range communication.
Replay Attack	Observing and recording a communication sequence to replay it later in order to spoof the system [9]–[13].
Rogue Node	Introducing a rogue communications node disguised as a legitimate node.
Firmware Attacks	
Code Injection	Introducing additional instructions with malicious intent (i.e. Sensor Parsing, Control Algorithm Adjustment, Memory Leaks, and Structured Query Language injection).
False Data Injection: Database-Based	Exploiting database vulnerabilities, typically by adding erroneous data [5], [6].
Firmware Modification	Modifying a subsystems firmware (i.e. router) to get to the ultimate target [16], [17]. Requires acquiring samples of an official firmware update, then analyzing, disassembling, and attempting to infer the method used by the device to validate updates.
Sleep Deprivation	Causing the system to rapidly exhaust its battery by forcing it to never sleep.
Sensor Attacks	
Supply Chain Attack	Gaining access to supplier computers and modifying the firmware, e.g., preinstalling back doors, malicious code, etc.
False Data Injection: Sensor-Based	Compromising a computer-controlled sensor by reporting false data collected by the sensor instead of the actual data.
GPS Jamming	Transmitting high power signals to impede reception of GPS signal (i.e., impacting GPS availability).
GPS Spoofing/ Meaconing	Synthesizing and transmitting a false GPS signal to deceive a target GPS receiver's location; Meaconing refers to capturing legitimate GPS signal and rebroadcasting with a delay, affecting the timing estimation and ultimately the GPS receiver's location.
Malware Infection	Inserting software into the system with deliberate harmful intent.
GCS Attacks	
Malware Infection	Inserting software into the system with deliberate harmful intent [18], [19].
Viruses	Parasitic programs that infect other programs, activated when the program is executed by the user.
Worms	Self-replicating programs that do not require human interaction.
Back Doors	Program that allows unauthorized access to the system.
Bots	Program that launches flooding or DOS attacks.
Root kits	Sets of software tools used to conceal the presence of an adversary who has already gained access.
Trojan Horses	Software that appears to be useful, but actually is harmful with malicious code.
Key loggers	Program that records key strokes on computers where they are installed.

Figure 2-1: List of CPS Exploits, Adapted from [8]

2.3 Hierarchical Embedded Cyber-Attack Defense System (HECAD)

In order to be able to address the issues above, the hierarchical embedded cyber-attack detection (HECAD) system is implemented. HECAD consists of a multi-level hierarchical hardware and software architecture [1] that is designed to monitor and detect the many different hardware, information and functional faults and attacks as discussed previously. Within the HECAD architecture there are four levels of abstrac-

tion designed around detection at multiple levels of operation on a unmanned system. This system is intended to verify mainly the integrity of information throughout the many different subcomponents involved within the flight control system as described in section 3. Figure 2-2 shows the general architecture of HECAD.

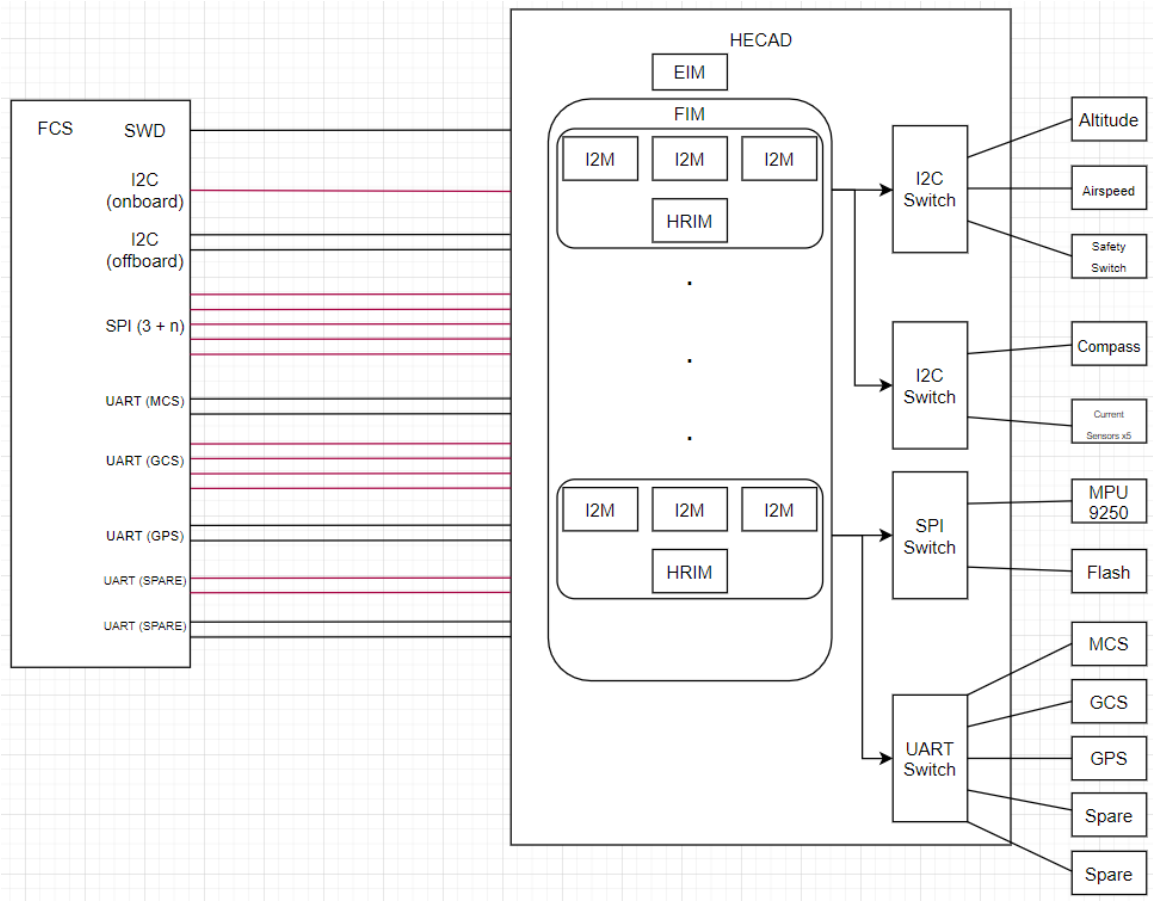

Figure 2-2: General HECAD Architecture in Zynq MPSoC

2.3.1 Hardware Resource Integrity Monitor

At the lowest level is the Hardware Resource Integrity Monitor (HRIM). The HRIM is designed to monitor the operation of a module on one of the communication ports: UART, I2C, and SPI. The HRIM monitors the hardware characteristics of a sensor, such as the operation of the communication protocol, specifically to ensure that the configuration of the protocol is the same on both the receiving and transmitting ends, as well as to ensure the configuration does not change mid-operation.

The HRIM sniffs data passively, and if an error in the communication bus has been detected, a switch can be triggered, detaching the faulty sensor until it has been repaired or reset, or simply alerting the operator. The HRIM can be applied to different applications, not just monitoring a hardware protocol.

2.3.2 Information Integrity Monitor

The next level up is the Information Integrity Monitor (I2M). As the name suggests, the I2M continuously checks for the integrity of the data. Unlike the HRIM, the I2M cannot be reused across different sensors. A dedicated I2M will need to be developed on a case-by-case basis for each sensor. This is to ensure that the I2M will be able to parse data from each different sensor and is designed to be independent from every other on-board sensor. However, despite the independence from the different sensors, there is a dependence on the HRIM, as the I2M receives correct data information from the HRIM. The purpose of the HRIM is to pull the data from the sensor and package it for analysis, while the I2M uses the packaged data for verification. In most cases, the data from each sensor will already be packaged into different registers, and the only task from here will be to identify the registers and their respective values. The I2M knows only about the data coming from the sensor that it is designed for: the operational ranges, typical values, and expected values.

2.3.3 Functional Integrity Monitor

Next comes the Functional Integrity Monitor (FIM). The FIM monitors the operation status of all of the sensors and actuators on-board from a higher level than the HRIM or I2M. At this level, data is collected from the sensors and actuators as well as the communication with the ground control station to ensure that there are no gradual changes to the data in accordance to its mission and functional requirements. An attack that cannot be detected by the HRIM and I2M will be analyzed at this level. As an example, an attack performed targeting a GPS spoof will allow attackers to manipulate the reference point of a GPS module. This in turns allows

for the attack to place their own GPS coordinates in the spoofing attack as desired. Because it is a spoof attack, the HRIM and the I2M will not detect the attack, since the data received by the GPS module will still transmit correctly over a specific communication protocol (UART) and will be within a reasonable range. A slow drift in the coordinates are categorized as correct data by both the HRIM and I2M. The data reported by the GPS module will be different according to the attackers's desired coordinates. With this, the attacker has the opportunity to slowly drift the coordinates away, causing the UAS to drift as a reaction to the change in coordinates. A sudden and large change in the GPS coordinates will be caught by the I2M. From here, the FIM detects that there has been a drift in the GPS coordinates, and uses another metric for validation. The goal of the FIM is to be able to detect changes that are forbidden at both the sensor / actuator level and the state of the flight controller itself but passes the lower level checks implemented at the HRIM and the I2M.

2.3.4 Execution Integrity Monitor

On top of the HRIM, I2M, and FIM comes the Execution Integrity Monitor. At the highest level of the HECAD hierarchy, the goal of the EIM is to be able to monitor and detect abnormal changes to the hardware resources used by the flight controller. While the goal of the HRIM, I2M (hardware and information in programmable fabric), and the FIM (functional level in software) is to detect lower level faults and attacks, the presence of malicious firmware and injected false data may not be caught by those respective monitors if they match the configurations and data formats of the sensors, especially if the firmware resides in the flight controller itself. Thus, the introduction of the EIM allows for the monitoring of the execution state and resources of the flight controller itself, including but not limited to: resource utilizing, memory status, state based operation, and event calculus. With resource utilization and memory status, the EIM can determine if there is malicious or alternative code running on the flight controller, resulting in unexpected behavior by the flight controller shown by the resource consumption or memory status. If there is malicious code that runs an alternative block of code that is not obvious to resource and memory metrics, then

a change in the state based operation represented by a control flow graph [6] or an event calculus model as explored in [11] can determine a potential fault within the system. In [4], Sutton et al. describe the method of using decoy processes, where a process is not intended to run in normal operating circumstances. If the presence of malware triggers running of this process, then it shows the detection of the malware. Furthermore, the access of specific resources such as a decoy file system or decoy I/O and data that is only accessible through the decoy process indicates the presence of malware. This can be used alongside an approach monitoring control flow as described in [6], because the access of decoy resources indicates a change in the state. In order for decoy processes to be effective, they should not consume as much resources as the main flight controller. An additional metric that can be used to evaluate correct execution characteristics exhibited by the flight controller. In [5], several power and heat characteristics of a system were observed. Alongside a decoy process, this can be used to ensure there is no change in the processing code of the flight controller.

2.3.5 Information Flow

The information flows upwards through the multiple levels of the hierarchy and consists of both the data coming off of the sensor into the bus, and the error information that causes an error to be detected. Going through the HRIM, the information is packaged into bytes that can be interpreted in specific ways in the I2M. This information does not contain configuration information such as start or stop bits in the UART protocol or the acknowledge and not acknowledge bits in the I2C bus. This configuration information is only used in the HRIM to verify that it is working correctly. In the I2M, every byte or bytes of data is taken and verified against the range of data that is reported in the respective data sheet. If there needs to be a parser to obtain the address and register of the data, this is also done here to ensure the slave addresses and the register addresses are valid. If they are not, this information is sent upwards to the FIM along with anything reported by the HRIM. The FIM then runs a higher level of processing on the information, but ONLY on the correct forms of data. Processing on incorrect data as determined on the lower levels of integrity will

also yield incorrect data on the functional level.

2.4 Related Works

In [6], Stracquodaine et al. provide a novel approach of securing unmanned systems using real-time software functionality analysis. This allows for the system to detect presence of any faulty malware within the flight control system (autopilot and operating system). By identifying internal code events (locations in the program's logic) as well as the software events, the flow of the system can be modeled and a detection methodology is used to identify intrusion. Their solution is to implement an embedded software solution within the flight controller itself. The issue with this is that providing a software based, embedded solution provides a way for the attacker to work around being detected. The presence of a profile is meant to safeguard this issue, but an embedded solution can go down along with the whole system if the attack is firmware based and injected long before the main autopilot code can even have the chance to run. The approach here would represent the functionality intended at the functional and executional integrity levels of the HECAD system.

A more independent approach is described in [12]. Sabaliauskaite and Mathur explores the use of intelligent checkers to verify the functionality of the specific devices that it is measuring. In this paper, an intelligent checker (IC) is used in a cyber-physical system control loop, where the data from an actuator is fed into a controlled process. The output of the controlled process feeds into both the sensor and the intelligent checker. The intelligent checker would ensure that the process makes valid commands. This can be adapted to be used on a UAS, and would be checking commands and operations from the flight controller as well as from the sensor. The purposes of the intelligent checker is to be able to measure various parameters such as temperature, light, and pressure. Depending on the sensor, certain parameters can vary as well. For GPS sensors, parameters that can be introduced are signal strength and direction of arrival, since they can be the most important factors in distinguishing real and spoofed GPS signals [13]. An attempt to spoof a GPS signal will exhibit

large changes in either of those parameters, and therefore an intelligent checker will be useful. In detecting the GPS spoofing signals, machine learning or the information from the narrow band receivers can also be used [14]. While these parameters are applicable for large CPS systems, the use of this for an unmanned system will introduce more environmental issues and less applicable parameters, specifically noise that is emitted by the on-board components or environment (vibrational noise or environmental noise). Therefore, this approach works well when the intended measurements are represented digitally. If this concept were to be introduced, it would sit within the hardware resource integrity monitor of the HECAD system.

Another approach to identifying abnormal functionality in a sensor network is by introducing neural networks. In general, neural networks prove to be very useful in identifying patterns and recognizing characteristics. While many of these uses generally revolve around a static set of data (i.e. images), an adapted neural network can be used to look for trends and characteristics in time-based data. In [15], Shin et al. introduce the use of Long Short-Term Memory and Gated Recurrent Unit (LSTM, GRU) neural networks to identify attacks using sensory information measured during real time. In this paper, it is shown that the computational cost is very low, with an execution time of 0.002104 seconds for LSTM and 0.001645 seconds for GRU approaches for detection. The neural networks generally need to have a model that considers the information correct and requires data sets that correspond with mission data. If mission data is missing, it is hard for the neural network to isolate correct versus incorrect sensor data.

In [16], Ding et al. explore the existing strategies to deal with falsely injected data: bayesian detection with binary hypothesis, weighted least square, kalman filters, and quasi-FDI. With the weighted least square approach, a framework is used to ensure that the data being measured does not exceed a predetermined threshold values. This is used as a way to ensure that a discrepancy in measurements are detected. For the kalman filter approach, a version of the kalman filter that is existent on the flight control system is adapted and modified, allowing for the processing of the raw sensor data as well as the estimation of the aircraft's physical state. By providing aircraft

information, additional information can be used to verify the estimation.

In [2], a similar application is used when the authors trained a convolutional neural network to identify abnormal events in a video. Their work was split into two stages, the first of which extracts features from a batch of inputs, and the second of which is used to detect abnormal features. The problem here though is that the training of the neural network relies on the use of still images, and requires a different approach when looking at time-series data such as sensor data. This approach can be very useful when dealing with mission specific applications, but not so much when working with free-fly operations. At the functional level, this can be introduced as a secondary detection mechanism that runs in parallel with the available existing solution. A similar approach is done in [3], where a new algorithm is introduced to detect faulty data injected into a UAV. In this implementation, an adaptive neural network structure is used and trained for fault detection. A different approach is used when determining fault classification. Instead of using conventional neural network procedures, a nonlinear observer output and sensor output is used to estimate faults. For weight updates in the neural network, an embedded kalmann filter is used to perform online tuning, allowing for quicker and more accurate attack detection. Online training is different from traditional training methods such as batch training. Batch training is when a large set of data is used for training, while online training is used for new data that comes into the system on the fly [17].

In [18], Sedjelmaci et al. explore the implementation of a hierarchical detection and response for cyber-attacks within a network of unmanned systems. In this paper, the system explores detection of attacks such as GPS spoofing, jamming, and gray / black hole attacks. By verifying information at both UAS nodes and the ground station, the intrusion detection system can identify the incoming attacks. This is a rule based approach when looking for specific attacks. This is a different approach from the HECAD system described above, as it spans across outside communications networks (flight control system and ground control station). HECAD is embedded within the flight control system, and monitors not only communications between the flight controller and the ground controller, but also operations within the flight

controller itself (sensor data, bus operations, and autopilot code).

Chapter 3

Subsystems

3.1 Implementation Platform: Avnet Ultrazed-EV

The Avnet Ultrazed-EV is a system on module that is based on the Zynq Multi-Processor System on Chip (Zynq MPSoC). It is an field programmable gate array with the ARM cortex-A53 processor. Within the Zynq MPSoC resides a processing unit alongside the programmable fabric. The HECAD system is designed to run in the Zynq MPSoC, with the HRIM and the I2M residing in the programmable fabric, and the FIM and EIM residing in the processing system. Communication across the programmable fabric with the processing system is done using memory mapped AXI transactions. In high performance processing, data can alternatively be streamed upwards into the processing system, instead of using memory mapped IO. Previous work to develop individual portions of HECAD (only the HRIM or I2M for a GPS module) were implemented on the Zynq-7000 System-On-Chip. After designing the HRIM and I2M for an additional UART devices (VACS Parser, explained below) and designing the HRIM and I2M for the I2C device (MS5611 Barometer), the amount of resources required to implement all of the logic exceeded the available amount in the Zynq-7000 chip. As a result, the entire design was exported to target the Zynq MPSoC.

3.2 Monitored System: Aries Flight Controller

In order to understand the HECAD architecture, the sources of information from which the data is coming to and from need to be identified. The purpose of HECAD is to be able to pull data coming from the ground control station to the flight controller and vice versa as well as GPS Module. This data is transmitted through the XBee communication device, which communicates with the flight controller through the UART bus. Next, data from the flight controller to the barometer sensors is pulled through the I2C bus. As such, HECAD needs to be able to parse data from the I2C bus and the UART bus.

The Aries flight controller is a heavily developed and tested flight controller that has been built from scratch as a result of several years of research, development and optimization by the VCU UAV laboratory. The Aries flight controller contains multiple buses to communicate with the variety of peripherals as highlighted in figure 2-2. On the I2C bus, there are 3 devices on the internal bus (barometer sensor, airspeed sensor, and safety switch), and 2 devices on the external bus (compass and current sensors). On the UART bus, there are 3 devices with several spares (mission control system, ground control system, and global positioning system). Of these devices, the barometer sensor and the GPS / VACS sensors are implemented in HECAD to ensure correct functionality at the bus level and the information level.

3.3 Sensors

3.3.1 NEO-M8 GPS Module

The NEO-M8 GPS module by uBlox is a versatile GPS module that supports multiple communication buses. For the purposes of this implementation, the M8 module is configured to UART at 115200. The M8 module allows for the NMEA sentences protocols, and communicates with the flight controller through the UART configuration. Out of the box configurations for the M8 module may not have the device configured to a UART port at 115200 baud. As a result, the flight controller

runs a series of codes, trying several options of baud rate until an understandable set of data is returned. This causes issues for HECAD as explained later.

3.3.2 VACS Packet Transmitter

The Virginia Commonwealth University Aerial Communication Standard (VACS Packet) is the communication protocol that is used by the Aries flight controller to communicate with the ground control station. The VACS packet is transmitted over an XBee wireless link. The XBee device communicates with the flight controller and the ground control station through UART.

For a specialized VACS packet, each byte of data represents a special header indicating sync bytes, destination address, source address, message id, data length, and checksum. As the name(s) suggests, the source indicates its source of transmission, the intended destination, the message header, the length of the message, the data payload, and the checksum. The checksum for the VACS packet is calculated as follows:

$$\text{checksum A} = \text{checksum A} + \text{current byte}$$
$$\text{checksum B} = \text{checksum B} + \text{checksum A}$$

The checksum calculation does not include the sync bytes or the checksum bytes transmitted by the sender. Each of these bytes has their own ranges, and every time they are read in from the HRIM, they are stored and checked against the expected ranges. For each packet that is transmitted, a checksum is calculated separately in real-time. If there is a mismatch in the checksum values reported, then an error flag is thrown, indicating at some point the data that has been transmitted has been either modified or dropped.

3.3.3 MS5611 Barometric Pressure Sensor

The MS5611 is a barometric pressure sensor with the ability to communicate with devices over both I2C and SPI communication devices. It allows for a 24-bit

ADC pressure and temperature value. The pressure and temperature has a 0.01 millibar and 0.01 Celcius precision respectively. Similar to the NEO M8 and the VACS Packets, the MS5611 barometer sensor contains fields, including calibration data, digital values and conversion values. Each of these has their own specified absolute and recommended register size bounds. What this means is that even if there is a recommended data type, the actual data size returned by the slave device may not use up all of the bits. For example, the MS5611 contains a register that has the difference between the reference and actual temperature. Even if the suggested register size is a signed 32 bit integer, the actual value returned contains a range of a 25 bit signed integer. For each of the registers, the ranges are checked for absolute values to ensure nothing reported is out of range.

The checks as described above are designed as rationality checks to detect any reported values that are not possible as shown in the data sheet. Within the I2M is where large deltas and discrepancies are detected as well. In the example of the MS5611, calculating the temperature and pressure will allow for range checking. A large delta or change between two samples may occur, but because the I2C bus operates at a high speed capacity, it is more realistic to keep a running average and throw an alert if the average changes by a certain threshold. The data is kept in an array, keeping a record of the average across the size of the array, for example 10 samples. If the averages change drastically (depending on the threshold that needs to be set), then an alert is thrown. A similar approach is taken for samples that are all zeroes, or all ones or flatlining in general. Realistically, they may return values that are very close to zero or ones or a set value. By taking the running average, any indication that the average is not changing indicates a no response or flatlined system. As a result, this will throw an error.

3.4 Communication Buses

3.4.1 UART & Vulnerabilities

The verification of the sensors at the bus level ensures correct functionality at the specific configuration at startup. The UART communication port is generally configured in a specific manner. This will typically be the transmission of data at a specific baud rate or correct number of data bits and start / stop and parity bits. As explained previously, there are two devices on the UART bus: a GPS module with NMEA sentences, and an Xbee device transmitting VACS packets. Both of these devices are configured to communicate with the flight controller over UART at 115200 speed, 1 stop bit, and no parity bits.

When a UART device attempts to communicate with a microcontroller at a configuration unknown to the microcontroller, several issues can happen. If the data is not received at the correct speed, the data received will be jumbled and not interpretable. If the timing for the baud rate is very large, control over the sensor will be lost, as there will be no way to know the status of the device (still transmitting, idle, stopped, etc). Attacks at this level can jam the UART bus with invalid or scrambled data.

3.4.2 I2C & Vulnerabilities

The I2C communication is a shared communication bus between devices. The master device can share a bus with multiple slaves, and the correct functionality depends on the master device correctly identifying a valid slave address. With correct operation, the master device will send the slave address (a 7 bit address) along with a 0 write bit / 1 read bit, receive acknowledgement from the respective slave. Next, the master will send a command (8 bit command), receive acknowledgement from the slave, and then the master will send the slave address again with a 0 write bit / 1 read bit if the command is intended to read data, or a 1 write bit / 0 read bit if the command is intended to write data to the slave. If the write bit is a 0, the slave will

send data back. If the write bit is a 1, the slave will acknowledge and wait for the next byte of data to be received. Communication between either devices relies on the SDA data line and the SCL clock line.

The operation of the I2C relies on the responsiveness of the SCL and SDA lines. The SCL lines allow for the clocking of the data out, and the SDA lines allow for the data to be changed when clocked out. If the SCL line is not responsive, it safe to assume that either the master device is holding the SCL line high or low(failing to clock data out), or that the slave device is holding the SCL line high or low (slave has stopped responding or timed out). Similarly, if the SDA fails to change, and if the SDA line is high, then either the master device has failed to acknowledge the data, or the slave has failed to acknowledge the command. Attacks at this level can manipulate either line to either lockup the bus, or to manipulate the data going in either direction.

Chapter 4

Implementation of HECAD Hardware and Information Monitors

There are two main protocols that will be explored in this implementation: Universal Asynchronous Receiver Transmitter (UART), and Inter-Integrated Circuit (I2C), also commonly known as the Two-Wire Interface over three levels of the HECAD hierarchy: HRIM, I2M, FIM. Each of the different sensors will communicate with the flight controller through a specific bus. At the lowest level of abstraction, a Hardware Resource Integrity Monitor (HRIM) is implemented in hardware to be able to detect correct and incorrect sensor functionality at the bus level. One level up, the Information Integrity Monitor is implemented in hardware to validate information integrity. One level higher above the I2M, an FIM is implemented to hold all valid data coming into HECAD from the two communication buses, but no further implementation is done for the FIM due its being out of scope. Figure 4-1 shows the implemented architecture of HECAD, with the expanded devices on each communication bus. For the purposes of this thesis, the detection logic for the various different attacks applicable to a GPS module, a VACS Packet transmitter, and a MS5611 barometer are implemented, and mitigation techniques are explored.

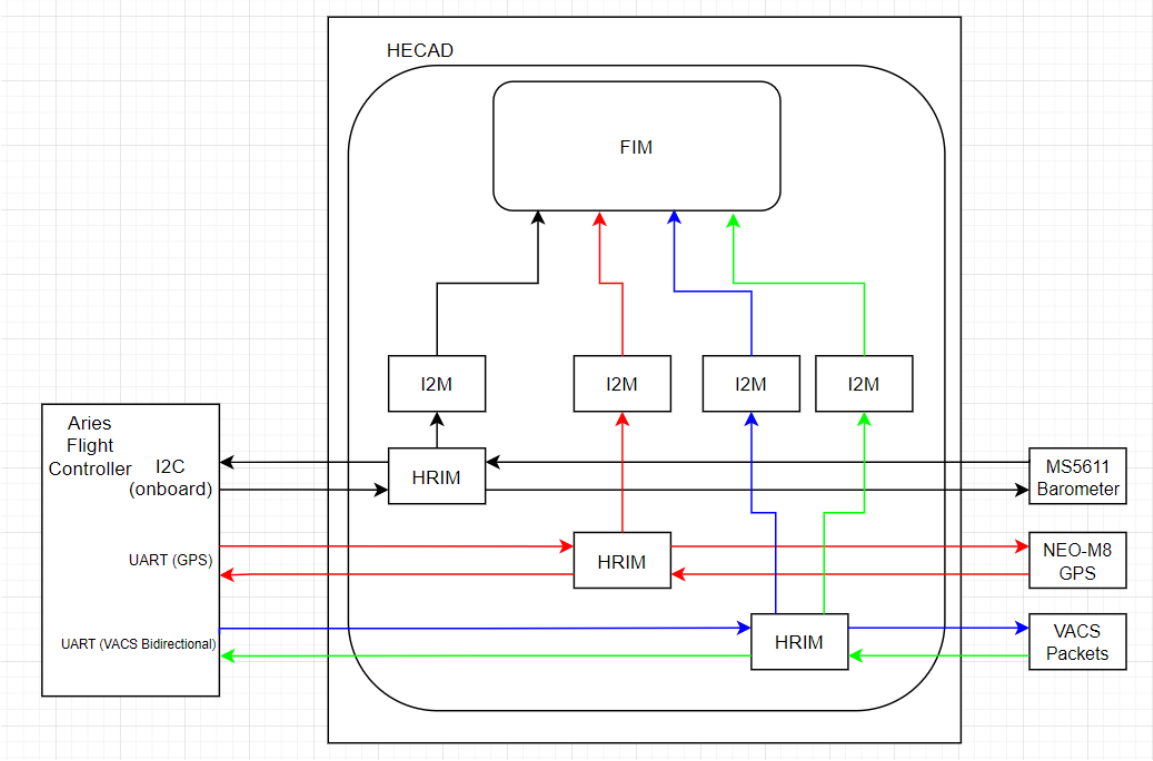

Figure 4-1: Implemented HECAD Architecture in Zynq MPSoC for I2C and UART Devices

4.1 Hardware Resource Integrity Monitor

In this implementation, the target of the HRIM was to be able to detect three main things: a change in the baud rate, unexpected parity and to verify the start and stop bits. An existing HRIM for the UART is taken from [1], and modified to work with the updated baud rate configuration. Using oversampling, a baud rate change is detected if there is a bit change outside of the middle of an oversampling clock. An oversampling clock of 16 times the baud rate is used along with a 16 bit register. If a change has occurred in the middle of the oversampling clock, half of the shift register capturing the data from the baud rate clock should be low and half should be high and vice-versa depending on the bit change. If the baud rate is different from what is expected, then this condition will fail. If there is a non-one parity detected, no stop bit detected, or more than 8 data bits at a time, then the HRIM will throw an error for the respective case. Since the UART device is the same for both devices, the logic

to detect errors is the same for both. Since UART is not a shared bus, the HRIM for UART is duplicated, despite having the same logic. In section 4, a more specific VHDL approach is described.

In developing the hardware resource integrity monitor for the two-wire interface, many of the concepts implemented for detecting changes on the UART bus can be carried over. Similarly, an HRIM has been adopted to parse the I2C data coming in from the MS5611 barometer sensor. For this sensor, several checks are implemented at the I2C HRIM. This includes the address bytes, the command bytes, and the characteristics of the SCL / SDA line.

Starting with the speed of transmission, it is known that the internal I2C bus needs to run at a specified speed. In this case, the speed of the bus is configured to be 336 kHz. From this, a frequency check can be made to ensure that the bus operates at the right speed within a certain threshold, typically a few percentages. Given the known frequency, the period of each SCL pulse is also known. Using the system clock of 50 Mhz, a calculation is made to determine how many system clock cycles are allocated to each SCL clock pulse. This same check can be applied for the SDA, but instead of frequency of the SDA line, the responsiveness is checked to ensure there is activity on either line.

In addition to the SDA and SCL line checks, address checking and command checking will be done at the HRIM. Address checking needs to be completed in order to properly parse data for incoming data. Because the I2C is a shared bus between multiple devices, the I2C protocol depends on the master communicating with the right slave device through addressing. For the I2C bus, the HRIM contains a parser that checks for a start and a stop condition, pulling in data when it detects a master read operation and checking commands when it is a master write operation. When there are multiple devices on the bus, the addresses are checked across all slave device addresses to ensure that the master is attempting to communicate with valid slave devices. Command checking can be a little more difficult. If the commands are to be checked in the HRIM, then all commands across all slave devices will need to be checked. As a result, the logic for the HRIM will increase exponentially. It is more

beneficial to combine individual command checks to the I2M for the I2C device, where the I2M is tailored to each specific sensor. This is further explained in section 4.

4.2 Information Integrity Monitor

On another level in the HECAD hierarchy, data may be able to pass the HRIM checks, but perhaps the data may not meet sensor specifications. For example, the communication link between a UAS and a ground controller may contain data that is organized into a specific protocol. The communication link between the ground controller and the flight control system transmits data packets organized in the VCU Aerial Communication Standard. While the UART components transmitting these VACS packets may be operating correctly, the data received at either end may not be. Over a wireless link as shown in figure 4-2, dropped data packets are common, resulting in entire packets if not partial parts of a data packet missing when received. As a result, partial data is received. In alternative cases, checks for information integrity are not limited to information formation such as a VACS packet. In specific sensors interfaced over SPI or I2C, it is harder to distinguish packets from raw data versus well formed data, especially since each sensor may transmit data differently. As a result, different sensors will have different checks put in place. The I2M is introduced, allowing for sensor information integrity verification at the information level. At this level, the I2M knows only about the type of information to be expected for the respective sensor. Since every sensor is almost guaranteed to transmit data in their own order and have their own register maps, a dedicated I2M needs to be made for individual sensors.

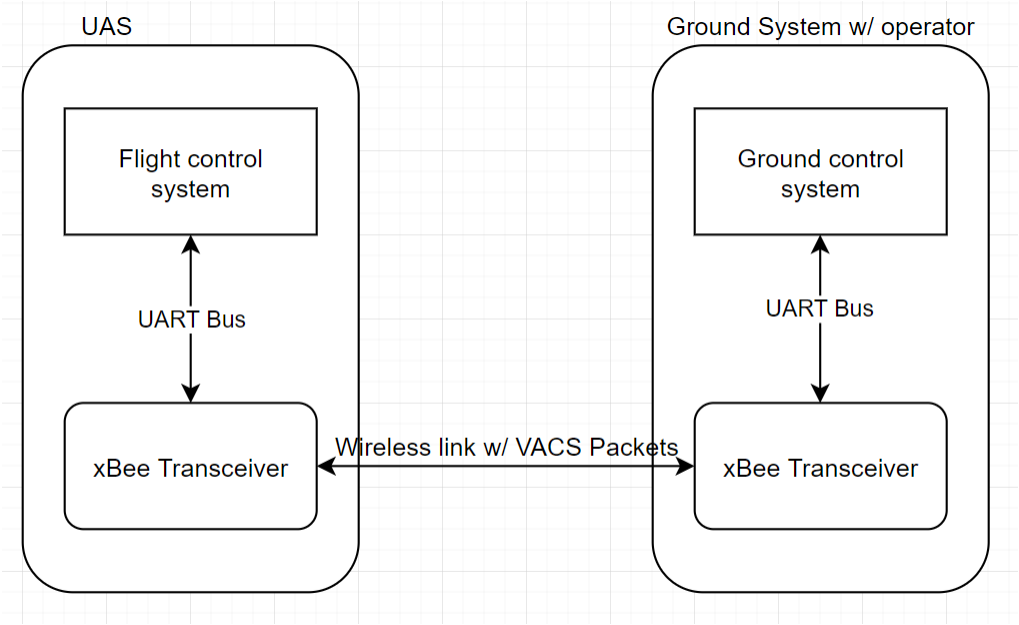

Figure 4-2: Communication Link between the UAS and the GCS

4.3 Functional Integrity Monitor

On an entire higher level of operation, an attack to the UAS can successfully pass the checks at the information and hardware level, and ultimately target the operation of the flight control system itself. While an adversary may not have direct access to the system, an attacker may be able to repeatedly spoof the data in such a way that slowly brings the system down. For a man-in-the-middle attack between the flight controller and the ground control station, an attacker may be able to listen in and successfully manipulate data that then gets sent out to its original destination. If an attacker wants to be able to bring down the UAS all-together, they may attack the reference points of the orientation and positional sensors to trick the system into going in a certain direction. A common example is a spoofing attack on the GPS sensor to slowly force the UAS to drift off into an opposite direction. In either case, bus transfer protocol will be correct, and information will still be well formed by the sensor. The Functional Integrity Monitor (FIM) is then introduced to monitor all of the data coming in from all of the sensors at any given point in time. Here,

the validity of the data from the sensors is not checked byte-by-byte or at the bus level, but rather multiple points of data will be taken over time. A higher level of processing is performed on this set of data to make sure that the data looks reasonable with respect to expected data values and mission-specific data. Multiple algorithms will be explored and evaluated in future work. In the previous section, the idea of taking a running average will be more useful and easily implemented here. Several running averages are taken for the pressure and temperature values.

4.4 Hardware Validation Checks

The detection at the hardware level allows for the ability to find faults at the bus level. Often times, an attack coming in from the outside may target the configuration of communication protocols. Some of these examples include changing the configuration of the UART ports or manipulating the I2C lines during operations. The following sections describe the process of detecting hardware faults and unexpected configurations.

4.4.1 UART

The detection and parsing of a UART bus utilizes an oversampling clock and a 16 bit shift register. Upon detecting a start bit, the over sampling clock counts until the middle of the start bit (8 clock cycles). From here, the counter is reset and starts counting until 16 oversample clocks have passed. When this happens, this will place the counter in the middle of the first transmitted bit. A secondary counter keeps track of how many bits have been received. Once a total of 8 bits have been received, the 9th bit is expected to be the stop bit. If this condition fails, then there is likely an issue with baud rate, the number of data bits, or the parity bit. This state machine is illustrated in figure 4-3.

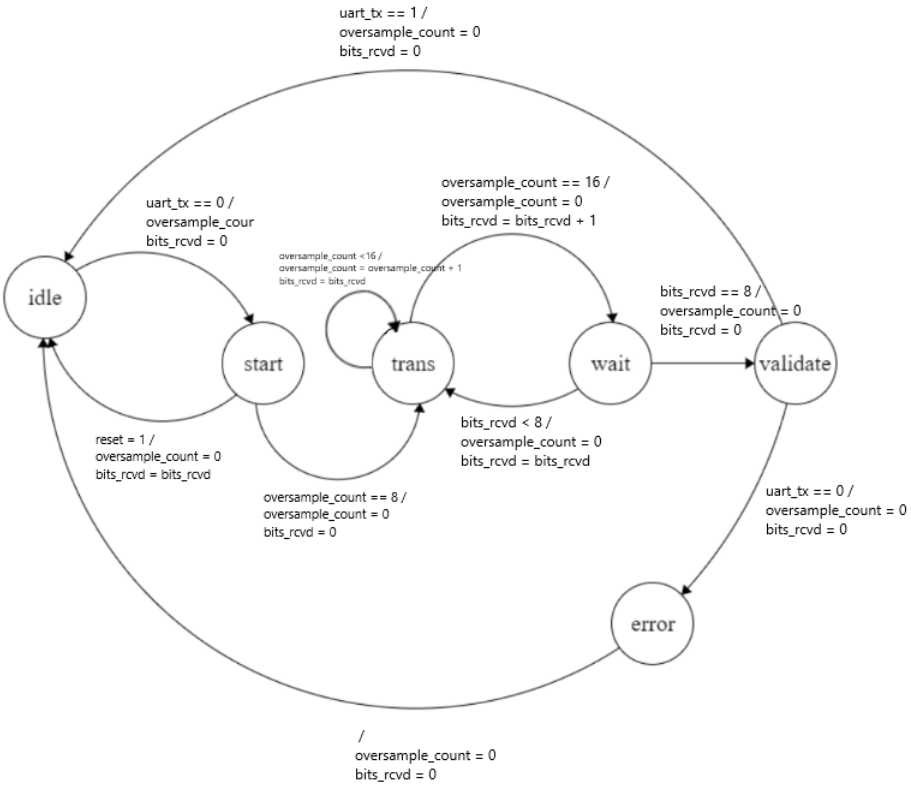

Figure 4-3: UART HRIM State Machine

Baud Rate

In order to be able to detect a change in baud rate, an oversampling clock is used. The oversampling counter is started when the start bit is detected, and then the counter is reset when the middle of the start bit is detected. In the next iteration when the oversampling clock reaches 16, the first transmitted bit is received. This bit is then shifted into an eight bit shift register. If the baud rate of the UART transmission is correct, the bit change will occur approximate halfway through a 16 bit count iteration. Within the shift register data, the upper 8 bits (previous transmitted bit) should equal the lower 8 bits (next transmitted bit) when an XOR operation is done across the bits. For example, a shift from a '0' to a '1' will result in the shift register containing:

0000000011111111

Where the leftmost value is the first bit transmitted (less significant bit) and the right most value being the second bit transmitted (more significant bit). If there is no bit change, and the previous bit was a '0', and the new bit is a '1', then the shift register will contain:

0000000011111111

Similarly, if the previous bit was a '0', and the new bit is also a '0', then the shift register will stay the same. On the contrary, if the previous bit was a '1', and the new bit is a '0', then the shift register will contain:

1111111100000000

If the previous bit transmitted is a '1', and the new incoming bit is a '1', then the shift register will contain:

1111111111111111

By checking the the XOR operation of the upper 8 bits and the lower 8 bits, it can be determined if there is a timing or framing issue. If the data that is being transmitted matches the expected baud rate, then data should settle into the shift register as expected. If the baud rate is higher or lower than expected, then the switching point will be skewed either to the left or the right when a bit change is detected. Figure 4-4 shows an example at 115200 baud rate. The state machine does not detect how many stop bits there are, rather just detecting a stop bit and then waiting for the next start bit for transmission.

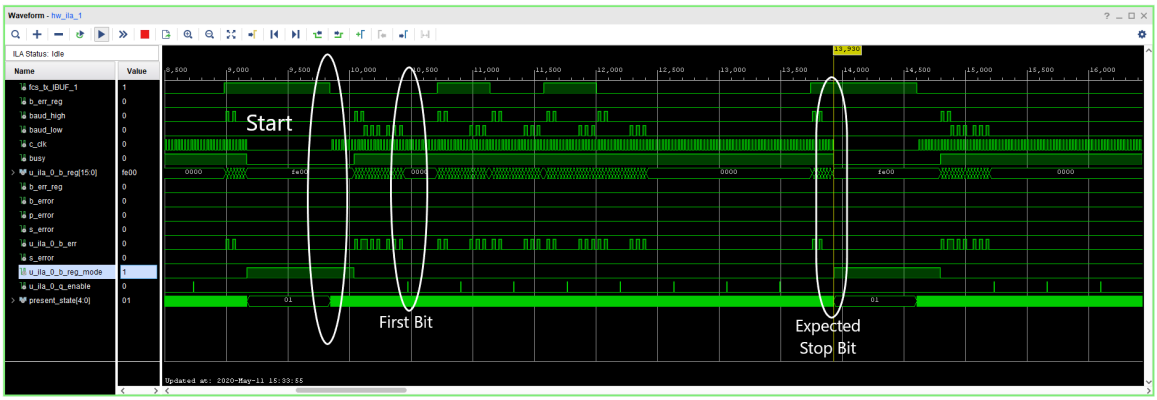

Figure 4-4: Sampled UART Transmission at 115200, 2 stop bits

At approximately 10900, the strobe on q_enable indicates the midpoint of the second transmitted bit. This bit is a 1. The first bit transmitted is a 0, shown at approximately 10500, where the data is shifted into the shift register. At 10700, it is the rising edge of the second transmitted bit. At this point in time, the second bit is shifted into the 16 bit register. Since the first half that has been shifted in is all 0's, the second half of the shifted register is all 1's. Since the timing is expected, exactly half of the shift register is all 0's and the other half is all 1's. The XOR operation on the upper half results in 0, and the XOR operation in the lower half results in 0. This indicates no error with the baud rate.

1111111100000000

In a figure 4-5 is a simulation of a UART transmission running at a slower rate. While the transmitted data captured is valid, a quick look at the tx line can easily distinguish a baud rate issue. After the 8th bit of transmission, the stop bit is expected to be present. A 0 bit is still present, and as a result, an error is thrown. The state machine is expecting it to be the 9th bit, the stop bit, but in reality, it is most likely 5th or 6th bit in the transmission.

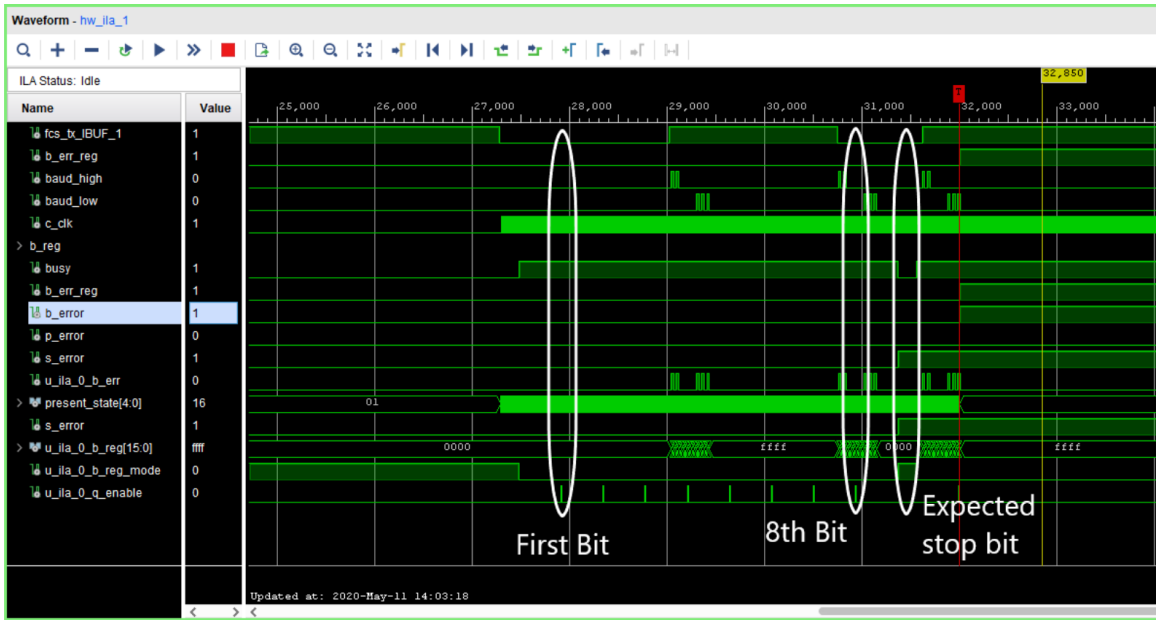

Figure 4-5: Sampled UART Transmission at 57600

Parity

The configuration for the UART transmission can be done with an optional parity bit. If this parity bit is transmitted, then an even parity bit will be '1' if the number of 1's in the transmitted byte is odd, and '0' if it is even. On the contrary, an odd parity bit will be '1' if the number of 1's in the transmitted byte is even, and the parity bit will be '0' if it is odd. In simpler terms, the parity represents the next bit to be added to make the number of 1's within the transmission either odd or even.

Stop Bits

The state machine developed for detection is designed to be able to detect the existence of a stop bit. If there is at least one, even if there is one and half or two, then it registers the transmission as successful. In the testing process, the state machine is tested with a single byte of data. The state machine looks for a stop bit, and the first stop bit detected by the state machine will reset the state machine to the idle state after a successful transmission. The overhead is not significant enough to cause a framing issue.

4.4.2 I2C

Within the hardware of the I2C, there are many opportunities to check for the correct operation of the I2C bus. Many of the concepts from the HRIM in the UART protocol carries over. Of these concepts, generally clocking frequency and responsiveness of the SCL and SDA lines are checked. At hardware level, since checking for the correct addresses allows for determining if the master is attempting to communicate with a valid device, then addressing and command checking are validated as well. While address checking and commands across all devices on the hardware level may provide advantage in identifying incorrect address and command operations, the combination of the number of commands and the number of devices makes the logic within the HRIM too large to be implemented. It is more efficient to complete this task in the I2M, where there is a dedicated I2M per slave device.

The I2C parser looks for the transactions as described in 4.4.2. The HRIM for the I2C device looks for a start condition, followed by the address of the slave. Once the address is received and validated, the HRIM looks for a slave acknowledge. Next, the read / not write bit is examined, and if it is a read, then the state machine pulls in the next byte along with a master acknowledge. If there is no acknowledge, the state machine returns to idle. If there is, the state machine looks for the next byte of data, and checks the master acknowledge again. This process will repeat as long as there is a master acknowledge. If the state machine is to perform write operation, the state machine picks up the next byte of data and looks for the slave acknowledge. This state machine is illustrated in figure 4-6.

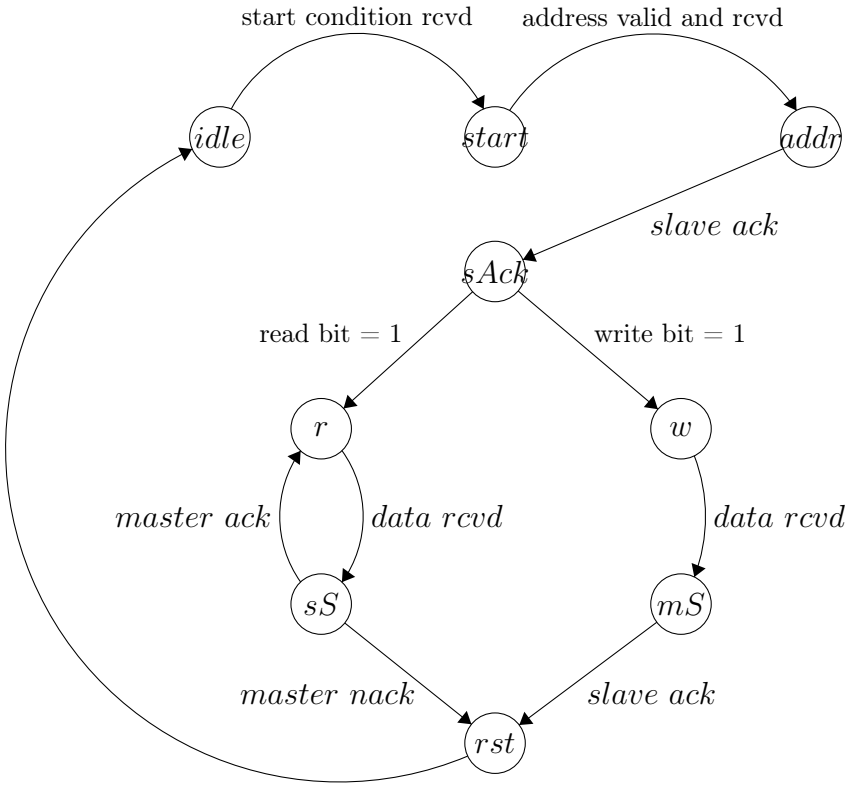

Figure 4-6: I2C HRIM State Machine

Slave and Master Acknowledgement

The state machine demonstrated in Figure 4-6 is used to parse data accordingly. Both the slave and master devices acknowledge the data received from the transmitter. However, the general acknowledgements by the slave devices must be made to ensure the slave can respond accordingly. In industry standards, the I2C bus models pull-up resistors. As a result, a high or undriven value indicates a no response from the slave or master depending on who is sending the data. In order to ensure that the master or slave is able to respond, the SDA lines need to be pulled down from either ends on the rising end of the SCL clock. If there is no acknowledgement, then this indicates either the end of the transmission or the failure to respond.

SCL Rate

When the clocking frequency is known for the SCL bus, then so are the periods of the high and low cycles of each clocking period. A state machine is developed here where both the frequency and the duty cycle of the pulse are checked. Upon detecting a start condition (when the SDA line drops when the SCL line is high), a counter starts to check the initial response of the SCL line (when SCL drops low). If the SCL line fails to drop low after a full cycle, then it is assumed to be held in high. Similarly, each transition during the transmission period is checked for each high and low cycle of the SCL clock pulses. If the clock stays too long in the low or the high cycles, then it gets thrown into an error state. Given the speed of the system clock, the number of system clock cycles is calculated to allow for checking of the SCL line.

For example, for the configuration with a 50 Mhz clock and a clock speed of 336 kHz, there needs to be no more than 148 system clock cycles in the entire period, or half for each rising / falling period:

$$\frac{50MHz}{336kHz} = \frac{50000000}{336000} = 148 \tag{4.1}$$

To avoid a strict bound, a certain percentage of error is given to the frequency to get a high bound and a low bound. For each high and low frequency, the maximum number of cycles is recalculated, and is used as the bounds in the frequency checker. If the number of clock cycles that has passed for each period falls below or exceeds the bounds, then the frequency for the measured SCL clock is higher or lower than expected. This results in an error with the clocking speed of the bus.

The same concept can be used to detect inactivity on the lines. After a start condition is detected, the SCL lines can possibly be held low or held high forever. The condition for a start condition is the falling edge of the SDA line while the SCL line is high, so the next step is to check for a SCL line held high. After the falling edge of the SDA line, the SCL line should drop no more than half the period later. However, because the duty cycle isn't necessarily 50 percent as in the ideal case, this threshold can be increased or decreased. If the SCL line is still high after this time,

then there is an issue on the bus and it is assumed that the line is not responding. Similarly, the SCL line stuck low error can occur if the number of system clock cycles bypasses that same threshold after the start condition.

A slave timeout can occur as well, and this error can be detected at either the HRIM or the I2M level. If after a certain amount of time has passed and there has been no activity from the lines, then it is safe to assume that there is no response. A bidirectional checking can be done from both the master and the slave devices. If the master sends the command and there is no acknowledgement for several commands, then the slave has timed out. Similarly if the slave returns data but there is no acknowledgement for the first byte of data returned, there will be a timeout. However, the state machine adapted to parse data in from the bus line will prevent reading in the first place if the slave device does not respond with an acknowledgement, even if the address is correct. This leads to the assumption of no activity. The detection of no activity on the SDA and SCL lines can indicate a bus crash as well. Generally, if the flight controller is functioning as expected, and it is constantly requesting data, then there needs to be activity on the SCL bus, at the very least. If there is no activity, it indicates a crash of the flight controller, or the slave is holding the SCL line low.

Command Checking

For the purposes of this implementation, the slave command checks are placed at the I2M level. Although the command checking can be placed within the I2M to minimize logic, the same effect can be achieved by using a state machine to check for valid addresses and commands. State machine 4-7 describes the process.

Figure 4-7 describes the process of checking for valid commands. The state machine starts off in an idle state, and upon detecting the address of the slave device of interest, then the state machines moves to a state waiting for the next command. Here, if a command is received and it is valid, then we move to a valid state, which then moves immediately back to the idle state. However, if the command is wrong, or if the command is never received, then the state machine moves to an invalid state.

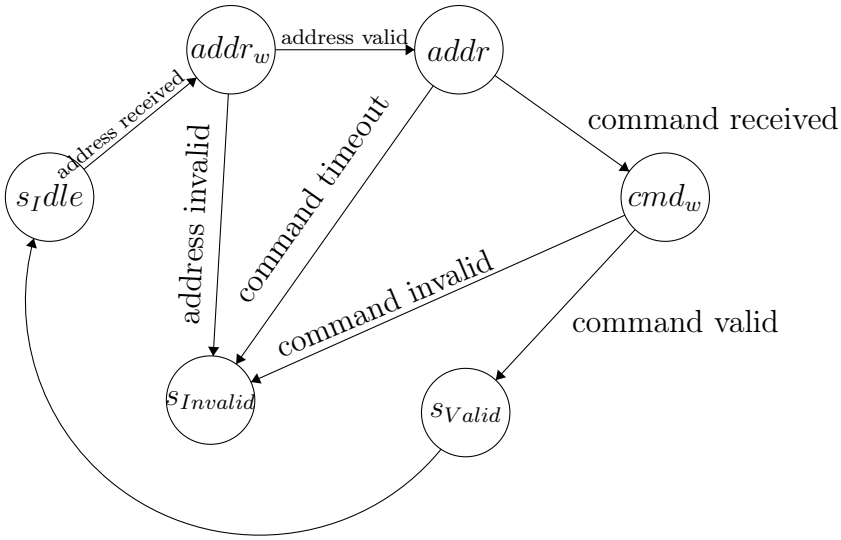

Figure 4-7: Command Checking State Machine

4.5 Information Validation Checks

While the detection at the hardware level may validate data, it only verifies that the data is operating within the specifications of the communication protocol. Data that has been validated at the hardware level may contain malicious data or data injected from a secondary source. The following sections describe the process and implementation of the different applicable checks at the information level.

4.5.1 Range

For every known register that a slave device is reporting, the corresponding data type and data range is checked. It is crucial to ensure that there is no out of range values returned because the data range may not take up the full range of a recommended data type. If there are out of range values, these issues will cause problems for other inner conversion calculations that need to be performed.

4.5.2 Flatlining

A check for flatlining is very straightforward. On the receiving end of the data (after the master has requested a valid register), the data can be validated against 4 metrics: all 0's, all 1's, same value, and same running average over a certain time window. The first two metrics can be tricky. Since data structures in which the data does not take up the entire structure are managed, comparing them to all 1's or all 0's are theoretically incorrect.

For example, suppose a register value is recommended to be a 16 bit signed value. This means that the range of values that can be returned is -32678 to 32767. But the sensor only returns values from -4000 to 8000. The extra overhead is used to store values resulting from arithmetic values used for other necessary calculations. The all 1's or all 0's for both the full range would result in -1 and 0, and which would be useless if we are looking at practical uses. The more applicable approach would be to check if the value is constantly -4000 or 8000, or constantly any number in between, since the sensor cannot return a value outside of this range. If it does, it will be caught in the range checking section.

On the other hand, detecting a non-changing value or a running average that has not varied by more than a specific threshold is a little more straightforward. For detecting a non-changing value, a process dedicated to the register or signal is used. A synthesized and implemented design checks the value of the register or signal every clock cycle, and a counter can be used to check the value of the register or signal every certain amount of time. If there is no change in the signal over a long period of time, then there is an issue with the data sent by the slave.

A similar approach is used when detecting a change in the running average of the data. Instead of checking the sample every clock cycle, a counter register can be used to add in a sample every time new data is read from the slave. Then, every new sample that comes in after a set amount (say 20 samples), the oldest sample can be removed, and the new sample can be added in. The new average is then recalculated. From here, the new average is compared to the old average, and if there

are no differences from the two, there is an issue with the data coming in and it will trigger an error. However, this is a very strict implementation, and the presence of a noise will most likely represent a change in the average. The more realistic approach is to keep the running average, but if the differences between a new and a old average has not differed by a set threshold, then there is an issue with the data. This takes a large amount of resources to maintain in the programmable fabric. The alternative to checking running averages is to do this process in the FIM, where it is easier to setup average checking over a set amount of due to the available data structures.

4.5.3 Discrepancy and Large Delta Changes

Detecting large discrepancy changes can be approached the same way as flatlining, but rather than comparing a current value with a sliding average of past values, a change in any two samples that are large will trigger an error. For example, in the MS5611, several processes were used to calculate the temperature and temperature compensated pressure. Between any two calculated samples, if there is a large enough change by 0.5 degrees (temperature) or 0.5 mbar (pressure), then an error is thrown.

The problem with this is that this method is checking for changes between two samples. Since this is a high speed bus, a practical fault (where the UAS is experiencing non-ideal environmental conditions), a sudden change in temperature or pressure will not necessarily be caught between two samples but rather over a large set of samples. Nevertheless, it is still important to have this check.

To solve the issue of discrepancy for a practical application, one would need to take the same approach as the flatlining approach: taking a running average, and checking for a change in the samples. Alternatively, another approach can be done by looking at the minimum and a maximum over a set of samples, and taking the difference between the two. This is a simpler approach to finding discrepancies, but may not necessarily cover a gradual change in data. The same ideas can be applied to longitude, latitude and altitude for a GPS module. Any sudden changes to any of these fields by either an average or a high to low will indicate a fault.

4.6 Information Flow

As explained previously, the information flows from the HRIM to the I2M to the FIM. While the HRIM is important in detecting higher level attacks, the implementation of the FIM is out of scope for this project. An HRIM and the I2M is implemented.

At the HRIM level, the hardware logic listens in on the bus and packages data into different bytes accordingly. For each set of data, the data is packaged into single bytes. For both the UART parser and the I2C parser, there is a state machine that checks for correct configuration in the UART and the addresses and the required slave acknowledge or master acknowledge. Every 8 bits of information that comes into the HRIM is packaged into an 8 bit register that is then passed onto the FIM. As explained previously, there are two devices that communicate on the UART bus. There is the NEO-M8 GPS module, and there is the VACS Protocol. For both UART buses, there are checks put into place the verify baud rate, parity, and start or stop bits.

At the I2M level, the data is interpreted in context. For the VACS packet parser, information such as the checksum validation bytes are checked. For the MS5611 barometer sensor, information such as temperature or pressure as well as their respective ranges are checked. Given the data sheet information, the suggested data types are used and the absolute ranges are given. Since the full range of the data type are often much larger than what can be reported by the sensor, a large part of the ranges are not used. This provides an opportunity to check for data ranges for all applicable data registers within the sensor. This is applied to both the VACS parser and the MS5611 barometer.

While the development of the FIM is out of this scope, the presence of memory mapped registers in the programmable fabric allows for data to travel up from the HRIM to the I2M and then up to the FIM through memory mapped registers. Only valid data as determined by the HRIM and the I2M is forwarded into the FIM. Figure 4-8 shows the flow of data all the way up from the communication bus to the highest

level of the HECAD system.

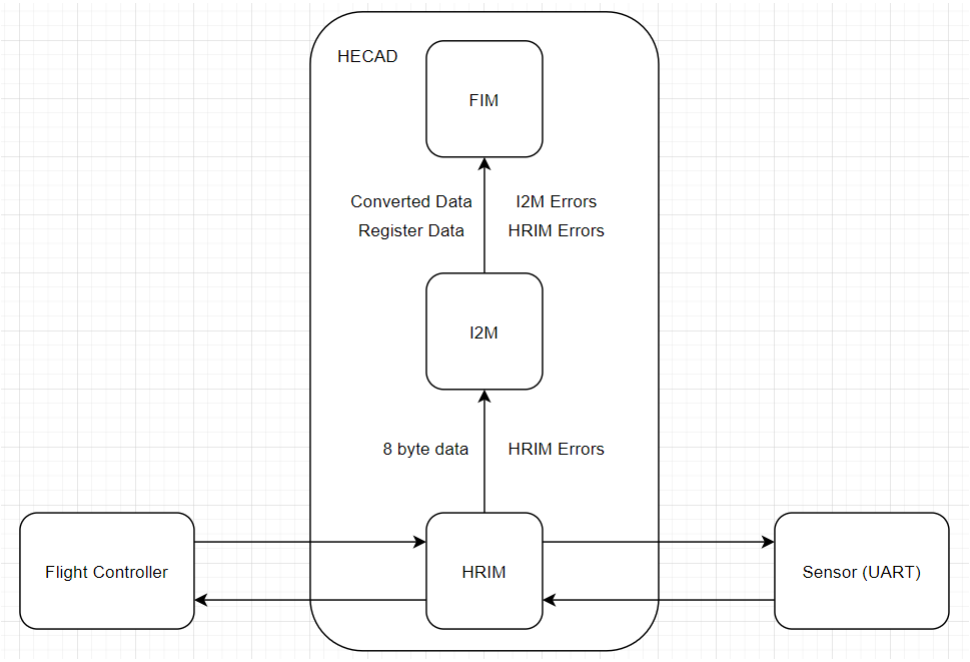

Figure 4-8: Information and Error flow within HECAD

4.7 Data Management

The HRIM does not consist of many data management layers, since it is simply checking the operation of the communication bus itself. While the HRIM can be used for many other checks besides bus operation verification, this version only checks for bus operation. As a result, there are generally no data to be managed here, except an 8 bit shift register and an 8 bit slave register for comparison. Here, both HRIM modules package incoming data into this shift register to be output to the FIM. For the HRIM in the I2C bus, the slave address register is used to verify the slave devices the master is attempting to communicate with is valid.

The data within the I2M is more in depth and detailed, and generally deals with the internal operations. For sensors such as the MS5611, there are additional operations that need to be completed before the data becomes relevant. For example, the MS5611 first reports calibration data such as pressure and temperature sensitivity,

temperature and pressure offset, and reference temperature. From here, the digital temperature and pressure are reported, and the actual temperature is calculated by the I2M using the coefficients and digital temperature and pressure. Then, the temperature compensated pressure is calculated. Within the programmable fabric, each of the values are stored in their respective and recommended data types. Using the I2M I2C parser, data is stored when read in accordance to the register addresses as described.

Within the FIM, data is stored as separate global registers. A specific register is used for each field of data that is extracted from the I2M that is important to determine a higher scale cyber-attack. For the MS5611 barometer, this will be the pressure and temperature. For the GPS module, this will be timestamp, latitude, longitude, and the altitude. For VACS, there are several fields of data extracted as described in table 4.1.

VACS	**GPS**	**MS5611**
Sync1	Timestamp	Digital Temperature
Sync2	Latitude	Digital Pressure
Source Addr	Longitude	Calibration 1
Destination Addr	Altitude	Calibration 2
Low MSG ID		Calibration 3
High MSG ID		Calibration 4
Low Data ID		Calibration 5
High Data ID		Calibration 6
Low Data Length		Actual Temperature
High Data Length		Actual Pressure
Data Payload		
Checksum A		
Checksum B		

Table 4.1: List of data fields - FIM

4.8 Error Management

In order to be able to manage data efficiently without locking up the HECAD system, there are multiple ways to handle error. As demonstrated in the results

section, there are instances where false data is injected into the system and HECAD locks up into a failed state, waiting for the user to manually clear or reset the errors. Implementing a lockup allows for users to be more directly involved with the HECAD system to monitor faults and errors, but in exchange prevents the detection of faults and errors that can arise during downtime. An initial startup of the flight control system may have many configurations unfamiliar to HECAD unintentionally, and can cause HECAD to trip to an error state. As a result, HECAD is correctly picking up unfamiliar operations at startup, but prevents any other detections during actual operation. This allows for an evaluation of the FCS startup code.

An alternative would be to allow the HECAD system to return to an idle state update detecting the fault or an error. An error strobe would be used to notify any of the higher up levels of HECAD along with any other debugging information (slave timeout, data error ranges, data flatlining, etc). This allows for tracing of system error during flight if the FCS ever crashes due to an unresponsive device or component on-board.

By providing a way to inform the other levels of a fault or an error, the HECAD system gives a channel of communication for HECAD to act as a secondary device in future works. If HECAD is to be able to fully replace the flight control system or provide a way to mitigate certain sensors and actuators, at the very minimum it needs to be able to be able to identify the source of an issue and the type of issue to take the appropriate action.

4.9 Mitigation Techniques

The detection of the existence of a cyber-attack allows for the intervention of a human operator for resolution. However, because the HECAD system is intended to run in real-time, requiring a user intervention can cause the HECAD system to fail to detect more important security incidents, especially if the first incident detected is minor. A modification to the error logic can be introduced to allow for intervention if the severity of the error is high enough. From here, the HECAD system does not

need to lock up, but rather the error logic can serve as a driver or enabler for the control logic for mitigation techniques. If an error has been detected, several steps can be taken as described below.

A safety switch component was imported in order to be able to allow a safety pilot to take-over manual operation of the aircraft in the event that the automated code on-board the aircraft fails or in an emergency where the pilot would need to take over. Because the flight controller is modularized to work with any air frame, the set of codes used for manual take-over will vary from frame to frame. For example, the operation requirements for a tricopter will be significantly different from a quadcopter which will be significantly different from a fixed wing vehicle. In all three frames, the same flight controller can be used, but a different set of autopilot codes can be used on the flight controller. The safety switch receives SBUS signals from the receiver and the PWM signals are generated and forwarded to the flight controller for processing. In the event anything in the autopilot software fails, or if there is any reason for the autopilot to override the autopilot control, the safety switch will detect that the manual mode switch has been triggered and will send the PWM signals directly from the safety pilot operator to the actuators, bypassing the flight controller itself. As the name suggests, the safety switch provides a mechanism for the safety pilot to control the aircraft in the case the autopilot software fails.

The presence of the safety switch provides a mitigation technique for a malfunction in the flight controller. However, ground pilots would only know there is an issue based on a visual judgement on the aircraft. In other words, ground pilots would flip the manual switch if it is determined to be in a failed state (when it does not appear to be performing required actions for mission, or if it is failing to maintaining a flight state). Because it is designed to be separate system from the flight controller, the risk of the flight controller being compromised by third party vendors is minimal. There is no way for the flight controller to communicate with the safety switch component. Furthermore, the safety switch is maintained by developers, so the risk of infection by a external attackers are also minimized. Nevertheless, additional monitoring of the PWM and SBUS signals can be performed between the safety switch and the

actuators, to ensure that the signals fall within specifications. This ensures that the HRIM can verify that the PWM and SBUS signals measured are expected (never 100 percent or 0 percent duty cycle) and the FIM can detect sudden changes to the actuators.

For hardware faults and cyber attacks detected on sensors, a method of mitigation for sensor attacks is to either swap out or power cycle the modules themselves. For example, a GPS module can experience a spoofing attack as described above. Depending on the GPS module, information regarding the signal strength and positional data may be extracted. A sudden change in the signal strength to the satellites or a change in the positional data may indicate an attack or fault [19]. If implemented, both indicators of faults can be caught by either the I2M or FIM and the operator can be notified, and the error signal can trip a power signal or be used to trigger reconfiguration logic.

While a direct GPS spoofing on a unmanned system is difficult, the injection of malicious firmware can provide the same effect of GPS spoofing attacks through supply chain attacks. If the malicious firmware awakes during runtime and begins to provide false data, then the unmanned system may no longer have a point of reference for its position. With a presence of multiple GPS components from different manufacturers, the prevention of a supply-chain attack may be picked up in the FIM in exchange for placing more GPS modules on-board. This can be done by cross referencing multiple unique GPS devices to ensure they agree with each other to a certain degree. Another approach can be done by cross referencing the other sensors utilized. By checking the changes in nearby sensors in the sensor network on-board, the FIM can potentially pick up a mismatch in the data reported by other sensors reference. For example, the FIM may receive false data reported by the compromised GPS that the unmanned system has recently changed direction. If the movement of the aircraft is independent on the IMU and barometer sensors, then checking the recent events on the IMU and the barometer sensors will determine that there has not been a change in the speed and direction of the aircraft as suggested by the GPS module. However, this assumption is valid only in the case where the UAS does not

utilize only GPS data to influence its direction and orientation. In the case that it does, then the barometer and the IMU will likely agree with the recent events as reported by the GPS, because the changing of orientation of the aircraft is based on the faulty GPS data.

Chapter 5

Injection and Verification

5.1 Injection Process

The process of injection into the different levels of the HECAD system involves separating the monitors out into their own workspace. When the inputs and outputs are defined for each level, it is easy to introduce a set of inputs and the expected outputs. For the UART HRIM, several VACS packets were sent at different configurations, including different baud rates, incorrect parity, and different data bit sizes. For the I2C HRIM, the SCL lines were manipulated in many different ways to see if the HRIM can detect a non-responsive data line.

In the I2M for the UART, a VACS packet parser was developed to ensure that received packets are uniform and fully packed. This is done by calculating the checksum in real-time. Similarly, the I2M for the I2C consisted of several validity, range and flatlining checks. This includes valid command checking, calibration checking, out of range checking, and pressure and temperature delta checking. These injections are fully examined in section 6. In Figure 5-1, an example of injection when the HRIM and I2M have been isolated is shown. At the HRIM, direct manipulations is done on the communication lines (UART configurations, I2C SCL line manipulations). Separating the outputs from the HRIM to the I2M, the I2M is injected with the different types of data that can be returned from the device. Here, the assumption is made that the data passes the HRIM configurations.

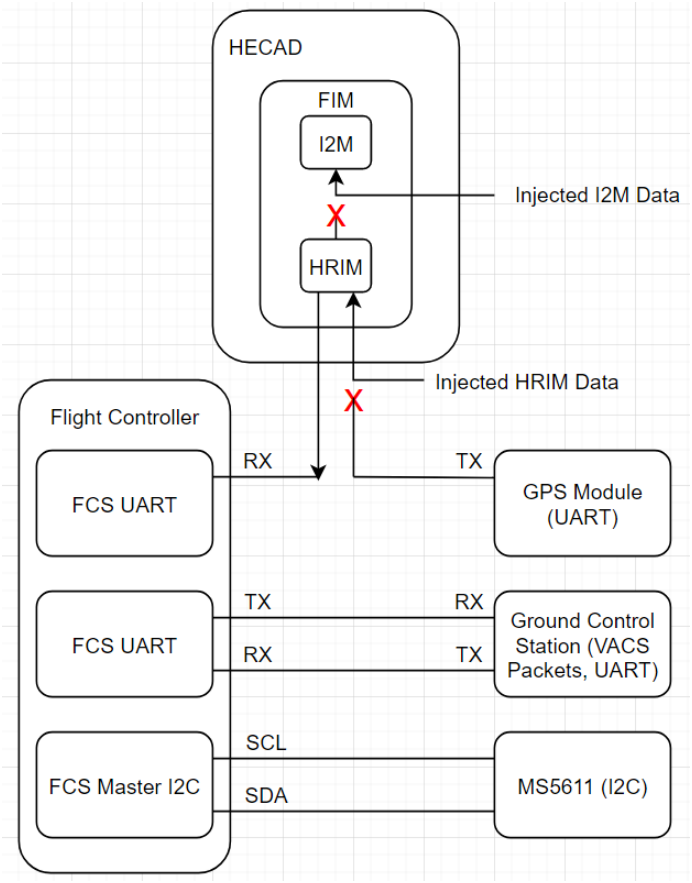

Figure 5-1: Injection Testbed for the GPS module

5.2 Hardware Level Injection

The testing of the implemented HECAD system involves injection data into two communication buses and four different components: the UART bus, the I2C bus, the two HRIM modules, and the two I2M modules. Each component is tested individually using a variety of methods. At the hardware level, data injected into the system is transmitted at different configurations using an FTDI chip. Because the UART configuration is expected to be at a transmission speed of 115200, no parity, and one stop bit, data is transmitted otherwise to check the detection of the discrepancies. Table 5.1 describes the injections into the HRIM module for UART devices.

Type	Target Field	Coverage
Configuration	Baud Rate	Incorrect Baud Rate
Configuration	Data Bits	Incorrect Data Lengths
Configuration	Parity Even	Parity Existence
Configuration	Parity Odd	Parity Existence
Configuration	Parity Space	Parity Existence

Table 5.1: HRIM Injections for UART Bus

For the I2C, a similar approach is used. The two wire interface consists of the SCL and SDA line used for bidirectional communication. First, a module to manipulate the SCL line is used. This module generates SCL lines at different frequencies, different duty cycles, lines that have been held high or low after a start condition and during a transmission. A similar approach is used to generate an improper SDA line. However, because the SDA generally represents data, the unexpected behavior of the SDA line (no slave or master acknowledgement) will be caught by the state machine parsing data coming into the HECAD system. Therefore, it makes sense for the data that is sent on the SDA line to be validated by the I2M. Table 5.2 describes the injections in the HRIM module for I2C devices.

Type	Target Field	Coverage
SCL Responsiveness	SCL activity	SCL held high after start
SCL Responsiveness	SCL activity	SCL held high during transmission
SCL Responsiveness	SCL activity	SCL held low after start
SCL Responsiveness	SCL activity	SCL held low during transmission
SCL Frequency	SCL speed	Frequency within a few percents of expected
SCL Duty Cycle	SCL behavior	Reasonable Duty cycle

Table 5.2: HRIM Injections for I2C Bus

5.3 Information Level Injection

Separating the I2M from the HRIM, each device is tested independently. This allows for testing based on the known inputs and outputs as well as the correct type of information. For the VACS transmitter on the UART bus, VACS will be injected,

with both correct and incorrect checksum calculations, as well as correct calculations but manipulated packets. This is to show that attackers may be able to change the information, but fail to properly change the verification bytes. Several changes are made, including the known fields and respective data lengths and types. Table 5.3 describes the injections at the I2M level for the UART VACS device.

Type	Target Field	Coverage
Checksum	Checksum A	Checksum incorrect
Checksum	Checksum B	Checksum incorrect
Data Change	Message Fields	Changed Data
Data Change	Message Fields	Dropped Data

Table 5.3: I2M Injections for VACS Device

For the MS5611 I2M device, injections involving every single field are considered. The processing of setting up and reading from the sensors involves reading in the calibration data, reading in digital temperature and pressure, and performing a calculation to translate it into real, interpretable data. First, the injection of calibration data is performed. There are two types of calibration injections: incorrect calibration data injected during the first read by the master device, and correct calibration data during the first read by the master device and incorrect calibration data injected during the second read by the master device. This covers checking for initial calibration data against known values and checking for changes in the calibration at any point during run time. Next, the injection of commands is performed. Any commands that are considered incorrect or unknown are ignored, and the HRIM throws an error. An injection is performed before and after a valid command has been received. Finally, changes in temperature and pressure injections are performed. Given that pressure generally is adjusted for temperature, an injected value that changes the temperature will also change the pressure. The opposite is not true, injections involving only pressure and only temperature are performed, and discrepancies large enough to be detected are shown. Table 5.4 describes the injections at the I2M level for the UART VACS device.

Type	Target Field	Coverage
Calibration	A1 Coefficient, First Read	Unexpected Value
Calibration	A2 Coefficient, First Read	Unexpected Value
Calibration	A3 Coefficient, First Read	Unexpected Value
Calibration	A4 Coefficient, First Read	Unexpected Value
Calibration	A5 Coefficient, First Read	Unexpected Value
Calibration	A6 Coefficient, First Read	Unexpected Value
Calibration	A1 Coefficient, Reread	Changed Value
Calibration	A2 Coefficient, Reread	Changed Value
Calibration	A3 Coefficient, Reread	Changed Value
Calibration	A4 Coefficient, Reread	Changed Value
Calibration	A5 Coefficient, Reread	Changed Value
Calibration	A6 Coefficient, Reread	Changed Value
Commands	All Commands	Invalid command
Pressure	Digital Pressure	Large Change in Pressure
Pressure	Digital Pressure	Large Change in Pressure
Temperature	Digital Temperature	Large Change in Temperature
Temperature	Digital Temperature	Large Change in Temperature

Table 5.4: I2M Injections for MS5611

Chapter 6

Preliminary Results and Verification

The goal of the HECAD system is to be able to identify any existing and incoming hardware faults and incoming errors as a result of either intended cyber-attacks or faulty hardware induced at any point of the development and distribution process and communicate this information with an operator. As such, faulty data is injected into the system at both the hardware and information level. In the previous sections, types of attacks are injected and the results are explored here.

6.1 HRIM

6.1.1 UART HRIM

The UART HRIM primarily detects changes in the UART configuration as well as changes in the information level on two devices: a GPS module and a VACS device. The configuration for both devices stays the same: transmission at 115200, one stop bit, and no parity bit. Correct GPS and VACS data is injected into the HRIM module at several different speeds, and at different configurations as shown below. In figure 6-1, data is sent into the module at a baud rate of 9600. Since this is such a low baud rate, the figure shows a couple of transmissions before the error is detected. This is because the low baud rate and the high oversampling rate catch a large part of the start bit as the transmission bits. After the first bit is actually transmitted, then the

error is caught.

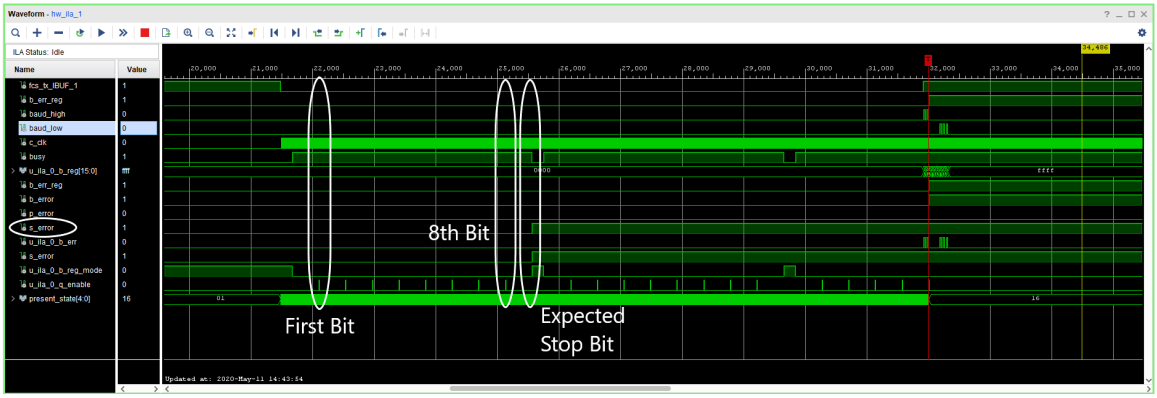

Figure 6-1: UART Injection at 9600 baud

In figures 6-2 and 6-3, data is sent into the module at a baud rate of 19200 and 57600 respectively. This is demonstrated to throw an error if the baud rate is a little bit lower than the intended frequency. In figure 6-2, the start of the transmission starts at approximately 29300, then after the first bit change, the error is caught. Similarly, figure 6-3 demonstrates the same effect, where the start of a transmission is at approximately 27500, and at approximately 31400 is the expected stop bit. In both figures, the baud is so slow that the data transmitted is caught in between the windows of the oversampling clock.

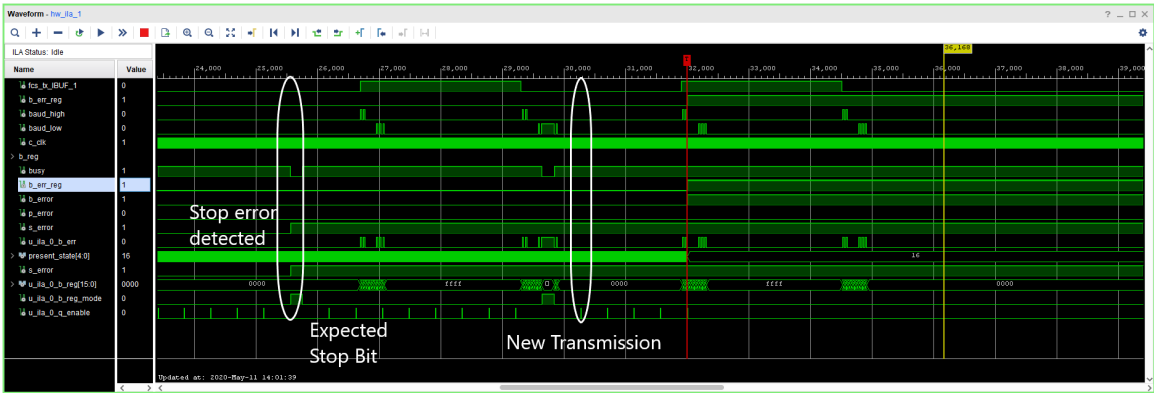

Figure 6-2: UART Injection at 19200 baud

The same experiments are used in figures 6-4 and 6-5 to demonstrate the same functionality for baud rates at a higher rate than intended. In these cases, data is sent into the module at a baud rate of 230000 and 500000 respectively. As expected,

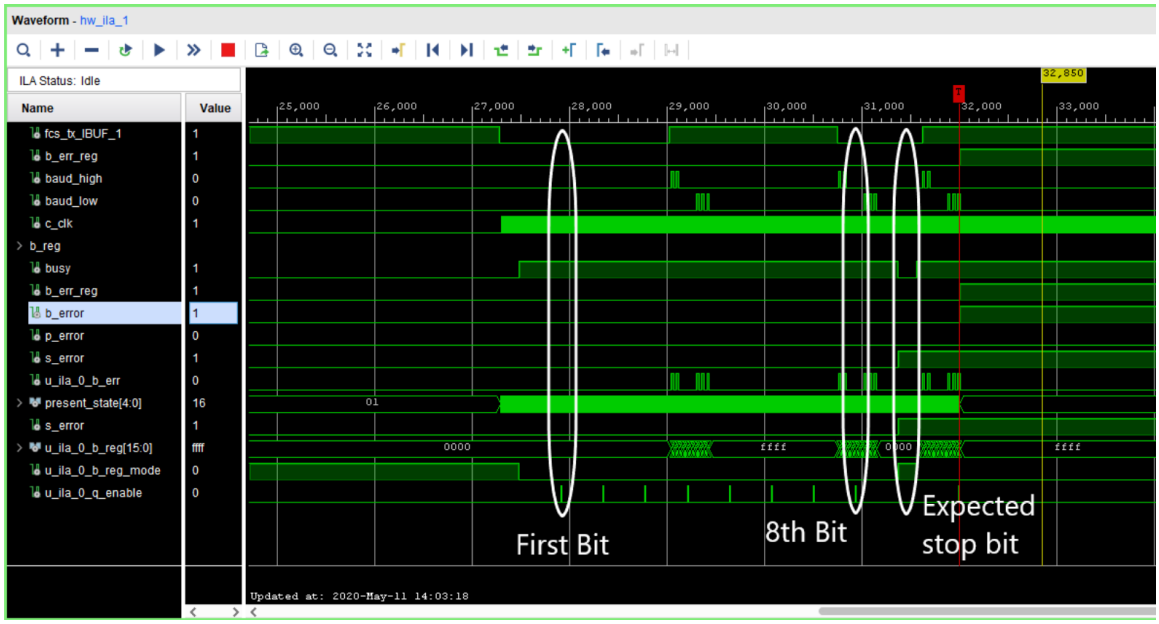

Figure 6-3: UART Injection at 57600 baud

the error is caught after a brief moment in both situations. This is because the point at which the bit has changed is not exactly in the middle of a 16-bit register generated by the oversampled clock.

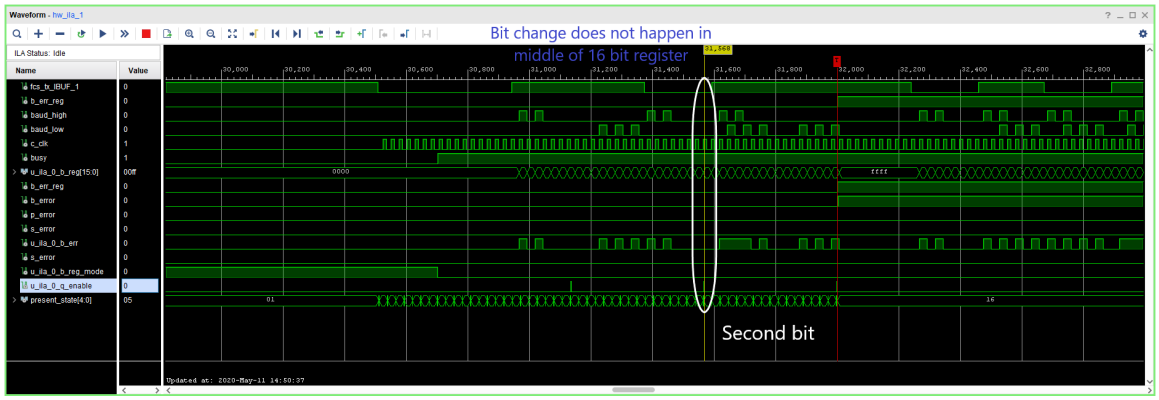

Figure 6-4: UART Injection at 230400 baud

In addition to the change in the baud rate, a different configuration for the parity bits was used. Since the state machine checks for a non-existing parity bit to be transmitted, the state machine throws an error if the parity bit is a '0', regardless if it is an even parity or an odd parity. A transmission of a '1' parity bit is considered a correct transmission since it is detected as the stop bit. A similar case appears if the

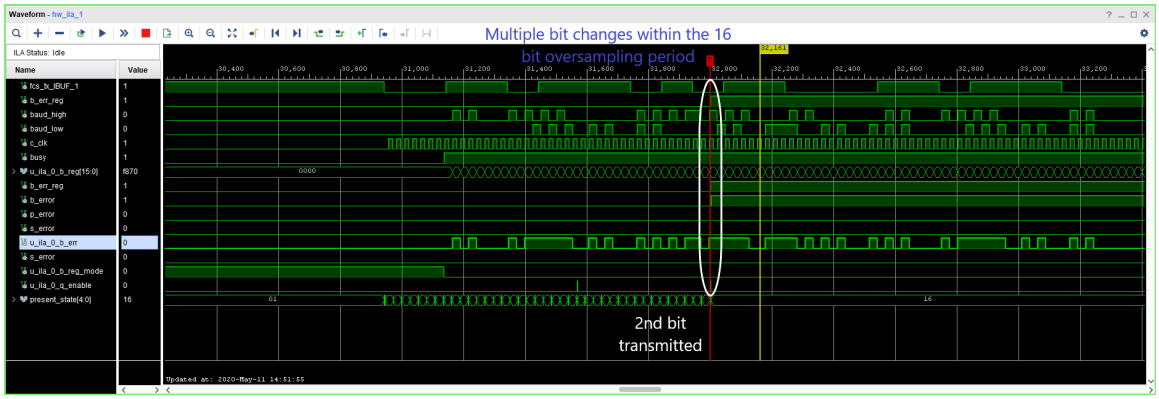

Figure 6-5: UART Injection at 500000 baud

the parity bit is a mark bit (where the bit is always a '1' and therefore is detected as a stop bit) or if the parity bit is a space bit (where the bit is always a '0'). In figure 6-6, at approximately 21910, the UART HRIM expects a '1' stop bit. The absence of the stop bit throws the stop error. Figures 6-7 and 6-8 show a similar case, where at the end of the transmission, a '1' stop bit is desired but a '0' bit is returned. Both cases are caught by the UART HRIM.

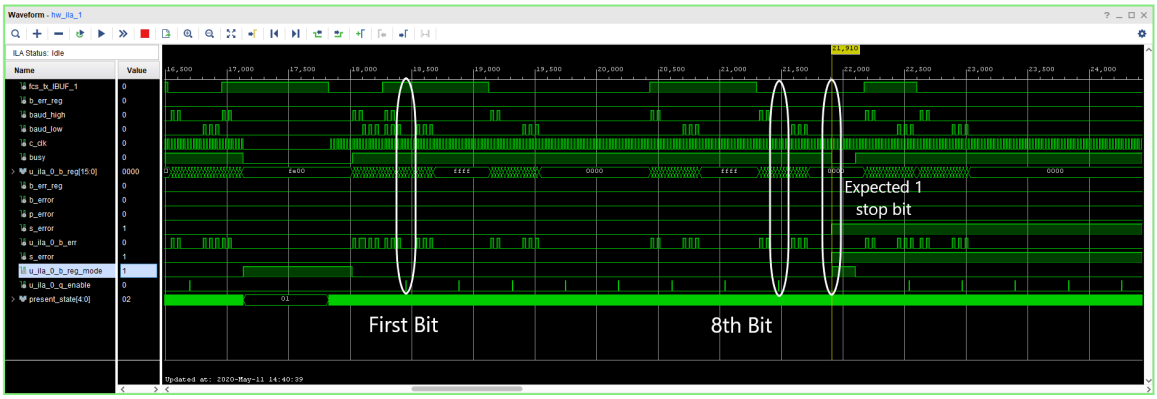

Figure 6-6: UART GPS with Even Parity Bit

The final injection involving the UART configuration is the stop bits. The non-existence of a stop bit is considered an error. However, the issue with injection is that there is no module that can transmit without a stop bit. The FTDI and Putty configuration does not allow the transmission of a byte without a stop bit. However, the presence of a '0' bit when a '1' bit is expected in the previous examples shows that the state machine can detect it as an error. In order to work with UART, the

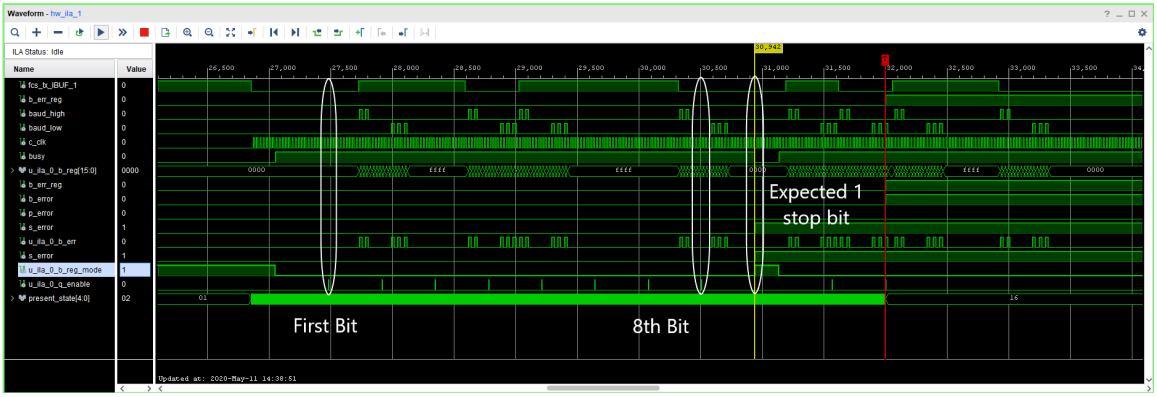

Figure 6-7: UART GPS with Odd Parity Bit

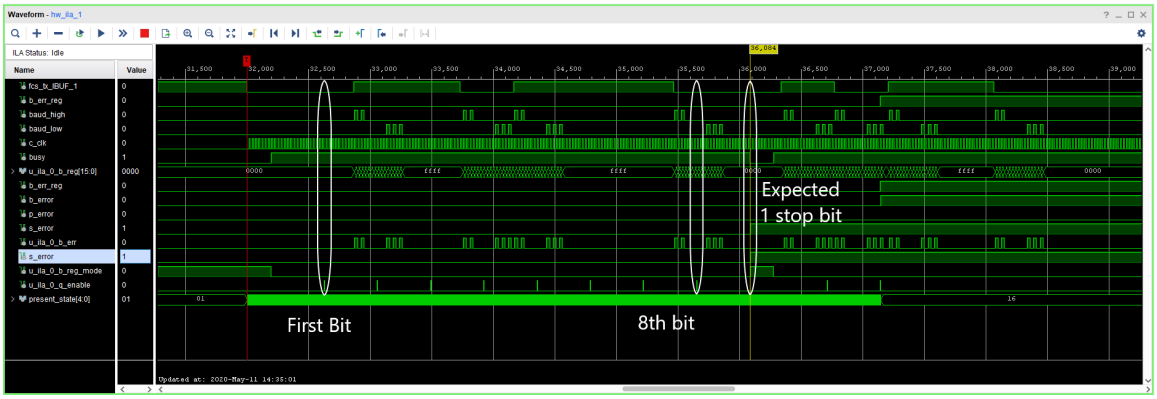

Figure 6-8: UART GPS with Space Parity Bit

next transmission requires the line to be pulled up high in order to have a start bit, so eventually the line will represent a stop bit, only at the wrong time, indicating a configuration error. The detection of two stop bits will not represent an error, since HRIM is simply looking for the first stop bit it detects. The rest is considered part of the idling period. Additionally, the overhead induced from the extra bit will not be significant enough to cause a framing error or baud error. An example of this is shown in figure 6-9.

In the previous section, it was noted that when the GPS module is initially configured to its factory settings out of the box, the flight controller runs a setup procedure where the flight controller attempts to communicate with the GPS module at different baud rates until a common baud rate is achieved. From here, the flight controller sends the commands to reconfigure the GPS module to the desired configuration. The

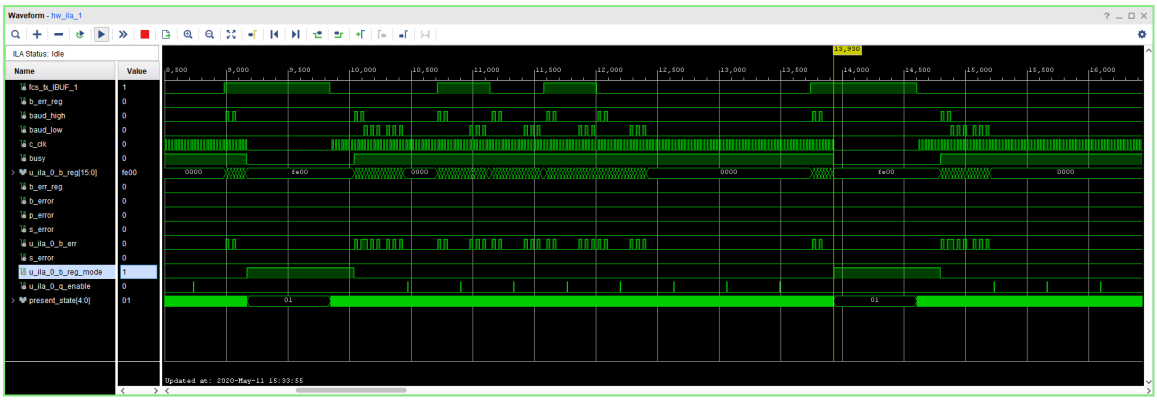

Figure 6-9: UART GPS with Two Stop Bits

issue with this setup is, when the flight controller boots up, the HECAD system also boots up, ideally at the same time. If the setup code attempts to communicate with flight controller, then the HECAD system will catch the error in the initial communication configuration. This will cause state machine to run out of sync. The state machine will have no way to distinguish the true start and stop points from the new configuration. To solve this problem, the flight controller and the GPS module should be configured to both communicate at the right configuration upon startup.

6.1.2 I2C HRIM

In the HRIM injection for the I2C device, the address validation still follows. Furthermore, the more thorough checks for the I2C hardware bus involve the SCL line and the SDA line to ensure that there is activity as well as response and frequency of the response on the line. Upon detecting a start condition on the line, if the SCL line is held low either after the start condition or during transmission, then the detector will throw an error. Similarly, if the SCL line responds but gets stuck during the initial transmission or during a transmission, then the detector will also throw an error. Figures 6-10 and 6-11 show an injected and modified SCL line that is held low after the start condition and during a transmission. After a approximately 75 percent of the SCL cycle has passed and no activity has been detected on the SCL line, then the I2C HRIM throws an error. In figure 6-11, the SCL line is broken during a transmission to show that the error can be caught outside of the start condition

as well. The number that is used as a threshold (75) can be modified so that it can catch the error after a longer or shorter amount of time, depending on the expected SCL bus speed.

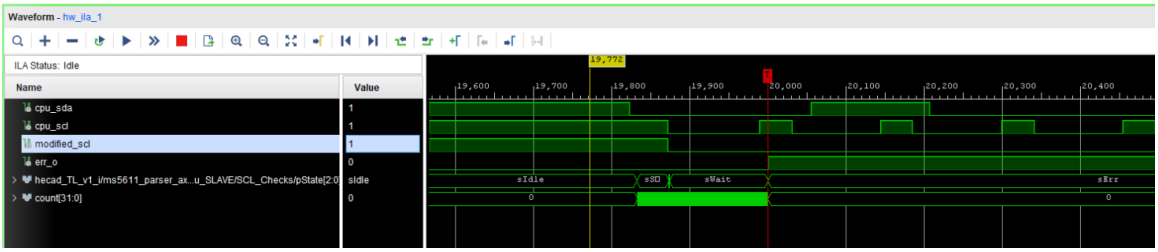

Figure 6-10: I2C SCL Held Low after Start condition

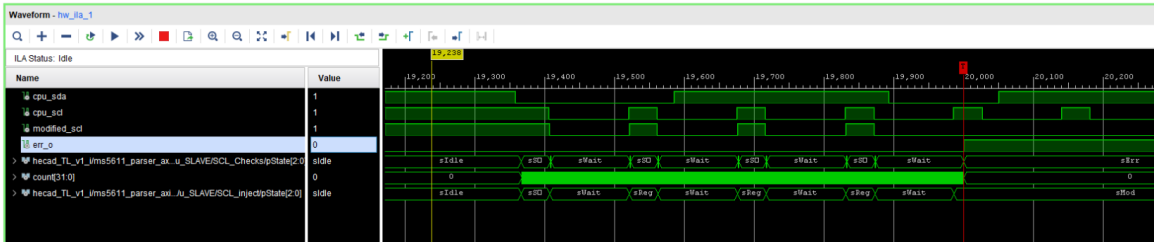

Figure 6-11: I2C SCL Held Low during transmission

Similarly, the same concept is used for the I2C lines being held high after the start or during the transmission. Figures 6-12 and 6-13 show the detection of the SCL lines held high after the start condition and during transmission.

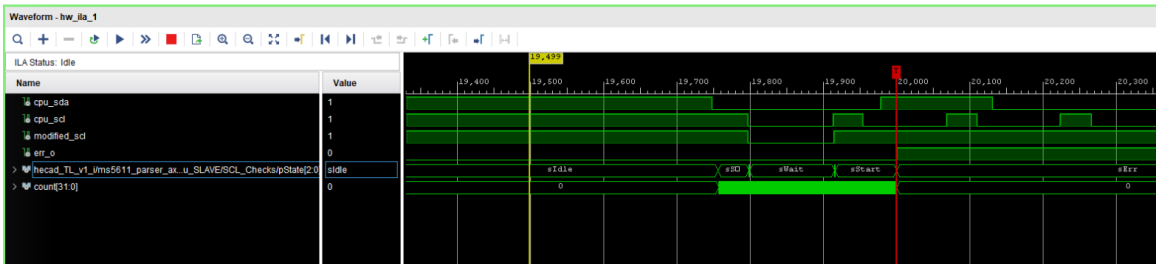

Figure 6-12: I2C SCL Held High after Start condition

Checking the frequency of the SCL line is the next injection. If the frequency of the SCL line is known, then checking the time between each rising and falling edge of the SCL line will allow for checking of the frequency. If the frequency is off by more than 5 percent, then the detector will also throw an error. Several injections

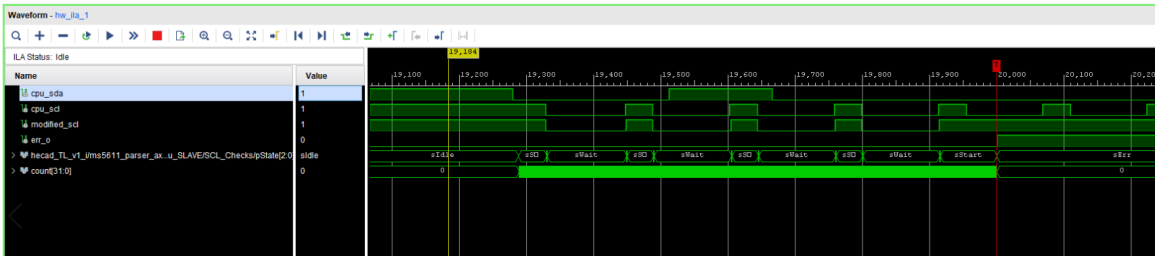

Figure 6-13: I2C SCL Held High during transmission

are done to show that the HRIM can detect a change in frequency. For figure 6-14, the frequency of the SCL line is modified to be 3.3 percent higher than the expected frequency, running at 347222 Hz, or approximately 347 kHz. For figure 6-15, the I2C SCL bus is modified to run at 250 kHz. For figure 6-16, the I2C SCL bus is modified to run at 2 Mhz. This method of checking is a little more difficult, especially when working with the actual speed of the SCL bus versus the intended speed of the bus. During the experimentation, several attempts to measure the speed of the bus varied between 312 kHz and 340 kHz. While this shows a reasonable drift on the bus, this may cause a false positive in identifying malfunction incidents on the I2C bus. As a result, the threshold in determining if there is a fault or an attack should be increased to a larger number.

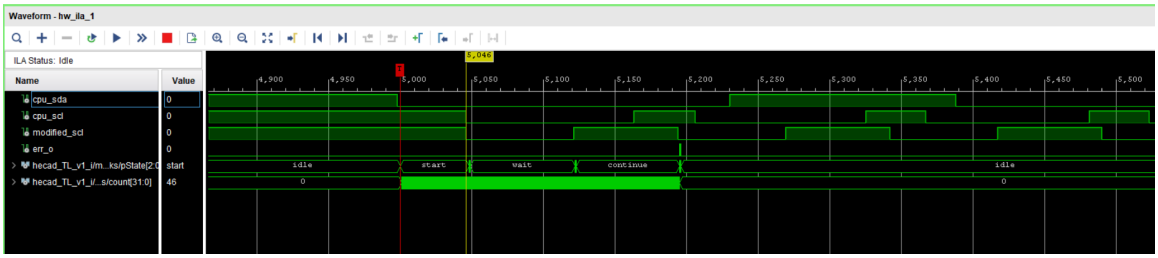

Figure 6-14: I2C Incorrect Frequency - 347222 Hz, +3.3 %

The HRIM also verifies the SCL duty cycle. If the frequency of the SCL line is known, then generally so is the duty cycle. The injector used previously is modified to output an SCL line with a different duty cycle that is unreasonable. In the injector, the modified SCL line that is injected in the HRIM has a duty cycle of 80 percent. This is to emulate an incorrect SCL line duty cycle, but not necessarily a non-responding I2C SCL line. Figure 6-17 shows the detection of a responding SCL at the wrong

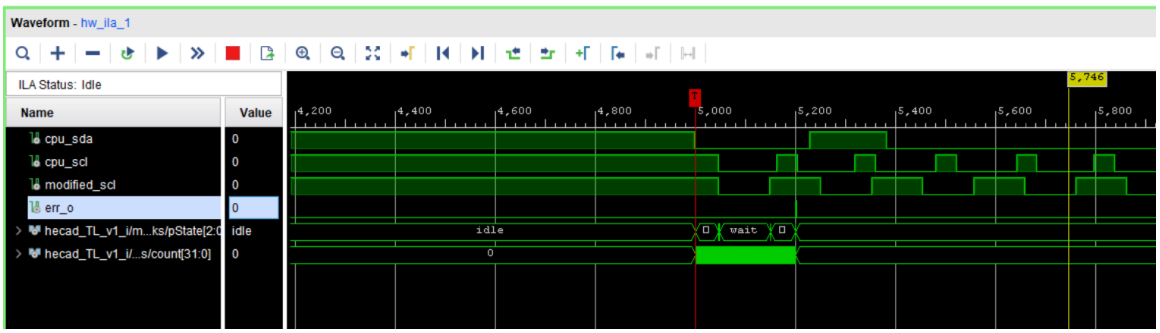

Figure 6-15: I2C Incorrect Frequency - 250 kHz, -25.6 %

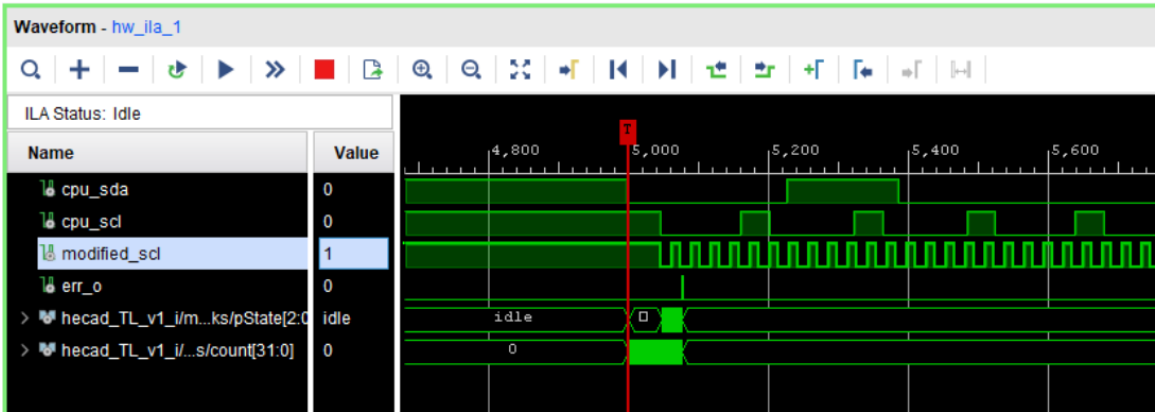

Figure 6-16: I2C Incorrect Frequency - 2 MHz, +495 %

duty cycle. Since the SCL lines driven by a master device generally do not have a 50 percent duty cycle, this check may not serve useful or important in determining existence of faults.

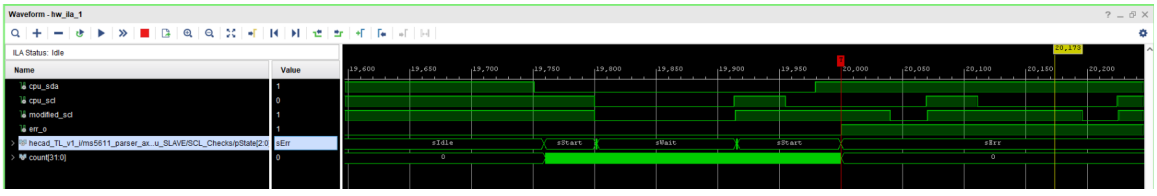

Figure 6-17: I2C Incorrect Duty Cycle

6.2 I2M

6.2.1 UART I2M

The injection at the UART I2M at the information level consists of working with changing VACS or GPS packets. To demonstrate a detection in the error of information received, the injections performed here will involve changing parts of a VACS packet, including the data section and the checksum section. Changing part of the data will result in an incorrect checksum and changing the checksum will lead to an error detection thrown by the VACS parser. The two types of injections are performed and shown here. In figures 6-18 and 6-19, a valid VACS packet is injected, but the checksum is changed. In figure 6-20, a VACS packet is injected, but parts of the packet itself are changed. In any situation, changing different parts of the VACS packet is an attempt to demonstrate the failure to receive uniform and unaltered data.

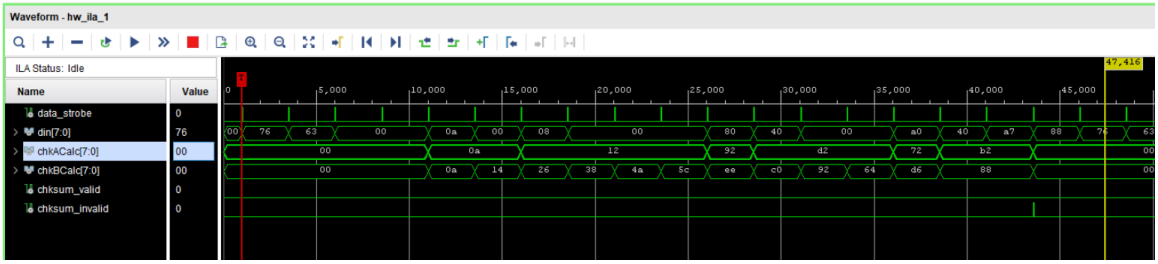

Figure 6-18: VACS Packet - Checksum A changed

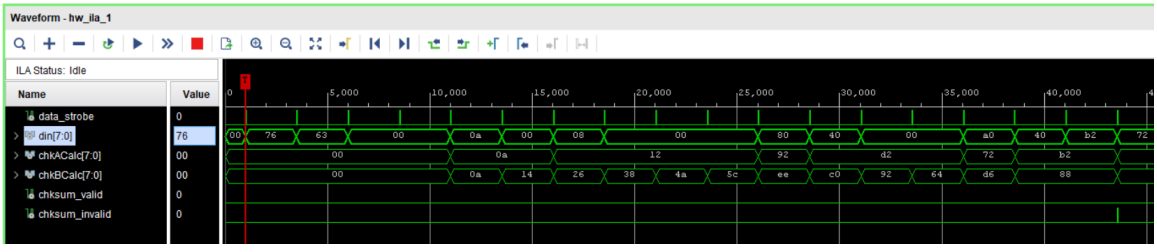

Figure 6-19: VACS Packet - Checksum B changed

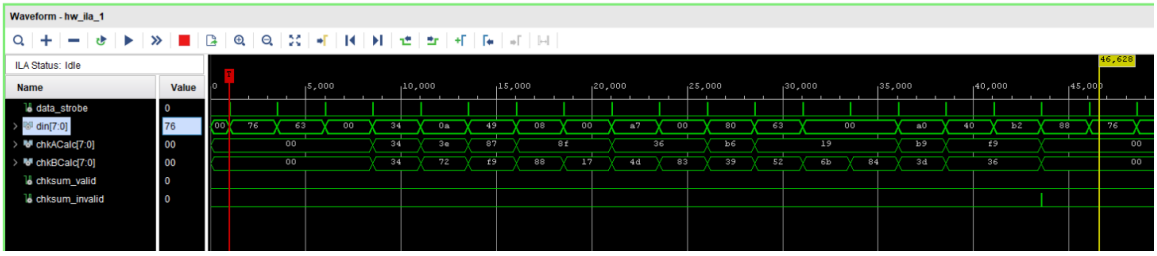

Figure 6-20: VACS Packet - Data Payload changed

6.2.2 I2C I2M

The injection at the I2C information level consists of several changes in the different registers that can be returned by the slave device. This is the more extensive injections test. First, the calibration registers are validated. Every slave device has factory burned data into the device memory, so the reading of the calibration registers should never change. As a result, data from the calibration register is read in and stored during the first iteration of the reading of specific calibration register and then checked against that stored data during the next iteration. There are a total of six different calibration registers. Figures 6-21 through 6-23 demonstrate the injection of an incorrect data value during the first iteration. This is intended to check the data read from the calibration registers. It is known what the calibration registers should be.

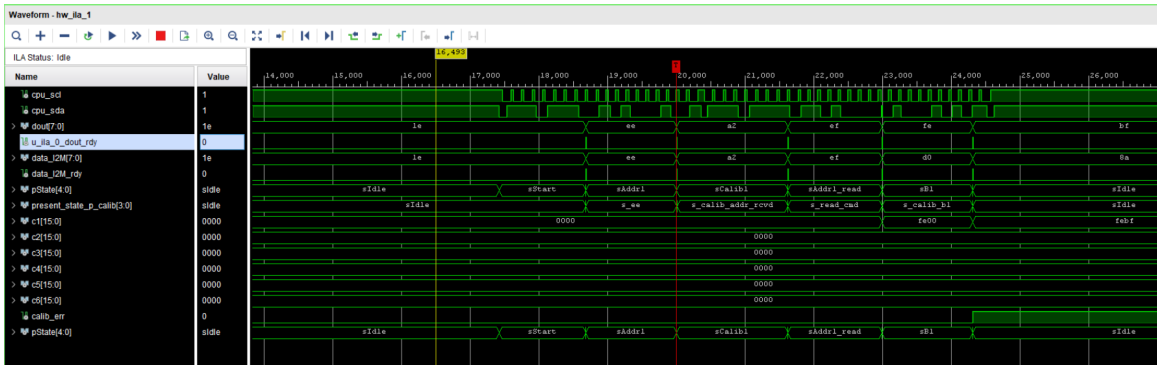

Figure 6-21: I2C C1 Calibration Injection

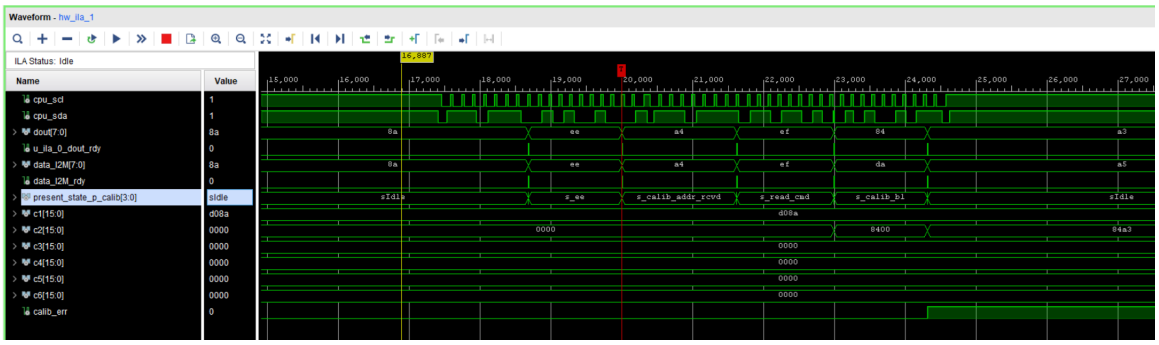

Figure 6-22: I2C C2 Calibration Injection

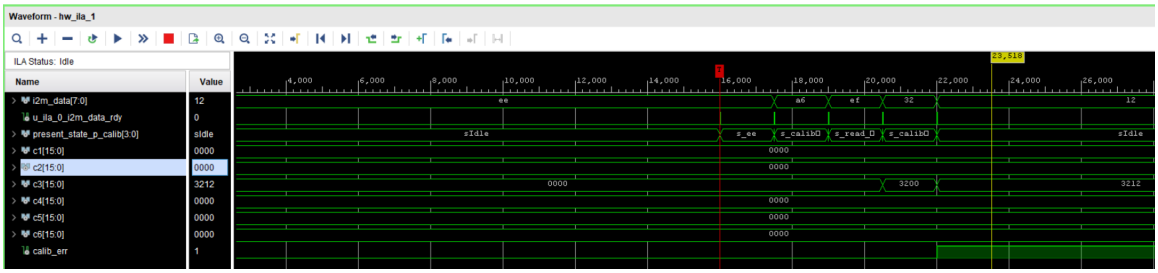

Figure 6-23: I2C C3 Calibration Injection

In the Aries flight controller, the autopilot code is set up to read the calibration registers once, and then store those values in memory. In the case where data returned is different, this can cause issues in calculations for the values that are calculated using the calibration data. In order to demonstrate a change in the calibration data, two iterations of reading calibration registers are performed: once with one set of data, and again with a false set of data. Figures 6-24 through 6-26 show the injection of the correct calibration value during the first iteration, and then a different calibration value during a second iteration.

Figure 6-24: I2C C1 Calibration Change

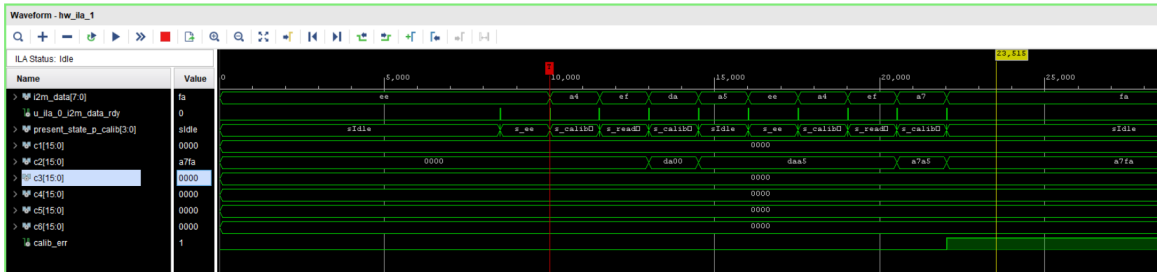

Figure 6-25: I2C C2 Calibration Change

Figure 6-26: I2C C3 Calibration Change

In addition to checking the calibration values in both initial setup and during program operation, every command initiated and transmitted with the intended slave address is checked against known commands. Given the datasheet for the MS5611, there are a total of 14 available commands, eight of which are not used. This is summarized in the previous section describing injection and validation techniques. Figures 6-27 through 6-29 indicate several attempts to send invalid commands to the correct slave address.

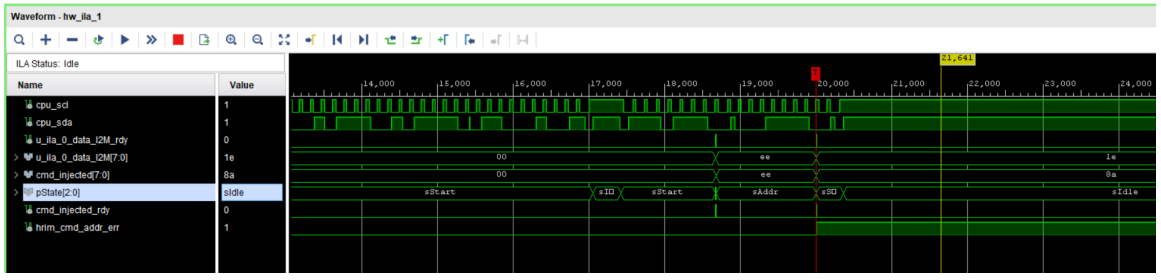

Figure 6-27: I2C Invalid Command Injection 1

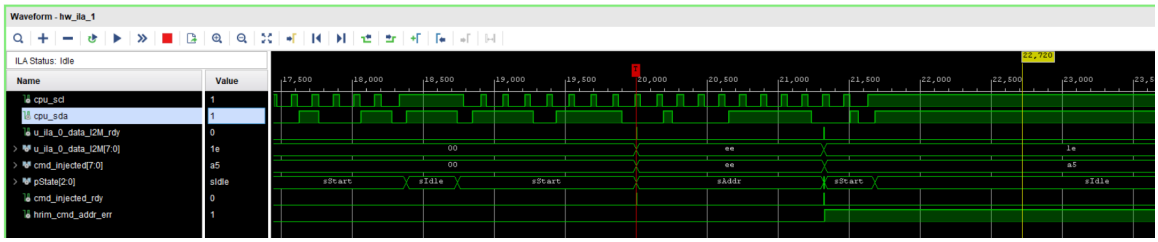

Figure 6-28: I2C Invalid Command Injection 2

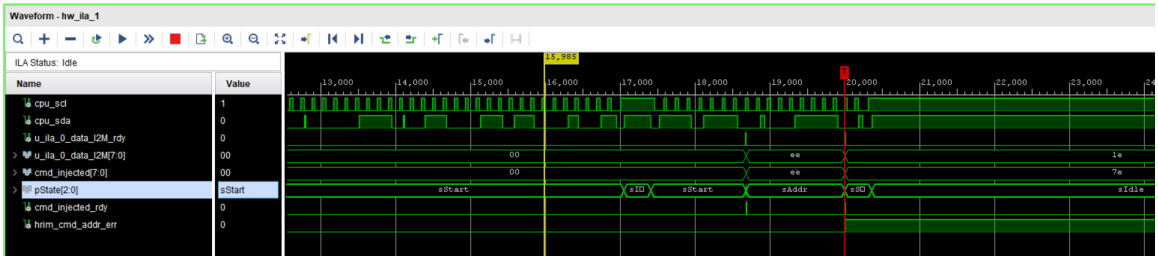

Figure 6-29: I2C Invalid Command Injection 3

At the final injection step at the information level, a change in the pressure and or temperature is injected. Since the pressure is temperature compensated and depends on the temperature, a change in the temperature warrants a change in the pressure by a large amount, but the opposite may not be a true. A change in the pressure alone will not affect a change in temperature. This is demonstrated in Figures 6-30 and 6-31.

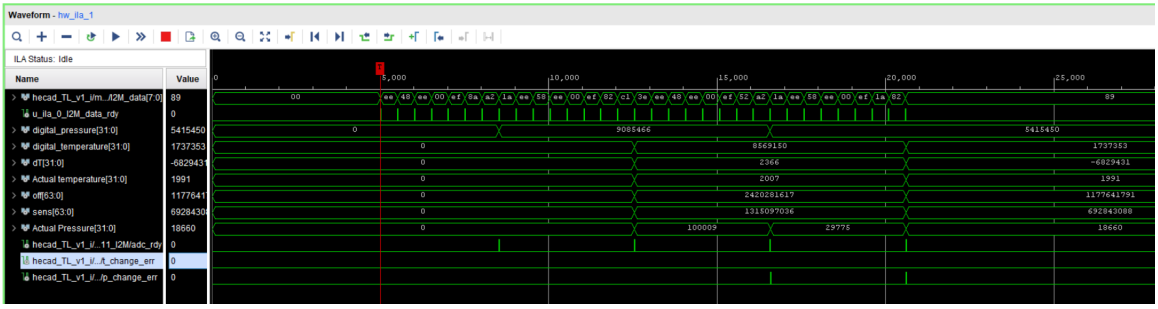

Figure 6-30: I2C Pressure Injection 1

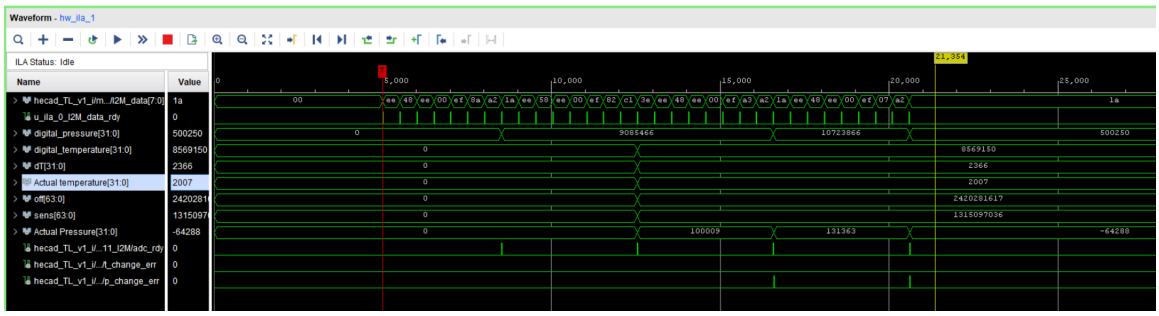

Figure 6-31: I2C Pressure Injection 2

To demonstrate that changing the temperature also changes pressure, figures 6-32 through 6-34 represent a change in pressure, a change in only temperature, or a change in both that is significant enough to warrant an error alert.

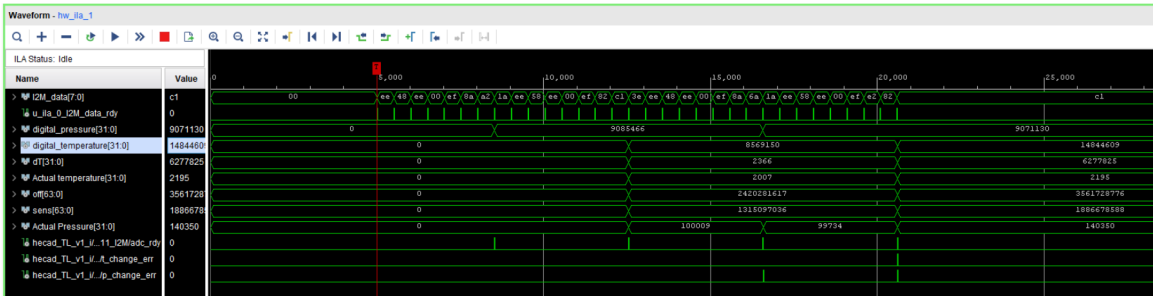

Figure 6-32: I2C Pressure Injection

Figure 6-33: I2C Temperature Injection

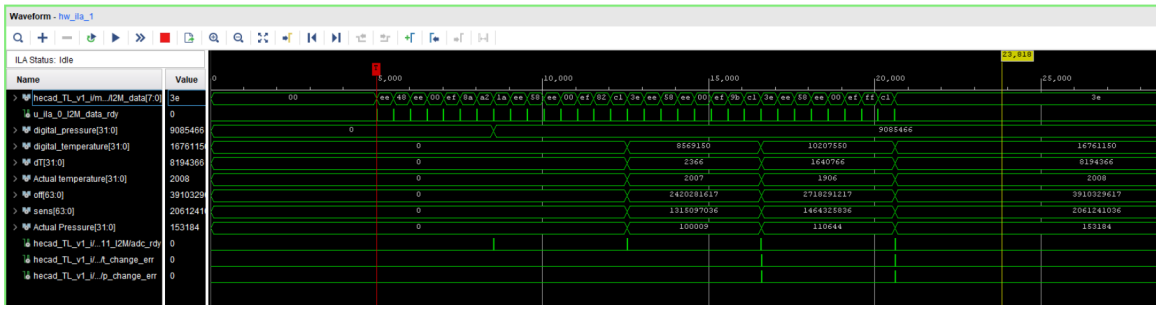

Figure 6-34: I2C Pressure and Temperature Injection

It is important to know that there are some assumptions being made here regarding the parts of the data that are being manipulated. For example, the injections involving the UART I2M VACS packets changed only a part of the data, either the data payload or the checksum. Often times, a well-coordinated attack will consist of attackers changing multiple parts of the protocol, especially if their goal is to be able to inject data that is recognized as valid and reasonable. For this reason, an attack that consists of injecting a data walkoff (where data drifts off constantly) will not be detected (unless there is a large discrepancy between any two samples) at the HRIM or I2M. A higher level of processing at the FIM will catch this data.

In many of the injections performed, HECAD demonstrated success detecting abnormalities throughout the system, at both the hardware and information level. At the functional level, all of the valid data that has been received (GPS data, VACS packets, and MS5611 data) at the hardware and information level are forwarded upwards and stored into global memory. From here, the higher level processing unit will preform actions such as average tracking or a neural network based time series classification. While the different hardware and information integrity modules highlight the errors that can occur during the transmission of data, the modules can further be refined and tested to detect a more specific configuration, such as a single versus two stop bits.

Chapter 7

Conclusions and Future Works

7.1 Conclusion

Over the years as the capabilities of unmanned systems have evolved, their area of applications have expanded. As a result, an increasing number of unmanned systems have become available to the general public. There results more opportunities for adversaries to find ways to reverse engineer these unmanned systems to gain access. There is a rising importance to find ways to secure these unmanned systems against ever evolving cyber-security threats.

A hierarchical embedded cyber-attack defense system is introduced and implemented as a non-intrusive detection and mitigation mechanism to safeguard against varying hardware faults and cyber-attacks. The HECAD system consists of a multi-level hierarchical detection system implemented in a FPGA, consisting of programmable fabric and a system on a chip. Within the programmable fabric resides the hardware integrity and information integrity monitors, which is intended to safeguard against false configurations and malicious information. Data is parsed from the communication bus in the HRIM and packaged into 8 bit registers to be sent up into the I2M. Within the I2M, the data is interpreted to its context, and its respective values and ranges are verified against what is known. If the data is valid, then the data is sent up to the FIM. In the FIM, data is managed using multiple global registers.

The HECAD device is implemented to work across the UART communication

bus and the I2C communication bus. On these two buses, the flight controller communicates with the ground control station, the GPS module, the barometer, and several other devices. In this thesis, implementations for checking, injecting and validating hardware configurations and operations along with information integrity is implemented for the VACS Packets, the GPS modules, and the barometer over the UART and the I2C bus respectively. This includes correct and incorrect UART configurations (baud rate, data bits, and parity bits), I2C line behavior (SCL lines held high or low), and VACS packet integrity (verification using checksums). Once valid information has been checked and passed by both the HRIM and the I2M, the data is then passed up to the FIM for further processing.

As shown in section 6, injections across both the hardware and information portions of a sensor functionality were performed. This includes changing bus configurations, tapping into bus lines and manipulating clock and data lines, injecting manipulated data packets for custom protocols, and injecting data against known constraints provided by data sheets. The implemented HECAD system was successfully able to identify the various faults and attacks, alerting the operator along with the error information.

7.2 Future work

In addition to the existing security checks implemented in place, the integration of HECAD will pave the way for the additional security measures as well. By being able to identify the sections of the UAS that have been compromised, HECAD will have enough information to take targeted measures for specific communication buses and devices. For specific devices, a simple power cycle will be sufficient to return the device to a working state. For compromised systems, a tougher mitigation strategy would need to be implemented.

7.2.1 Power Cycling

For devices capable of being power cycled, providing control logic for HECAD to power cycle certain devices would be a form of a mitigation strategy. If a communication bus fails to respond after a reasonable amount of time, reset logic can be tied to the power transistors of the I2C devices and used to power cycle the device. This way, there is the ability to repair a broken communication link without utilizing more resources to completely swap out a device for a new one. This tackles the issue of a malfunction on a communication bus, but not necessarily the malfunction of the device itself, especially if there is active malware on the sensor.

7.2.2 Hot Swapping

The purpose of using a hierarchical detection system such as HECAD is to allow for the detection of issues on hardware information and functional level. For devices that have been compromised as a supply chain attack (intentionally or not) there may be existing conditions on multiple levels of transmission. If there are enough evidence of a sensor or component malfunction, then it gives HECAD enough grounds for it to implement hot swapping logic. However, in the case of intentional supply chain attacks, it would only prove beneficial if the device that is being swapped out comes from a different supplier and distributor, all while maintaining performance and power requirements.

The introduction of using hard swapping also gives the ability to potentially repair devices. If a device has the ability to be repaired, then the hard swapping logic can also have recovery logic tied in as well. This works especially with devices that have been affected by single-event upsets and power cycling on the spot will result in an significant downtime.

7.2.3 Intelligent Detection

The existence of the FIM and EIM is to be able to detect higher level attacks that are not easily detectable at the hardware or information level. These generally overlap

with a large portion of the supply-chain attack that manipulates attacks at a much higher level. As an alternative to algorithms at the FIM level, a more open-minded approach to detecting anomalies is to use a neural network or a convolutional neural network to identify specific attack patterns on incoming data.

Convolutional neural networks trained to identify patterns and processes perform better and more efficiently than rule based approaches such as supervised learning. The concept may be applied here as well, where patterns and noise can be used as training for the convolutional neural network. As a result, any unfamiliar behavior that is happening with the UAS (sudden changes, drifts, plummets, etc) will trigger an error. This approach works for mission-specific situations too. For waypoint-to-waypoint missions, the behaviors described apply to a convolutional neural network. Mission data fed into the network will serve as training for determining whether or not UAS behavior is correct.

While the integration of a neural network is not imperative for proper functionality, the ideal of self-awareness or reconfiguration paves the way for HECAD to act as a secondary system. If HECAD can successfully identify issues with the flight control system itself, HECAD can take over all of the subcomponents and act as a backup flight controller.

7.2.4 HECAD as Flight Controller System

The HECAD system contains programmable fabric as well as a Quad-core, Zynq-MPSoC processing system. The programmable fabric serves as hardware based verification logic to verify hardware and information level data. The A53 processor can be used to run the FIM and communicate with the EIM. A final design should have enough resources left over to run flight controller code. A part of the processor can be dedicated to running the flight controller system.

An ideal system that comes together to form a minimally intrusive HECAD system will consist of the programmable fabric used to verify data integrity at the HRIM and at the I2M. As data comes into the HRIM, data is packaged and passed to the I2M and then to the FIM. Within the FIM resides either a neural network or a simple

processor alongside a backup flight controller system.

In the near future, different and more in-depth cyber-attack detection methods and mitigation techniques will be added into the various parts of the HECAD system. Within the FIM, higher level algorithms can be introduced, or the neural network approach can be applied. The EIM will be introduced along with the single wire interface to communicate with the main HECAD system for debugging and execution analysis. If it is possible to isolate the cores and dedicate them to specific parts of an application program, then one or two cores will be dedicated to running autopilot code on the side. Several interfaces will need to be used to allow for proper routing of components to the backup system. The combination of the different levels of the HECAD system along with the backup flight control system will allow for a tightly coupled secure UAS.

REFERENCES

[1] Matthew Leccadito. "A Hierarchical Architectural Framework for Securing Unmanned Aerial Systems". In: *Theses and Dissertations* (Jan. 2017). DOI: https://doi.org/10.25772/0DK3-E418. URL: https://scholarscompass.vcu.edu/etd/5037.

[2] Samir Bouindour, Mohamad Mazen Hittawe, Sandy Mahfouz, et al. "Abnormal event detection using convolutional neural networks and 1-class SVM classifier". In: *8th International Conference on Imaging for Crime Detection and Prevention (ICDP 2017)*. Dec. 2017, pp. 1–6. DOI: 10.1049/ic.2017.0040.

[3] Alireza Abbaspour, Kang K. Yen, Shirin Noei, et al. "Detection of Fault Data Injection Attack on UAV Using Adaptive Neural Network". en. In: *Procedia Computer Science*. Complex Adaptive Systems Los Angeles, CA November 2-4, 2016 95 (Jan. 2016), pp. 193–200. ISSN: 1877-0509. DOI: 10.1016/j.procs.2016.09.312. URL: http://www.sciencedirect.com/science/article/pii/S1877050916324851 (visited on 05/20/2020).

[4] Sara Sutton, Benjamin Bond, Sementa Tahiri, et al. "Countering Malware Via Decoy Processes with Improved Resource Utilization Consistency". en. In: *2019 First IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA)*. Los Angeles, CA, USA: IEEE, Dec. 2019, pp. 110–119. ISBN: 978-1-72816-741-1. DOI: 10.1109/TPS-ISA48467.2019.00022. URL: https://ieeexplore.ieee.org/document/9014383/ (visited on 05/20/2020).

[5] Zeinab Abbasi, Mehdi Kargahi, and Morteza Mohaqeqi. "Anomaly detection in embedded systems using simultaneous power and temperature monitoring". In: *2014 11th International ISC Conference on Information Security and Cryptology*. Sept. 2014, pp. 115–119. DOI: 10.1109/ISCISC.2014.6994033.

[6] Christina Stracquodaine, Andrey Dolgikh, Matthew Davis, et al. "Unmanned Aerial System security using real-time autopilot software analysis". In: *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*. Meeting Name: 2016 International Conference on Unmanned Aircraft Systems (ICUAS) Reporter: 2016 International Conference on Unmanned Aircraft Systems (ICUAS) ISSN: null. June 2016, pp. 830–839. DOI: 10.1109/ICUAS.2016.7502633.

[7] Pratyusa K. Manadhata and Jeannette M. Wing. "An Attack Surface Metric". In: *IEEE Transactions on Software Engineering* 37.3 (May 2011). Conference Name: IEEE Transactions on Software Engineering, pp. 371–386. ISSN: 1939-3520. DOI: 10.1109/TSE.2010.60.

[8] Matthew Leccadito, Tim Bakker, Robert Klenke, et al. "A survey on securing UAS cyber physical systems". In: *IEEE Aerospace and Electronic Systems Magazine* 33.10 (Oct. 2018). Number: 10 Reporter: IEEE Aerospace and Electronic Systems Magazine, pp. 22–32. ISSN: 1557-959X. DOI: 10.1109/MAES.2018.160145.

[9] Bharat B Madan, Manoj Banik, and Doina Bein. "Securing unmanned autonomous systems from cyber threats". In: *The Journal of Defense Modeling and Simulation* 16.2 (Apr. 2019). Publisher: SAGE Publications, pp. 119–136. ISSN: 1548-5129. DOI: 10.1177/1548512916628335. URL: https://doi.org/10.1177/1548512916628335 (visited on 05/20/2020).

[10] C. G. Leela Krishna and Robin R. Murphy. "A review on cybersecurity vulnerabilities for unmanned aerial vehicles". In: *2017 IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*. ISSN: 2475-8426. Oct. 2017, pp. 194–199. DOI: 10.1109/SSRR.2017.8088163.

[11] Ke Yue, Li Wang, Shangping Ren, et al. "An Adaptive Discrete Event Model for Cyber-Physical System". en. In: *Analytic Virtual Integration of Cyber-Physical Systems Workshop, USA* (2010), pp. 9–15.

[12] Giedre Sabaliauskaite and Aditya P. Mathur. "Intelligent Checkers to Improve Attack Detection in Cyber Physical Systems". In: *2013 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*. Meeting Name: 2013 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery Reporter: 2013 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery ISSN: null. Oct. 2013, pp. 27–30. DOI: 10.1109/CyberC.2013.14.

[13] J. Magiera and R. Katulski. "Detection and Mitigation of GPS Spoofing Based on Antenna Array Processing". en. In: *Journal of Applied Research and Technology* 13.1 (Feb. 2015), pp. 45–57. ISSN: 1665-6423. DOI: 10.1016/S1665-6423(15)30004-3. URL: http://www.sciencedirect.com/science/article/pii/S1665642315300043 (visited on 05/07/2020).

[14] Mohsen Riahi Manesh and Naima Kaabouch. "Cyber-attacks on unmanned aerial system networks: Detection, countermeasure, and future research directions". en. In: *Computers & Security* 85 (Aug. 2019), pp. 386–401. ISSN: 0167-4048. DOI: 10.1016/j.cose.2019.05.003. URL: http://www.sciencedirect.com/science/article/pii/S0167404819300963 (visited on 05/11/2020).

[15] Jongho Shin, Youngmi Baek, Yongsoon Eun, et al. "Intelligent sensor attack detection and identification for automotive cyber-physical systems". In: *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*. Meeting Name: 2017 IEEE Symposium Series on Computational Intelligence (SSCI) Reporter: 2017 IEEE Symposium Series on Computational Intelligence (SSCI) ISSN: null. Nov. 2017, pp. 1–8. DOI: 10.1109/SSCI.2017.8280915.

[16] Derui Ding, Qing-Long Han, Yang Xiang, et al. "A survey on security control and attack detection for industrial cyber-physical systems". en. In: *Neurocomputing* 275 (Jan. 2018), pp. 1674–1683. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2017.10.009. URL: http://www.sciencedirect.com/science/article/pii/S0925231217316351 (visited on 05/07/2020).

[17] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: *arXiv:1609.04747 [cs]* (June 2017). arXiv: 1609.04747. URL: http://arxiv.org/abs/1609.04747 (visited on 05/20/2020).

[18] Hichem Sedjelmaci, Sidi Mohammed Senouci, and Nirwan Ansari. "A Hierarchical Detection and Response System to Enhance Security Against Lethal Cyber-Attacks in UAV Networks". In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 48.9 (Sept. 2018). Conference Name: IEEE Transactions on Systems, Man, and Cybernetics: Systems, pp. 1594–1606. ISSN: 2168-2232. DOI: 10.1109/TSMC.2017.2681698.

[19] Nils Ole Tippenhauer, Christina Pöpper, Kasper Bonne Rasmussen, et al. "On the requirements for successful GPS spoofing attacks". In: *Proceedings of the 18th ACM conference on Computer and communications security*. CCS '11. Chicago, Illinois, USA: Association for Computing Machinery, Oct. 2011, pp. 75–86. ISBN: 978-1-4503-0948-6. DOI: 10.1145/2046707.2046719. URL: https://doi.org/10.1145/2046707.2046719 (visited on 05/11/2020).