2017

A Hierarchical Architectural Framework for Securing Unmanned Aerial Systems

Matthew Leccadito
Virginia Commonwealth University

A Hierarchical Architectural Framework for Securing Unmanned Aerial Systems

A Dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at Virginia Commonwealth University.

By

MATTHEW LECCADITO

Master of Science, Virginia Commonwealth University, 2013

Director: Dr. Robert H. Klenke,

Professor, Department of Electrical & Computer Engineering

Virginia Commonwealth University

Richmond, Virginia

August, 2017

Acknowledgement

Firstly, I would like to express my sincere gratitude to my advisor Dr. Robert H. Klenke for the continuous support, patience, motivation, and immense knowledge. I could not have imagined having a better advisor and mentor for my graduate studies. Secondly, I would like to thank Dr. Tim Baker, who has helped me every step of the way through my graduate career. I would not be where I am today without the many hours he devoted to me, whether it be discussing solutions, debugging hardware and software, or teaching me various techniques that would contribute to the skills I possess today.

I would like to thank the members of my advisory committee: Dr. Carl Elks, Dr. Milos Manic, and Dr. Edward L. Boone for their insightful comments and encouragement, and for asking the right questions which helped guide me and keep me on the right path. I would like to thank the Virginia Commonwealth University School of Engineering for supporting and funding my academic research in the UAV lab. I would also like to thank Andy Fabian and Andrew Ward for assisting me in the implementation of my research and the insight they have given me. Finally, I must express my profound gratitude to my friends and family for providing me with unfailing support and continuous encouragement through my years of study and through the process of researching and writing this dissertation. This accomplishment would not have been possible without them. Thank you.

Table of Contents

LIST OF TABLES

LIST OF FIGURES

Abstract

A Hierarchical Architectural Framework for Securing Unmanned Aerial Systems

By Matt Leccadito

A Dissertation submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy at Virginia Commonwealth University.

Virginia Commonwealth University, 2017.

Director: Dr. Robert H. Klenke,

Professor, Department of Electrical & Computer Engineering

Unmanned Aerial Systems (UAS) are becoming more widely used in the new era of evolving technology; increasing performance while decreasing size, weight, and cost. A UAS equipped with a Flight Control System (FCS) that can be used to fly semi- or fully-autonomous is a prime example of a Cyber Physical and Safety Critical system. Current Cyber-Physical defenses against malicious attacks are structured around security standards for best practices involving the development of protocols and the digital software implementation. Thus far, few attempts have been made to embed security into the architecture of the system considering security as a holistic problem. Therefore, a Hierarchical, Embedded, Cyber Attack Detection (HECAD) framework is developed to provide security in a holistic manor, providing resiliency against cyber-attacks as well as introducing strategies for mitigating and dealing with component failures. Traversing the hardware/software barrier, HECAD provides detection of malicious faults at the hardware and software level; verified through the development of an FPGA implementation and tested using a UAS FCS.

CHAPTER 1

INTRODUCTION

Cyber Physical Systems (CPS) play an important role in providing critical services in industries such as: autonomous vehicle systems, energy, health, manufacturing, etc., by integrating computation, physical control, and networking. Most of these systems are not only cyber-physical, but also operate in a safety-critical application where a failure or malfunction could result in damage or even loss of life. An Unmanned Aerial System (UAS) meets the requirements of a CPS and safety-critical system with its dependence on wireless communication, sensors, and algorithms that work synergistically to perform its functionality. Innovation in UAS technology has followed the paradigm of enhancing performance as a main priority, with security as either an afterthought or not considered at all, causing a lack of security against cyber-attacks in most UAS. In the past, UAS were expensive, heavy, and most commonly used by the military, however, cost, size, and weight have decreased drastically, while their capabilities, attributed to technology, have increased substantially. Technological advances are rapidly increasing in unmanned systems and secure solutions must keep-up with the technology to maintain safety and assurance. The increased interest in UAS due to semi and fully autonomous flight has benefited the civilian and military community and lead to increased efforts to incorporate UASs into industry [1]–[3]. The shift of control from a human pilot to an automated, computerized autopilot has tremendous advantages; however, the dependence on embedded electronics exposes UASs to new threats of cyber-attacks.

The cyber threats to UASs are becoming more evident and research in the area of securing safety-critical CPSs is increasing [4]. Current research is focused on creating attack assessments and discovering vulnerabilities [5]–[8], but has not significantly addressed detection and prevention of cyber-attacks on UASs, and more specifically the UAS Flight

Control System (FCS). A survey has been conducted on cyber-attack vulnerabilities and defenses for flight control systems in [9], as a preliminary assessment to developing a hardware architecture to detect and prevent attacks on a UAS. Vulnerability assessments typically focus on wireless and Global Positioning System (GPS) attacks [10]–[18] and less on malicious attacks from within the system. Minimal research has been conducted on increasing resilience of the FCS itself by investigating inter-component communication functionality, detecting sensor data anomalies, and fingerprinting of normal execution behavior. This research presents comprehensive solutions capable of detecting malicious attacks at various levels and parts of the system, such as a cyber-attack in the form of malicious code injected into the onboard sensors and firmware on the FCS with the ability to potentially mitigate the attack and if necessary quarantine the malicious on-board component.. The insight gained from the assessments has been used to develop an architecture to secure a UAS FCS from cyber-attacks. A block diagram of a typical UAS FCS is shown in figure 1 and can be used as a guide to examine the vulnerabilities of the FCS. The FCS consists of an autopilot, which is typically the main processor performing navigation and control algorithms that control the UAS while simultaneously polling sensors, controlling physical actuators and managing command, control and communication with a Ground Control Station (GCS) operator.

The communication links between the FCS and the other on-board sensors are open to vulnerabilities, which can be easily identified: sensors, actuators, and wireless communication. The fourth vulnerability is the firmware running on the main processor, which can consist of not only the navigational and control algorithms, but the lower level driver software. Not only is the code written by an error prone human, but may also need to be updated for various reasons, which is typically performed by the UAS operator who may have little knowledge of the potential security vulnerabilities within the FCS. It is common

for a user to purchase a ready to fly, Commercial Off-The-Shelf (COTS) UAS which does not require any technical understanding of the UAS, creating additional vulnerabilities.

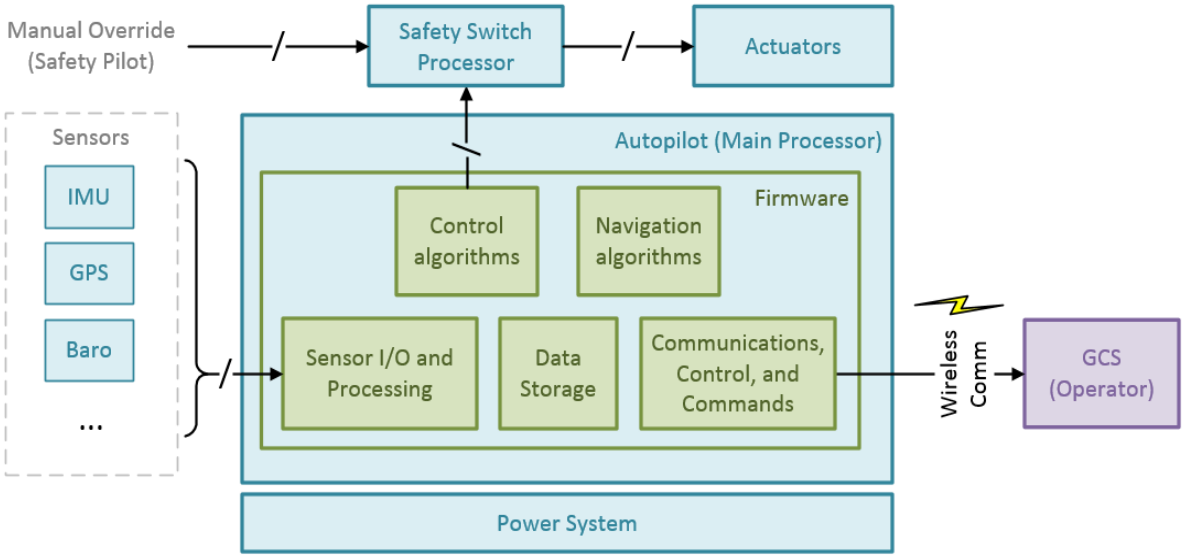

Figure 1 High Level Abstraction of a UAS FCS architecture

The identified vulnerabilities can be divided into four categories: firmware, sensors, communications, and actuators. Many sub-components of a UAS FCS contain embedded firmware vulnerable to attacks through the manufacturing and distribution chain or during run-time by another component having the access and authority to modify or update the sensor firmware.

One of the mechanisms for inserting malicious code onto the autopilot may be through website spoofing [19]–[21], where an attacker may clone a legitimate website and the user would download malicious code. Other ways can include a software developer with malicious intent or through maintenance of the autopilot, which would require physical access, where an un-trusted source can program malicious firmware onto the autopilot. Analogous to the autopilot main processor firmware is the sensor firmware. The sensor firmware may be corrupted through the manufacturer, distributor, assembly, or during programming of the autopilot. The malicious firmware can affect the communication

between the sensor and autopilot at the protocol level or interfere with the sensor information through false data injection. A sensor may rely on an external reference, where an attacker could possibly spoof the external reference, providing false readings.

The communications network is typically a wireless connection between the autopilot and a GCS operator. The autopilot communicates through a modem on-board periodically transmitting the status of the UAS and receiving configuration and control parameters as well as commands. An attacker within distance of the wireless transmission and with certain knowledge of the signal can receive and transmit data to the modem. The attacker can scan for activity on a specific frequency, or can jam all frequencies through transmitting noise, using up all the available the bandwidth, rendering the autopilot and/or GCS incapable of receiving any legitimate data. If an attacker has knowledge of the wireless protocol as well, they may have the ability to sniff the data communication and perform a man-in-the-middle attack by intercepting and re-transmitting the data, or a replay attack through recording a message and transmitting it later. The attacker can use fuzzing techniques, by sending random data to the modem and waiting for a physical response of the vehicle. If they have knowledge of the command and control, it is possible to send commands and/or control parameters to cause harm or take control of the vehicle.

The actuators are also vulnerable to attacks, which depending on the capabilities of the actuator, it may contain firmware or can be modified through physical manipulation. It is also possible to attack the actuators through the autopilot or the network, which may require modification of the autopilot firmware or parameters that may be configurable through the network. The UAS FCS has various parts that can be targets for cyber-attacks such as the main processor executing the autopilot firmware, on-board components and sensor firmware, wireless network, and ground control station. There currently does not exist any solution capable of monitoring the on-board components, integrated on the FCS, that

connects between the main processor and on-board components allowing the ability to monitor on-board functionality.

Objectives of the Dissertation

The objective of this dissertation is to present the design, development, and testing of an FPGA implementation of the hierarchical architecture accompanied by a set of detection algorithms that can be integrated into a UAS FCS (an example of a CPS) to monitor, detect, and react against cyber-attacks.

A hierarchical architecture was developed to secure a UAS FCS from malicious attacks at all levels of integration without compromising performance and functionality of the system while in a normal monitoring state through passively sniffing the on-board signals. As is the case of many CPS applications, there are significant size, weight, and power limitations. Furthermore, the monitoring and detection system is designed so that it does not create additional vulnerabilities for the UAS FCS. It performs in real-time and is embedded into the system such that it has the capability to monitor all communication and components within the system and have the ability to isolate any compromised subsystem from the main processor.

Contributions of the Dissertation

To achieve the objectives outlined above, two primary contributions are presented in this dissertation.

The first contribution is a novel methodology that monitors a UAS FCS through monitoring the physical signals between the main processor and on-board components, performing multi-level verification on the systems specification characteristics, dynamic runtime attributes, and system level, functional parameter assertions. In addition, the

methodology allows the ability to use the multi-level verification collaboratively to develop a more concise decision to reduce the amount of false positive attack detections.

Second, the detailed design of the embedded cyber security monitor and detection hardware architecture is presented. It is designed to be integrated directly within the FCS architecture, on-board the PCB and does not require hardware and software external to the FCS. The presented architecture not only allows monitoring of external communication but allows monitoring at all the functional levels of the CPS including hardware level communication protocols, sensor data integrity, and firmware verification. Furthermore, the presented architecture is positioned between the autopilot main processor and on-board sensors and actuators, allowing isolation of compromised components.

CHAPTER 2
BACKGROUND AND RELATED WORK

2.1 Vulnerabilities of a generic CPS

A vulnerability can be defined as a weakness or flaw in the system which an exploit can take advantage of to perform malicious activity. The vulnerabilities of a CPS can vary depending on the application. To characterize the vulnerabilities that exist in a CPS, a taxonomy of vulnerabilities was developed as shown in Figure 2. Although it is not a complete taxonomy listing all vulnerabilities in a CPS, it can be generalized to apply to most systems. The taxonomy does not differentiate between deliberately and non-deliberately created vulnerabilities, as it demonstrates where a vulnerability is created in a general manner. Who created the vulnerability and for what purpose can be considered irrelevant for the purposes of this work.

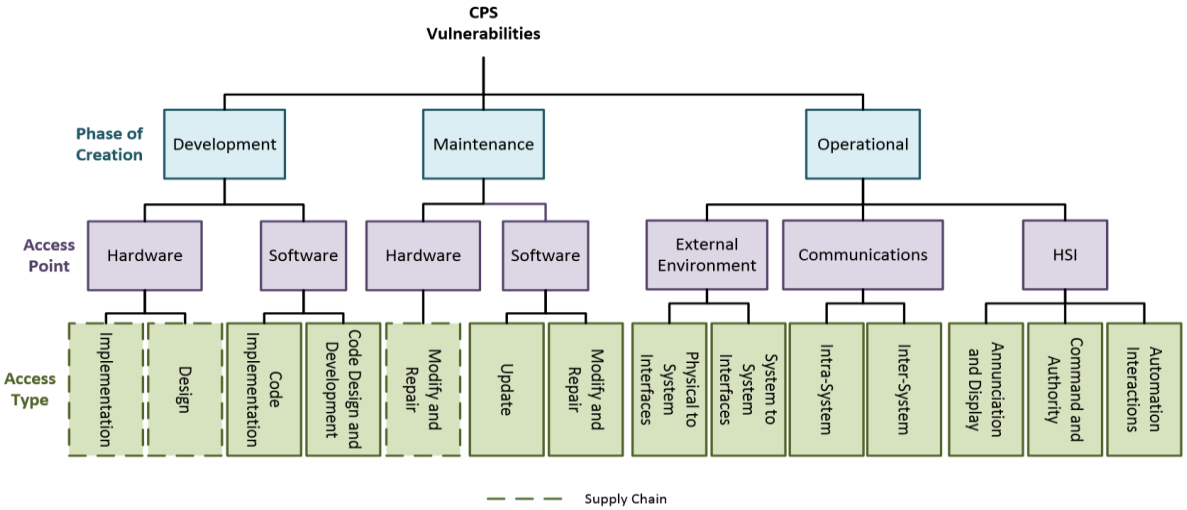

Figure 2 CPS Vulnerability Taxonomy

The taxonomy shows where the vulnerability lies in the various steps of the life cycle of a CPS. There are three levels of the taxonomy: Phase of Creation, Access Point, and Access Type. The three phases of creation describe where a vulnerability may be developed:

development, maintenance, and operation. The development phase is where the system is initially designed and implemented by the development team which can have access to many parts of the system a typical user would not have access to. This can also be a reoccurring phase depending on the application where a system may continuously be in development. The maintenance phase consists of system upgrades or updates, or any additional modifications to the system which can be carried out by a technician or the user. Vulnerabilities can be induced by the software developer or engineer providing the update to the software or modifications to the system. Lastly, the operational phase refers to vulnerabilities introduced during the operation of the system, such as a communications link that transmits and receives when the system is up and running or external factors that can only be introduced during operation. Vulnerabilities may exist during the operational phase; however, the taxonomy encapsulates the creation of vulnerabilities, not when the vulnerability is exposed.

The access points describe where a malicious attacker would exploit the vulnerability, such as through hardware by modifying the physical components, or software through exploiting a programming bug or inserting malicious code, or eavesdropping on a wireless communication. At the bottom of the taxonomy lies the access types which describes more specifically the type of access a malicious adversary would perform to exploit the vulnerability, e.g. inserting an intentional bug into the code during the development stage, updating the autopilot with corrupt firmware code, or replacing physical components on the FCS.

2.2 Vulnerabilities of a UAS CPS

An assessment on CPS exploits performed in [22] aided in the development of the following list of exploits for a UAS Cyber Physical System. The exploits can be placed into four main

categories: Network, Firmware, Sensor, and Ground Control Station. The first series of exploits consist of network attacks. A network attack can include any type of wired or wireless communications attack. Second are firmware attacks, which are related to the exploits on the software/firmware executing of the main processor of the system. Third are the sensor attacks which refer to the firmware on the physical sensor, external reference manipulation, and any on-board component such as the actuators. Last, are the Ground Control Station (GCS) attacks which are performed on the computer and/or software the operator uses to control and communicate with the UAS. A list of possible exploits are shown in detail in Table 1.

Table 1 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 [23]
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 [5], [24].
Network Isolation	Attacks aimed at isolating a particular physical geographical area from a network [25].
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 [7], [11], [14], [17], [26].
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 SQL injection).
False Data Injection:	Exploiting database vulnerabilities, typically by adding erroneous data [27], [28].
Firmware Modification	Modifying a subsystems firmware (i.e. router) to get to the ultimate target [29], [30]. 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., pre-installing 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 [8], [31].
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 denial of service 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 is harmful with malicious code.
Key loggers	Program that records key strokes on computers where they are installed.

Traditionally, UAS flight control system (i.e., autopilot) software and hardware developers are not fully aware of CPS cyber security vulnerabilities. Cyber threat analysis is typically conducted at a higher level, separate from the design process, by experts in the field. However, conventional cyber security techniques need to evolve from the paradigm of build it, then secure it, to a more concurrent design approach. It is necessary to not only assess the UAS from a high-level system view, but also from a low-level hardware perspective using a more detailed holistic approach. The objective of a cyber-attack on a UAS can be categorized as one of two types: Mission Envelope Failure or Flight Envelope Failure. Mission Envelope Failure is defined as restrictions placed on the vehicle by a cyber-attack (i.e. altitude limits,

GPS boundaries, etc.) that result in failure to successfully complete the UASs mission. Flight Envelope Failure refers to a failure of the flight control system to maintain the vehicles state within the acceptable operating envelope, resulting in failure of the vehicles air-frame and often vehicle destruction.

Network Attacks

UASs typically use wireless communications links for command, control, and communications (C3). The C3 link allows a GCS operator to send navigational commands to change the vehicles flight path, altitude, airspeed, etc. as well as receiving system status and health information from the vehicle. The data communications link, if applicable, allows the real-time transmission of data gathered by the vehicle (e.g. video) to be sent to the operator or other entities. Usually, the data communications link does not affect the operation of the vehicle meaning that it is not safety-critical, although interrupting it may effectively cause a Mission Envelope Failure.

Small, low-cost, COTS UASs were not historically designed with the consideration that the wireless communication channel would be a vulnerability since they were considered to be used as a hobby. It has become evident, however, that the vulnerabilities in UAS communication links can be exploited. With UASs becoming more popular, the use of new, low cost, small, lightweight, and fast encryption capabilities will reduce the probability of wireless security problems. Techniques for securing the wireless communication link in UASs are presented in [10], [17]. In addition, an assessment of different wireless communication channels and sensor attack analysis is performed in [7]; In UAS communication, there are four basic communication architectures [14]: Direct Link/Ad-hoc, Satellite, Cellular, and Mesh Networking. Direct link refers to a connection that is reliable with low latency and little to no interference, satellite communication improves coverage, however has limited bandwidth; cellular refers to using downlink towers that can be

extended for better coverage; and a mesh architecture treats each node as a relay in the system.

Two types of wireless communication for UASs exist, line-of-sight (LOS) and indirect. Indirect communication typically refers to satellite communication (SATCOM). The most common frequency bands used for wireless communication in Unmanned Systems are 400 MHz, 900MHz, 15.15GHz – 15.35GHz, 4GHz – 8GHz, 2.4 GHz and 5.8GHz. The 400MHz band is commonly used throughout Europe and the rest of the world (excluding US); lower power required, lower path loss than 900MHz, translating to less power for the same range. However, with the lower frequency channel bandwidths can be limited to 10kbits/s or less. The 900MHz Band is commonly used for UASs for narrow band lower data rate applications than the 2.4GHz 5.8GHz band. The 900 MHz Band offers better performance than 2.4GHz or higher bands when obstructions such as trees and water are present. It is often used in non-line-of-sight applications [26]. The 15.15GHz - 15.35GHz up link and 14.40 - 14.85 down link is referred to as the TCDL Ku-Band; a secured link developed by military and susceptible to interference, and can be affected by rain/snow, and air humidity. The 4GHz - 8GHz range is referred to as the C-Band. The UAS range is typically 4.4GHz-4.94GHz and 5.25GHz - 5.75GHz, which are less susceptible to air humidity, but subject to interference by many COTS devices. The most commonly used band is Wi-Fi a/b/g/n: 2.4GHz - 2.48GHz and 5.15GHz - 5.75GHz; which uses MIMO and up to 4 antennas. Omnidirectional antennas have an antenna pattern that radiates in all directions perform better outdoors, however, there is an increased risk of eavesdropping due to its increased radius and the unlicensed commercial nature of the 2.4 and 5.8 GHz bands.

A UAS has three main security breaches associated with its control and telemetry wireless data link [11]: packet forwarding, eavesdropping, and hijacking. Packet forwarding is a technique where the data packets are delivered in an irregular way such as dropping of packets, duplication, or modification. Eavesdropping refers to an adversary silently listening

to the communication, and hijacking is when an adversary gains control of the vehicle through mimicking the GCS.

A typical UAS-GCS communication can be classified as a Mobile ADHOC Network (MANET) like a wireless sensor network (WSN). A WSN typically has static nodes and a MANET consists of dynamic nodes. They pose many of the same vulnerabilities, however certain attacks, such as proximity based attacks (i.e. jamming), would be less effective because the attacker must be mobile. To understand how a UAS may be vulnerable to wireless attacks, WSNs vulnerabilities are investigated. The major attacks on a wireless network can be categorized as follows:

- *Denial of Service (DoS)*
    - Physical Layer: Jamming, tampering
    - Link Layer: Collision exhaustion, unfairness
    - Network Layer: Neglect and greed, homing, misdirection, black holes
    - Transport Layer: Flooding, de-synchronization
- *Attack on Information in Transit*: Information may be spoofed, replayed, or vanished.
- *Sybil*: Forging an identity; in cases where more than one UAS work together to accomplish a task, an adversary node can pretend to be a peer node or a member of a group when it is not. This can in turn degrade the integrity of data, security and resource utilization, routing, voting, and resource allocation system functions.
- *Black Hole/Sinkhole Wireless Attack*: Attacker listens to network and finds the highest quality or shortest path between the GCS and a UAS, then attracts all traffic between the two entities and corrupts or destroys it.
- *Hello Flood Wireless Attack*: In a group of UASs, an adversary sends a Hello type message to initiate friendly contact with a neighboring node, the neighboring node acknowledges this as a friendly node and sends messages through the adversary to

the ground station; typically used in large areas where a mesh type network is utilized.

- *Wormhole Wireless Attack*: Tunneling or retransmitting data that does not require compromising the network; attacker gives a node a false pretense that the recipient of a message is only one hop away, when it is multi-hops away.
- *Botnets*: The term botnet refers to insertion of a compromised computer into a network.

UASs can be small, cheap, and easily equipped with wireless networking capability, providing a platform to create a wireless botnet in the skies [32]. Instead of using a fixed computer, attackers can send commands through the non-stationary UAS making it difficult to track the attacker. It is possible to get directly between the GCS and the vehicle in the air, creating a new type of intrusion. Using this method, it is possible to get close enough to attack the wireless network functionality. They can be used to attack a Wi-Fi network by exploiting vulnerabilities in Wi-Fi as well as 3G and GSM [33], acting as an antenna that cell phones would connect through to make calls. UASs can be used to survey a wireless network and gather data such as BSSID, SSID, encryption type, channel, or the MAC addresses of the communicating entities.

Soon, the Automatic Dependent Surveillance Broadcast (ADS-B) will be a major cornerstone of aircraft de-confliction in the National Airspace System (NAS). ADS-B is the next generation of air traffic modernization. It transmits information about the aircraft's state (i.e. altitude and airspeed) and positional data to other aircraft and ground stations in the vicinity [29]. Two types of corruption of the ADS-B message have been identified, message misuse which is caused by an adversary identifying, locating, and tracking the entity, and message delay, caused by jamming. The ADS-B system will introduce another attack surface into the UAS ecosystem. A white paper released in 2012 by Eurecom referenced in [30], that presented techniques to address the vulnerabilities in the ADS-B

system which include: message authentication to protect from unauthorized users, message signatures to protect from tampering of messages, encryption to protect sensitive data, message integrity to assure validity of the message, and message freshness to protect against replay attacks.

Firmware Attacks

The autopilot contains control and navigational algorithms that are executing in software on the main processor along with low level drivers to communicate with on-board sensors. A firmware attack may include modifying the source code that is executing on the autopilot, which can be inserted during development or maintenance of the UAS. During development of the autopilot software a vulnerability may be inserted intentionally or unintentionally by either a malicious developer or an inexperienced programmer. It is also possible for an adversary to mirror a website that is hosting the source code for the autopilot code, and malicious code can be programmed onto the autopilot by the operator unintentionally.

Sensor Attacks

The sensors on board a UAS include an Inertial Measurement Unit (IMU), absolute and differential pressure sensors, and optical and sonar sensors. Software attacks can occur from insertion of malicious code into the sensor firmware which can then create, modify, or stop data from being sent to the main processor. Software attacks also include, fuzzing attacks that induce noise into the sensor. Unexpected, invalid, completely random noise added to the sensor data flowing into the processor can cause unexpected problems, aircraft instability, process lock-up, and invalid outputs to next process. Digital update rate attacks can delay the frequency of sensor output, causing inconsistent processing of sensor information. Software and hardware can be used to create longer sampling periods to increase the

probability of aliasing. They can also be used to change sampling times of analog-to-digital converters through buffer overflow or hardware manipulation. Denial of service is a threat as well, which can prevent the processor from running at desired update rate. With physical access to the autopilot, malicious hardware can also be inserted between components directly onto the autopilot and/or sensors can be replaced with corrupt sensors to disturb the operations of the autopilot. GPS attacks are the most common types of attacks, since a typical GPS receiver does not use any type of encryption and can be spoofed with little knowledge of the UAS. The most common types of GPS attacks [5] are shown in Table 2.

Table 2 GPS Vulnerabilities

Method	Description
GPS Attacks	The GPS is an external reference providing position data for the UAS. A GPS receiver can be attacked through manipulation of signal, software, or data [7].
GPS Data Level Attack	Manipulation of real-time data with a valid GPS signal, such as resetting the current year of the GPS receiver to the year of UNIX epoch rollover.
GPS Receiver Software Attack	Exploitation of OS vulnerabilities, and in some cases a network stack on high end devices. A GPS receiver is a device and can easily go unpatched against certain attacks.
Reference Station Attack	Attacking a reference station that maintains a static position and allows civilian GPS to calculate position with greater accuracy, can be spoofed by re-transmitting faulty data to dependent GPS receivers.
Spoofing	Attacker generates a counterfeit signal with incorrect positional and timing data [8], [9]. GPS spoofing is a proximity based attack and was successfully demonstrated at 0.62km at the University of Texas [10]. An additional limitation to this type is the need to track all GPS signals known to the target UAS.
Signal Delay	Using a GPS receiver, the GPS signal is decoded and re-broadcasted creating a timing delay, ultimately generating incorrect positional data; it is only possible to achieve gradual error with this method [11], however, a delay attack can be achieved on military GPS receivers [12].
Jamming	An attacker transmits significant RF noise through which the GPS signal cannot be deciphered.

Ground Control Station Attacks

The GCS is used by the operator to send commands, navigate, monitor, and in some cases manually control the UAS. Sometimes it is the only link to the UAS during a mission. Not only is wireless communication between the GCS and the UAS a vulnerability, but the GCS itself can be exploited. Thus, the GCS computer must be independently secured before it is connected to the UAS. The GCS can be a desktop or laptop computer with a keyboard, and consequently a key-logging virus can be inserted that results in the loss of sensitive data or compromising the UAS [31]. An attack may not necessarily occur between the UAS and GCS directly, but indirectly through a separate network that the GCS connects to that ultimately can be used to access the UAS communication link.

As smart device technology increases, the robustness and usability also increases. Desktop computers in the field are becoming antiquated as they are replaced by laptops and mobile devices. With this paradigm shift comes a new level of cyber vulnerability. Not only do most smart devices have more versatile network connection capabilities, but most software development occurs on open source software and third-party applications. A threat model has been developed assessing the vulnerabilities that exist in smart device ground control stations [8]. It shows that smart devices operating systems and software must be updated frequently, along with use of anti-virus software, as they can easily be manipulated to monitor soldiers' communication and movement. A recursive parameter estimation algorithm was developed in [34] to perform anomaly detection during flight. It can detect cyber-attacks on electronic hardware and the resulting degradation and failure. Adapting the Recursive Least Squares method, it can detect significant changes compared to previously known parameters and alert the GCS operator of an attack.

2.3 Mitigations, Countermeasures, and Defense Strategies for a UAS CPS

A UAS has a more restrictive set of requirements for features such as size, weight, energy usage, and computational power imposed by its physical limitations than a typical CPS, such as a control system for a nuclear power plant or even a ground-based autonomous vehicle. This can make it more difficult to incorporate traditional counter measures that can be applied to other types of CPS. Cyber security techniques for embedded systems related to UASs and other unmanned systems will be discussed in the following section.

Current autonomous vehicle systems that employ some type of detection and protection from cyber-attacks incorporate the following characteristics [32]:

- Multi-Agent System Architecture – Multiple systems composed of subsystems, creating a system of systems
- Redundancy - Prevents a single point of failure
- Diversity - Prevents a single attack vector from becoming effective
- Defense-in-Depth - Blocks the pathway for an unauthorized user, providing multiple layers of security countermeasures throughout the system; defending the system from any attack using multiple independent methods
- Authentication - Prevents humans and other devices from impersonating another entity
- Micro-Kernel - Uses a minimum-sized OS kernel. All unused device drivers and other software are removed, only the required micro kernel is running

A platform to increase safety and security for unmanned systems known as Sphere was developed in [35]. The purpose of Sphere is to create component usage policies by categorizing modules by different levels of criticality to mission success. A catastrophic effect could result from the failure of a critical component such as the autopilot processor, GPS, or airspeed sensor. Furthermore, the components must be isolated and disabled upon failure.

Secondly, Sphere created a protocol structure for proof of authenticity among the different subsystems, which must authenticate themselves upon start up.

Fault handling is the process of attempting to correct or work around an error in the system once it is detected. It is difficult to assess usefulness of fault handling mechanisms; most common techniques include triple redundancy and fail-safe states. Although these mechanisms cause restrictions on the size, weight, and power of UASs, they help keep the mission alive. It is shown in [7] that fail safe states can impose a risk to the UAS if the wrong state is chosen for a given situation. Fault handling must consider both transient and permanent failures. Transient failures are irregular and un-predictable faults which included external disturbances, while permanent failures typically derive from hardware damage.

ADS-B

As UAS are introduced into the national airspace, it is imperative that the ADS-B system can be as secure as possible from cyber-attacks. Attacks on the availability of ADS-B, such as jamming or denial of service, could be mitigated by employing secure, low frequency, land based alternatives such as terrestrial land based radio navigation systems; a plane to plane type of backup communication could be employed, however another plane could be spoofed [36], allowing a malicious entity to cause further harm. Confidentiality can be augmented using a common group identifier like IP communications and integrity can enhanced through a time-difference-of-arrival technique that helps identify the source of the ADS-B transmission. A lightweight Public Key Infrastructure (PKI) would be a viable solution adding integrity verification to messages [30] through the use of key-hash message authentication (HMAC) to verify message authentication.

GPS Spoofing

Interest in detecting GPS spoofing has increased in the past decade, leading to the development of techniques applied to GPS signal characteristics, including the position solution, Doppler shift, and SNR [37]–[42]. A method is developed in [12] that uses a reference receiver to compare the cross-correlation of the C/A and P(Y) codes. By isolating the P(Y) code, a large correlation can be observed in a non-spoofed signal. A new type of GPS detection was developed using a monopole-patch hybrid antenna connected to two independent GPS receivers that compares the signal to noise ratio to identify a fault [13]. A similar approach for GPS spoof detection was developed in [43] using two or more antennas with known relative positions, and comparing the resulting position solutions as opposed to the RF signal.

There are four main observable ways to spoof a GPS: the GPS message, code ranges, fractional phase ranges and Doppler shift. Various methods that have been developed to counter GPS spoofing attacks [44] are shown in Table 3.

Table 3 GPS Spoofing Counter Attack Methods

Method	Description
Monitor the absolute power of each carrier	A maximum power can be set to determine the spoofed signal.
Monitor signal power rate changing	RF signal radiation from a point source. Any positional changes near earth should not result in a dramatic signal power change. magnetic induction
Monitor the relative powers	Modern GPS have two signals, the relative power ratios should be fixed.
Bound and compare range rates	When phase range is manipulated with respect to code range, phase range rate will most likely be sacrificed and can be detected.
Doppler shift check	Relative speed of receiver compared to each satellite can be derived; attacker would need one spoofer per satellite to go undetected.
Cross Correlation of L1 and L2	The signal on L2 is slower than that on L1 due to ionosphere effect, the sign of cross-correlation shift is known.
Residual analysis	A GPS signal consists of the spoofed signal, authentic signal, and noise. It is possible to remove the spoofed signal; however, the additional noise may be too strong to recover the weak authentic signal

L1-L2 range differences	Range differences are caused by ionosphere effects; phase and code ranges should conform, a spoofed signal propagating in lower levels behaves differently
Verify received ephemeris data	Compare spoofed data with almanac data to ensure the GPS message does not provide fixed satellite information
Jump Detection	Monitor changes in observables and power consumption are within a tolerable range

Redundancy as well as multi sensor data fusion can be used as a cross-checking mechanism to reduce the risk of integrity or availability loss [45]. A GPS can be combined with an Inertial Navigation System (INS) and an optical sensor [46], which can provide additional information to compare the GPS information against, to reduce the security risk of incorrect position and altitude.

Botnets

A UAS being versatile can be quickly deployed to eavesdrop on a private or corporate wireless network, and potentially be used as the controlling unit in a botnet. A botnet refers to a large number of computers that are infected with code and controlled by an outside source. A UAS that is performing as a botnet can be difficult to detect, unless there is clear visibility of the UAS being hijacked. A secure defense requires observing the behavior of the UAS and discovering the Geo location of the UAS, or detecting the addition of new hosts in a network that then alert the system to the presence of a botnet.

Low Cost UAS Wireless Encryption

In small UASs, the most widely used wireless modem is the XBEE RF module, more specifically, the XBEE Pro 900HP [47] and the XBEE 802.15.4 [48]. These wireless modules are low power (between 45mA and 215mA), light weight (3g), and small size (2.438cm x 2.761cm). The most common way to secure a wireless network is to add encryption to the data so that it is not decipherable to a casual observer. The addition of encryption, however,

imposes limitations on a wireless network. In an ideal setting, there are unlimited resources available; this is not the case for UASs, especially low-cost hobby and civilian grade UASs. Wireless security in low cost UASs has three main trade-offs: bandwidth, latency, and energy consumption. In a low-cost system, there are greater constraints on the processing capabilities, which limits the usage of encryption algorithms. Less intensive algorithms have been investigated such as Rivest Cipher 4 (RC4), a simple and fast stream cipher, which requires only a small amount of resources and computation.

The XBEE wireless modules have been shown to work with their encryption capabilities enabled for agricultural purposes with low power and cost efficiency [49]. A study was conducted on the performance with and without using encryption on an Arduino UNO Atmega328P processing platform utilizing a Series 1 XBEE wireless module [50]. It was shown that the additional encryption capability of RC4 increased the delay of data transfer from 1ms to 150ms, and the use of RC4 with hash increased the delay from 1ms to 400ms. In [51], wireless vulnerabilities are analyzed between Bluetooth, WIFI, and the XBEE ZigBee wireless modules [52]. The research showed multiple vulnerabilities in the wireless communication, including configuration, non-encryption, and the interface by which the UAS and ground station would connect through, such as a serial port or socket communication. The XBEE wireless module is capable of handling the Advanced Encryption Standard (AES) 128 encryption algorithm on the module itself, which is based on the Rijndael cipher with a block size of 128bits [53]. However, the modules also have enough bandwidth to communicate in typical UAS applications with an AES 256 encryption algorithm performed off-board of the XBEE using an FPGA [54]. The Skipjack algorithm is less resource intensive and just as secure as the RC5 [55] derived from the RC4 algorithm with a few advancements. Applying these less intensive encryption algorithms is a first step into securing the wireless communication links and will prevent most prevalent attacks, however it is also the users' responsibility to actively enable, monitor, and update security measures. Securing UASs is

only useful if the security measures are actively being used during operation and are shared amongst the user community. For example, it is known that wireless communications are vulnerable to cyber-attack, however, it is not common for COTS UASs to use encrypted communications in civilian applications.

This section has examined the current mitigation techniques, counter measures and defenses for attacks on a UAS CPS and its various subsystems encompassing the hardware, software, wireless communication, GCS, and other potential threats. Many vulnerabilities that lie in a UAS CPS are application specific, however, many of the vulnerabilities can be applied to any CPS.

2.4 Architectural Schemes for securing a UAS CPS

There are currently, two approaches that have been proposed as solutions to securing a low-cost UAV FCS. The first is a System-Aware approach that uses a non-invasive architecture referred to as the Sentinel, and the second uses mathematical modelling and Embedded Domain Specific Language techniques, referred to as HACMS.

The System-Aware Cyber Architecture [56] was developed at the University of Virginia consisting of an integration of diverse, interchangeable, redundant subsystems from multiple vendors. The sub-systems are capable of rapidly changing their configurations and using data consistency checks and a voting scheme to determine a fault versus a natural failure. The system was referred to as the Sentinel, a non-invasive, external system proposed to be capable of monitoring and detecting cyber-attacks on a UAV and has been demonstrated using a Piccolo FCS [57]. The architecture depends on a communication channel between the FCS and the Sentinel. The Piccolo provides a serial communication channel which transmits the information of the autopilot and can also be used to send commands to the Piccolo. The Sentinel architecture polls the information from the autopilot

and performs four different techniques on the data: diversity, configuration hopping, data consistency, and tactical forensics. Implementing diversity requires multiple redundant subsystems from different vendors that perform a voting scheme to monitor and detect any inconsistency in the data. The Sentinel is also capable of changing the autopilot configuration during run-time. Using a voting scheme, data consistency is monitored using redundant subsystems and forensics to rapidly identify the attack, alert the operator, and restore the system. The Sentinel uses a triple redundancy methodology, comprised of three separate Raspberry Pis and a SiCore SHIELD 2 Coprocessor as a central interface point between the autopilot and the Sentinel system.

In most architectural designed systems, horizontal integration is used. This is the concept of developing different subsystems independently and interfacing them to interact together.

A new type of architectural design incorporates vertical integration, using architectural models to ensure compatibility of independent modules within the architecture [37], with the challenge being the trade-off of complexity versus precision of analysis. This approach is built upon using high level models that can develop low level code (i.e. MATLAB Simulink), minimizing the area for error in developing secure code. The vulnerabilities that are embedded in software are typically due to unsafe code written by developers or an unforeseen bug that may appear.

High-Assurance Cyber Military Systems (HACMS) has created an Embedded Domain-Specific Language (EDSL) specifically for embedded systems that is capable of generating safe-C [38]. HACMS was developed by DARPA in an initiative to harden Unmanned Systems from security vulnerabilities. The goal of HACMS is to develop a synthesizer capable of producing a machine-checkable proof that the generated code satisfies functional specifications as well as security and safety policies [36]. It is based on the philosophy of creating a programming language that minimizes all the vulnerabilities in current embedded

system programming languages. HACMS has done a lot of work in CPS with regard to: resilience [40], [41], [58], robustness [59], and attack detection [42].

The current proposed solutions to secure a UAV FCS each have their advantages and disadvantages. The design decision to make the Sentinel a non-invasive system, has the advantage of lower complexities when migrating to another UAV FCS, and does not modify the systems characteristics, and can be easier to maintain. However, in the research conducted in [57] there is a dependency on the existence of a serial communication interface as well as the inability to quarantine and monitor the on-board sensors, verify the firmware, and monitor the run-time execution of the autopilot. The HACMS architecture relies on mathematical proofs and formal verification, as well as an embedded domain specific language to prevent error prone code. However, this approach is very specific and would be difficult to apply to other autopilots, since the entire autopilot would need to be redesigned and recoded. Formal methods may be used to secure the autopilot; however, it can be difficult to detect malicious sensor firmware, communications and actuators attacks by solely incorporating formal methods into the firmware on the autopilot through the use of embedded DSLs.

CHAPTER 3
HIERARCHICAL MONITOR SYSTEM ARCHITECTURAL PRINCIPLES

This section describes the design decisions for the various components and reasons behind the approach of using a hierarchical architecture and subcomponents that make up the proposed methodology. Subsequently, to understand the cyber-attack domain, the system model is defined and followed by a description of what makes an attack detectable or undetectable. Secondly, various architectural methodologies are presented. The following equations will follow a standard model in cyber-physical literature e.g. [60]–[62]. The attacker model is integrated into the system model and consists of attacks on the hardware communication between the sensors and the main processing unit, the firmware on the sensors, the software executing on the main processor, and the actuators that manipulate the control surfaces.

3.1 System Model

Consider the linear control system given by the following equation:

$$x_{k+1} = Ax_k + Ba_k + w_k$$

$$y_k = Cx_k + Da_k + v_k$$

The system can be defined as Σ = (A, B, C, D) where matrices A and C describe the system and matrices B and D describe the capabilities of the attacker. In the above equation, $x_k \in \mathbb{R}^n$ represents the state of the system for k ∈ {1,…,n} ,w_k represents the noise in the system $y_k \in \mathbb{R}^p$ represents the measurements from the sensors for k ∈ {1,…,p}, v_k represents the noise in the measurements, and $a_k \in \mathbb{R}^d$ represents the attack injections into the system at time k for k ∈ {1,…,d}; $\mathbb{R}$ signifies the real numbers set. The system is in normal operating conditions if $k > 0$ and $a_k = 0$.

Detectable and Undetectable Cyber Attacks

There exists the set of all attacks, which can be separated into detectable and undetectable attacks. There are three main attributes that must be exhibited for an attack to be detectable. First an attack must be observable, where it must be possible to view the attack. Second, it must be accessible, meaning that there must be some way of retrieving the data, such as using a hardware IP core connected to a physical signal to extract the bits from the communication and parse the data based on the protocol. Lastly, it must be verifiable, meaning that there needs to be something to compare the result against in order to verify that it is correct.

To examine what makes an attack detectable vs. undetectable, we use the following equations derived from [62]. First, we make the assumption that the pair (A, C) is observable. This model allows for simultaneous attacks on the sensors, main processing unit, and actuators. First we consider the following sequences, first the output sequence:

$$Y_k = [\ y(0)^T\ \ y(1)^T \dots\ y(k)^T\ ]^T,$$

and secondly the unknown attack sequence injected by the attacker:

$$E_k = [\ a(0)^T\ a(1)^T \dots a(k)^T\ ]^T,$$

with k > 0.

As stated previously, the pair (A, C) are observable, and can be used to create the observability matrix:

$$O_k = \begin{bmatrix} C \\ CA \\ \vdots \\ CA^T \end{bmatrix},$$

A detector for a given system Σ, can be defined as:

$$\partial : \mathbb{R}^{p(k+1)} \times \mathbb{R}^q \to \{1, 0\},$$

Where "1" represents that an attack has occurred and "0" represents that no attack has occurred. An attack E_k is considered undetectable if $\partial(O_k\gamma) = 0$ for all $\gamma \in \mathbb{R}^n$ where γ represents any time step where $a_k \neq 0$. A detectable attack is any attack that is not undetectable and there exists an occurrence where $\partial(O_k\gamma) = 1$. The set of detectable and undetectable attacks can be combined into the set of all possible attacks $\mathbb{R}^{d(k+1)}$. This set of all cyber-attacks with the detectable vs. undetectable attacks are shown in Figure 3. When developing a monitoring architecture, we must analyze what types of attacks are detectable and undetectable and try to extract as much information as possible from the system.

3.2 Partitioning Cyber Attacks on a UAV CPS

To determine which cyber-attacks against a UAS CPS are detectable and undetectable, we must first define which functions of a UAS are observable:

Function 1 (Sensor communication). *The communication between the main processor and the on-board sensors, including the communication between the on-board modem and main processor.*

Function 2 (Wireless Communication). *The Command, Control, and Communications (C3C) between the Ground Control Station (GCS) and the UAV.*

Function 3 (Actuator Outputs). *The actuator outputs between the main processor/receiver and the actuators controlling the UAV control surfaces.*

Function 4 (Autopilot software). *The software programmed onto the autopilots main processor running navigational, and control algorithms as well as hardware peripheral firmware.*

Sensor Communication

The communication between the sensors and the main processor can be directly monitored and the attacks on the sensors can be partitioned into various types of attacks. First, a hardware protocol attack, which consists of changing the configuration of the sensors hardware peripheral communication protocol or injecting a fault into the physical signal. Secondly, a firmware attack which is injected by modifying the firmware on the sensor, which can typically occur during a supply chain attack by the manufacturer or distributor. Lastly, a spoofing attack, which requires a sensor to rely on an external reference, where an attacker imitates the external reference with malicious intent.

Wireless Communication

The wireless communication between the UAS and the GCS can easily be eavesdropped on by an adversary within the operating distance of the UAS. The wireless communication operates through a wireless modem which can be monitored onboard the UAV. Attacks may be directed against the wireless communication channel through methods such as Denial of Service (DoS), replay, fuzzing, etc. which mostly are detectable attacks, or indirectly by attacking the GCS and sending command/controls through the wireless channel, which can result in an undetectable attack.

Actuator Outputs

Actuator outputs can be monitored on the FCS directly between the main processor and the actuator. The actuator outputs, depending on the FCS configuration can be driven by either a manual receiver, augmented using a combination of manual and autopilot, or completely by the autopilot in a fully autonomous mode.

Autopilot Software

The autopilot software is programmed into the main processors flash memory, and can be monitored through a debug interface such as JTAG or SWD depending on whether the pins are accessible on the FCS. There are programs available that can make use of formal methods to create statically verified code to minimize error prone code generation.

Information vs. Security Tradeoff

The properties define the observability of the FCS on the UAV and the functionality may contain detectable and undetectable attacks. Figure 3 displays visually how there is a set of all cyber-attacks, where the objective of a cyber-attack monitor is to bridge the gap and minimize the separation of the detectable attacks vs. undetectable attacks.

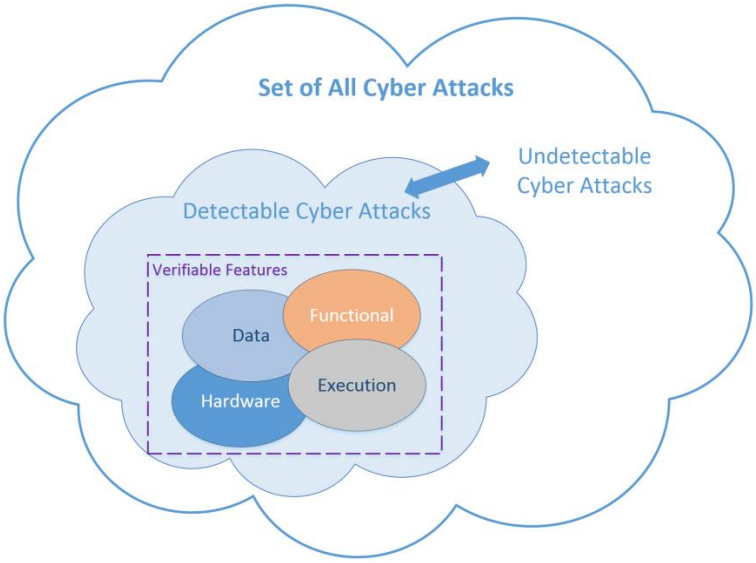

Figure 3 Set of Cyber Attacks

The objective of a cyber-attack detection monitor is to gather as much information possible to detect an attack. With respect to the monitor obtaining the maximum amount of

information, a trade-off must be made whether the monitor will make a more informed decision and be less secure, or a less informed decision and be a more secure. For example, to detect attacks and/or isolate sensors at the hardware communication level between the sensors and the main processor, it is necessary to create a monitor that passes through or blocks communication between the main processor and the sensor. The capability of blocking communication in the monitor, creates a vulnerability, but it does allow a greater amount of information to be gathered by the monitor to determine that there is an attack ongoing and where it originates. With a greater amount of observability in the system, the amount of detectable errors increases which also allows monitoring at the hardware, data, execution, and functional levels of the system. With a detailed set of observable properties defined, the following sub sections will describe different architectures for creating the best security monitor for cyber-attack detection on a UAV CPS.

3.3 Embedded vs. External

The first decision in developing the proposed methodology is whether the monitor should be an embedded or external isolated system. Embedding the monitor within the system provides a monitor with an increased amount of information, through access to the on-board components that can be used in the fault/attack detection decision making. It allows accessibility of the low-level physical signals that provide internal communication between the main processor as well as the on-board components such as the actuators and power rails. An embedded approach also allows for the ability to physically isolate a sensor from the main processor and perform a mitigation action. An external system such as the Sentinel discussed in related work, has the advantage that it is a separate system and can be isolated and does not require any modification of the system it is monitoring. However, it is not able to perform analysis of the on-board components since it relies on the system being monitored to provide the data necessary to perform the

monitoring function at the software level through an external communication line. Because the embedded monitor approach enables the detection of a larger number of cyber attacks and provides the capability to reconfigure the system to mitigate many types of cyber attacks, it was selected as the approach used in this work.

Figure 4 shows the difference between an embedded and an external solution. The embedded version of the monitor is integrated onto the FCS Printed Circuit Board (PCB) as an additional hardware component to connect as a hub between the on-board sensors and the main processor. The external version connects externally through a communication line that the main processor sends out state information.

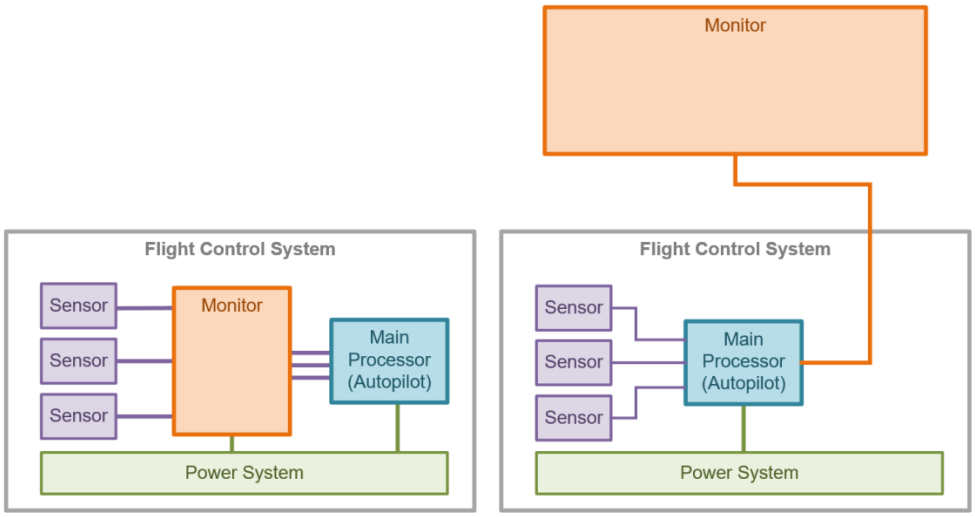

Figure 4 Embedded vs. External Monitor

3.4 Monitor Architecture

The selection of an embedded monitor approach leads to the requirement to select the overall architecture of the monitor itself. An embedded monitor can be organized as either a centralized design, a distributed design, or a hierarchical design.

3.4.1 Centralized Architecture

A centralized architecture relies on a single entity that all the subsystems report to determine if an attack has been detected or not. Each subsystem contains its own unique functionality that aids in the overall system fault detection. In this architectural scheme, all the processing within the submodules would be sequential. The advantage of having a centralized architecture would be the sharing of memory and resources, which would allow a great amount of information to be shared between each of the subsystems. The disadvantage would be that everything must flow through the central entity, where if one of the submodules were to fail, the whole system would fail. A layout of the centralized architecture with the various submodules in a security monitor is shown in Figure 5.

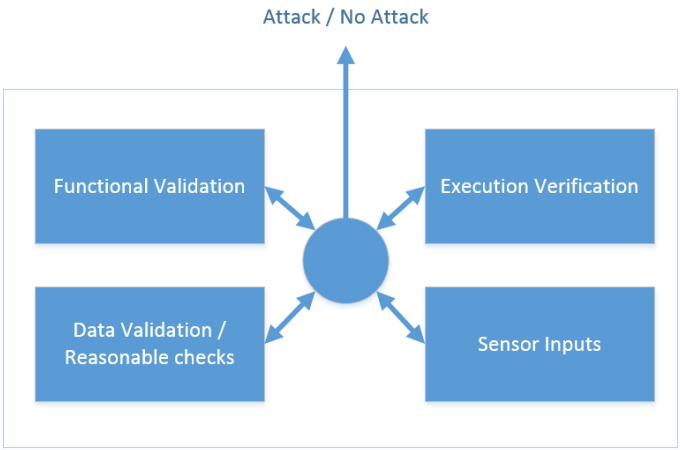

Figure 5 Centralized Architectural Layout of Security Monitor

3.4.2 Distributed Architecture

The distributed architecture allows each of the subsystems to work separately from one another and independently report on attacks. Using a distributed architecture alleviates the disadvantage of having a single centralized process deciding on whether an attack has been detected, by allowing each subsystem to alert the target system when an attack has been detected while not disturbing the other sub modules. Also, a distributed architecture allows

for sharing of resources between closely related submodules. An advantage is that there is not a single point of failure, as opposed to a centralized architecture. A disadvantage is that data is still passed to all the subsystems regardless of whether there is a fault detected or not; a case may be where a fault occurs at the hardware level and detected at one or more of the other subsystems, which can make it difficult to determine where the attack originated. A distributed system would also allow for improvements to the overall system to be made much more easily since each submodule can be upgraded individually as opposed to upgrading the entire system.

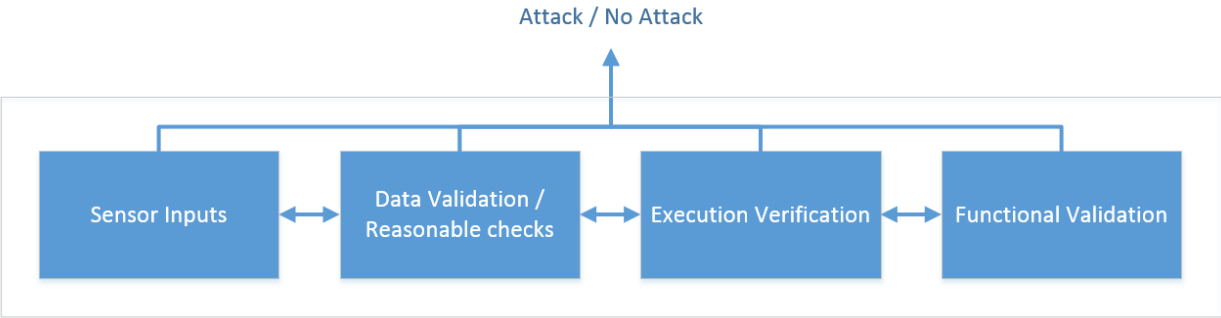

Figure 6 Distributed Architectural Layout of Security Monitor

3.4.3 Hierarchical Architecture

The hierarchical architecture follows similar principles of the distributed architecture; however, it adds an additional level of hierarchy that can provide an added benefit of multi-level verification. Hierarchy in this architectural sense refers to the collection of subsystems that depend on lower levels for operation, such as retrieving the data from the hardware resource, passing it to the next subsystem to translate from raw to sensor data, then performing system level analysis on the data. A hierarchical approach allows each submodule to verify the data at each level, so that if there is a fault at the lowest level, the submodule can notify the upper levels, allowing separation of tasks and collaboration amongst the various sub modules. It not only allows for collaboration between the

subsystems, but also creates isolation by parallelizing the execution flow for each sensor, creating an individual hierarchy amongst the various detection modules for each individual sensor so that if a failure is to occur within the architecture, it would allow for the rest of the system to operate normally. An example of a hierarchical architecture is shown in Figure 7.

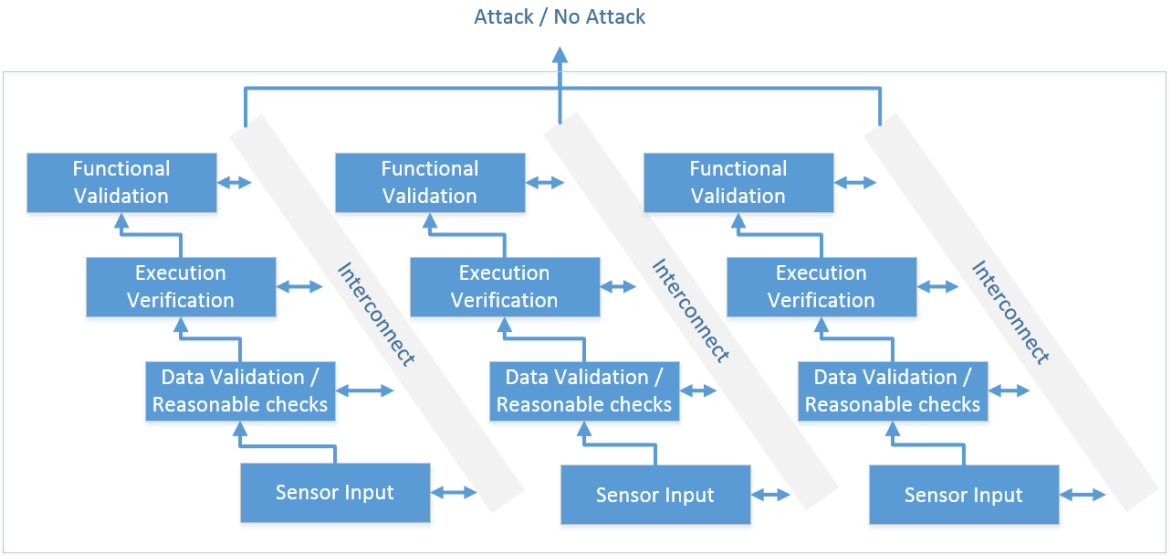

Figure 7 Hierarchical Architectural Layout of Security Monitor

The hierarchy allows many advantages, such as improved collaboration between the subsystems since they all have the ability to communicate. In the event of a fault or attack, a subsystem higher up in the hierarchy can check against a subsystem lower in the hierarchy for any abnormal behavior that may not have caused an alert, potentially providing more information on an attack with multi-level verification. An example would be a hardware level anomaly such as an incorrect parity reading on a UART which may just throw the byte away instead of passing it to the next subsystem. In this case, the UART may trigger a warning that a parity bit was incorrect, but may continue functioning normally unless the parity bit is continuously incorrect. It can create an isolated flow of information which can improve the location of detected faults/attacks that may occur in the system. It also contributes to better

security within the monitor itself, in the event of a failure or lock-up in one of the sub-modules of a particular sensor, it would not affect all of the other operations in the system. The following chapter will apply hierarchical properties to the development of a proposed architecture to monitor a UAV CPS against cyber-attacks.

A hierarchical architecture allows for multi-level verification, where system level analysis is performed on the sensor data. It is necessary that the information can be verified that it is retrieved from the sensors correctly, which depends on the hardware peripherals protocol configuration. A foundation is laid where future development can use the hardware level information, in a collaborative nature, to create a better decision using a collection of the information gathered at the hardware and information layers. This method can be used to reduce the number of false positives. It is also necessary to have a sequential data flow instead of a parallel flow that would allow checking different aspects of the information. If the data received at the hardware level is corrupted, it is possible to go undetected, and can be difficult to locate the source of the fault/attack. This type of a system requires a sequential data flow that can provide verification at multiple levels so that each subsystem in the monitor can provide separate analysis on the different stages of the information flow.

Using the hierarchical architecture with a sequential data flow leads into the design decision of developing a monitoring system with four separate subsystems. The data flow and the decision flow of the monitor work inversely whereas the data flows upward from monitoring the physical signal characteristics and retrieving the data to passing the data to the next level that translates the data from raw data into sensor data and performs specification checks, and lastly passes the data to a system level module that verifies the mission level, functionality of the system. It is also necessary to monitor the execution of the main processor to verify that it is performing correctly in conjunction with monitoring the on-board components. The decision flow is performed in a top down topology where

in the upper layers performing analysis on the sensor data and system level functions, trigger the lower level monitoring the physical signals, to isolate the sensor and change the flow of data from receiving to transmitting, depending on the mitigation sequence.

3.5 (FPGA) Field Programmable Gate Array Implementation

An FPGA implementation allows the ability to customize a soft hardware protocol IP core to perform analysis on the protocol configuration specifications. In addition, the FPGA allows the development of a customized crossbar switch implemented in digital logic. The crossbar switch allows the monitoring system to isolate an on-board component by physically switching the connections within the FPGA. If a malicious sensor is detected, the monitoring system can connect the autopilot signal to an IDLE state on the hardware peripheral, and connect the sensor to a custom internal hardware peripheral in the FPGA that can be used to mitigate the sensor through methods such as power cycling or sending reconfiguration commands, and can even reconnect the sensor to the autopilot if necessary. An FPGA also allows the ability to incorporate a soft-core processor over using a general-purpose processor. The advantage of using a soft-core processor is that it can be customized to use only the necessary components for the intended application, saving resources for other uses, as well as removing un-predictable functionality of the processor such as interrupts that can make it difficult to determine how the processor will perform its functions. With an FPGA implementation, an interconnect is needed to transfer data between the various subsystems.

3.6 Interconnect

To implement a multi-level architecture in an FPGA, an interconnect is required to transfer the data between the various subsystems. An open-source interconnect would

have the advantage of customizability vs. a licensed interconnect which may perform better in certain applications, however, a licensed interconnect cannot be modified to fit the needs of the application. A lightweight architecture with minimal signals required and short transaction time allow for a non-complex design and would allow for easier customization and would be less prone to faults. The interconnect would benefit from having a master slave architecture since the data from the physical signal layer can be polled and retrieved by the next layer whenever data is available. It is not necessary to incorporate an interconnect that is made up of various modules since it would simply be transferring data from one module to the next and no complex functionality is required. If a master-slave architecture is used, it would allow for each of the sensor modules being monitored to have a paired slave, allowing for a single, shared data bus which would provide a less complex solution where each of the sensors can use the same logic to write and read from the bus and would only need a unique address based on the master-slave pair.

CHAPTER 4

HIERARCHICAL EMBEDDED CYBER ATTACK DETECTION SYSTEM (HECAD)

To protect a UAS FCS from vulnerabilities, a hierarchical hardware architectural design is proposed to monitor, detect, and react to malicious faults. The architecture is intended to be embedded into the hardware design of a UAS FCS. A programmable hardware implementation is recommended over a software solution because it is possible to simulate and verify every component of the system, and can be more difficult to penetrate over a micro-controller, and has other advantages that will be discussed further in section IV. This section will provide the architectural layout, specifying the hardware requirements to integrate HECAD into an FCS.

4.1 Overview

The hardware design of the HECAD, allows for the ability to monitor, detect, and react on a resource constrained system. This separation of the FCS and HECAD also incorporates an extra level of security in the case of the autopilot being compromised, HECAD remains operational as opposed to executing detection algorithms on the autopilot itself. It is essential for HECAD to be an embedded hardware solution in order to intercept communication between the autopilot and sensors at the hardware level and for counter measures to be implemented in real time. A block diagram overview of the architecture is shown in Figure 8. It consists of four main functional blocks: Execution, Functional, Hardware Resource, and Information Integrity Monitors (respectively EIM, FIM, HRIM, and I2M). The functional blocks are isolated from one another and operate concurrently. HECAD is designed to be integrated into the architecture of the FCS, external to the main processor, creating a separation of autopilot algorithms and security. The hardware implementation of

HECAD is secured to make it very difficult for an adversary to attack, allowing the ability to completely isolate on-board sensors as well as completely override the autopilot in a secure manner.

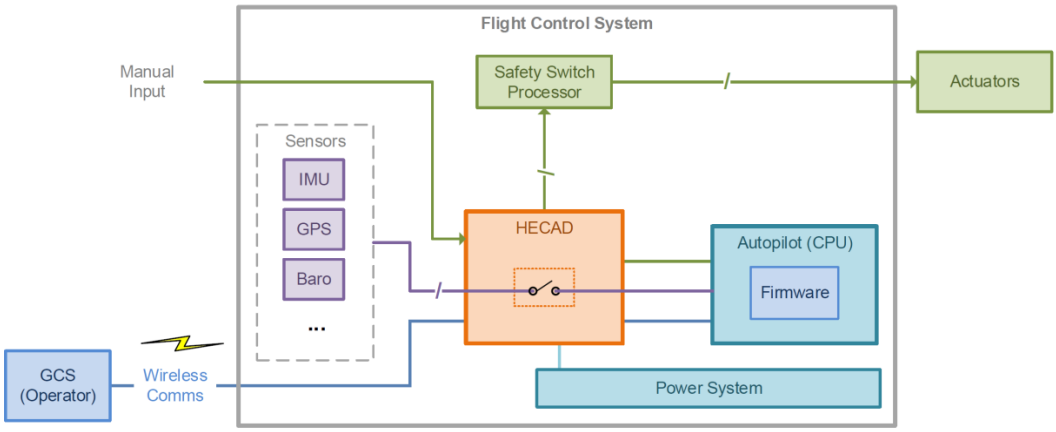

Figure 8 HECAD System Overview

HECAD has the ability to monitor the system on an individual component level as well as monitoring the system from a holistic view. The monitor's data flow works in a bottom up methodology beginning with the HRIM verifying the hardware resources through monitoring the communication protocols of the on-board sensors and retrieving then transferring the raw data to the I2M which converts the raw data into relevant sensor data. After successful packets, have been retrieved by the I2M, the data is transformed into sensor data and passed to the EIM which verifies the firmware and run-time execution of the autopilot and the FIM, which performs analysis at the functional level. The fault alarms flow in the opposite direction, whereas when a fault is detected, it will most likely require the HRIM to isolate the corrupt sensor and initiate a reconfiguration. There exists a multi-master, multi-slave data bus where each sensor has a master-slave pair connecting the I2M sensor to the HRIM IP core. The sensors each have their own data registers for storage of data that can be read from the FIM for further processing and validation. For example, the HRIM contains a UART IP designed to poll data from the GPS sensor, this data is collected from the

UART protocol, then fetched by the I2M, transformed into relevant sensor data such as: Latitude, Longitude, Altitude, etc. and transferred to the FIM to validate the functional correctness of the data. The following section describes the requirements to integrate HECAD into a UAS FCS will be discussed followed by a description of each component of HECAD.

The first and lowest level of the architecture is the HRIM, which analyzes the physical signal characteristics, verifying that the hardware protocol conforms to the configuration and is within its specifications. At this level, it will sniff the data from the communications line passively, minimizing the effect on the characteristics of the system. This is another advantage where the monitor does not need to delay the information transmitting between the autopilot and sensor. The next subsystem is the I2M, which analyzes the information layer, where the data is translated from ones and zeros into sensor data information and can be verified against the sensor specifications. After the data has passed the integrity checks it can be further analyzed in the next subsystem, the FIM. Once the data can be verified that it is within its specifications and legitimate data, further mission specific assertions can be performed on all of the data retrieved from the sensors on the FCS for a more complete solution to determine if a fault has occurred. It is also necessary to not only perform verification checks on the on-board components and sensor data, but to verify that the autopilot firmware executing on the main processor is performing as expected, which can be performed in the last subsystem, the EIM. This allows the monitoring system to perform run-time, functional agnostic checks on the main processor that can verify that it is performing within its intended purpose and the internal control algorithms are executing within their timing constraints. In order to develop a system capable of being embedded on the FCS and monitor the physical signals an FPGA implementation is proposed.

4.2 HECAD Requirements

Integrating HECAD into a UAS FCS has various requirements to maximize its capability for monitoring, detecting, and reacting to attacks or component failures. Each of the requirements may be implemented by one or more subsystems. Requirements for the HECAD include:

- Verification of the firmware and check for consistency
- Fingerprint normal execution of the main processing system
- Verification of hardware level I/O specifications and sensor specifications, and configurations
- Verification and validate system level operation
- Verification and validate the command, control, and communication

In the proposed architecture, HECAD does not notify the main processor in regards to fault alerts (i.e. it does not communicate to the CPU to indicate that a fault has occurred or any information regarding the fault). This disconnect of communication provides an extra layer of security to isolate HECAD from the main processor. Since there is no communication, the autopilot must have the necessary methods in place for when a sensor is disconnected from the main processor to react accordingly. Integrating HECAD onto the PCB of the FCS allows the ability to monitor the hardware level internal communications between the main processor and the on-board sensors on the FCS. An overview of HECAD is shown in Figure 9 the subsystems that collectively form HECAD is contained in the following sub-sections.

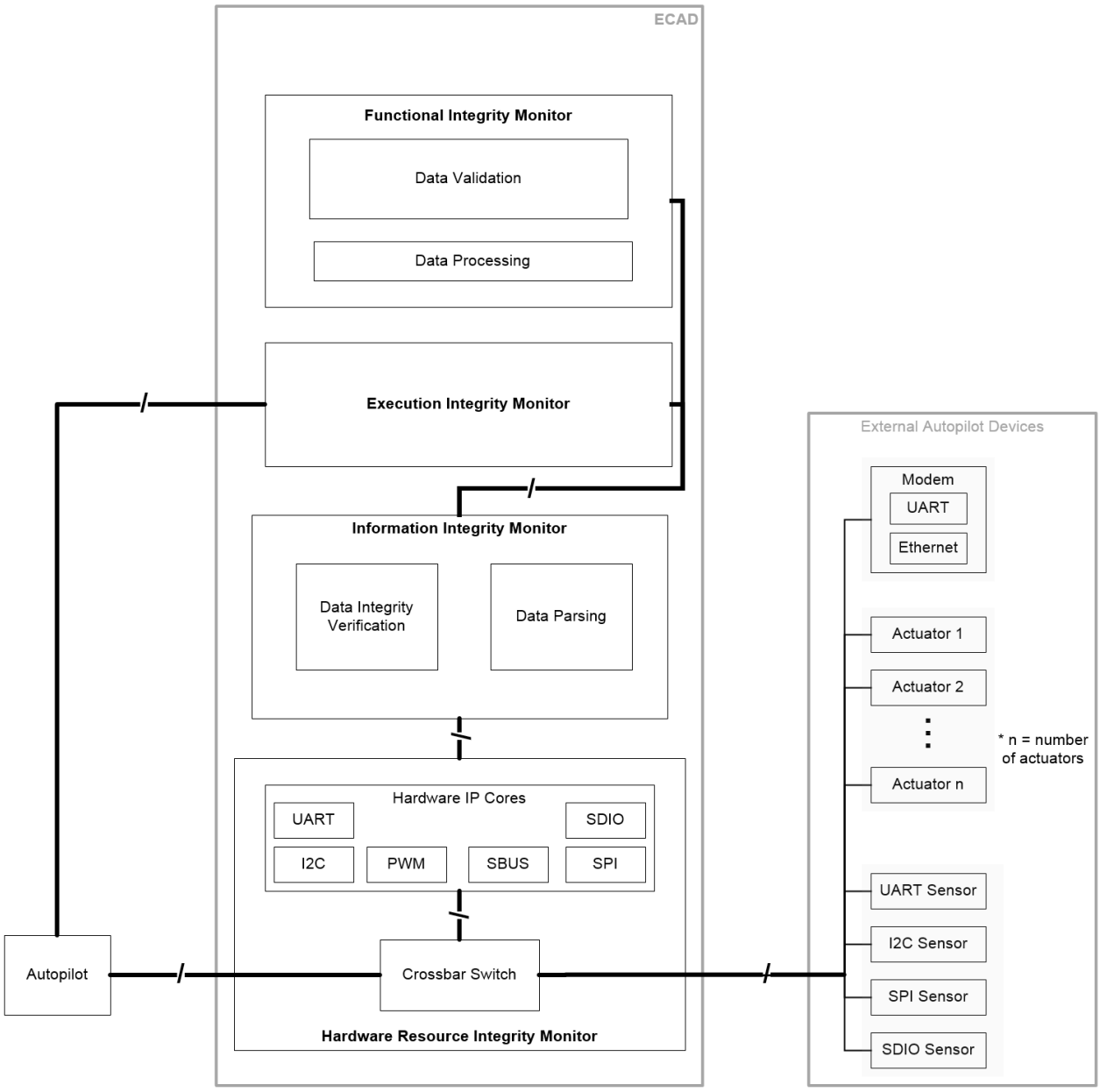

Figure 9 HECAD FCS Detailed Overview

4.3 Hardware Resource Integrity Monitor (HRIM)

The hardware resources in a UAS FCS consist of the data communicating from the on-board sensors to the autopilot, connected through internal communication. At this level, there is no data transformation or knowledge of the sensor data; the module is only aware of the communication protocol specifications. Typical FCS internal communication may consist of but are not limited to: UART, I2C, SPI, and SDIO protocols. The I2C protocol is a shared bus and will be used as an example shown in Figure 10. It is important to note that a protocol such as UART does not share an RX and TX line, as it is not a shared bus protocol. The principles applied to the I2C protocol may be applied to any IP Core that may exist in the HRIM.

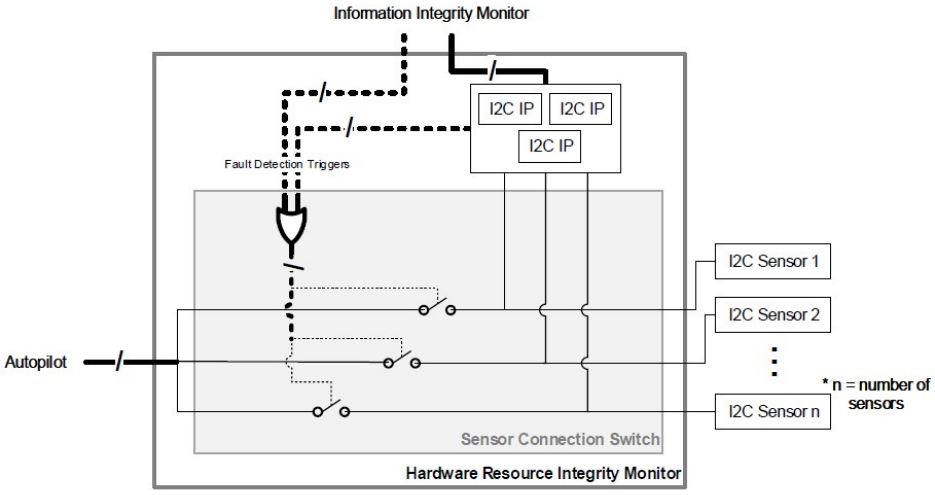

Figure 10 Hardware Resource Integrity Monitor

The HRIM as shown in Figure 10, acts as a hub between the autopilot and on-board sensors acting as a bus protocol monitor, with an additional switch like ability, where in the event of malicious fault detection it can disconnect the respective sensor. The HRIM operates in PASS TROUGH mode until a fault is detected. In this mode, the data from the sensors are

sent directly to the autopilot. The HRIM passively, monitors the data from the line, and directs it to an I2C Intellectual Property (IP) core. Each I2C device is connected through its own individual I2C communication line, and there is an IP core for each device as well. This allows the HRIM to disconnect individual I2C devices without interfering with I2C communication between the other sensors and autopilot.

The HRIM consists of many individual cores, one core per sensor and has knowledge of the bus protocol used for its specified sensor. The detection algorithms at this level consist of hardware peripheral protocol configuration verification and should be simple, real-time, and reliable. A representation of a sensor IP core is shown in Figure 11, consisting of a crossbar switch, protocol specific hardware IP core with a wrapper around the hardware core, and detection logic that uses techniques to detect malicious faults injected into the protocol configuration as well as physical signal faults.

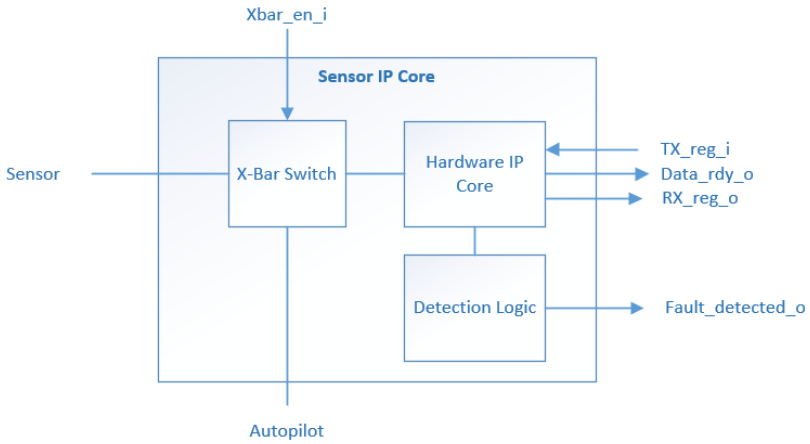

Figure 11 Sensor IP Core

Potential faults that could occur include preventing the data to be sent by holding the line in an idle condition, changing configuration of the communication protocol, sending bogus commands to or from the autopilot, etc. In a typical operation, the update rate of the sensor remains constant, and is monitored as well since slight delays in update rates can

exponentially affect the navigation and control algorithms by accumulating errors over time. When a fault is detected, the sensor is switched to the DISCONNECTED mode, which removes the connection between the sensor and the autopilot. The sensor is still connected to the IP core which may have the ability to react to the fault. Some of the reaction techniques may consist of: resetting the sensor, switching to an alternate sensor, power cycling the sensor, or disconnecting the sensor.

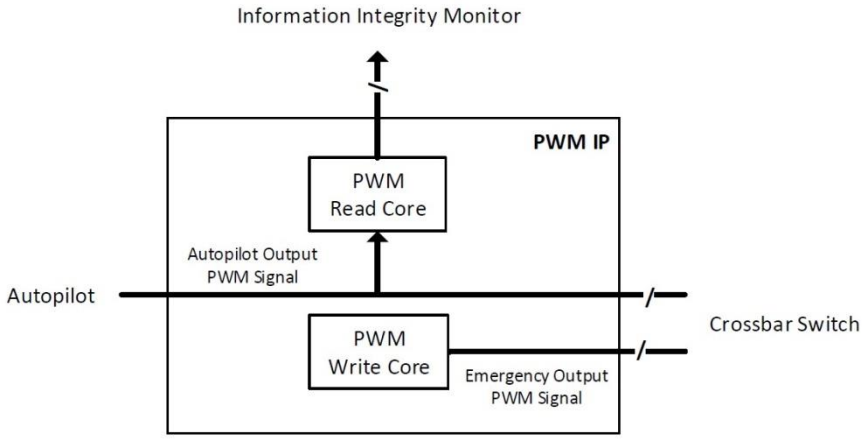

Figure 12 Pulse Width Modulation IP Core

The HRIM monitors all hardware resources, this includes not only sensor communication protocols, but also Pulse Width Modulation (PWM) signals used to control the actuators on the UAS. The IP cores can be customized to accommodate different applications. The PWM IP core monitors the PWM outputs of the autopilot that are controlling the actuators. The PWM IP as shown in Figure 12, has the capability to read/monitor the PWM commands sent to the actuators as well as control the actuators in the event of a fault. Faults on the physical PWM signal can be detected such as monitoring the timing of the pulse width in the HRIM, furthermore the bounds can be checked in the I2M, and lastly the actuator position can be cross checked with the other information from the sensors in the FIM. The HRIM provides the data to the I2M from the hardware IP core,

allowing the I2M to focus on only information integrity, as well as detect faults that occur at the hardware IP core level before it reaches the I2M. A fault that has occurred at the HRIM level may not be detected by the I2M; data can still be received; however, it will be incorrect data and can disrupt the system undetected. With the HRIM in place, it allows multi-level verification of the data so that the I2M has a level of assurance that the data is read correctly and can perform information integrity analysis with fewer false positives.

4.3.1 HRIM UART Example

This section describes the application of the HRIM to a sensor or actuator which uses the Universal Asynchronous Receive and Transmit (UART) protocol. The UART protocol is asynchronous, which means it does not have a clock, and only consists of two signal lines and requires both sender and receiver to be configured with the same settings. This type of setup can easily allow a fault to occur that can be difficult to detect. In a typical processor, the UART is a hardware component that notifies the upper level software that data has been received, however there is usually no indication of any errors with the protocol itself. The UART has various configuration settings, which consist of: Baud Rate, Parity, Stop Bits, and Data Bits. At the software level, if any of these configuration settings are changed during runtime, there will be no indication except for incorrect data being received by the user. The UART relies on the signal being pulled high or low depending on what the IDLE state is, which would signify a start, and uses timing until the whole byte is read into the RX register.

An example of a transmitted byte on a UART is shown in Figure 13 with configuration settings of eight data bits, parity enabled, and one stop bit. The received data byte would be a binary 10101010, or in hex 0xAA. If the configuration of the sender or receiver is changed, the timing after the start bit is initiated will be off and all of the subsequent bits will be misinterpreted.

Figure 13 UART Byte Example

Most hardware UARTs implement a 16X oversampled clock to read each bit received on the UART. Using the oversampled clock, it is possible to detect changes in baud rate. One technique used in HECAD is to record the signal for the first and second half of the 16 clock cycles it takes for each bit. If the configuration of both sender and receiver are the same the signal should remain stable for the whole 16 clock cycles, to account for error all of the clock cycles do not need to be used, however it can reduce the number of faults detected. A technique is used by initially counting half, eight of the oversample clock cycles after a start has been initiated, and then the full 16 for each subsequent bit, this enables the receiver to not only reduce jitter but grab the middle of the bit instead of the beginning or end. By using the oversampled clock, it allows the capability to detect if the baud rate has been modified. This may not always work due to a high jitter signal, however, under normal circumstances, it allows accurate monitoring of the signals to determine correctly a working UART. Correct sampling of a receiving UART is shown in Figure 14, where the oversample clock is storing the middle of each bit, and it can be seen that each half of the 16 clock cycles stays constant if there is a bit change. There can be a worst and best-case scenario where if the data consists of all zeros or all ones it may not be able to detect an error, however the best case would be that the first bit is a high signal, or the opposite of a start signal.

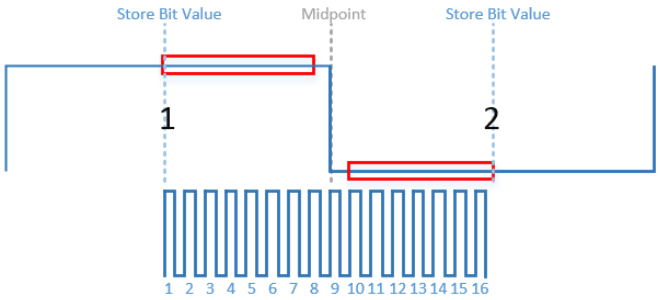

Figure 14 UART Overclock Sample Baud Rate Configuration @ 57600

In Figure 15 is a depiction of what it would look like if the baud rates would differ between the sender and receiver. It shows a higher baud rate from the sender than the receiver. It can be seen that the first and second half of the 16 clock cycles are not consistent and can easily be detected under normal operating condition.

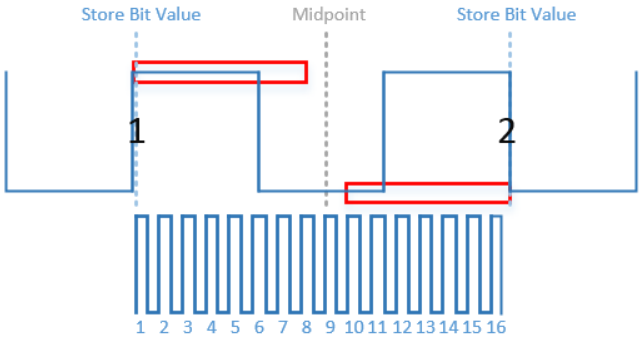

Figure 15 UART Overclock Sample Baud Rate Configuration @ 115200

Other faults that may occur on the UART pertain to the data bits, parity, and stop bits. These faults depend on one another and may not be easy to determine exactly which fault has occurred, however it should still be able to determine that it is a configuration fault. The simplest of the three is the parity, where the parity of the byte can be calculated on the receiving end and determined if the parity bit is incorrect. The stop bits can be checked for

an IDLE signal, if the UART is IDLE high and the stop bits are not also each high, then a fault has occurred. An attack on the data bit configuration would require counting the number of bits, and could possible go undetected, since it would also require the detection of an incorrect stop bit. To determine the correct amount of data bits, would require counting bits from the initiation of the start bit to the stop bit going to IDLE. Lastly, based on the configuration settings the UART is able to also determine if the signal is being held in an active state NOT-IDLE, by counting how long each byte transfer should take. For example, a configuration of 8 data bits, no parity, and one stop bit at a baud rate of 57600, it is possible to determine the time it takes to transfer a whole byte. As seen in Equation 1, it is calculated that the signal should go high approximately 173.6 us after the signal goes low, when IDLE is high.

$$\frac{1}{57600} * (\, 8\, Data\ Bits + 1\, Start\ Bit + 1\, Stop\ Bit\,) = 173.6\, us$$

Equation 1: UART byte transfer time calculation

4.3.2 HRIM Operational Characteristics

This section describes the operational characteristics of the HRIM including what data it shares with other HECAD systems and how it shares that data, and what types of attacks it is intended to detect.

HRIM information provided to other systems

HRIM provides the data read from the hardware IP core and a status register containing the status of the hardware IP core. When data is received on the hardware IP core, a signal is emitted to the I2M sensor module and a read command is issued on the interconnect bus

that requests the data and the status registers from the HRIM. When a fault is detected in the hardware IP core, a fault detected signal is emitted to the I2M and the FIM.

HRIM communication and isolation

The HRIM hardware IP core continuously reads data from the sensors and stores the data into a data register that can be read from an internal data bus connecting the subsystems. When data is received, a signal is emitted to the I2M to request a read from the data register, which occurs in two system clock cycles, which should be much less than a data register update on the hardware IP core. Since the architecture is implemented on an FPGA, there is no dependence between sensor modules at the same level, or within the same data flow of a single sensor module. In the event of a failure at the I2M or FIM level, the HRIM continues to operate without disruption, even if the I2M were to continuously read from the data register, it has read-only access and does not write to the data register, thus the HRIM would be unaffected.

Information Flow between HRIM and other blocks

The information flows from the HRIM to the I2M during normal operation, and when a fault has occurred the information flows from the I2M to the HRIM. In normal operation, data is received by the HRIM and stored in registers. When a fault has occurred, the crossbar is enabled and the sensor is isolated from the system. A mitigation technique is engaged when the crossbar is enabled, which may require commands to be transmitted to the sensor, which is sent from the I2M to the HRIM.

Acceptable region of operation of the HRIM

The acceptable operating region of the HRIM can be defined as a successfully operating protocol without any errors detected in the hardware IP core. The HRIM does not have any knowledge of the sensor information itself, thus in the case of a data fault injection attack, the HRIM is not aware of the attack as long as the hardware protocol is functioning correctly.

HRIM Cyber Attack Characteristics

Faults that are detected at the HRIM can be characterized by configuration parameters of the protocol and the characteristics of the physical signal. An attack in the HRIM level can be caused by firmware modification of a sensor or hardware modification. Along with monitoring the sensors, the HRIM will monitor the power through an ADC which can monitor current or voltage spikes and possibly signatures can be created for operation which can be used to verify the power consumption of the system. An attack on the system may be caused by creating noise or current drain through overconsumption by a specific sensor and/or the main processor through malicious firmware. It is also possible the actuator control signals are maliciously altered through modification of the PWM signal or adding noise on the physical signal.

4.4 Information Integrity Monitor (I2M)

The I2M is responsible for parsing and verifying the integrity of the data received from the HRIM. The I2M consists of a collection of individual sensor modules to represent each sensor being monitored in the system, an actuator module, a power system module, and a communications processing and control module. An overview of the I2M is shown in Figure 16, displaying the configuration where each of the sensors on-board the UAS FCS has a

dedicated module in the I2M which will perform the parsing of the data which translates the raw data into usable sensor data. The I2M is customizable and can be modified to add more sensors or remove sensors depending on the application, since each module is independent of each other in the I2M; e.g. the GPS module does not depend on the IMU module. The actuator module is used to monitor the actuator outputs from the autopilot as well as the inputs received from the manual operator. The power system module is used for performing analysis on the power system of the autopilot, this can include various power sources such as the input power source to the FCS, and any other regulated power source on the FCS. The communications processing and control filters through the communications between the autopilot and the GCS operator, which consists of status updates relayed from the autopilot, commands sent from the operator, and control inputs from the operator such as parameters, gain values, limits, etc.

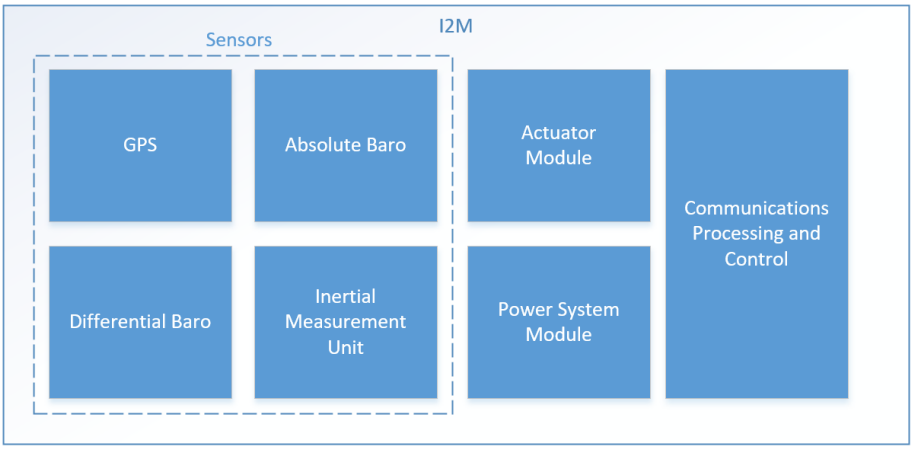

Figure 16: I2M Overview

An individual sensor module is shown in Figure 17, more specifically, in this case, a GPS sensor. Each module in the I2M is connected with an IP core in the HRIM which signals when data is ready on the data bus, at which time the module in the I2M reads the data placed on the bus. The I2M has a direct dependency on the HRIM, since it can only parse data that has

been read from the sensor using the hardware IP core. If the HRIM were to lock up or become corrupt, the I2M would not receive data from the sensor, and would therefore trigger a fault since it will expect data to be received at a specified frequency, and will treat it as a sensor attack due to the inactivity.

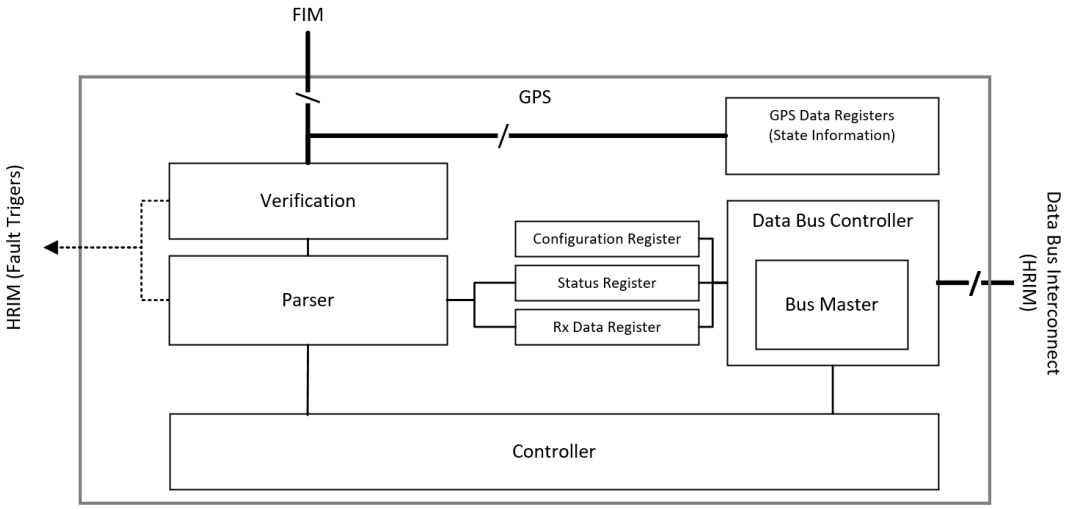

Figure 17 Information Integrity Monitor

Each sensor-specific module in the I2M is equipped with a similar set of blocks. As the data is received on the data bus in the I2M, it is placed in the parser. After the data is parsed, it is stored in a data register and passed to data verification checking if the information meets the sensors specifications. Each sensor module within the I2M contains its own set of 64-bit registers to store data in fixed point representation. The configuration register in the I2M is used to configure the IP core at the HRIM level. The status register contains the status of the integrity of the data, such that if data verification identifies an attack, other subsystems of HECAD can be notified. The I2M is aware of the type of sensor and can parse the raw data to create the relevant sensor data. The integrity of the data can be verified by using techniques that detect faults that may occur in the form of: parser errors, repeat values, data update rate, all zeros, and bounds checking. The data is verified and passed to the FIM for further

processing in parallel. The I2M verifies that the information received meets the sensors specifications and current configuration. Parser errors can occur through a man in the middle attack where the data is corrupted before it is received by the processor. Repeat values can indicate that a sensor has been corrupted in a way where the values do not change or the sensor contains a faulty update rate. Bounds checking can be applied to the sensors data through the known minimum and maximum values which are typically set by the user in the configuration, or could be determined by the type of data, such as latitude measurements which should lie between ±90°. In addition to sensors the I2M provides verification of the command, control, and communications between the UAV and the GCS.

The I2M is where the raw data read from the sensor is translated into sensor data which can be further analyzed for faults. Holistically, it is adding a level of verification after the HRIM and before the EIM and FIM, which verifies that the data the other subsystems are receiving is legitimate data and not faulty data, potentially allowing for better detection of faults/attacks as well as locating where the fault originated. This also allows multi-level verifiable fault detection method, where the detection can be a three-state detection scheme instead of a two state, which the states would be NO FAULT and FAULT. In a three-state system, the states could be NO FAULT, POSSIBLT FAULT, and FAULT; where the subsystems can work together to verify that a possible fault is in fact an actual fault, reducing the number of false positives.

4.4.1 Communications Processing and Control

The main operation of the Communications Processing and Control (CPC) module in the I2M as shown in Figure 18, is to monitor the communications channel between the UAS and the ground control station operator. This channel contains the periodic transmission of the status of the system, commands sent from the operator to the autopilot, expected response from commands, and parameters set in the autopilot by the operator. The CPC currently

contains a Cyclic Redundancy Check (CRC) that verifies the checksum of the packets sent to the autopilot. The CPC may include techniques to detect denial of service either by loss of communication from the autopilot or a sudden increase in bandwidth of data. The commands sent from the operator are to be verified using techniques to determine the sender, as well as checking for correct status of the system and the type of command set, i.e. setting the system to IDLE during flight. In the case of malicious command packets being received, the integrity of each command message is verified to identify replay attacks as well as malicious data injected into the received data.

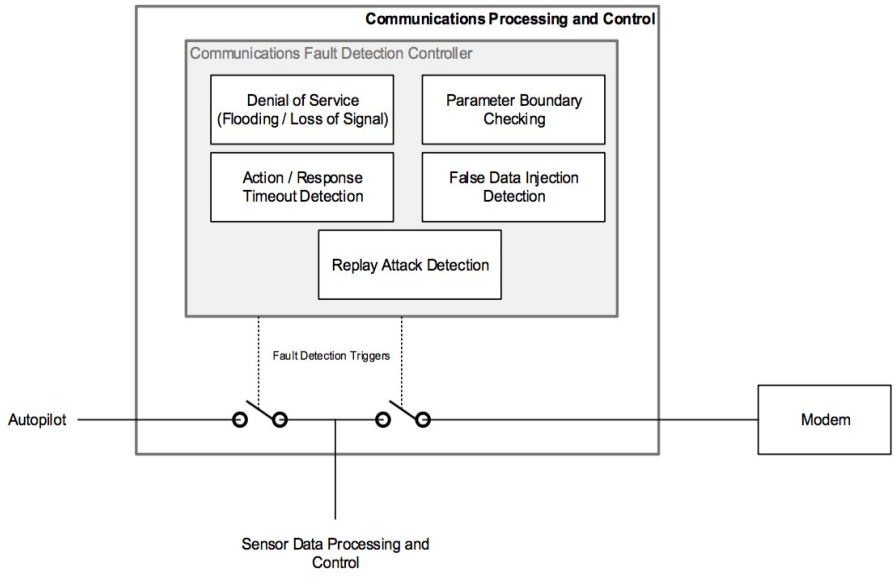

Figure 18 Communications Processing and Control

Lastly, parameters may be sent through communication to the autopilot from the operator, such as actuator limits, control algorithm gains, parameter limits, etc. These parameter values are verified for correctness to make sure that a malicious parameter does not hinder operation and the UAS can maintain stable operation.

The CPC requires knowledge of the commands sent from the GCS operator to the autopilot, configuration, parameters, and gain values of the control input from the GCS, and lastly

knowledge of the communication protocol. Detection algorithms of attacks in the command subsection would consist of anomaly detection with the aid of a state estimation cross-verification algorithm and comparison operators to verify that the navigation and flight limit parameter values are within reason.

The control module could detect anomalies as well the distance of the values from previous values to verify gain value changes and ensure correct stable flight. A threshold and comparison operation can be used to verify the min and max of the parameter values, as well as comparing the state estimation outputs in the communication status updates to the GCS with the physical actuator outputs for an extra level of verification.

Lastly, the communication module would require an algorithm to analyze the data received from the GCS to the autopilot as well as vice-versa. This allows the detection of a firmware attack as well as a network attack from the communication channel. The I2M detects various attacks such as: an increase in update rate, verification of time-stamps on packets, and detection of random malicious configuration packets. This minimizes the vulnerability to denial of service, fuzzing, and replay attacks.

4.4.2 I2M Operational Characteristics

This section describes the operational characteristics of the I2M including what data it shares with other HECAD systems and how it shares that data, and what types of attacks it is intended to detect.

I2M communication and isolation

The I2M stays in an IDLE state and waits for the HRIM to trigger a data ready signal, which means the hardware IP core has received data; the I2M immediately notifies the data bus master controller, and requests data on the bus from the HRIM. After the data is received, the data is fed through the sensor parser in the I2M and if a sensor data update has been

made, i.e. a whole packet has been successfully parsed or data has passed all checks, a signal is triggered to notify the FIM and EIM that new sensor data is available.

The information flows from the HRIM to the I2M, and from the I2M to the EIM and FIM when in normal operating conditions; if a fault is to occur in any of the subsystems, the information flow changes direction between the HRIM and I2M, and depending on the mitigation procedure, data flows from the I2M to the HRIM, (e.g. commands to reset the sensor) and the information flow stops between the I2M and EIM and FIM.

I2M information provided to other systems

The I2M translates the raw data read from the HRIM into legible sensor data that can be used for further analysis. The parsed data is stored in memory on HECAD that can be directly accessed by the FIM and the EIM. A signal is triggered when new data is available and the other subsystems can read from the data registers. The I2M also provides the recovery data to the HRIM in the case of a sensor reconfiguration The HRIM is configured through the I2M and has no knowledge of the sensor specifications or configurations prior to receiving its configuration from the I2M.

I2M Cyber Attack Characteristics

Attacks in the I2M would be modifications of the sensor data that would exceed the specifications of the sensor, such as changing the data rate of the sensor, modifying the timestamp, or providing data out of bounds of the sensor specifications reading for a GPS sensor. These attacks should not be application specific, but sensor specific, since the verification is done per the sensor specs.

4.4.3 I2M GPS Sensor Example

To demonstrate the operations of the I2M, the GPS module will be used as an example. The I2M verifies the integrity of the data through various techniques that may consist of the techniques in this example but are not limited to them. The verification at the I2M can be application specific or taken from the specifications of the sensor. For the purpose of this example the GPS sensor I2M verification will adhere to UAV autonomous waypoint navigation.

A control flow graph of the GPS module in the I2M is shown in Figure 19. Every sensor module in the I2M remains in an IDLE state until a data ready signal is triggered from the HRIM to alert the I2M that data has been received from its respective sensor. Once the data ready signal has been triggered the I2M begins to perform its verification, beginning with requesting the data on the bus.

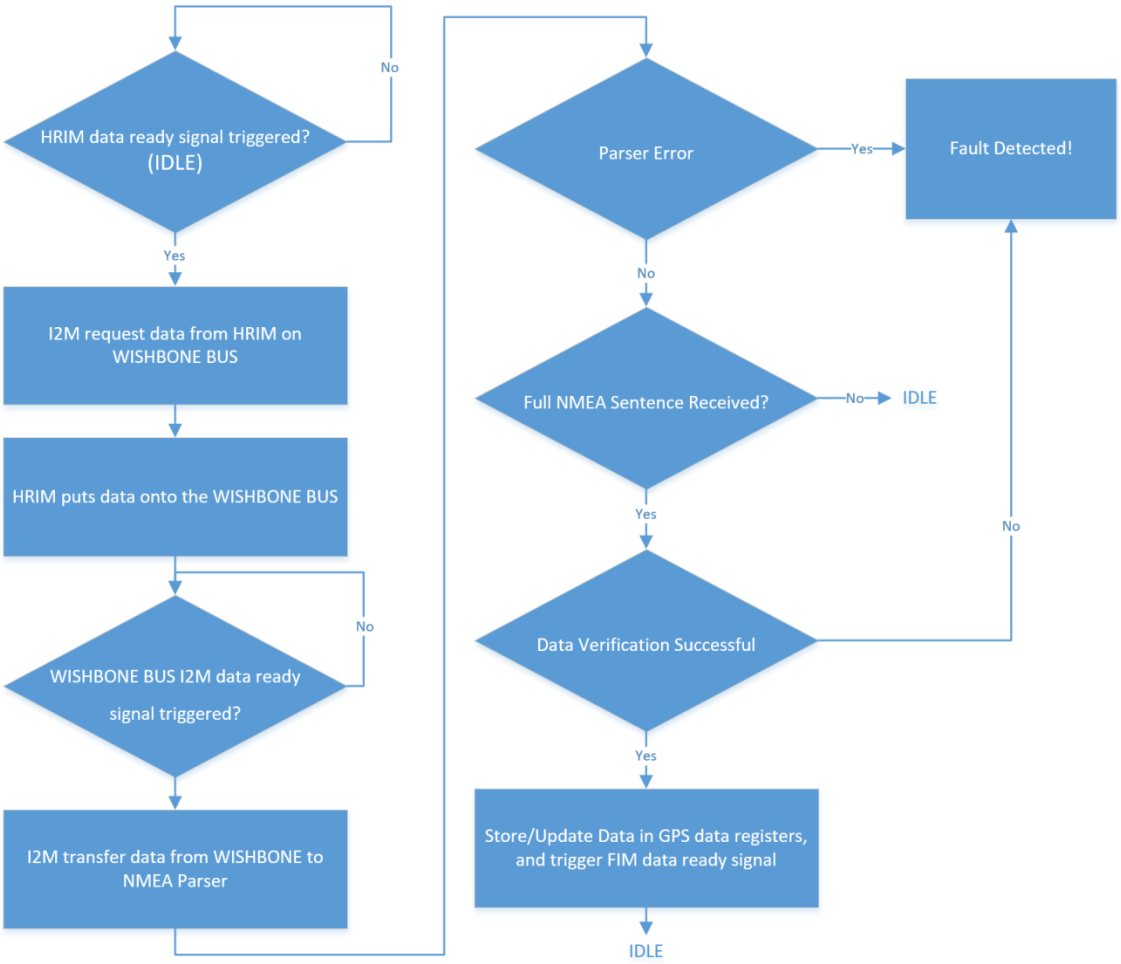

Figure 19 Control Flow graph of GPS module in I2M

The GPS module in the HRIM responds to the request by placing the data received by the hardware IP core onto the bus. Once the data is transferred to the I2M it is passed to the NMEA parser. The NMEA parser verifies that the data received is a valid ASCII character (specified by NMEA sentence requirements) and triggers an error signal if the data does not meet the specifications or once a whole NMEA sentence has been received and the checksum is incorrect. If a full NMEA sentence is received and the checksum is valid it is passed to the data verification module, otherwise, it goes back to IDLE and waits for the next byte. The specific data verification techniques are not in the scope of this research, however rudimentary techniques can be used to demonstrate the ability of the I2M. In the I2M, the

verification techniques are currently performed on the raw data, where as some sensors may require a conversion that relies on floating point mathematics; to improve performance and save area on the FPGA, these types of conversions are performed in the FIM which uses a soft core processor to perform the conversion instead of digital logic.

A technique used for data verification is bounds checking. To perform bounds checking the I2M must be aware of the sensors specifications as well as the applications parameters. The bounds of the GPS data can be verified on, for example, the Longitude and Latitude which must remain within ±180° and ±90° respectively. The altitude of the UAV can be bounded to positive measurements and a ceiling specified by the application. Other measurements such as airspeed and velocity can be bounded by the capability of the UAV. Once the data has successfully passed all of the verification techniques, it is stored in data registers, and a signal is triggered alerting the FIM that new data has been updated and the FIM requests the data on a second bus separate from the bus that communicates between the HRIM and I2M.

In the event that a fault is detected on the I2M, a signal trigger enables the crossbar switch in the HRIM isolating the sensor from the main processor. Currently the sensor is isolated by asserting an IDLE state on the RX and TX lines of the autopilot to prevent the faulty sensor from communicating with the autopilot. After the sensor is isolated, the I2M initiates a mitigation routine which may consist of power cycling the sensor, issuing a reset command, or switching to a redundant sensor. Mitigation techniques are out of the scope of this research and can be discussed in future work. To demonstrate the capability of the I2M, a baud rate attack is performed on a GPS sensor in the results section, where the HRIM detects the attack, isolates the sensor, notifies the I2M and the I2M resets, reconfigures, and reconnects the sensor to the autopilot

4.5 Execution Integrity Monitor (EIM)

Autopilot firmware is updated for various improvements regarding bug fixes, new hardware support, security, etc. This paradigm of updating the firmware on a FCS autopilot creates a vulnerability where malicious code can be inserted into the firmware. Malicious code can be inserted at many stages of the development of the FCS which begins in manufacturing, then to distribution, operator firmware upload, and maintenance.

Malicious firmware may be inserted in the manufacturing or distribution phase with default firmware loaded onto the hardware. It can also be inserted through the form of a website mirror attack, or through a malicious maintenance technician. It is not common for low cost COTS FCSs to have any type of on-board checking of firmware since most autopilots used by the civilian community are constantly updated and modified by the operator. Once the FCS has been delivered to the intended operator from manufacturing and distribution, new firmware may be downloaded from the web and programmed onto the autopilot. In this case, vulnerabilities may lie in downloading the firmware from a website, which may be re-directed or mirrored causing a download of malicious firmware. Once the firmware is downloaded, the hardware programming device used to load the firmware onto the autopilot may be compromised

The HECAD framework accounts for firmware vulnerabilities within the EIM through hardware and software techniques to monitor the control flow of the firmware and perform run-time verification. The module is connected to the autopilot's main processor with the capability of reading internal registers and program memory contents for firmware verification as well as any other necessary connections needed to monitor execution during run-time. It also contains a fault detection alarm to alert the other functional blocks when a malicious fault has been detected. The EIM works in conjunction mostly with the HRIM and I2M to access the debug interface of the main processor as well as cross verify the data of the internal registers of the main processor with the data received from the sensors. It will also

be operating with the FIM for higher level analysis of the application specific functionality providing timing and code execution information.

The initial firmware verification is performed on boot-up/initialization of the autopilot, at which time techniques are used to verify the memory contents of the autopilot CPU, such as using a hash chain [32] with a random challenge, which may require a static hash to compare to. If malicious code is inserted in a way that it is disguised from the firmware detection techniques, run-time monitors check for any abnormal behaviors during execution. Some techniques that can be used for run-time execution can be divided into software and hardware solutions. Software based solutions may require an invasive solution, such as inserting redundant instructions into the program to detect deviations in the program execution flow [33], [63] [64]. Numerical safety assurance can be used to periodically monitor the autopilot to check for inconsistencies in performance or memory contents during run time execution Hardware based methods may require external hardware such as a watchdog processor [65] [66] and lock-stepping [65] which monitor the state performance and state of the master processor, checking the execution flow of the program or trace memory accesses. Another technique is to monitor the instantaneous current drain on the system through power fingerprint monitoring [66]. This also requires certain specifications of the autopilot operations to be known by the HECAD, in the form of instructions, algorithm frequencies, sensor update rates, etc. Upon successful verification of the autopilot firmware, HECAD transfers authority of the on-board sensors to the main processor. In the case of a fault in the firmware verification, HECAD has the ability to prevent the main processor from communicating with the sensors and controlling the actuators. The EIM observes the internal memory of the main processor to verify register contents and possibly the program counter and/or instruction registers to validate program execution. Future work will include implementing the aforementioned techniques as well as artificial neural networks, fuzzy logic, and other methods of reinforcement learning techniques.

4.5.1 EIM Operational Characteristics

This section describes the operational characteristics of the EIM including what data it shares with other HECAD systems and how it shares that data, and what types of attacks it is intended to detect.

EIM communication and isolation

Upon bootup the EIM should enable the crossbar switch to keep the sensors from communicating with the autopilot until it has successfully verified the contents of the firmware on the autopilot. Once the EIM passes the initial program memory check, a signal is passed to the HRIM to disable the crossbar switch and allow the sensor communication to pass through to the autopilot. During run-time, the EIM communicates with the I2M and FIM periodically to verify that the data registers match the sensor data as well as provide information with regards to internal control and navigational algorithms such as timing characteristics, program execution, or registers used to hold dynamic information.

The EIM is mostly separated from the other blocks since the HRIM, I2M, and FIM mostly deal with the communication between the autopilot and other on-board systems. The EIM communicates to the HRIM and I2M to access the main processor through a debug interface such as JTAG, as well as power fingerprinting. It will pass data from the main processor to the I2M and the FIM if internal register data is needed for comparison, as well as signal whether a fault/attack has been detected.

EIM information provided to other systems

Depending on the observability of the system, the EIM may be able to provide the I2M and FIM information regarding the systems calculations and received data. If the EIM is capable of reading internal registers, it can cross-verify the data read by the sensor in the

system with the data read by HECAD in the I2M. The FIM would be able to also take advantage of the control and navigation algorithms to check for runtime manipulation such as Not a Number (NAN), divide by zero, or other values that may cause the system to lock up or behave unexpectedly. The EIM would also be able to verify if the configuration of the hardware peripherals changes during runtime unexpectedly.

EIM Cyber Attack Characteristics

The EIM monitors the systems static and dynamic runtime operations to verify that the execution integrity of the system is correct. Static attacks may include adding, removing, or modifying the software that is programmed on the main processor. Dynamic attacks may be stealthy attacks that can be hard to detect, possibly a multi-level attack such as modification of the data rate of a sensor that could offset the timing of the algorithms executing on the main processor. It may also be in the form of a passive listener inserted into the system that listens and gathers information without disrupting the system.

4.6 Functional Integrity Monitor (FIM)

The FIM monitors the functional integrity of the system holistically, validating its behavior according to its specifications. The FIM actively monitors the communications between the operator and the UAV, the actuator commands, and the sensor data through application specific requirements to determine if a fault has occurred. An overview of the FIM is shown in Figure 20. The FIM performs analysis in parallel with the system being monitored, which may include performing a rudimentary state estimation of the system, which could also be used to control the UAS in the event of a failure in the autopilot software. Another technique that can be used by HECAD for fault/attack detection is cross sensor validation which requires a system to be equipped with multiple sensors that

provide similar measurements, such as altitude and/or airspeed provided by a barometric pressure sensor and a GPS.

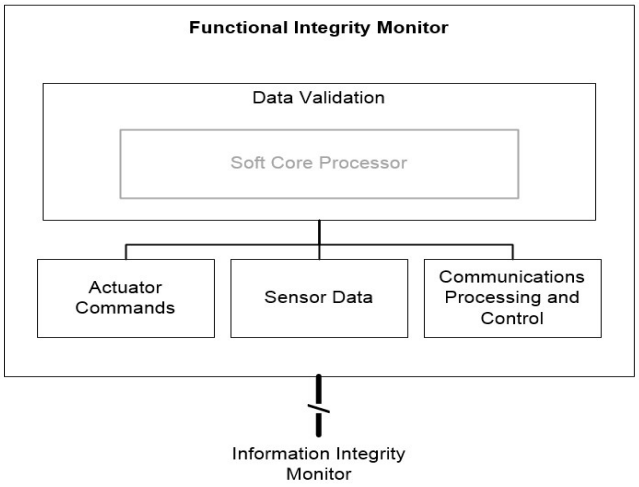

Figure 20 Functional Integrity Monitor

The FIM can monitor the rates at which each of the sensors is changing and determine if one of the sensors deviates past a certain threshold, which can be a part of the requirements for a specific application. The FIM can be tailored to perform any higher-level analysis within the capabilities of an FPGA, to validate the user-specified requirements. The specific algorithms used in the FIM are out of the scope of this research and is a topic of future research. To get a better understanding of how the FIM will operate, a use case will be presented to demonstrate how a list of requirements may be developed for a specific application.

4.6.1 Use Case: UAV Surveillance Requirements

One of the many applications a UAS may be used for is surveillance of a specified area, typically including a payload of a camera actively recording video the area below, and for this example it will stream the video down to a ground control operator. The first set of

requirements will be designed for the control of the UAS. For the purposes of this example, the UAS has 5 different modes of operation: IDLE (on the ground, throttle in the off position), TAKE OFF, WAYPOINT, SURVEILLANCE, and LANDING. Since the FIM has access to the communications information between the operator and the UAS, it can detect which mode the UAS is currently in and change requirements dependent on this information. The following set of requirements was developed to be applied towards the SURVEILLANCE mode of operation:

- The UAS will not roll greater than 45° when in SURVEILLANCE mode
- The UAS will not pitch greater than 10° when in SURVEILLANCE mode
- The UAS airspeed will not exceed 40 knots and maintain a ± 3 knot variance when in SURVEILLANCE mode
- The UAS altitude will not exceed 200 meters and should maintain a ± 5-meter variance when in SURVEILLANCE mode
- The UAS will not roll or pitch at a rate greater than 10°/s when in SURVEILLANCE mode

The next list of requirements applies to mode changes:

- Mode change WAYPOINT/SURVEILLANCE -> IDLE is prohibited
- Mode change IDLE -> WAYPOINT/SURVEILLANCE/LANDING is prohibited
- Mode change TAKE-OFF -> IDLE is prohibited when altitude > 0
- Mode change LANDING -> IDLE is prohibited when airspeed > 0

The requirements are then placed into code onto the FIM where they can be used to verify the assertions on the data in HECAD. A block diagram of a high-level view of HECAD is shown

in Figure 21 with an example of a false data injection attack. A false data injection attack may occur at the software level where the data is modified; if the data makes it to the FIM level then it is a more difficult attack to detect since it passed all of the validation of the I2M integrity as well as packet checksum verification. The block diagram shows the physical signal data received at the hardware level in the HRIM, then the raw data is passed to the I2M, where it is parsed and validated for integrity, and finally it is read into the FIM, which for this example uses a previous value vs. the current value to check for consistency. It can be seen that the altitude suddenly jumps from 60 to 150 meters, assuming the target altitude is approximately 60 meters, the control algorithms would attempt to correct the current altitude to reach the target altitude, causing a sudden dive in the UAS potentially crashing the plane.

Since the requirements are in place HECAD will react to the malicious fault immediately as shown in Figure 22. The FIM retrieves the sensor data from the I2M, and performs a cross-verification analysis on the data and detects that there is a large deviation between the altitude measurements. The FIM notifies the I2M that a fault has occurred, then the I2M triggers the HRIM crossbar switch enable signal, which will isolate the GPS sensor from the autopilot. Once the sensor is isolated the I2M stops receiving data from the HRIM and initiates its mitigation procedure.

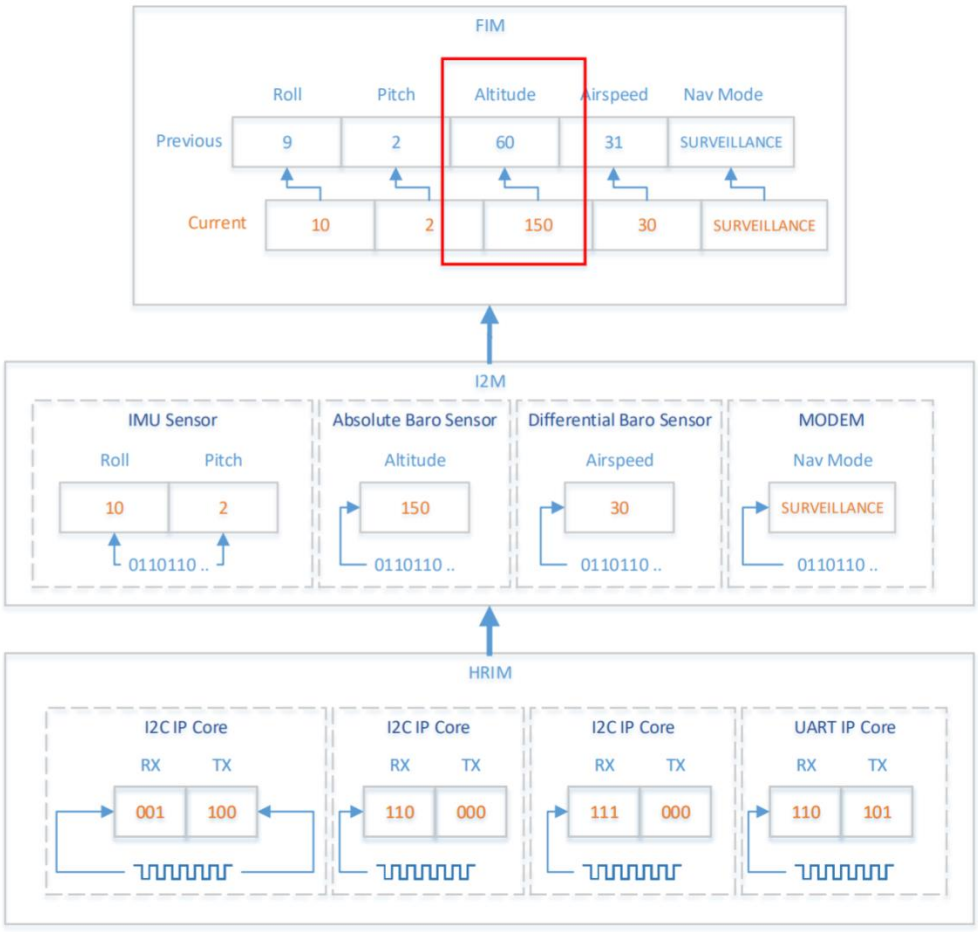

Figure 21 FIM Requirements Example False Data Injection

The mitigation procedure can range from many operations which may include resetting the sensor or switching to another on-board redundant sensor, the method used for mitigation faults are outside of the scope of this research and will be explored in future work. The list of requirements is application specific and can be as complex as needed. A detailed finite tree including the requirements and any application specific UAS tasks can be constructed to create a series of assertions to be used in the FIM alongside cross sensor verification and any other methods that may be necessary to assure integrity at the functional level of the system.

The FIM requirements can be determined offline and should be tailored to a specific application that the UAS will be performing. The FIM has the advantage that it will be executing on a soft-core microprocessor, so reconfiguration of the methods is relatively easy compared to reconfiguring algorithms developed in programmable logic as well as perform necessary calculations which may require floating point operations. One example of a soft-core processor is the Xilinx MicroBlaze, which is used for implementation, testing, and validation of the HECAD architecture. A hard-core microprocessor can have unpredictable behavior in an interrupt driven application or with the use of additional accelerator hardware. In the case of a soft-core processor, such as the Xilinx Micro Blaze, these unpredictable features can be disabled to improve predictability and decrease security risk.

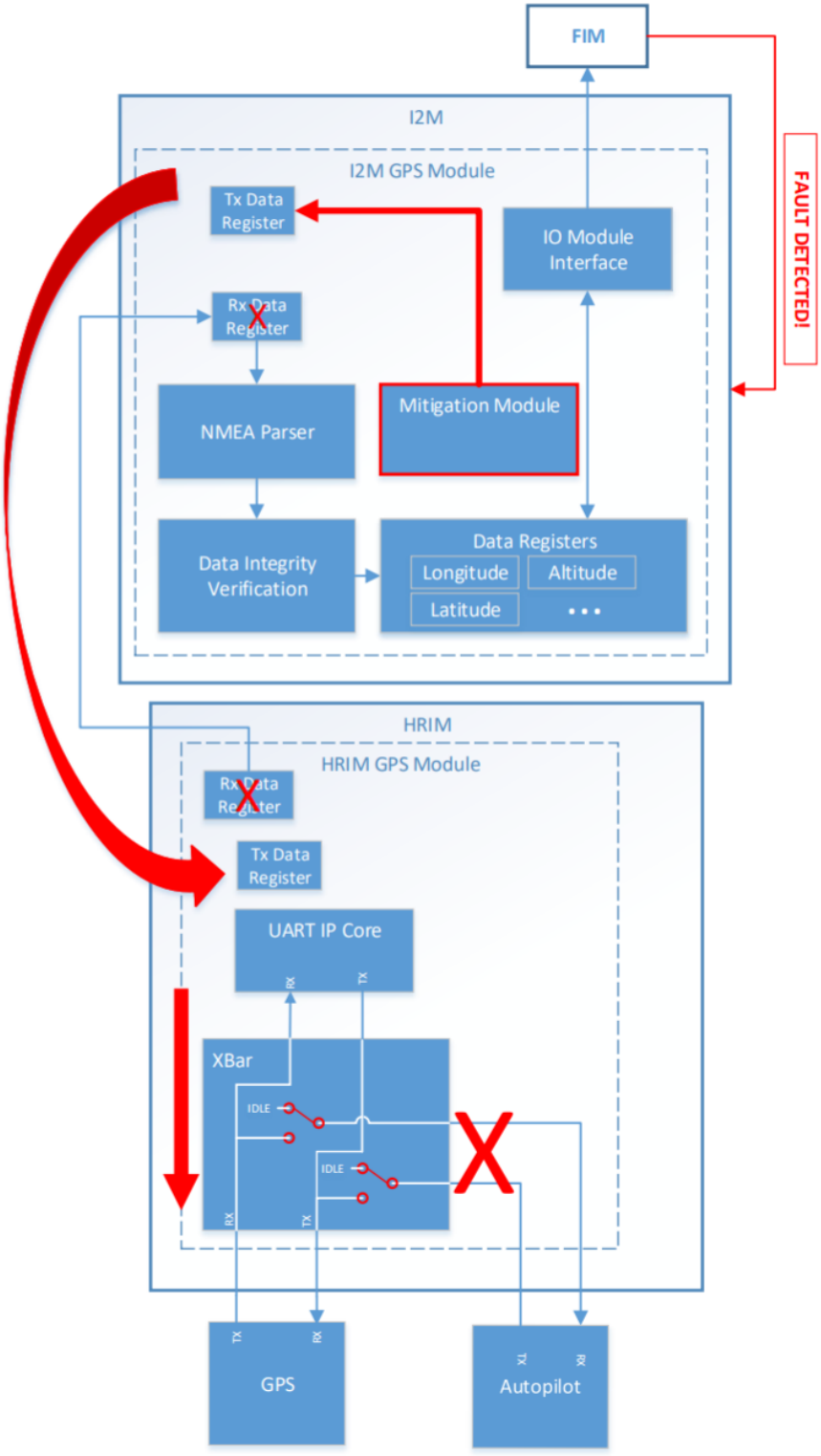

Figure 22 FIM Fault Detection and HECAD change of information flow

4.6.2 FIM Operational Characteristics

This section describes the operational characteristics of the FIM including how it communicates with other HECAD systems, and what types of attacks it is intended to detect.

FIM information provided to other systems

The FIM is the top level of the hierarchical architecture and does not supply any information to the other sub-systems, however it can be used collaboratively with all of the other sub-systems to collectively decide if a fault/attack has been detected. It does supply signals to the other sub-systems when a fault/attack has been detected. The FIM communicates to the other sub-modules only to specify if a fault/attack has been detected. If a lock up or failure is to occur in the FIM, since it is at the top of the hierarchy, it will not affect the rest of HECAD since the other sub-modules do not rely on any information from the FIM.

FIM Cyber Attack Characteristics

Characteristics of a cyber-attack at the FIM level may be the most difficult type of attack to detect since it requires detailed domain knowledge of the application as well as the capabilities of the UAS. An attack may be in the form of a GPS walk off attack where over time the UAS slowly drifts and may stay within the requirements, rendering it a stealthy attack. False data injection attacks may occur in the FIM where all integrity checks pass, requiring more experience from an adversary and extensive knowledge of the sensor.

4.7 WISHBONE Bus

The WISHBONE Bus is a lightweight interconnect architecture designed for interfacing portable IP cores in reuse designs that need flexibility. It was originally created by the

Silicore Corporation and the stewardship was later transferred to the OpenCores project. The other similar bus architectures include: AMBA, Avalon Bus, IBM CoreConnect, PLB, and OCP, however these architectures are not open source and require a license to use. The WISHBONE Bus is lightweight, which requires minimal number of signals, and can be fully customized for various architectures. The WISHBONE bus is used in the HECAD to implement the communications between the various hierarchical levels of the architecture.

WISHBONE Signals

The WISHBONE architecture consists of 6 signals, an address bus, and a read and write data bus. The address bus is 16 bits and the data bus is set to 32 bits; however, each bus can be modified if necessary. The control signals controlled by the master are the CYC, STB, SEL, and WE signal. The SEL signal is not used in this design. The STB and CYC signals are set high when the master would like access to the bus, when they are high it means that the bus is busy and a master is currently transmitting or receiving data. The WE signal is the write enable signal, it specifies if it is a write (high) or a read (low).

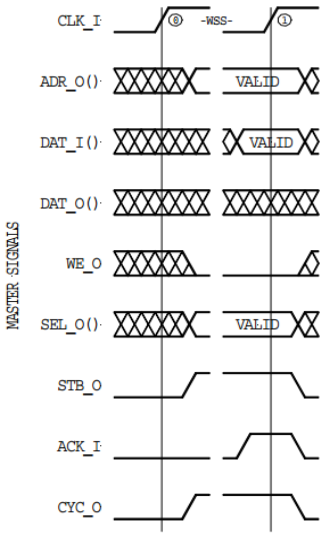

Figure 23 WISHBONE Single Read Cycle [67]

The slave control signals are ACK, ERR, and RTY. The slave sets ACK high on the following clock cycle after a master asserts STB and CYC. If there is an issue, such as incorrect addressing, the slave asserts the ERR signal, if the bus is busy the RTY signal is asserted. A transaction on the WISHBONE bus takes 2 clock cycles. A single read transaction is shown in Figure 23.

Shared Bus Architecture

HECAD uses a shared bus architecture, which requires a slightly more complicated interconnect, however it allows for less additional logic required for each additional master outside of the WISHBONE bus. The shared bus architecture in HECAD uses a multi-master multi-slave bus. Each of the sensor modules in the I2M contains a WISHBONE master, and each sensor hardware IP core contains a WISHBONE slave.

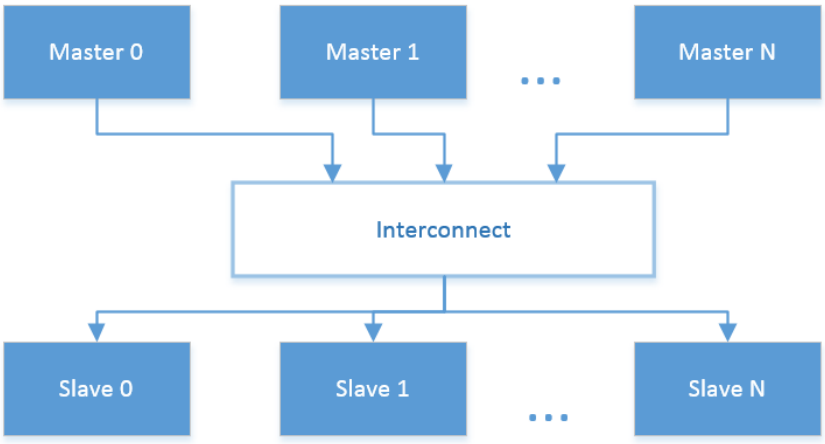

Figure 24 WISHBONE Shared Bus Architecture

WISHBONE Interconnect

The WISHBONE interconnect is the hub that connects all of the masters to the slaves. To accommodate a multi-master multi-slave architecture, some modifications were necessary such as integrating an arbiter, which determines which master can use the bus; only one master can have access to the bus at any given time. An example of the arbiter's connections is shown in Figure 25, demonstrating one master and one slave on the shared bus to reduce complexity, however it can be expanded to multi-master and multi-slave.

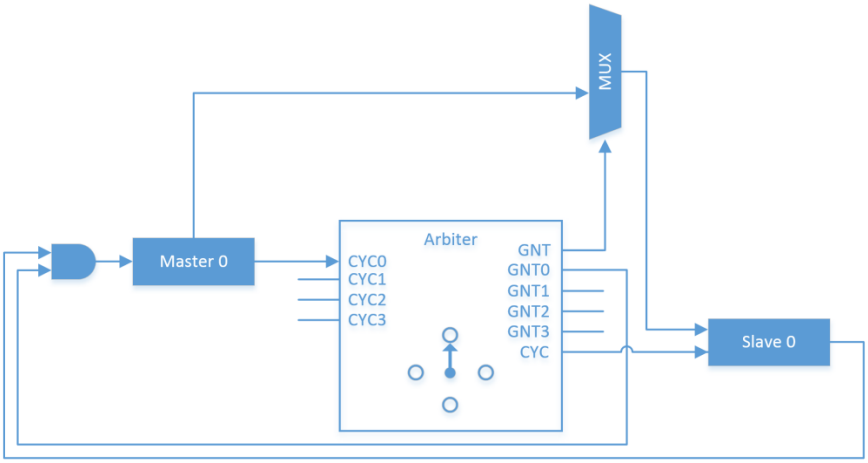

Figure 25 WISHBONE Bus interconnect arbiter connection example

The chosen arbiter method for HECAD is the round-robin arbiter, which allows each master a chance to access the bus. There is no need for priority since each transaction on the WISHBONE bus takes two clock cycles, which assuming a typical UAV has < 10 onboard sensors polling at a worst-case scenario of 200 Hz. Assuming the system clock frequency of HECAD is 100MHz, this would allow for a maximum of 50M transactions per second, which is much greater than 10 sensors polling at 200 samples/second, which would result in 2000 transactions per second. At the worst case, a sensor would have to wait (2 clock cycles) * (# of masters on the bus), which for 10 sensors would be 20 clock cycles, at 100MHz would be a maximum of 5MHz sampling rate per sensor, which verifies that a round robin arbiter suits

a UAV application. The advantage of using the custom open source bus is that the arbiter may be customized to accommodate various applications, in the case that a priority architecture is necessary.

The operation of the interconnect starts once the CYCX signal of the master is asserted, as long as no other master is using the bus, the GNT signal is asserted which allows the signals of the master to pass through to the slaves on the bus, the interconnect asserts the signals for the specified slave device determined by the address put on the bus, where the interconnect enables the specific mux passing through the master signals to the slave. The master that owns the bus at any time will have the GNTX signal asserted, and the slave signals are passed through to the master for acknowledgment of the transaction.

4.8 Securing HECAD

When designing a security monitor, it is necessary to consider the trade-off of security vs. information gathering ability. The idea behind developing HECAD was to design a cyber-attack monitor that was capable of gathering as much information about the system as possible to maximize the detection capabilities. In order to cross the hardware/software boundary and monitor the hardware peripherals of the system, it was required to integrate the monitor onto the hardware of the FCS in order to not only monitor the hardware peripherals but isolate the slave devices. By integrating the security monitor on-board it also creates an additional attack surface, where an adversary could disable HECAD, therefore disabling the whole system. It was decided that the amount of effort required to disable HECAD would be very high and require physical access to the FCS, and therefore focused on gathering as much information as possible over securing the security. Nevertheless, various methods for securing FPGAs do exist. Common threats to FPGAs may include an adversary cloning an existing FPGA bitstream and promoting it as their own, reverse engineering a

bitstream, where an adversary recovers the bitstream through the circuit design, tampering through added logic, spoofing by replacing the bitstream and re-selling the device, or destruction of the FPGA and/or replacing the FPGA with an identically manufactured one. Methods to secure FPGAs were surveyed in [68] and a brief overview of the methods will follow.

SRAM and Anti-fuse

The FPGA bitstream can be stored on the FPGA using SRAM, Flash, or Anti-fuse. An SRAM configuration is volatile memory, therefore requiring power to keep its contents and requires external permanent storage, which is possible for an adversary to attack. Flash memory is a form of non-volatile permanent storage; however, it commonly requires In System Programming (ISP), which exposes the FPGA to the same issue as SRAM. Lastly, an Antifuse FPGA is a one-time programmable method that is disconnected by the manufacturer, providing a non-accessible permanent memory storage, however since it can only be programmed once, it poses other security concerns.

Bitstream Encryption

An FPGA implementation is inherently more secure over a general processor since it uses a bitstream over program memory, and would take significantly more time to reverse engineer, however, for further protection bitstream encryption can be used. The most common method of securing and FPGA is to use a method of encryption on the bitstream. FPGA vendors such as Xilinx and Altera provide tools that allow the bitstream to be encrypted and written into SRAM. Encrypting the bitstream defends against Trojan insertions where an adversary could partially reconfigure the FPGA and insert malicious logic into the bitstream. The advantage that encryption has is that the design tools provided

by Xilinx and Altera will not allow partial reconfiguration mixing encrypted and unencrypted reconfigurations.

Battery Backup Random Access Memory (BBRAM)

Key storage secrecy is fundamental in security, in order to keep the key secret it requires a way of keeping the register in memory confidential on the FPGA. One way of keeping the key secret is through using BBRAM, which allows for key agility and zeroization. When primary power is applied, the BBRAM is powered through the power supply, which would permit replacing the battery in the field. Xilinx provides an internal interface to zero out the specified key space in the event of an adversary tampering with the device. However, it does have the disadvantage that if the BBRAM loses connection or battery power when no primary power is available, the contents could be erased. BBRAM is said to be more physically secure than nonvolatile key storage, and to steal the key an adversary would have to scan the bits using expensive hard to obtain equipment, and attacking BBRAM is said to be beyond all but the most sophisticated adversaries.

eFuse

An alternative to using BBRAM is to use an eFuse memory, which is a one-time programmable non-volatile memory, typically programmed by electro-migration using high current. Additional processing complexity and/or high voltage are not required for eFuse; however, it does require a lot of logic circuitry and is only practical for low memory key storage.

Internal Temperature and Power Monitoring

An embedded analog to digital converter (ADC) is included in Xilinx FPGAs to monitor voltage and temperature inside and outside of the FPGA; users can configure the circuitry to accommodate the operating temperatures of the environment for specific applications. If a threshold is reached, user specific actions can be taken such as clearing crypto registers, RAM, zeroing keys, or clearing configurations and shutting the device down.

Fencing and Redundancy

Xilinx Isolation Design Flow (IDF) provides containment of faults at the FPGA level, allowing for the use of modular redundancy, watchdog alarms, segregation, and isolation of test logic to minimize single chip faults. The concept of isolation in the FPGA is performed by using configurable logic blocks as a fencing mechanism to isolate different systems in the programmable logic. Triple modular redundancy is the most popular method of reducing single point failures as well as providing and extra level of security through verification of sub systems, commonly incorporated through a voting scheme [69].

4.9 Benefit of HECAD

Integrating HECAD into a UAV CPS offers many benefits that can help improve cyber-attack detection in small, low power, and low cost UAVs. It offers a solution that can be integrated onto the hardware of the FCS, using an FPGA design approach, to aid in quick attack detection and isolation of malicious on-board sensor behavior. The hierarchical architecture offers benefits of collaboration between sub-modules to increase the ability to detect and locate malicious attacks, as well as potentially increasing the amount of information gathered about the system using a hardware-software solution.

4.9.1 Sub-module collaboration and increased information

Traditional approaches to monitoring systems typical monitor a systems software at the information level. At the information level, there is no information about a fault/attack that may occur below the software layer of the system; for example, if the hardware peripheral on the system or hardware on the slave device that the system is receiving data from is tampered with. At the software layer, the data may show signs of tampering, however it can be difficult to determine where the attack originated. HECAD has the capability to detect hardware peripheral attacks, such as an adversary modifying a sensors hardware protocol configuration or tampering with the physical signal between the main processor and a sensor. An example of a hardware peripheral attack is shown in Figure 26. It demonstrates how an attack at the hardware level may go unnoticed by other methods of detection, since an attack may not modify the data to an extent that can easily be identified. It is also shown how the attack can trickle through the various levels of HECAD, where it gets immediately detected at the HRIM sub-module, however, the data passes through the information layer checks such as bounds checking, and the information received from the sensor is similar to the data received on the main processor in the EIM.

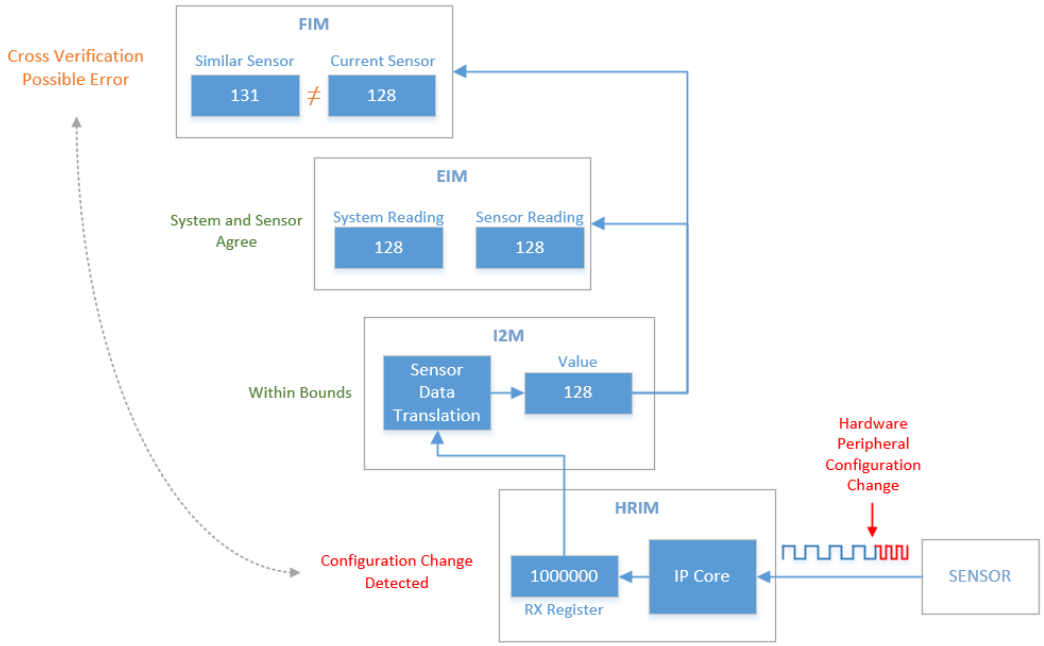

Figure 26 Hardware Peripheral Attack Example

However, in the FIM, Cross-verification throws a flag that there is a possible error between two sensors, since the error may not be drastic enough to be caught by the requirements or restrictions placed in the FIM, it may not be able to accurately determine if a fault/attack has occurred. This demonstrates HECADs ability to use collaboration amongst the sub-modules to potentially gather more information on a fault/attack.

4.9.2 Improved Isolation and Location Detection of Faults

A fault/attack at the hardware level not only allows HECAD to make a more informed decision, it also allows for better isolation and determination of the location of the fault/attack. When a fault is detected at any level, HECAD has the ability to isolate the slave device from the main processor; a continuation of the previous example is shown in Figure 27. Once the fault/attack has been detected by the HRIM, a signal is emitted to all of the other sub-modules that are in the hierarchy of that specific slave device, and mitigation can begin.

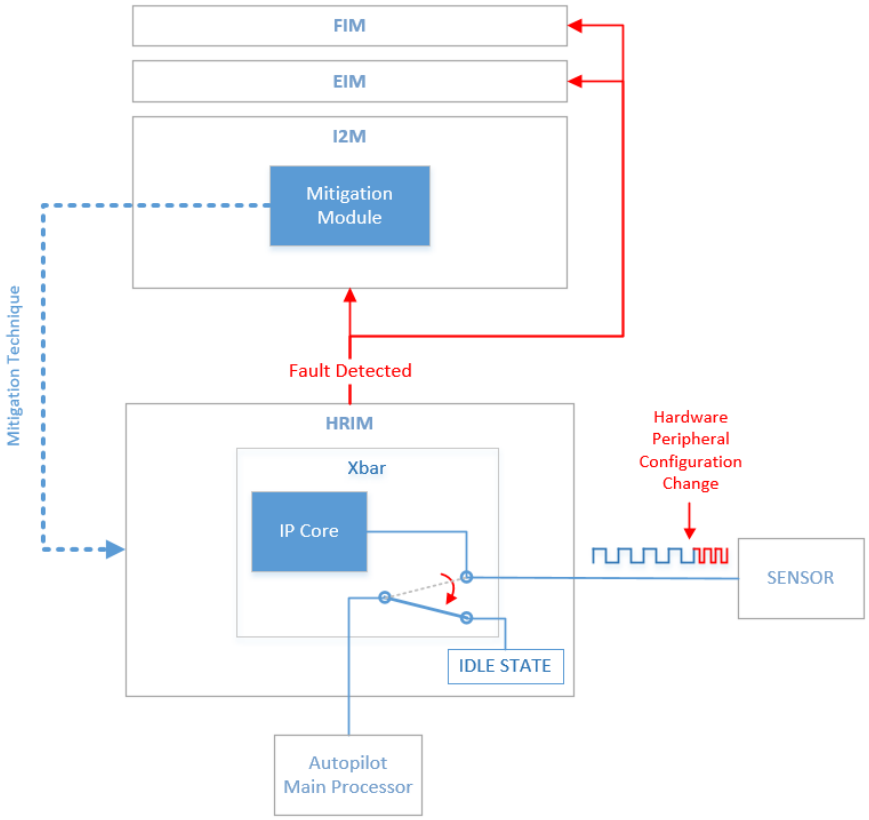

Figure 27 Sensor isolation upon HRIM attack detection

The mitigation techniques for the slave devices on the UAV FCS are out of the scope of this research and will be considered in future work. It is important to note that HECAD provides the foundation for mitigation techniques to be possible to implement on a UAV FCS, where there is currently no framework that allows such an integrated approach. Currently, when a fault/attack is detected, the slave device is isolated from the main processor, and the hardware peripheral on the main processor is held at an IDLE state, while HECAD provides a mitigation technique such as performing a reconfiguration routine to reset the slave device and reconnect the device once it has been successfully reconfigured.

4.9.3 Hardware-Software Boundary Attack Detection

In most monitoring systems that are developed for cyber-attack detection, the emphasis is focused on the information layer, which is where the software is contained and do not verify the hardware peripherals that communicate between the main processor and the on-board slave devices. HECAD takes into account the possibility that the firmware on a sensor can be a potential attack surface and HECAD has the capability to monitor the information layer as well as the hardware peripheral layer. This additional monitoring capability can potentially provide more information allowing for quicker fault/attack detection, isolation, and location. It also provides a basis for a hierarchical architecture allowing for multi-level verification and collaboration between different sub-modules that monitor different characteristics of the system.

Breaking the hardware/software barrier through monitoring hardware resources as well as software on a system is becoming more necessary among Cyber-Physical systems where a principle of assessing the physical processes as well as the cyber aspect of the system is necessary to prevent system failure [70], [71] . Software failures remain to be the leading cause for computer based cyber-physical systems, however hardware failures have a larger impact with the number of devices affected [72]. With COTS UAS becoming more widely available amongst the civilian community, sensor firmware attacks can potentially have more of an impact during the manufacturing and distribution process. A firmware attack can remain undetected longer and be more difficult to detect, since autopilot firmware can easily be patches and updated, hardware sensor firmware is not.

CHAPTER 5

DEVELOPMENT PLATFORM AND DESIGN TOOLS

5.1 Field Programmable Gate Array

A Field Programmable Gate Array (FPGA) provides a foundation in which ECAD can be implemented as a collection of modules that can perform individual operations simultaneously. An FPGA-based design was chosen as the implementation platform because of its fully customizable features, simulation capabilities, and ability to meet real-time requirements. An FPGA has many features that make it advantageous over a hardcore processing system [34]. A main advantage of using an FPGA architecture, is that it can be developed entirely using a Hardware Description Language (HDL), which provides many features such as: customizability, observability, testability, hardware/software partitioning, parallelization, and isolation. An FPGA also allows for partial reconfiguration which allows for greater innovation, allowing the developer to modify parts of the overall design without needing to reconfigure the whole FPGA design.

Inside of an, FPGA the programmable logic typically consists of Look up tables (LUTs) and flip-flops. The LUTs provide programmable combinational logic, and the flip flops provide for efficient implementation of sequential logic such as registers, counters, and state machines; integrated together, a group of LUTs and flip flops create a resource referred to as a slice. Counting the number of these resources used to implement a design can be used to measure the utilization of an FPGA.

Diligent Nexys4 DDR

The Nexys 4 DDR integrates an Artix-7 FPGA onto a ready-to-use development platform. The Artix-7 chip used is the XC7A100T, contains, 15,850 logic slices. A logic slice

consists of 4 LUTs and 8 flip-flops, where in about $1/4^{th}$ of the logic slices can be used for distributed memory. The Xilinx Artix-7 is a member of the Xilinx 7-Series family and optimized for low power, low cost, and high throughput applications. The Artix-7 FPGA is available in various size packages that contain different amounts of resources. Table 4 displays the smallest, middle, and largest size packages with specifications of the resources available in each package.

The Artix-7 FPGA is similar in physical size compared to the ARM Cortex M4 micro controller, which is commonly used as the main processor for low cost autopilots in the open source community, typically around 14x14mm in size.

Table 4 ARTIX-7 Specifications

Chip	Physical Size (mm)	LUTs	Slices	Clock Speed (MHz)	Cost
XC7A12T	10x10 , 15x15	12,800	2000	464 - 628	$25
XC7A50T	10x10 - 23x23	52,160	8,150	464 - 628	$50
XC7A100T	15x15 – 27x27	101,440	15,850	464 - 628	$125
XC7A200T	19x19 - 35x35	215,360	33,650	464 - 628	$256

The Nexys 4 has 24 I/O ports, which can be used to connect the autopilot and the sensors to HECAD through communication protocols created through custom IP cores in HECAD. There are 16 switches, 4 buttons, 16 LEDs, and an ethernet port that can be used for debugging and testing. There is also an internal temperature sensor that can be used to monitor the temperature of the FPGA. The Nexys 4 has a 100MHz clock integrated onto the board to drive the programmable logic clock source. The Nexys 4 is shown in Figure 28

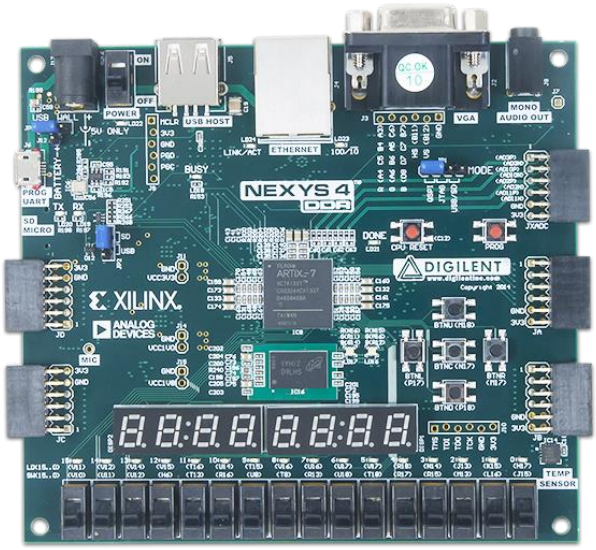

Figure 28 Digilent Nexys 4 DDR Development Board

Xilinx MicroBlaze (FIM Soft Core Processor)

The MicroBlaze is a full-featured, FPGA optimized 32-bit Reduced Instruction Set Computer (RISC) soft core processor. As a soft-core processor, it can be instantiated completely in programmable logic on the FPGA. There are many configurable options which may optimize the functionality for a specific use such as: performance optimized, performance optimized with branch optimizations, area optimized, and frequency optimized with branch optimization. The optimization used for HECAD is the area optimization option, minimizing the utilization of programmable logic allowing HECAD to contain more functionality. The MicroBlaze can be optimized for performance later on if necessary, depending on the application. The amount of resource consumed in the programmable logic depends on the optimization configuration chosen, the lowest is area optimization with a minimal configuration which utilizes approximately 600 LUTs, and the highest is the performance optimization, which uses approximately 6000 LUTs.

Xilinx recently released a lightweight version of the MicroBlaze processor, referred to as the MicroBlaze Micro Controller System (MCS); which can be used for simple control

applications. The local memory of the processor is limited to 4KB – 128KB, however its footprint is approximately 1000 LUTs. The current HECAD design incorporates this lightweight version of the MicroBlaze MCS, consuming 1.7% of the available programmable logic on the Artix-7 FPGA. As the complexity of the FIM increases the configuration of the MicroBlaze will need to be changed to accommodate the detection methods. The full featured MicroBlaze has the option to instantiate a Floating-Point Unit (FPU), which can drastically speed up floating point arithmetic that may be needed to calculate an accurate state estimate of the UAS. There is also an option to create a lock-step functionality where Xilinx instantiates two MicroBlaze processors that perform identical operations and can be used as a form of redundancy to detect tampering attempts, transient faults, or permanent hardware faults that may occur within the FPGA.

The MicroBlaze is necessary in the FIM to perform more complex algorithmic analysis that may be difficult to implement in a Hardware Descriptive Language (HDL) on the FPGA. The MicroBlaze communicates to the I2M through an IO module, which is a light weight Xilinx proprietary communication protocol. The MicroBlaze also has multiple GPIO pins, which are used to signal the MicroBlaze to fetch the sensor data out of the I2M data registers, as well as used to notify the I2M in the event of a fault/attack detection in the FIM. The FIM exists entirely inside the MicroBlaze and does not require any external logic in the FPGA, therefore expanding the FIM typically will only affect the FPGA block memory utilization, unless the MicroBlaze needs to be reconfigured and/or another MicroBlaze is used for lock-step or another operation to aid in cyber-attack detection.

5.2 VCU ARIES TINY FCS

The Aries Tiny is a custom FCS designed and developed by students in the UAV lab at VCU. The Aries Tiny FCS was chosen as a candidate for a testing platform because of its

customizability, accessibility, and familiarity with its hardware and software. The FCS contains an I2C bus connecting the Inertial Measurement Unit (IMU), absolute and differential barometric pressure units to measure body forces, altitude and air speed, 2 UARTs connecting the GPS and modem for position and communication, SBUS input for manual control, PWM outputs to control the actuators and multiple Analog to Digital Converters (ADC) to monitor power. The FCS is shown in Figure 29.

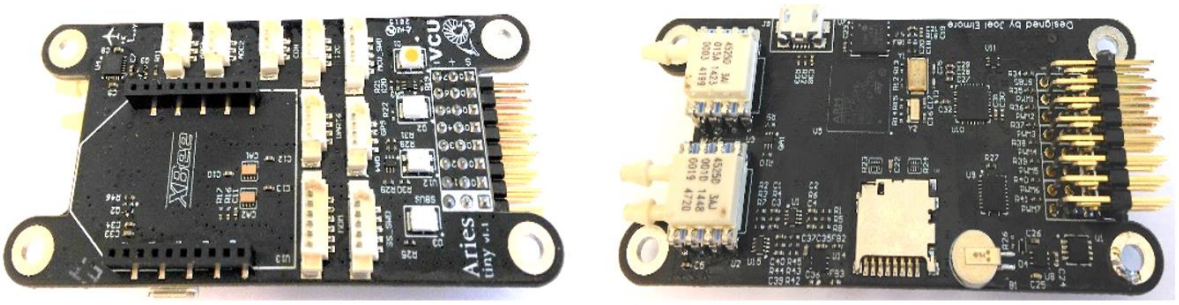

Figure 29 VCU Aries Tiny Flight Control System

The Aries Tiny has been tested and verified for semi and fully automated flight for several years. The FCS has all of its on-board communication routed to external connectors, which allows the ability to configure all of the on-board sensors to be disabled, and external sensors connected for testing purposes. This capability allows HECAD to be directly embedded between the autopilot main processor which provides HECAD the necessary functionality to connect and isolate the sensors from the autopilot. VCU also has a Hardware in the Loop Simulator (HILS) that can be used for further testing and future development of HECAD allowing for fault injection at the hardware and software level.

5.3 Xilinx Vivado

Xilinx Vivado is a design suite that is used for synthesis and analysis of HDL designs. When developing a digital system, the typical design flow uses a process that requires a 3-step approach: simulation, synthesis, and implementation. In a simulation environment, everything is in an "ideal" state, where there are no real-world delays, and nothing gets optimized out, and everything works as anticipated as long as the design is correct. The first step after compilation in a design suite such as Vivado, is synthesis, this process is where the tools take the HDL and try to convert it into a design implementation in the form of logic gates that can be directly programmed onto the FPGA. During this process, many things can happen during optimization that the user may not be aware of.

Coming from previous experience in embedded software development, compile times are negligible, and code that is not written in an optimized manner, does not slow down the compile times noticeably. In the FPGA design tools, this is the exact opposite case, where not only does the synthesis time take a lot longer to synthesize, but the design tools will try and optimize your code, with unanticipated results. Xilinx Vivado has multiple optimization options for synthesis with the two most popular being, runtime for speed and area. The default is runtime optimization, which will use however many logic gates necessary to allow the design to meet specifications and reduce delays of signals as much as possible. The issue with this method is that the design tools may remove certain parts of the design without the user's knowledge possibly due to an unintended HDL design issue, such as a signal that gets triggered during run-time that Vivado assumes is not used, and when programmed onto the FPGA, it will not behave as expected. In the opposite case, when selecting area optimized, unless the user has accounted for all of the anticipated delays correctly in their simulation, the design tools will have optimized the design, and more often than not, will not meet timing requirements, therefore, the best case is to use the default option.

CHAPTER 6

RESULTS

6.1 Hardware / low-level communication protocol monitor

One of the main contributions of the HECAD architectural framework is the ability to detect faults in the inter communication at the hardware level between the autopilot main processor and on-board components. In addition to having the ability to detect a fault, HECAD can isolate a slave device from the main processor of the FCS in the HRIM. The HRIM monitors the physical signal characteristics of the communication protocols of the on-board sensors as previously established. An example demonstrated the process of a hardware level fault occurring with knowledge of the UART protocol and configuration settings, HECAD is able to detect the fault in the HRIM based on the timing characteristics of the protocol at the signal level.

To demonstrate the fault detection and isolation capability of HECAD, a simulation and real-world experiment was performed which consisted of a GPS sensor baud-rate change during run-time as shown previously in section 4.3.1, where it was shown that the HRIM could detect physical signal faults. A simulation and real-world example of a physical signal fault injected into the FCS validates HECADs capabilities for detecting attacks/faults as well as isolating sensors using the crossbar switch. Validating this functionality proves that using an FPGA is possible to detect configuration attacks as well as isolate a sensor, therefore for future work the concepts can be extended for other communication protocols to create a complete cyber-attack detection system for a UAS FCS.

The first experiment shown in Figure 30, was a simulation performed in ModelSim consisting of a fault injected into the GPS sensor TX line that changed the baud-rate in the middle of a transaction. The signals shown are as follows: *clk* is the system clock, *rst* is the

system reset, *busy* is the UART busy signal, *uart_0_rx_sensor_o* is the UART transmit line from the HECAD to the GPS sensor, *uart_0_tx_sensor_i* is the UART receive line from the GPS sensor to HECAD, *uart_0_rx_cpu_o* is the UART receive from HECAD to the autopilot, *uart_0_tx_cpu_i* is the UART transmit line from the autopilot to HECAD, *xbar_en_s* is HRIM crossbar enable signal, *uart_0_fault_detected* is the fault detected in HRIM signal, *uart_0_fault_data_reg_s* is the transmit register used to send mitigation data out on UART, *present_state* is the I2M GPS sensor state machine status, *REG_DATA_OUT_s* is the data sent from the I2M to the HRIM on the WISHBONE bus, and *REG_DATA_IN_s* is the data sent by the HRIM to the I2M on the WISHBONE bus.

When observing the *uart_0_cpu_o* and *uart_0_tx_sensor_i* lines, it is shown that they are equivalent which demonstrates that the data received by the GPS sensor is being passed through to the autopilot. A UART start transaction is observed on these lines where the signal goes from high to low.

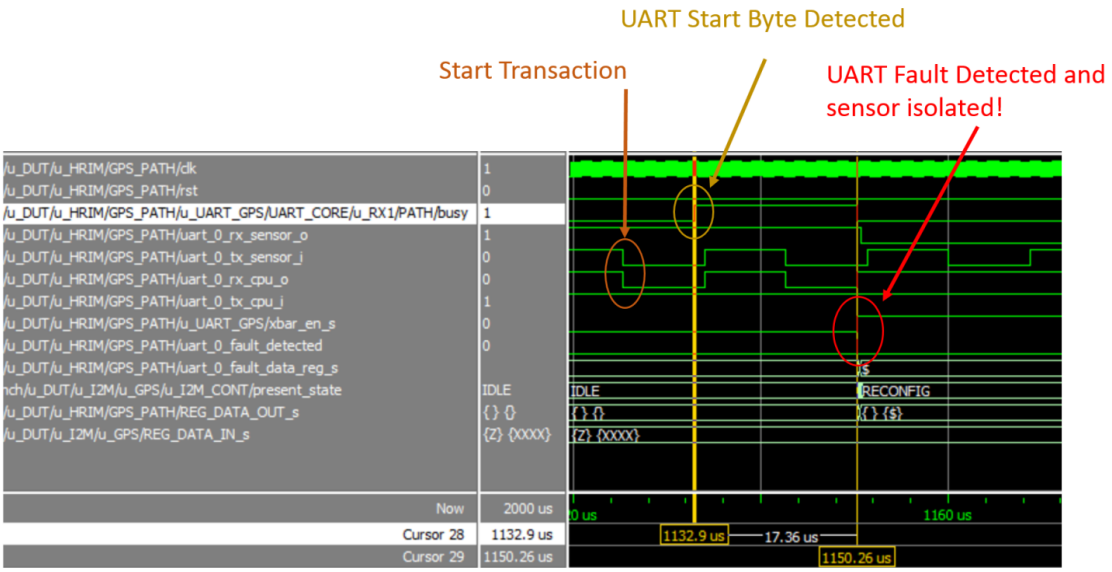

Figure 30 GPS Baud Rate fault injection performed in ModelSim (Zoomed In)

The HRIM UART busy line goes high when a start condition is detected in the UART. A start condition typically takes half the amount of time a bit takes to be detected based on the

baud-rate of the UART, to allow for rejection of signal jitter. Once the busy line is activated, the HRIM UART begins to read the data, however as shown in the figure, the current baud-rate of the UART transmission expects 57600, where a single bit should take 17.36 us, and if the correct baud-rate is observed by the UART, the signal transition should appear half way. In the figure, the signal changes multiple times within the baud-rate period which signifies an error on the UART. The HRIM detects the error, and immediately enables the cross-bar module which isolates the sensor and the UART signal connected to the autopilot (*uart_0_cpu_o*) goes high, which is an IDLE state on the UART. The GPS sensor is disconnected from the autopilot, and the I2M changes its state to RECONFIG, which is the mitigation state that attempts to reset and reconfigure the GPS sensor by the I2M sending a reconfiguration sequence to the HRIM through the WISHBONE bus, which can be seen by the data updating on *REG_DATA_IN_s* and the data begins to be populated on *REG_DATA_OUT_s.*

To better observe the fault in action, Figure 31 shows the same fault zoomed out. It may be easier to identify that the baud-rate attack has occurred. It is important to note that HECAD has successfully detected the baud-rate change, isolated the sensor from the autopilot by asserting the *xbar_en_s* signal, and began mitigating the fault/attack through notifying the I2M and changing the data flow from HRIM -> I2M, to I2M -> HRIM which can be observed on the WISHBONE bus through the *REG_DATA* signals and the *uart_0_rx_sensor_o* where the reconfiguration data is sent to the GPS sensor on the UART line.

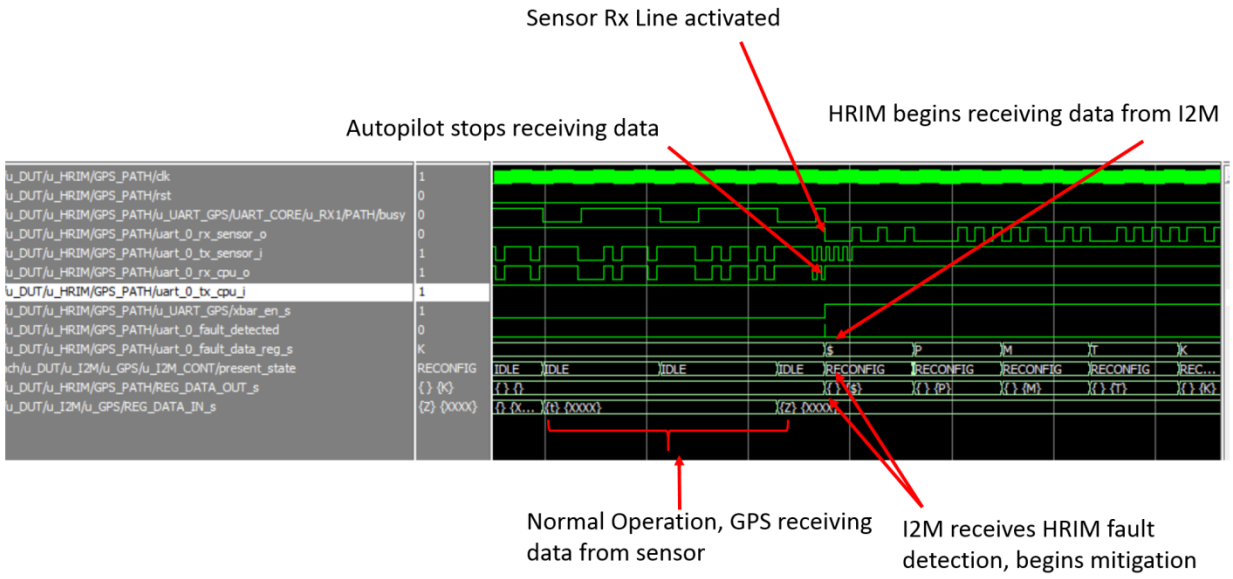

Figure 31 GPS Baud Rate fault injection performed in ModelSim (Zoomed Out)

A simplified block diagram of HECAD is shown in Figure 32. It is display the information flow of normal operation vs when a fault is detected and mitigation has been initiated. In normal operation, the green arrows show the information flowing from the GPS sensor to the autopilot through HECAD, which also passively sniffs the data on the communication line. HECAD does not interrupt the communication between the GPS sensor and the autopilot, by using the FPGA, it allows HECAD to separate the data into two parallel paths as it comes into the HRIM providing the ability to pass the data to a hardware IP core to parse the communication protocol without interrupting the communication between the sensor and the autopilot. Once a successful piece of data has been received by the hardware IP core, it is passed to the I2M through the WISHBONE bus, and further processed and eventually retrieved by the FIM.

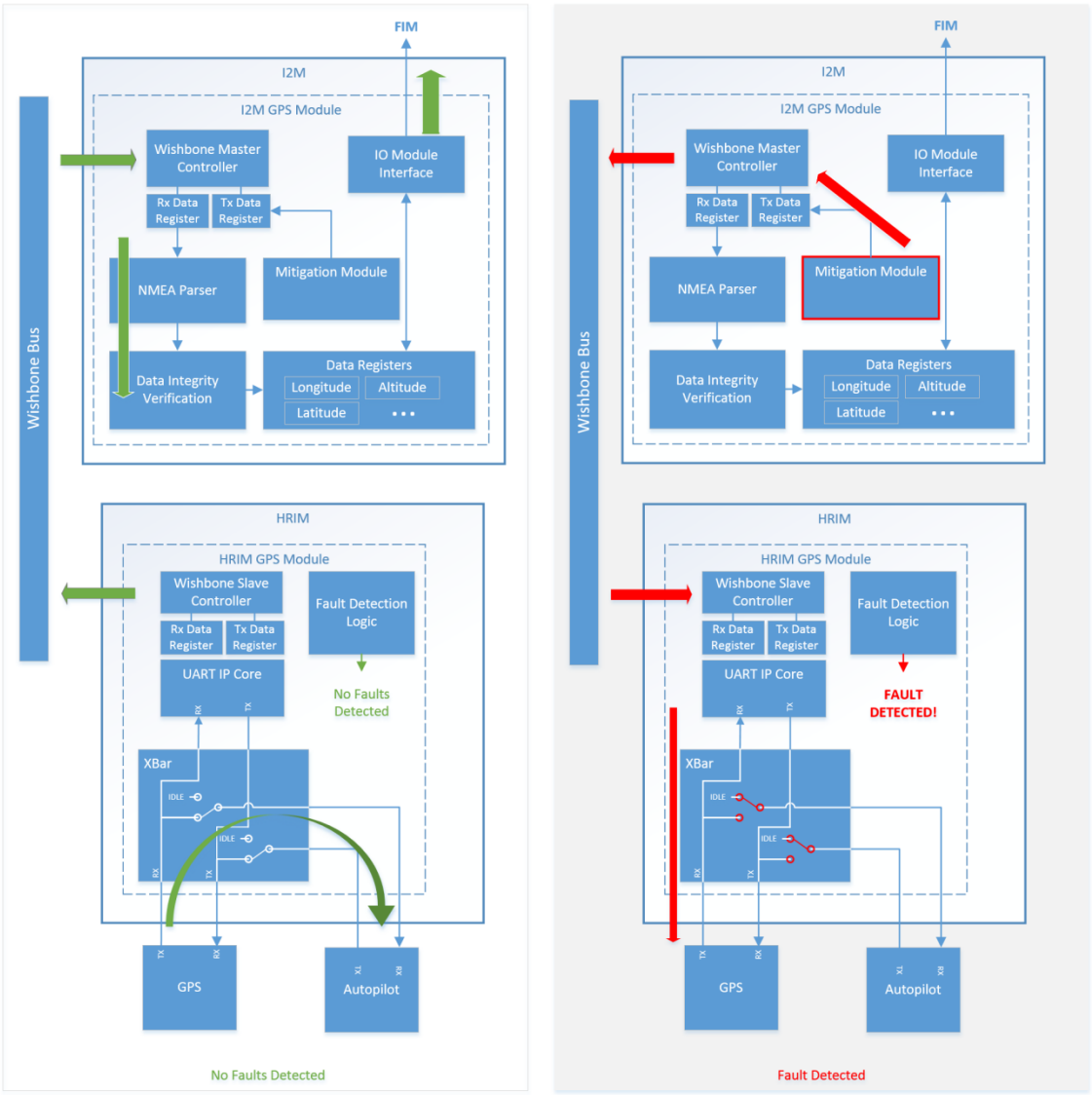

Figure 32 Block diagram of HECAD in normal vs mitigation operation

When a fault/attack is detected, the HRIM crossbar is enabled, causing the sensor to be isolated from the autopilot. When a sensor is isolated from the autopilot, it is done through the FPGA, where the HRIM connects the sensor side to the internal hardware IP core and the autopilot side gets forced to an IDLE state while mitigation is performed. After the crossbar is enabled, the information flow stops, and begins to flow the opposite way where the mitigation module in the I2M is initiated and performs its task.

Once the simulation was successfully performed, the architecture was synthesized and implemented on the Nexys 4 DDR FPGA and tested in a real-world scenario connected to the VCU FCS and a GPS emulator that allowed for a reconfiguration of the baud-rate during run-time. Resembling the simulation experiment, the UART baud-rate of the autopilot and the GPS sensor were both configured to 57600. During run-time after the initialization sequence of the autopilot, the baud-rate should remain the same. The baud-rate change fault was injected a few seconds after the autopilot boot up and expected normal operation. Shown in Figure 33 is the logic analyzer trace capture of the UART signals between the autopilot and GPS emulator.

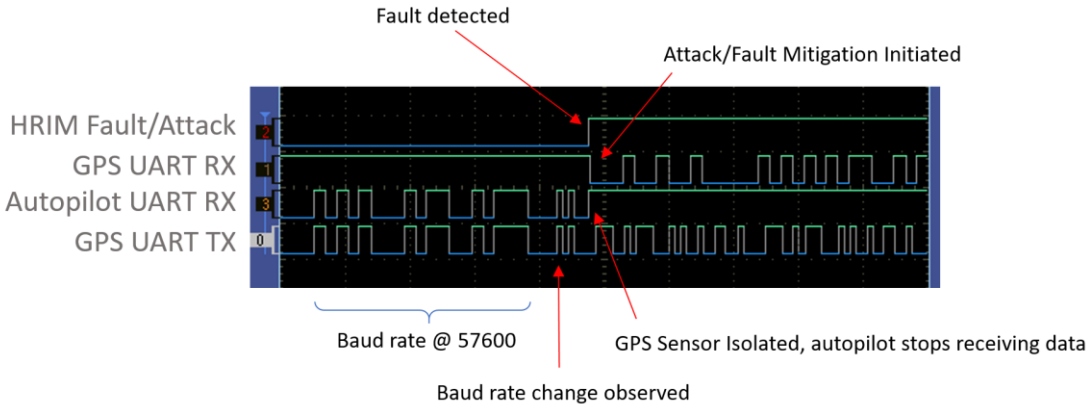

Figure 33 Logic analyzer capture of a real-world autopilot GPS baud-rate fault

The *HRIM fault/attack* signal is asserted when the HRIM detects an attack, as well as enables the cross-bar module isolating the sensor. The *GPS UART Rx* signal is the GPS sensor UART receive line, the *Autopilot UART Rx* signal is the autopilot UART receive line, the *GPS UART TX* signal is the GPS sensor UART transmit line. It can be Observed that the baud-rate is changed from 57600 to a higher baud-rate, which is detected by HECAD and the autopilot UART receive line is forced to IDLE by HECAD, disconnected from the GPS sensor which is still sending out data, and the mitigation sequence has been initiated, which is shown by the data being sent out on the GPS sensor UART receive line.

6.2 Cross-sensor data validation in the FIM

HECAD has the ability to detect attacks at various levels from the hardware to the software level. The first results section highlighted on hardware level detection, whereas this section will demonstrate HECADs ability to detect attacks at the software level. The first level of software fault/attack detection lies in the I2M, which verifies the integrity of the data, and is functional agnostic; consisting of methods such as bounds checking, timestamp verification, etc. The second level of detection resides in the FIM, which detects more functional types of faults and is application specific.

To validate HECADs ability to perform high level, functional analysis in the FIM, a rudimentary cross sensor verification method was used to detect a fault in the sensor data that passes all checks from the HRIM, I2M, and EIM. Established previously in section 4.4.3, the FIM is capable of verifying functional integrity using a rudimentary method for the purposes of demonstrating the ability to apply analysis to sensor data that otherwise would not create an alarm based on functional agnostic methods.

The experiment was performed using a GPS sensor emulator and an absolute barometric pressure sensor modelled in VHDL, which can be used to calculate the altitude of the UAS. The sensors are connected to the autopilot in the same configuration as previously where the sensors are connected to HECAD which passes the data through to the FCS autopilot. In this example, the GPS is connected through UART and the absolute barometric pressure sensor is connected through I2C. A fault was injected into the GPS altitude data after the initialization sequence successfully completed, during normal operation.

The plot shown in Figure 34 displays the data from the FIM data registers in HECAD as well as the HRIM crossbar switches enable signal. The data from the GPS sensor is read in through a dedicated UART core in the HRIM, and the data from the barometric pressure

sensor is read in through a dedicated I2C core in the HRIM. Each sensor has its own sensor specific parser that translates the raw data read on the bus into sensor data that can be converted into altitude data in the FIM. Once a complete packet is parsed in the I2M and the integrity is validated (bounds checking, etc.) a signal is triggered to the FIM, and the FIM will retrieve the data out of a block of memory where all of the sensor data is stored. The FIM will perform analysis on the data per sensor as well as combined with multiple sensors. For this example, a rudimentary cross verification method was used, where a threshold of ±10 ft. of altitude was specified; a rudimentary algorithm is used to demonstrate the functionality of HECAD, which is the aim of this research, however more complex methods will be used in the FIM and is a topic for future development. The fault injected into the GPS altitude is shown in Figure 34, where the GPS and pressure sensor read similar altitudes of approximately 570 feet, up to a point where the GPS reading jumps to > 800ft. In a real-world scenario, a UAS may react by performing a sudden dive which could potentially be unrecoverable. The plot shows how HECAD immediately catches the fault through cross verification and isolates the sensor and begins mitigating.

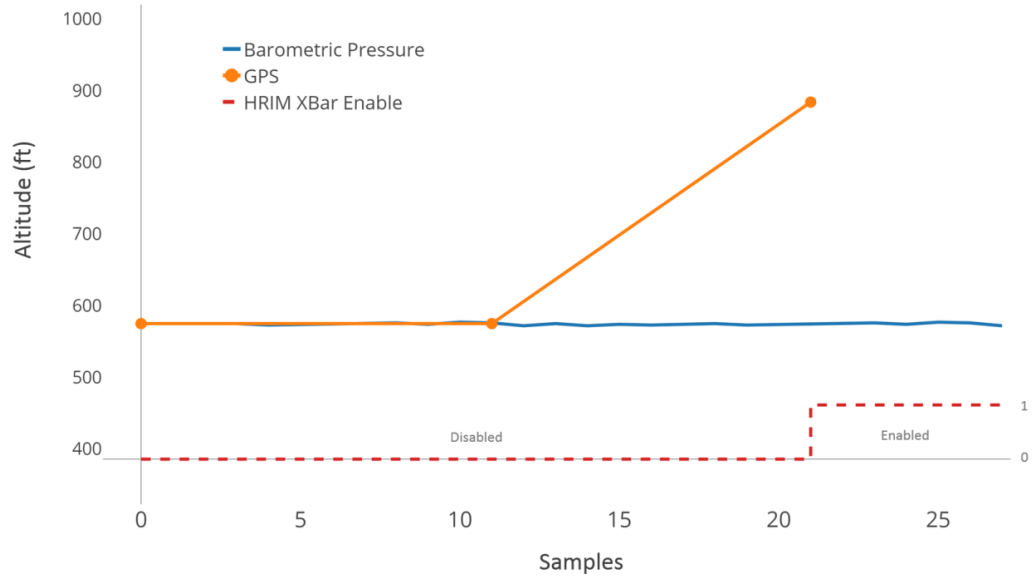

Figure 34 FIM readings of GPS and Baro altitude and HRIM XBAR enable signal

The signal trace of the ModelSim simulation is shown in Figure 35. The signals are as follows:

- *fault_detected*: Fault detected signal in the FIM that notifies the I2M
- *gps_altitude*, *baro_altitude* : GPS and barometric pressure sensor altitudes parsed in the I2M that reside in the FIM
- *hrim_gps_xbar_en*: Crossbar switches enable signal that isolates the GPS sensor when enabled which is triggered from I2M to the HRIM
- *fim_fault_detected*: Signal received by the I2M triggered by the FIM
- *gps_output_updated_s*, *abs_baro_output_updated_s:* Triggered when a complete packet is received by HECAD from the GPS and pressure sensor respectively
- *rx_sensor:* GPS UART receive line from HECAD
- *rx_cpu*: Autopilot UART receive line from HECAD (passthrough from GPS sensor to the autopilot)
- *tx_cpu* is the GPS UART receive line from HECAD (passthrough from the autopilot to the GPS)
- *xbar_en* is the HRIM crossbar switch enable signal

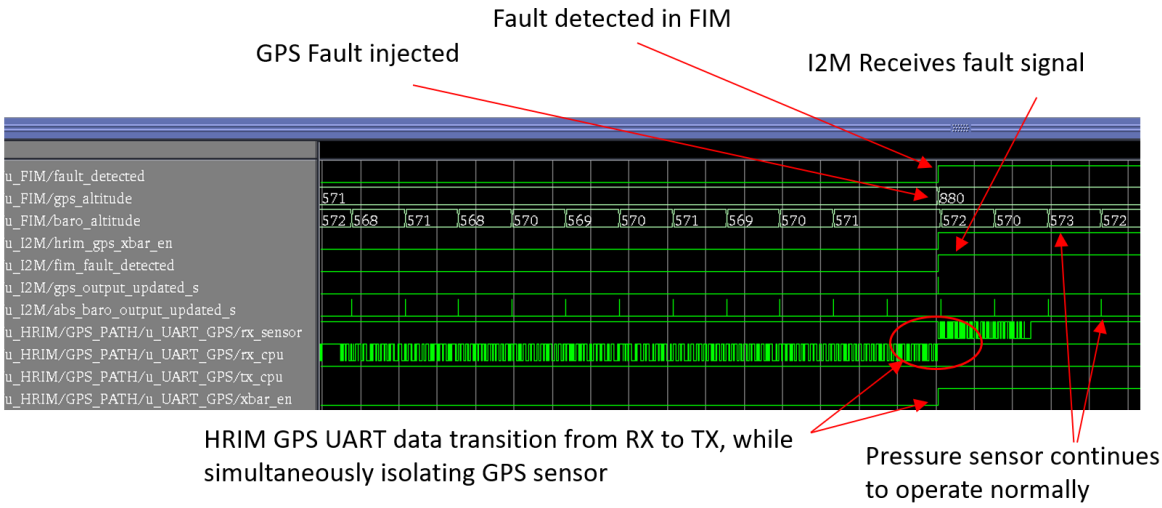

Figure 35 GPS and barometric pressure altitude fault injection attack signal trace

The GPS fault injection is in the form of an altitude jump as shown in Figure 35, where the altitude value changes from 571 ft. to 880 ft. HECAD immediately catches on the fault in the FIM, which is performing cross verification analysis on the barometric pressure sensor and GPS sensor, using consistency checking of past values to compare the GPS vs Baro altitude measurements. When HECAD detects the fault in the FIM, a signal is immediately sent to the I2M, which sequentially triggers the HRIM to enable the crossbar switch module and isolate the GPS sensor and initiate the mitigation sequence; exploiting the hierarchy. It is important to note that through this process of detecting, isolating, and mitigation that the pressure sensor is not disturbed and continues to function normally.

A block diagram of a high-level description of the information flow of HECAD during normal operation and when the FIM detects the GPS fault injection, is shown in Figure 36. The left diagram displays the FIM, I2M, and HRIM working under normal operation, where the information flows upwards from the HRIM, passing the data to the I2M, and the FIM fetching the data for cross verification sensor analysis.

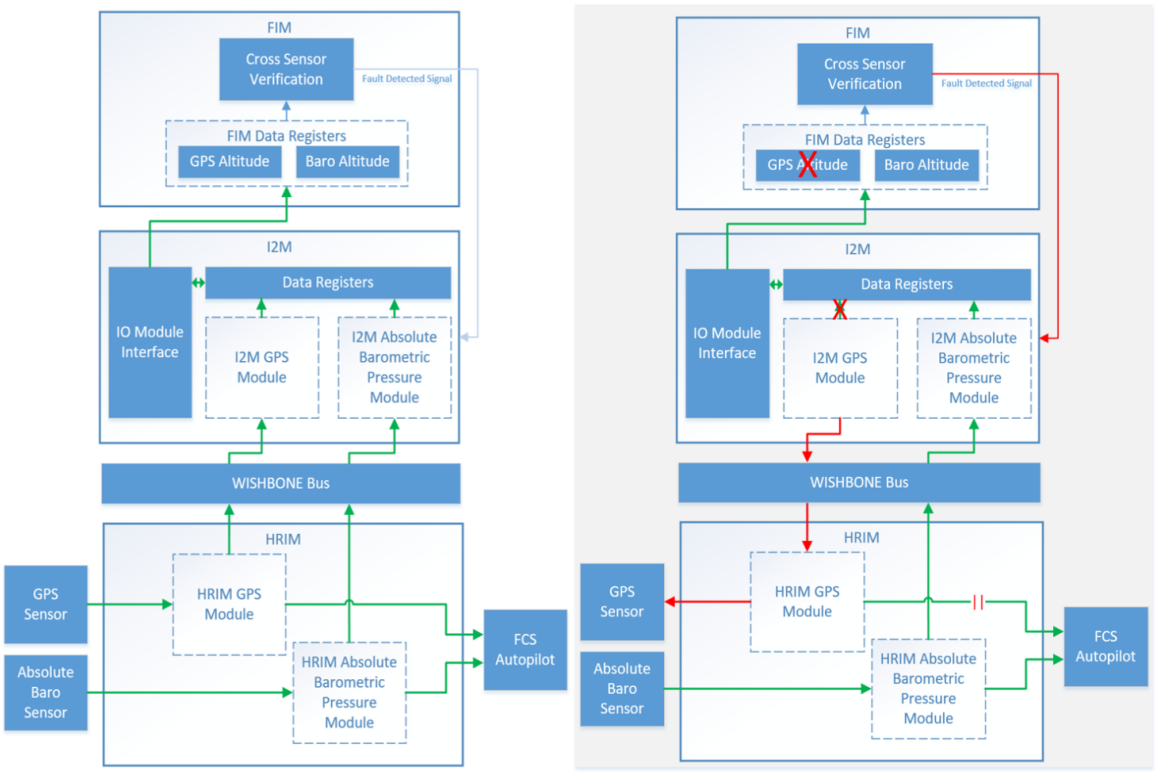

Figure 36 HECAD block diagram of a GPS fault injection detected in the FIM

When the fault is detected at the FIM level, the GPS data registers in the FIM do not continue to update since no data is available, the I2M stops updating and the information flow transitions to sending data to the HRIM to perform the mitigation sequence. The flow of information in HECAD follows the same trend for an attack detected in each subsystem of HECAD, where the HRIM isolates the sensor and the I2M transitions to mitigation.

6.3 HECAD subsystem isolation

One of the many advantages of HECAD is the ability to utilize the parallelism and isolation of the subsystem hierarchies derived from the FPGA implementation. Every slave device connected to HECAD has its own information flow path, using its hierarchical properties while adopting a distributed workflow. In a general processor, the event of a

system failure is more likely to occur vs. an FPGA when a subsystem is to fail due to the sequential processing nature of a general processor vs. digital logic which allows for parallel execution. HECADs architecture allows for the system as a whole to keep operating in the event that a subsystem is to fail. An experiment was performed where the HRIM GPS subsystem is frozen in its current state to simulate the event of a subsystem failure.

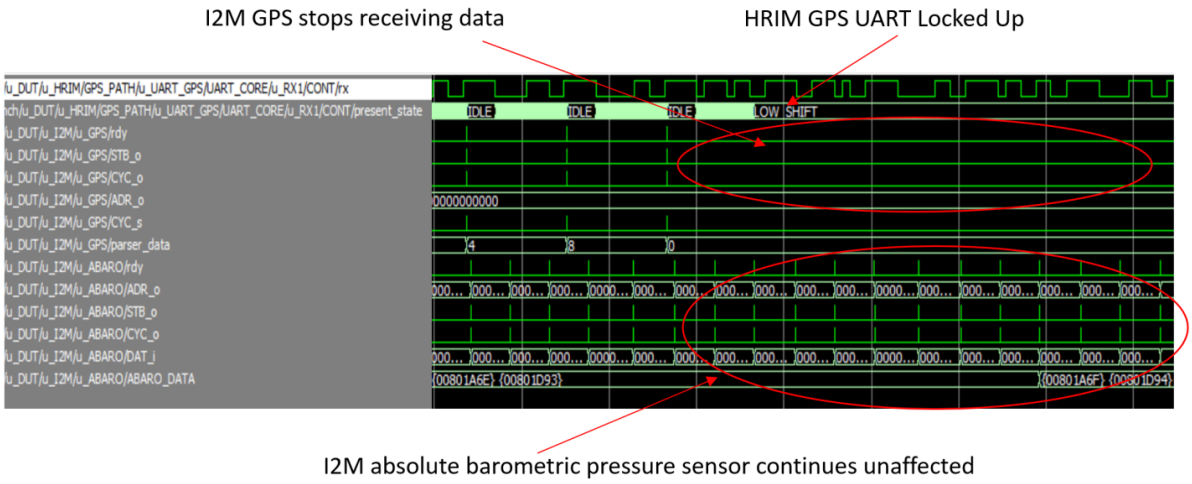

Figure 37 ModelSim signal trace of a subsystem failure

The ModelSim signal trace is shown in Figure 37, where it can be observed that the I2M GPS subsystem stops receiving data on the WISHBONE bus from the HRIM due to the fault that was injected forcing the HRIM GPS subsystem to halt its operation. The result of this fault is the GPS sensor unable to be monitored by HECAD since the data flow from the HRIM to the I2M has stopped, however, the I2M is expecting data at a certain frequency, which will enable the crossbar switch in the HRIM, which will not allow for the sensor to pass through any data to the autopilot.

A block diagram of the subsystem failure is shown in Figure 38, displaying the hierarchy of the GPS subsystem blocks halting operation in all of the levels of HECAD.

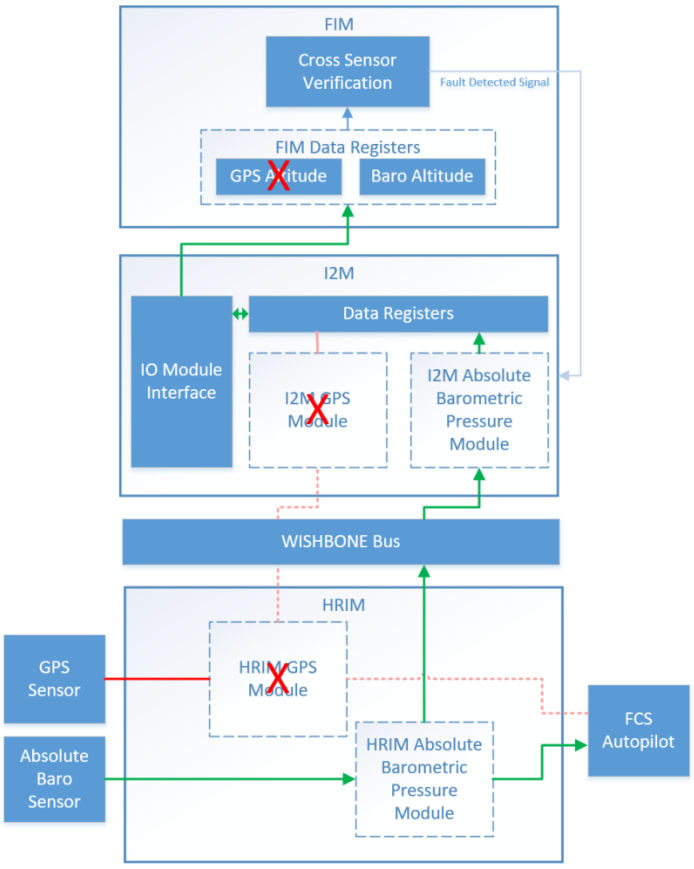

Figure 38 Block diagram of HECAD subsystem failure

The same experimental setup is used as previously where a GPS and an absolute barometric pressure sensor are connected to HECAD through a UART and I2C respectively. Subsystem isolation is imperative in a monitoring system because if an unexpected event occurs in the monitor itself, it is necessary that the system being monitored is affected as minimally as possible. In HECAD this allows for the failures to remain local, preventing cross subsystems to be affected.

6.4 HECAD Resource Utilization

HECAD was simulated, synthesized, and implemented on the Nexys 4 DDR. Although the whole system was not tested and implemented, the functional aspects of the architecture

were tested to verify that the hierarchical architecture design on an FPGA was possible and capable of monitoring, detecting, and isolating cyber-attacks on a UAV FCS. As mentioned previously, the goal of this research is to develop an architectural framework with the ability to monitor, detect, and isolate malicious cyber-attacks at the hardware and software level; the specific algorithms in each block are out of the scope of this research and reserved for future work.

The current state of HECAD includes a full stack implementation of the GPS sub modules from the HRIM, I2M and FIM with rudimentary algorithms capable of detecting an attack. The absolute barometric pressure sensor and modem sub modules have been implemented for basic functionality within HECAD, which includes the ability to sniff the communications line, collect the sensor data, pass it through the WISHBONE bus up to the I2M to parse the raw data into sensor data, and notify the FIM to retrieve the sensor data to perform a rudimentary analysis. The differential barometric pressure sensor and IMU sensors can use a similar template to the absolute barometric pressure sensor, whereas they will behave the same functionally whereas different sensors will have respective parsing algorithms and sensor specific verification e.g bounds checking.. The actuator sub modules will need to incorporate a PWM IP core for the inputs and the outputs of the FCS, and then can be dropped into the skeleton template set in place. The WISHBONE bus is fully functional and implemented ready for additional sub modules. The EIM has a rudimentary method of monitoring the system during run-time which takes advantage of a watch-dog timer that can connect to a GPIO on the Autopilot that allows HECAD to monitor the run-time execution of the main processor through knowledge of the execution loops such as the control algorithms running at 50Hz. The Autopilot periodically will signal the EIM every 20ms, allowing to verify that no extra code is running or sensors have been modified to change how long an execution loop will take.

The current progress of the HECAD implementation is shown in Figure 39. The complete blocks are shown with a filled in blue background, partially complete have a blue outline, and the un-implemented blocks are greyed out. In the HRIM and I2M the WISHBONE bus master and slave interfaces are instantiated and ready to incorporate the functionality of the respective sub modules. Also, for testing purposes a debug module was instantiated in order to communicate the status of the system to a user via UART. The debug module has direct access to the status registers inside each subsystem, which would alert the engineer if an attack has been detected or not. It is not decided yet if the final design will have communication with the autopilot, and the decision will be made in future work.

The implemented features have been simulated, synthesized, and implemented using the Xilinx Vivado Design Suite. The current utilization of resources in the Artix-7, *XC7A100T* FPGA programmable logic based on consumed logic slices is 21%. The utilization accounts for about half of the system design with rudimentary algorithms, and can be used to estimate that the entire system barebones would consume about 50% of the resources of the Artix-7 FPGA. Depending on the complexity of the algorithms in the FIM level, more RAM may be needed to extend the size of the MicroBlaze memory to increase the functionality by adding more analysis algorithms.

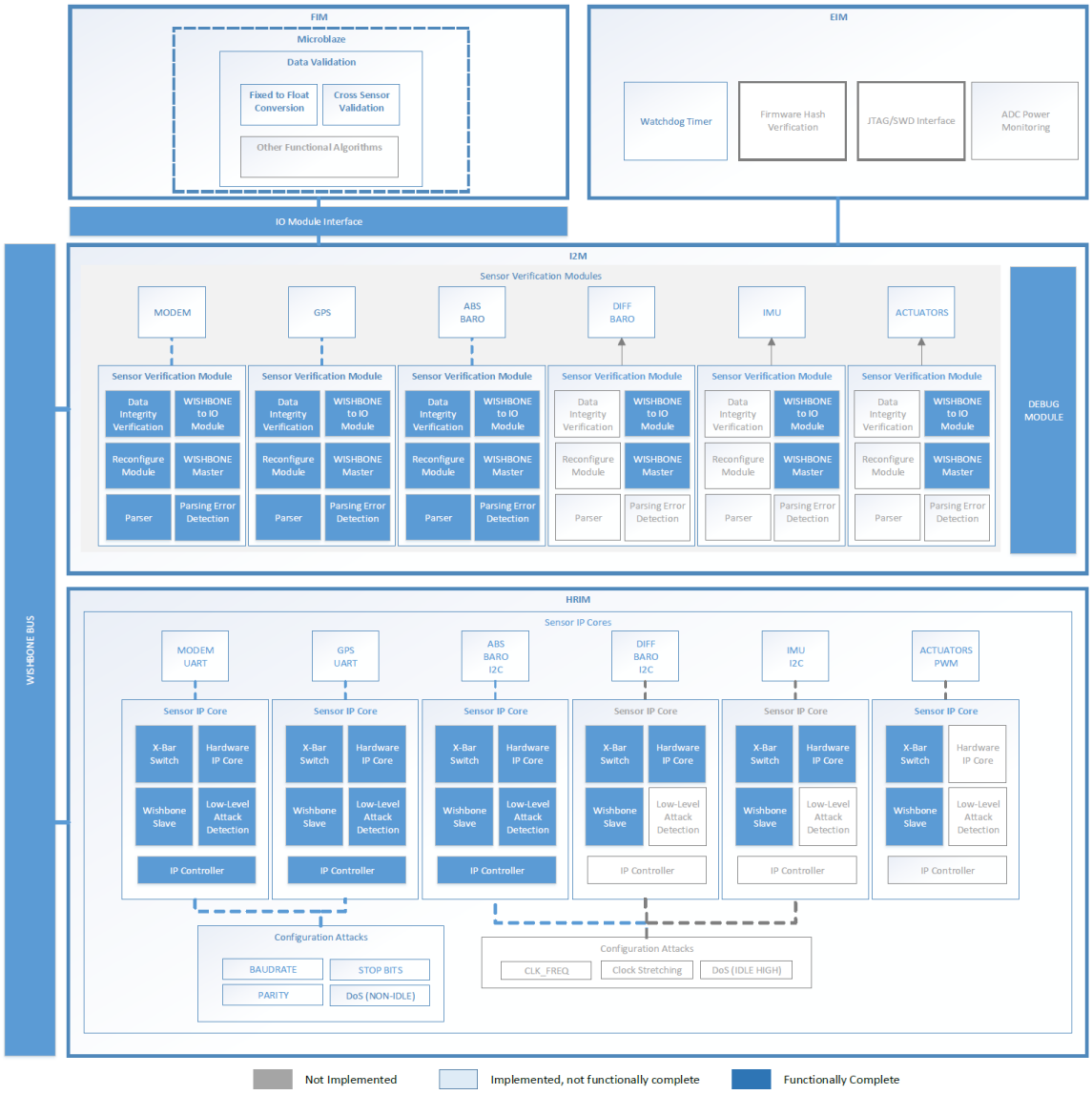

Figure 39 HECAD Block Diagram Overview of all the implemented subsystems

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

Cyber-attacks are becoming more sophisticated as technology improves, and it is not only integral to prevent faults from occurring, but it is becoming necessary to monitor and detect malicious faults that can have devastating consequences in safety critical applications. Current approaches to monitor UASs focus on the software level and utilize an external system capitalizing on triple modular redundancy or a DSL developed for specific applications using formal methods implemented in line with the system software. The research performed in this dissertation introduces a holistic approach of securing a UAS FCS at the various levels, traversing the hardware/software boundary, monitoring individual component and actuator failures at the hardware level in the HRIM, while simultaneously monitoring the system at the software level through the EIM run-time execution verification, I2M component specification verifications, and FIM system level, mission envelope validation. HECAD allows the use of a combination of system level, mission specific methods that utilize statically defined assertions offline, and functional agnostic methods to monitor the dynamic runtime execution of the system.

The methodology presented in this paper creates a framework that can be used as a general guide in securing a UAS CPS by embedding an FPGA based solution onto the FCS. This approach creates the ability to monitor low-level communication directly between the autopilot main processor and the on-board sensors maximizing the amount of information that can be collected to monitor the system. Before HECAD, no solution existed that permitted the ability to monitor the low-level communications between the autopilot and the on-board sensors without affecting the system. The HECAD methodology opens new doors into securing a UAS against cyber-attacks by incorporating a platform onto a UAS FCS that not only has access to the main processor, on-board sensors, and actuators, but also the

ability to isolate malicious sensors and potentially override the autopilot if a total system failure is to occur. HECAD provides the capability to monitor, detect, and mitigate potential cyber-attacks that are unobservable by external or complete software DSL solutions. With this accessibility to the on-board components, new methods can be explored to secure a UAS FCS. An FPGA implementation allows the possibility of developing detection algorithms in software (on a soft-core processor) was well as hardware (FPGA programmable logic) and can be used by future integrators to secure a CPS to improve resilience, reliability, and correctness of the system.

7.1 Future work

The goal for future work is to complete the implementation of the HECAD architecture on an FPGA and test it on a UAS FCS using a Hardware in the Loop Simulator (HILS). New algorithms will be developed to detect attacks on the physical signal and communication protocol to monitor and detect sensor firmware cyber-attacks. Algorithms are to be developed to detect cyber-attacks at each level in HECAD, as well as investigating techniques to monitor the execution of the main processor dynamically during run time and developing solutions on the FPGA in digital logic. More sophisticated algorithms are to be implemented in the FIM to combine all of the sensor data as well as information about the application, possibly developing a state estimation algorithm that can utilize the sensor data and potentially create a mitigation solution that can control the UAS in the event of a total system failure. The hardware resource information, sensor data integrity verification, and mission specific functional integrity validation can be used holistically to develop collaborative detection algorithms utilizing all of the levels of fault detection to maximize the ability to secure a UAS.

CHAPTER 8

REFERENCES

[1] K. Robillard and A. Byers, "Amazon drones: Obstacles to the bezos dream," *Politico*, 2013.

[2] G. Bensinger, "Before the Drones Come, Amazon Lets Loose the Robots," *Wall Str. J.*, vol. 9, 2013.

[3] L. Summers, "Why stagnation might prove to be the new normal," *Financ. Times*, vol. 15, 2013.

[4] "Hacking Drones … Overview of the Main Threats - InfoSec Resources." [Online]. Available: http://resources.infosecinstitute.com/hacking-drones-overview-of-the-main-threats/. [Accessed: 18-Mar-2016].

[5] A. Kim, B. Wampler, J. Goppert, I. Hwang, and H. Aldridge, "Cyber attack vulnerabilities analysis for unmanned aerial vehicles," *Infotech Aerosp.*, 2012.

[6] A. Y. Javaid, W. Sun, V. K. Devabhaktuni, and M. Alam, "Cyber security threat analysis and modeling of an unmanned aerial vehicle system," in *Homeland Security (HST), 2012 IEEE Conference on Technologies for*, 2012, pp. 585–590.

[7] K. Hartmann and C. Steup, "The vulnerability of UAVs to cyber attacks-An approach to the risk assessment," in *Cyber Conflict (CyCon), 2013 5th International Conference on*, 2013, pp. 1–23.

[8] K. Mansfield, T. Eveleigh, T. H. Holzer, and S. Sarkani, "Unmanned aerial vehicle smart device ground control station cyber security threat model," in *Technologies for Homeland Security (HST), 2013 IEEE International Conference on*, 2013, pp. 722–728.

[9] M. T. Leccadito, T. Bakker, R.H. Klenke, and C. R. Elks, "A Survey On Securing UAV Cyber Physical Systems," *IEEE Aerosp. Electron. Syst. Mag.*, to be published.

[10] D. Rudinskas, Z. Goraj, and J. Stankūnas, "Security analysis of uav radio communication system," *Aviation*, vol. 13, no. 4, pp. 116–121, 2009.

[11] N. Butcher, A. Stewart, and S. Biaz, "Securing the MAVLink Communication Protocol for Unmanned Aircraft Systems."

[12] B. W. O'Hanlon, M. L. Psiaki, T. E. Humphreys, and J. A. Bhatti, "Real-time spoofing detection using correlation between two civil GPS receiver," in *Proceedings of the ION GNSS Meeting*, 2012.

[13] Z. Zhang, M. Trinkle, L. Qian, and H. Li, "Quickest detection of GPS spoofing attack," in *MILITARY COMMUNICATIONS CONFERENCE, 2012-MILCOM 2012*, 2012, pp. 1–6.

[14] E. W. Frew and T. X. Brown, "Networking issues for small unmanned aircraft systems," *J. Intell. Robot. Syst.*, vol. 54, no. 1–3, pp. 21–37, 2009.

[15] D. P. Shepard, J. A. Bhatti, T. E. Humphreys, and A. A. Fansler, "Evaluation of smart grid and civilian UAV vulnerability to GPS spoofing attacks," in *Proceedings of the ION GNSS Meeting*, 2012, vol. 3.

[16] T. E. Humphreys, B. M. Ledvina, M. L. Psiaki, B. W. O'Hanlon, and P. M. Kintner Jr, "Assessing the spoofing threat: Development of a portable GPS civilian spoofer," in *Proceedings of the ION GNSS international technical meeting of the satellite division*, 2008, vol. 55, p. 56.

[17] E. M. Puchaty, D. DeLaurentis, and others, “A performance study of UAV-based sensor networks under cyber attack,” in *System of Systems Engineering (SoSE), 2011 6th International Conference on*, 2011, pp. 214–219.
[18] N. M. Rodday, “Exploring Security Vulnerabilities of Unmanned Aerial Vehicles,” 2015.
[19] T. Moore and R. Clayton, “Examining the impact of website take-down on phishing,” in *Proceedings of the anti-phishing working groups 2nd annual eCrime researchers summit*, 2007, pp. 1–13.
[20] R. Dhamija, J. D. Tygar, and M. Hearst, “Why phishing works,” in *Proceedings of the SIGCHI conference on Human Factors in computing systems*, 2006, pp. 581–590.
[21] S. Egelman, L. F. Cranor, and J. Hong, “You’ve been warned: an empirical study of the effectiveness of web browser phishing warnings,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2008, pp. 1065–1074.
[22] G. Loukas, “Cyber-Physical Attacks: A Growing Invisible Threat,” 2015.
[23] M. Strohmeier, V. Lenders, and I. Martinovic, “Lightweight Location Verification in Air Traffic Surveillance Networks,” in *Proceedings of the 1st ACM Workshop on Cyber-Physical System Security*, 2015, pp. 49–60.
[24] Y. Son *et al.*, “Rocking drones with intentional sound noise on gyroscopic sensors,” in *Proceedings of the 24th USENIX Conference on Security Symposium*, 2015, pp. 881–896.
[25] Y. Gao *et al.*, “Analysis of security threats and vulnerability for cyber-physical systems,” in *Computer Science and Network Technology (ICCSNT), 2013 3rd International Conference on*, 2013, pp. 50–55.
[26] “Advantages and Disadvantages of ISM Band Frequencies,” *L-com Global Connectivity*. [Online]. Available: http://www.L-com.com/content/Article.aspx?Type=N&ID=10421. [Accessed: 07-Jan-2016].
[27] M. Pini, M. Fantino, A. Cavaleri, S. Ugazio, and L. L. Presti, “Signal quality monitoring applied to spoofing detection,” in *Proceedings of the 24th International Technical Meeting of The Satellite Division of the Institute of Navigation (ION GNSS 2011)*, 2001, pp. 1888–1896.
[28] S. Daneshmand, A. Jafarnia-Jahromi, A. Broumandan, and G. Lachapelle, “Low-complexity spoofing mitigation,” *GPS World*, vol. 22, no. 12, pp. 44–46, 2011.
[29] “Air Traffic Services Brief -- Automatic Dependent Surveillance-Broadcast (ADS-B).” [Online]. Available: http://www.aopa.org/Advocacy/Air-Traffic-Services-,-a-,-Technology/Air-Traffic-Services-Brief-Automatic-Dependent-Surveillance-Broadcast-ADS-B. [Accessed: 06-Nov-2015].
[30] A. Costin and A. Francillon, “Ghost in the Air (Traffic): On insecurity of ADS-B protocol and practical attacks on ADS-B devices,” *Black Hat USA*, 2012.
[31] N. Shachtman, “Exclusive: Computer Virus Hits U.S. Drone Fleet,” *WIRED*, 07-Oct-2011. [Online]. Available: http://www.wired.com/2011/10/virus-hits-drone-fleet/. [Accessed: 07-Oct-2015].
[32] E. Yağdereli, C. Gemci, and A. Z. Aktaş, “A study on cyber-security of autonomous and unmanned vehicles,” *J. Def. Model. Simul. Appl. Methodol. Technol.*, p. 1548512915575803, 2015.
[33] A. Cavaleri, B. Motella, M. Pini, and M. Fantino, “Detection of spoofed GPS signals at code and carrier tracking level,” in *Satellite Navigation Technologies and European Workshop*

on GNSS Signals and Signal Processing (NAVITEC), 2010 5th ESA Workshop on, 2010, pp. 1–6.
[34] Z. Birnbaum, A. Dolgikh, V. Skormin, E. O'Brien, and D. Muller, "Unmanned Aerial Vehicle security using Recursive parameter estimation," in *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2014, pp. 692–702.
[35] D. F. Pigatto, J. Smith, K. R. Lucas, and J. Castelo Branco, "Sphere: A novel platform for increasing safety amp; security on Unmanned Systems," in *2015 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2015, pp. 1059–1066.
[36] "Experimental Analysis of Attacks on Next Generation Air Traffic Communication - Google Search." [Online]. Available: https://www.google.com/search?q=Experimental+Analysis+of+Attacks+on+Next+Generation+Air+Traffic+Communication&oq=Experimental+Analysis+of+Attacks+on+Next+Generation+Air+Traffic+Communication&aqs=chrome..69i57.152j0j4&sourceid=chrome&ie=UTF-8. [Accessed: 21-Mar-2017].
[37] O. Sokolsky, M. Pajic, N. Bezzo, and I. Lee, "Architecture-Centric Software Development for Cyber-Physical Systems," 2014.
[38] P. C. Hickey, L. Pike, T. Elliott, J. Bielman, and J. Launchbury, "Building embedded systems with embedded DSLs," in *Proceedings of the 19th ACM SIGPLAN international conference on Functional programming*, 2014, pp. 3–9.
[39] "High-Assurance Cyber Military Systems (HACMS)." [Online]. Available: http://www.darpa.mil/program/high-assurance-cyber-military-systems. [Accessed: 03-Nov-2015].
[40] R. Ivanov, M. Pajic, and I. Lee, "Resilient multidimensional sensor fusion using measurement history," in *Proceedings of the 3rd international conference on High confidence networked systems*, 2014, pp. 1–10.
[41] R. Ivanov, M. Pajic, and I. Lee, "Attack-resilient sensor fusion," in *Proceedings of the conference on Design, Automation & Test in Europe*, 2014, p. 54.
[42] J. Park, R. Ivanov, J. Weimer, M. Pajic, and I. Lee, "Sensor attack detection in the presence of transient faults," in *Proceedings of the ACM/IEEE Sixth International Conference on Cyber-Physical Systems*, 2015, pp. 1–10.
[43] P. F. Swaszek and R. J. Hartnett, "Spoof detection using multiple COTS receivers in safety critical applications," 2013.
[44] H. Wen, P. Y.-R. Huang, J. Dyer, A. Archinal, and J. Fagan, "Countermeasures for GPS signal spoofing," in *ION GNSS*, 2005, pp. 13–16.
[45] D. L. Hall and J. Llinas, "An introduction to multisensor data fusion," *Proc. IEEE*, vol. 85, no. 1, pp. 6–23, 1997.
[46] P.-J. Bristeau, F. Callou, D. Vissiere, N. Petit, and others, "The navigation and control technology inside the ar. drone micro uav," in *18th IFAC world congress*, 2011, vol. 18, pp. 1477–1484.
[47] "XBee-PRO 900HP - Digi International." [Online]. Available: http://www.digi.com/products/xbee-rf-solutions/modules/xbee-pro-900hp. [Accessed: 16-Nov-2015].
[48] "XBee 802.15.4 - Digi International." [Online]. Available: http://www.digi.com/products/xbee-rf-solutions/modules/xbee-series1-module. [Accessed: 16-Nov-2015].

[49] D. Pilankar, T. Sawant, R. Sule, and S. Mahadeshwar, "Implementing Agricultural Applications Using Wireless Sensor Network and Securing Them Using Advanced Encryption Standard," *J. Eng. Res. Appl. IJERA ISSN*, vol. 2248, p. 9622, 2013.
[50] A. F. M. Sultanul Kabir, M. A. Shorif, H. Li, and Q. Yu, "A study of secured wireless sensor networks with XBee and Arduino," in *Systems and Informatics (ICSAI), 2014 2nd International Conference on*, 2014, pp. 492–496.
[51] P. Luong, "Securing Embedded Systems for Autonomous Aerial Vehicles," WORCESTER POLYTECHNIC INSTITUTE, 2013.
[52] "XBee ZigBee - Digi International." [Online]. Available: http://www.digi.com/products/xbee-rf-solutions/modules/xbee-zigbee. [Accessed: 16-Nov-2015].
[53] "What is Advanced Encryption Standard (AES)? - Definition from WhatIs.com," *SearchSecurity*. [Online]. Available: http://searchsecurity.techtarget.com/definition/Advanced-Encryption-Standard. [Accessed: 16-Nov-2015].
[54] M. 22 and 2013 Lou Frenzel | Electronic Design, "What's The Difference Between IEEE 802.15.4 And ZigBee Wireless?" [Online]. Available: http://electronicdesign.com/what-s-difference-between/what-s-difference-between-ieee-802154-and-zigbee-wireless. [Accessed: 02-Jun-2016].
[55] C. Karlof, N. Sastry, and D. Wagner, "TinySec: a link layer security architecture for wireless sensor networks," in *Proceedings of the 2nd international conference on Embedded networked sensor systems*, 2004, pp. 162–175.
[56] R. A. Jones and B. Horowitz, "A System-Aware Cyber Security architecture," *Syst. Eng.*, vol. 15, no. 2, pp. 225–240, 2012.
[57] S. Malhotra, "Technical Report: Security Engineering Project: System Aware Cyber Security for an Autonomous Surveillance System On Board an Unmanned Arial Vehicle | Systems Engineering Research Center." .
[58] J. Weimer, O. Sokolsky, N. Bezzo, and I. Lee, "Towards Assurance Cases for Resilient Control Systems," in *Cyber-Physical Systems, Networks, and Applications (CPSNA), 2014 IEEE International Conference on*, 2014, pp. 1–6.
[59] M. Pajic *et al.*, "Robustness of attack-resilient state estimators," in *ICCPS'14: ACM/IEEE 5th International Conference on Cyber-Physical Systems (with CPS Week 2014)*, 2014, pp. 163–174.
[60] H. Fawzi, P. Tabuada, and S. Diggavi, "Secure state-estimation for dynamical systems under active adversaries," in *Communication, Control, and Computing (Allerton), 2011 49th Annual Allerton Conference on*, 2011, pp. 337–344.
[61] A. Teixeira, I. Shames, H. Sandberg, and K. H. Johansson, "Revealing stealthy attacks in control systems," in *Communication, Control, and Computing (Allerton), 2012 50th Annual Allerton Conference on*, 2012, pp. 1806–1813.
[62] Y. Chen, S. Kar, and J. M. Moura, "Dynamic attack detection in cyber-physical systems with side initial state information," *IEEE Trans. Autom. Control*, 2016.
[63] M. Sugihara, "A dynamic continuous signature monitoring technique for reliable microprocessors," *IEICE Trans. Electron.*, vol. 94, no. 4, pp. 477–486, 2011.
[64] Y.-Y. Chen and K.-L. Leu, "Signature-monitoring technique based on instruction-bit grouping," *IEE Proc.-Comput. Digit. Tech.*, vol. 152, no. 4, pp. 527–536, 2005.

[65] R. W. Horst, R. L. Harris, and R. L. Jardine, “Multiple instruction issue in the NonStop Cyclone processor,” in *ACM SIGARCH Computer Architecture News*, 1990, vol. 18, pp. 216–226.
[66] C. R. A. Gonzalez and J. H. Reed, “Detecting unauthorized software execution in SDR using power fingerprinting,” in *MILITARY COMMUNICATIONS CONFERENCE, 2010-MILCOM 2010*, 2010, pp. 2211–2216.
[67] “Wishbone system-on-ship (SoC) interconnection architecture for portable ip cores,” https://opencores.org/opencores,wishbone, 2002.
[68] S. M. Trimberger and J. J. Moore, “FPGA security: Motivations, features, and applications,” *Proc. IEEE*, vol. 102, no. 8, pp. 1248–1265, 2014.
[69] J. M. Johnson and M. J. Wirthlin, “Voter insertion algorithms for FPGA designs using triple modular redundancy,” in *Proceedings of the 18th annual ACM/SIGDA international symposium on Field programmable gate arrays*, 2010, pp. 249–258.
[70] H. Alemzadeh, Z. Jin, Z. Kalbarczyk, and R. K. Iyer, “An embedded reconfigurable architecture for patient-specific multi-paramater medical monitoring,” in *Engineering in Medicine and Biology Society, EMBC, 2011 Annual International Conference of the IEEE*, 2011, pp. 1896–1900.
[71] H. Lin, H. Alemzadeh, D. Chen, Z. Kalbarczyk, and R. K. Iyer, “Safety-critical cyber-physical attacks: Analysis, detection, and mitigation,” in *Proceedings of the Symposium and Bootcamp on the Science of Security*, 2016, pp. 82–89.
[72] H. Alemzadeh, R. K. Iyer, Z. Kalbarczyk, and J. Raman, “Analysis of safety-critical computer failures in medical devices,” *IEEE Secur. Priv.*, vol. 11, no. 4, pp. 14–26, 2013.