

INDUSTRIAL CONTROL SYSTEM DATA RESILIENCY

by

Daniel A. Bovard



A thesis

submitted in partial fulfillment

of requirements for the degree of

Master of Science in Electrical and Computer Engineering

Boise State University

August 2021

© 2021

Daniel A. Bovard

ALL RIGHTS RESERVED

BOISE STATE UNIVERSITY GRADUATE COLLEGE

DEFENSE COMMITTEE AND FINAL READING APPROVALS

of the thesis submitted by

Daniel A. Bovard

Thesis Title: Industrial Control System Data Resiliency

Date of Final Oral Examination: 11 March 2021

The following individuals read and discussed the thesis submitted by student Daniel A. Bovard, and they evaluated the student's presentation and response to questions during the final oral examination. They found that the student passed the final oral examination.

Sing Ming Loo, PhD. Chair, Supervisory Committee

Charmaine C. Sample, DSc. Member, Supervisory Committee

Liljana Babinkostova, PhD. Member, Supervisory Committee

The final reading approval of the thesis was granted by Sin Ming Loo, Ph.D., Chair of the Supervisory Committee. The thesis was approved by the Graduate College.

DEDICATION

This thesis is dedicated...

To my father Robert, my role model in life, who has guided and inspired me to apply my utmost effort, talents, and energy into each of life's challenges, no matter how big or small. Thank you for being a source of inspiration and strong foundation I can always look up to for advice and support.

To my mother Jane, who has nurtured and cared for me in every stage of my life, giving me motivation and strength in my times of weakness and despair. Thank you for being a shoulder to lean on and reminding me to enjoy and honor the life God gave me by giving more to the world than I receive.

To my fiancé Natalie, who throughout our journey together has supported all my endeavors in my long-distance pursuit of my education and career. Thank you for your unconditional love and support as we reach the end of our separation and move onto the next act of our lives together.

ACKNOWLEDGEMENTS

I would like to express my deep gratitude to the committee chairman and my thesis advisor Dr. Sin Ming Loo. Thank you for the many opportunities, much guidance, ideas, and mentorship in the pursuit of this research and my degree. Your support has allowed me to succeed in both my education and my career sooner than I thought possible.

I would like to thank Idaho National Laboratory and Dr. Charmaine Sample for the internship and sponsorship while pursuing this research. Thank you for enabling me to pursue this thesis and making my research efforts possible. I hope this research is of benefit, and should the need arise, I look forward to the opportunity to assist in any further developments.

I am grateful for the guidance of Mark Laverty in the production of the hardware developed for this thesis. Thank you for your guidance and wisdom in electronics! Thank you for helping me in addressing the power problems I struggled with daily. I was lucky to have been able to work with you and benefit from your knowledge.

I would like to thank Dale Reese and Gavin Huggins of Idaho Scientific for my internship and employment throughout this entire process. Thank you for giving me the time and flexibility to properly finish my education.

Lastly, I want to thank my friends and colleagues who believed in me from the beginning.

ABSTRACT

This thesis identifies and fortifies against a critical vulnerability in industrial control system (ICS) security. A properly designed ICS security framework consists of a multi-layered approach starting with heavy fortifications in information technology and ending with control information of operational technology. Currently, ICS security frameworks lack visibility and place blind trust in devices at the lowest level of the control hierarchy. Attaining control data visibility at the lowest level of the control hierarchy is critical to increasing the resiliency of an ICS security posture. This thesis demonstrates how this data can be captured at the lowest level of the control hierarchy, and then synthesized with existing network and system data to form a more complete picture of operational technology's behavior.

TABLE OF CONTENTS

DEDICATION	iv
ACKNOWLEDGEMENTS	v
ABSTRACT	vi
LIST OF FIGURES	xi
LIST OF ABBREVIATIONS	xiv
CHAPTER 1: INTRODUCTION	1
System Thinking	2
System Degradation & Security Threats	3
Control Systems and Visibility	4
Thesis Topics	7
Layout of This Thesis.....	8
CHAPTER 2: PREVIOUS RESEARCH AND TECHNOLOGY	10
Typical ICS Facility Structure.....	10
External Network	13
Enterprise Layer	14
Supervisory Layer	14
Field Layer	16
Control Process Layer	16
Advanced Persistent Threats	17

ICS Cyber-Attacks.....	18
Stuxnet.....	18
Ukrainian Electrical Grid	19
Best ICS Cybersecurity Practices.....	20
Removal of Unnecessary Features.....	21
Data Integrity.....	22
Authentication & Authorization	29
Secure Communication Protocols.....	34
Intrusion Detection Systems.....	39
Network Enclave.....	46
Threat Modeling & Intelligence	48
MITRE ATT&CK Framework.....	49
Threat Intelligence	52
Detection & Analysis	52
Emulation	53
Assessments.....	54
CHAPTER 3: RESILIENT ICS DATA.....	56
The Proposed System	57
CHAPTER 4: CYBER RESILIENCY IN SYSTEMS & DATA.....	61
System Thinking.....	62
Cyber Resiliency	64
System Resiliency	67
Data Resiliency	68

CHAPTER 5: DATA ACQUISITION SYSTEM.....	71
Example Design.....	71
Degradation & Infiltration.....	74
Degradation.....	74
Infiltration.....	76
System Integration.....	77
CHAPTER 6: HARDWARE DESIGN.....	80
Control Tap Overview.....	80
Amplifier Sensor Circuits.....	81
Current Sensor Circuit.....	83
Voltage Sensor Circuit.....	86
PWM Sensor Circuit.....	86
ADC Interface.....	88
CHAPTER 7: SOFTWARE DESIGN.....	94
ADC Reader API.....	96
PLC Reader API.....	98
Control Monitoring Process.....	99
CHAPTER 8: FUTURE RESEARCH.....	104
Artificial Intelligence & Machine Learning.....	104
FPGA Data Acquisition.....	105
CHAPTER 9: CONCLUSION.....	111
REFERENCES.....	115
APPENDIX A.....	119

APPENDIX B.....	126
APPENDIX C.....	134

LIST OF FIGURES

Figure 1.1	DCS Layout.....	5
Figure 1.2	Control Feedback Loop	6
Figure 2.1	Purdue Model for Control Hierarchy	12
Figure 2.2	DMZ Architecture	13
Figure 2.3	Disabling of Unused Features	22
Figure 2.4	Data Integrity System	29
Figure 2.5	EAP - Pre-Shared Key	33
Figure 2.6	TLS Handshake	38
Figure 2.7	Network-Based IDS.....	43
Figure 2.8	Host-Based IDS.....	45
Figure 2.9	ICS Network Enclave	48
Figure 2.10	ATT&CK Navigator Heat Map	51
Figure 3.1	Proposed Thesis Contribution.....	59
Figure 5.1	Water Tank Control System.....	72
Figure 5.2	Water Tank Control Loop.....	73
Figure 5.3	Recorded Control Process.....	74
Figure 5.4	Water Tank Control System with Monitor	79
Figure 6.1	Amplifier Sensor Circuit Block Diagram	81
Figure 6.2	Analog to Digital Conversion Process.....	82

Figure 6.3	Differential Amplifier Circuit.....	84
Figure 6.4	Current Amplification Stages.....	85
Figure 6.5	Current Sensor Amplifier Circuit.....	85
Figure 6.6	Voltage Sensing Amplifier Circuit.....	86
Figure 6.7	Pulse Width Modulation.....	87
Figure 6.8	PWM Sensing Amplifier Circuit.....	88
Figure 6.9	Raspberry Pi GPIO Interface.....	89
Figure 6.10	Typical SPI Structure.....	90
Figure 6.11	ADC Chip Selecting Circuit.....	91
Figure 6.12	PCB and Raspberry Pi (front).....	92
Figure 6.13	PCB and Raspberry Pi (back).....	93
Figure 7.1	IDS Software Flow.....	95
Figure 7.2	BCM2835 SoC Block Diagram.....	96
Figure 7.3	ADC SPI Transaction Format.....	97
Figure 7.5	Normal Control Process Behavior.....	102
Figure 7.4	Disturbed Control Process Behavior.....	102
Figure 7.6	Undetected Disturbance.....	103
Figure 7.7	False Control Process.....	103
Figure 8.1	FPGA Resources.....	106
Figure 8.2	FPGA Data Acquisition System.....	107
Figure 8.3	FPGA Block Design.....	108
Figure 8.4	FPGA / PCB Connection.....	108
Figure 8.5	SPI Waveform.....	109

Figure 8.6 Serial Terminal Data Responses109

LIST OF ABBREVIATIONS

ADC	Analog-to-Digital Converter
AH	Authentication Header
API	Application Program Interface
APT	Advanced Persistent Threat
CHAP	Challenge Handshake Authentication Protocol
CLB	Configurable Logic Block
CS	Chip Select
DAC	Digital-to-Analog Converter
DC	Direct Current
DCS	Distributed Control System
DDoS	Distributed Denial of Service
DMZ	Demilitarized Zone
DNS	Domain Name System
DoS	Denial of Service
EAP	Extensible Authentication Protocol
ECC	Error Correction Code
ESP	Encapsulated Security Payload

FPGA	Field Programmable Gate Array
FTP	File Transfer Protocol
GPIO	General Purpose Input/Output
HIDS	Host-based Intrusion Detection System
HMAC	Hash-Based Message Authentication Code
HMI	Human-Machine Interface
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IC	Integrated Circuit
ICS	Industrial Control System
IDS	Intrusion Detection System
IETF	Internet Engineering Task Force
I/O	Inputs/Outputs
IP	Internet Protocol
(networking)	
IP	Intellectual Property
(property)	
IPSec	Internet Protocol Security
IT	Information Technology
LAN	Local Area Network
MAC	Message Authentication Code

MCU	Microcontroller Unit
MISO	Master Input / Slave Output
MITM	Man-in-the-middle
MOSI	Master Output / Slave Input
NIDS	Network-based Intrusion Detection System
NTFS	New Technology File System
OS	Operating System
OSI	Open Systems Interconnection
OT	Operational Technology
OTP	One-Time Password
PAP	Password Authentication Protocol
PCB	Printed Circuit Board
PID	Proportional-Integral-Derivative
PLC	Programmable Logic Controller
PSK	Pre-Shared Key
PWM	Pulse Width Modulation
RAID	Redundant Array of Inexpensive Disks
SCADA	Supervisory Control and Data Acquisition
SCLK	System Clock
SMB	Server Message Block

SPI	Serial Peripheral Interface
SSH	Secure Shell
TCP	Transmission Control Protocol
TFA	Two-Factor Authentication
TLS	Transport Layer Security
VPN	Virtual Private Network
WAN	Wide Area Network
WPA	Wifi-Protected Access

CHAPTER 1: INTRODUCTION

Industrial control systems (ICSs) are used to control different forms of automation and are classified into two main categories. There are Supervisory Control and Data Acquisition (SCADA) systems and Distributed Control Systems (DCSs). Each type of control system has process defined inputs and outputs (I/O) which vary between different use cases and applications. SCADA systems tend to be used for the automated monitoring, flow, and distribution of necessary resources such as gas, electricity, water, and material. DCSs tend to be utilized when controlling an automatic physical process in an industrial control facility via Programmable Logic Controllers (PLCs).

Together, these two types of systems comprise a large portion of ICSs found in an industrial control facility. Every control system relies on a controller receiving control inputs to produce control outputs which return feedback signals. The controller receives these feedback signals and adjusts the controls accordingly. This concept is known as a control feedback loop and is at the core of control system theory. Control inputs and outputs are also reported to a DCS, which orchestrates higher level control and organization between multiple control processes.

Protecting a facility's capital, employees, and intellectual property is of utmost importance. Therefore, the control systems entrusted with this duty, along with their communication channels, must be reliable and secured. Additionally, control systems, their supporting systems, and networks within the facility should be designed and

assembled to be resistant and resilient to component degradation, as well as various intrusions.

System Thinking

Industrial control facilities can be conceptualized as a system of systems. There are communication systems, computer systems, security systems, control systems, etc. At some point, each of these systems interact with each other within the facility and work together to deliver services. Fully understanding the ICS environment requires an understanding of how these systems interact with one another.

First, many systems are structured with many different channels of feedback. The connectedness of systems means that they move together, rather than independent subsystems. Within human nature and different cultures, humans tend to think in terms of causality, where A causes B. However, that is not always the case as the universe is a very complex system of systems, where often B may also cause A [11].

Second, all systems share common features. One such feature is the notion that the capability of an overarching system will be greater than the sum of all system components. Similarly, an understanding of system interfaces, components, and their relationships, can lead to a greater understanding of system performance and reliability.

By understanding systems, security mechanisms can be implemented with greater wisdom to their cascading side effects with the rest of the system. This way, security features that are added to the whole security infrastructure can be more impactful by protecting the attack surface while adding less vulnerabilities. The concepts defined in system thinking are critical to understanding security challenges that modern ICS environments face and aid in forming holistic engineering solutions.

System Degradation & Security Threats

Computer systems, network systems, security systems, and control systems all exist within the scope of an industrial control facility. This means that each of these systems are subject to system degradation over time, and failures can vary from memory cells not being freed, networking devices losing their speed or reliability, or erosion of a motor's axle. Each of these types of component degradation can lead to system-wide degradation and, ultimately system-wide failures. Further, when lacking proper evaluation, these failures may be unexpected, causing problems ranging from minor inconveniences to catastrophic events.

A common sign that systems may be degrading is that the systems' performance and/or behavior lies outside the scope of the system boundaries. While these discrepancies may be mild in nature, they can signal the initial degradation of a component within the system. In control system environments, the consequences of failure can range from the halting of production to physical damage to loss of human life and capital. Monitoring and recognizing when control systems start to degrade is essential. Detection of systems misbehavior can trigger corrective action. Critical components must be identified and well maintained while operating efficiently. Systems can be preemptively halted, inspected for degradation, and promptly repaired.

However, degradation is not the only cause of failures within ICS environments. Another critical area of vulnerability comes from security threats. ICSs are used in many different industries and nations globally, and for this reason are exposed to a wide variety of types and severities of threats. The goals of these threat actors may vary, as do the tactics, techniques, and procedures (TTP) these threats follow. Intruders' main goals tend

to be espionage, intellectual property theft, damage to capital or human life, or the shutting down of operations. Intruders could be industry rivals, criminal groups, disgruntled insiders, foreign adversaries, etc. These threats may be divided into three main categories: basic threats, advanced threats, or advanced persistent threats (APTs).

Basic threats generally stem from amateurs who employ well-known and documented attacks such as social engineering, phishing attacks, or open-source attacks which may be found in the open environment of the internet. While social engineering attacks are considered simple or trivial, they are commonly used because human nature can be coerced with far less effort than computer technology. Advanced threats stem from developed hacker groups or industrial spies who may perform attacks such as distributed denial of service (DDoS) attacks, private data extraction, or extortion via the use of more sophisticated attacks. APTs tend to be sophisticated, organized, and well-funded attackers. They are usually employed by international adversaries who leverage well-developed and cutting edge TTPs to perform new attacks which are referred to as zero-day attacks. APTs are constantly advancing since once a cutting-edge attack has been used, the attack vector loses value. The knowledge gained from the attack is released for defense purposes, thereby making it available to amateur attackers.

Control Systems and Visibility

While SCADA systems are important to the continual operation of an industrial control facility, most physical action resides within DCSs and their subcomponents. A typical DCS will oversee multiple control systems participating in one or many advanced control processes [6]. Within each control system governed by a DCS, multiple components are integrated to achieve the control of a physical process. Figure 1.1 [26]

depicts an example of a DCS layout where multiple control processes are working in parallel to achieve a goal of producing an output product from given input resources. There are programmable logic controllers (PLCs) that are responsible for the observation and control of a process. Such processes can include controlling the pressure inside of a tank, the flowrate through a pipe, or the temperature of an oven. The PLC manages these control outputs (pressure, flow, temperature, etc.) by sending an analog electrical signal (current or voltage) to an actuator such as a pump, heater, motor, etc. The PLCs are programmed with a control process that is oriented to control outputs, such as temperature, by driving the actuators.

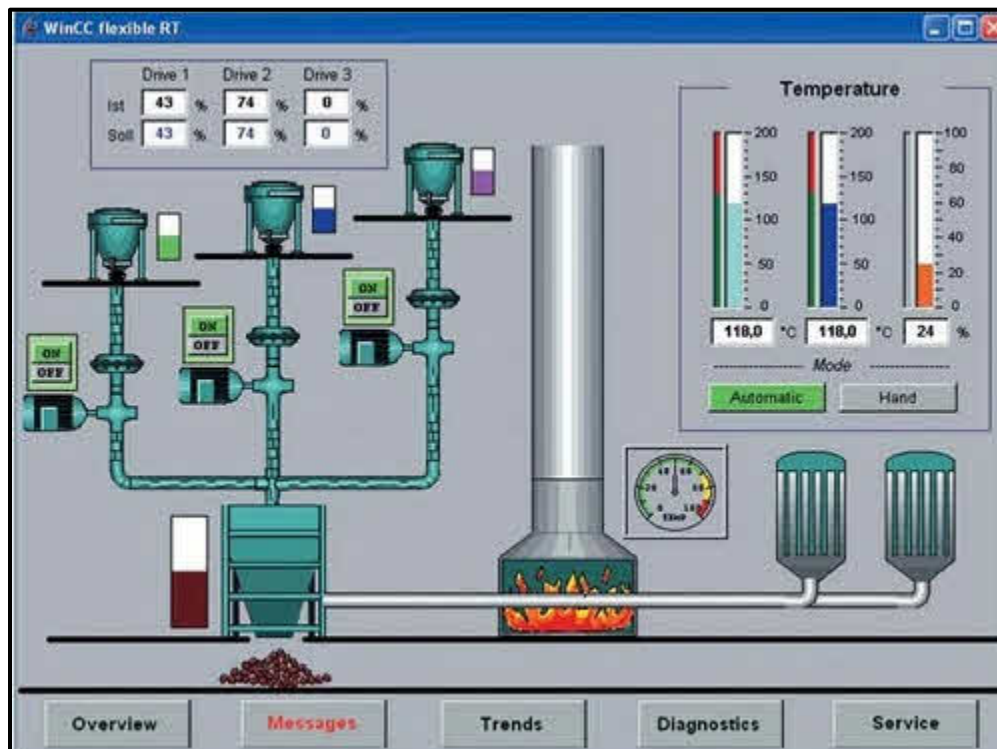


Figure 1.1 DCS Layout

However, this can only be achieved by knowing the current state of the system via the usage of feedback signals. These feedback signals are returned from sensors that inform the PLC of the system status. Such signals can include the actual pressure, flow,

or temperature. With the known state of the system, the PLC can choose to adjust or maintain the electrical outputs that drive the actuators. This concept is known as a control feedback loop. Figure 1.2 depicts a typical control feedback loop.

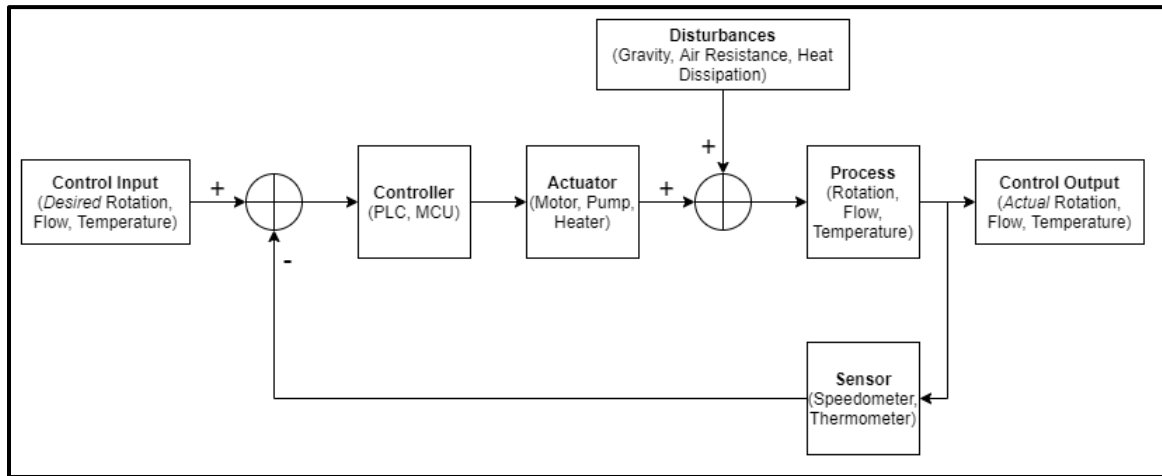


Figure 1.2 Control Feedback Loop

One of the major challenges in protecting systems from degradation and cyber threats is the concept of system visibility. System visibility is the concept of truthfully knowing the state of a system with as little obfuscation as possible. In an environment where detecting degradation and cyber intrusions are important, having extensive visibility on a system is critical. However, when components degrade and attackers infiltrate, the veracity of returned data which is used to detect these events begins to erode. Sensors and PLCs may not report legitimate information when components degrade or have been compromised by an attacker. Degradation and infiltration are difficult to nearly impossible to detect, in part because security products do not collect and process operational technology (OT) data.

To handle this paradoxical problem, a new solution must be derived that breaks the paradox and allows for system visibility even in this face of degradation and cyber

infiltration. This thesis contributes to the field of ICS security by presenting an additional method of data acquisition that increases ICS visibility and data resiliency.

Thesis Topics

This thesis identifies what requirements are needed to make these systems resilient and secure from anomalous device behavior, while limiting the false positive problem that plagues anomaly detection solutions. This thesis also discusses what the term resilient means in the context of ICS security, and thus give system administrators and security engineers a well-defined perspective on which to construct solid foundation upon which security mechanisms can be built.

This thesis reviews some of the vulnerable aspects of industrial control systems infrastructure and provides infamous examples of exploits leveraged against these vulnerabilities. This thesis presents the best cybersecurity practices utilized by ICSs and critical infrastructure today. These practices consist of the removal of unwanted services, data integrity, secure communication protocols, authentication and authorization, intrusion detection systems, and network enclave.

The remainder of this thesis discusses a new type of data acquisition and control monitoring system which may be employed at the lowest level of the control system hierarchy, the physical environment. APTs may have already infiltrated facility networks, infected PLCs, or compromised sensors or actuators. Additionally, control system components such as sensors and actuators may degrade over time causing undesirable system behavior. For this reason, this thesis recommends monitoring and recording the physical responses of a system through an out-of-band channel of information. By doing so, security systems gain visibility via an additional point of reference to detect degrading

systems and the intrusions of adversaries which may alter or disrupt control process behavior. This gap in intrusion detection has been recognized by researchers [2], and this problem is addressed in the pages to follow.

Much of the work in the design and assembly of a data acquisition system is documented in the later chapters of the thesis. The proposed data acquisition system is capable of monitoring and recording the physical behavior of the systems under control. With an out-of-band channel of information, the data acquisition system may be employed as an additional lens and cross-reference to untrusted system signals. The data acquisition system's data can be synthesized with in-band network and system data. This union of data may be used in the generation of alerts when discrepancies arise between information being reported from PLCs, and the data being reported from the data acquisition system. With such a system, system administrators can be alerted of anomalous events and detect device degradation and infiltration when either event occurs. With the data acquisition system, the previous statement holds true even if the existing control infrastructure (ie. PLC) is reporting illegitimate data.

Layout of This Thesis

To thoroughly address the issues raised by this thesis, the following chapters are structured in such a way as to give context to the proposed data acquisition system. Chapter 2 presents the previous research related to this thesis and highlights existing infrastructure, threats, and technology. Chapter 3 elaborates the broader context of this thesis and gives perspective to the proposed resilience-bolstering control monitoring system. Chapter 4 provides background into the underpinnings of the thesis involving system thinking and data resiliency.

The proposed data acquisition and control monitoring system is detailed in Chapter 5. Chapter 6 explains the hardware design of the proposed control monitoring system. The software design and some control monitoring results are presented in Chapter 7. Related future research and avenues of application are presented in Chapter 8. Summary and conclusions are presented in Chapter 9.

CHAPTER 2: PREVIOUS RESEARCH AND TECHNOLOGY

Typical ICS Facility Structure

While the ICS is the core of automation and production in a facility, large amounts of infrastructure exist to schedule, monitor, and supply all production processes taking place. As industrial control facilities grow more sophisticated, the communication, monitoring, and control is further abstracted for ease of management. For this reason, multiple network layers are usually implemented between the facility's marketing or sales departments and the physical control processes. This ICS facility layout is known as the Purdue Model for Control Hierarchy and provides a more layered approach to facility-wide operation and security [5, 13, 14]. An example of an industrial control facility layout following the Purdue Model for Control Hierarchy is presented in Figure 2.1 [30].

Larger ICS facilities may need to communicate with other entities such as management, sales, and suppliers. To establish communication, an external network connection is required to allow communications between facilities, suppliers, and customers. The first layer of an ICS is an external network layer, where internet communication and security are handled. Once sales have been made, the production needs to be scheduled with the lower layers of the ICS facility to meet the demands of the customers in a timely fashion. This layer is typically known as the enterprise layer, and this is where highly abstracted systems allow enterprise users to manage resources and schedule production to meet demands. To meet the demands of the enterprise layer,

production schedules are achieved via the orchestration of engineers that understand the operation of ICSs and their capacities. This layer is known as the supervisory layer, where command and control of ICSs occurs to produce the desired quantity of product. With a plethora of facility-wide information available, engineers may operate control processes via some form of human-machine interface (HMI).

Once control has been executed by the engineers at the supervisory layer, the control information is passed to the next layer. This layer is known as the field layer, where the control inputs direct the ICSs to perform their control processes on the given input resources. All the PLCs, sensors, and actuators exist here. These devices report real-time information about the status of the control processes taking place.

The control layer is essentially a part of the field layer but is differentiated as the control or physical process layer due to its composition of physical system components. This is the layer where the physical processes are taking place, whether it be the controlling of the pressure in a tank, the temperature of a weld, the flow rate of fluid, etc. This thesis proposes that the control layer's analog information may be extracted via an out-of-band data acquisition system whose data can be synthesized with existing in-band network data to gain visibility at the lowest level of the control hierarchy.

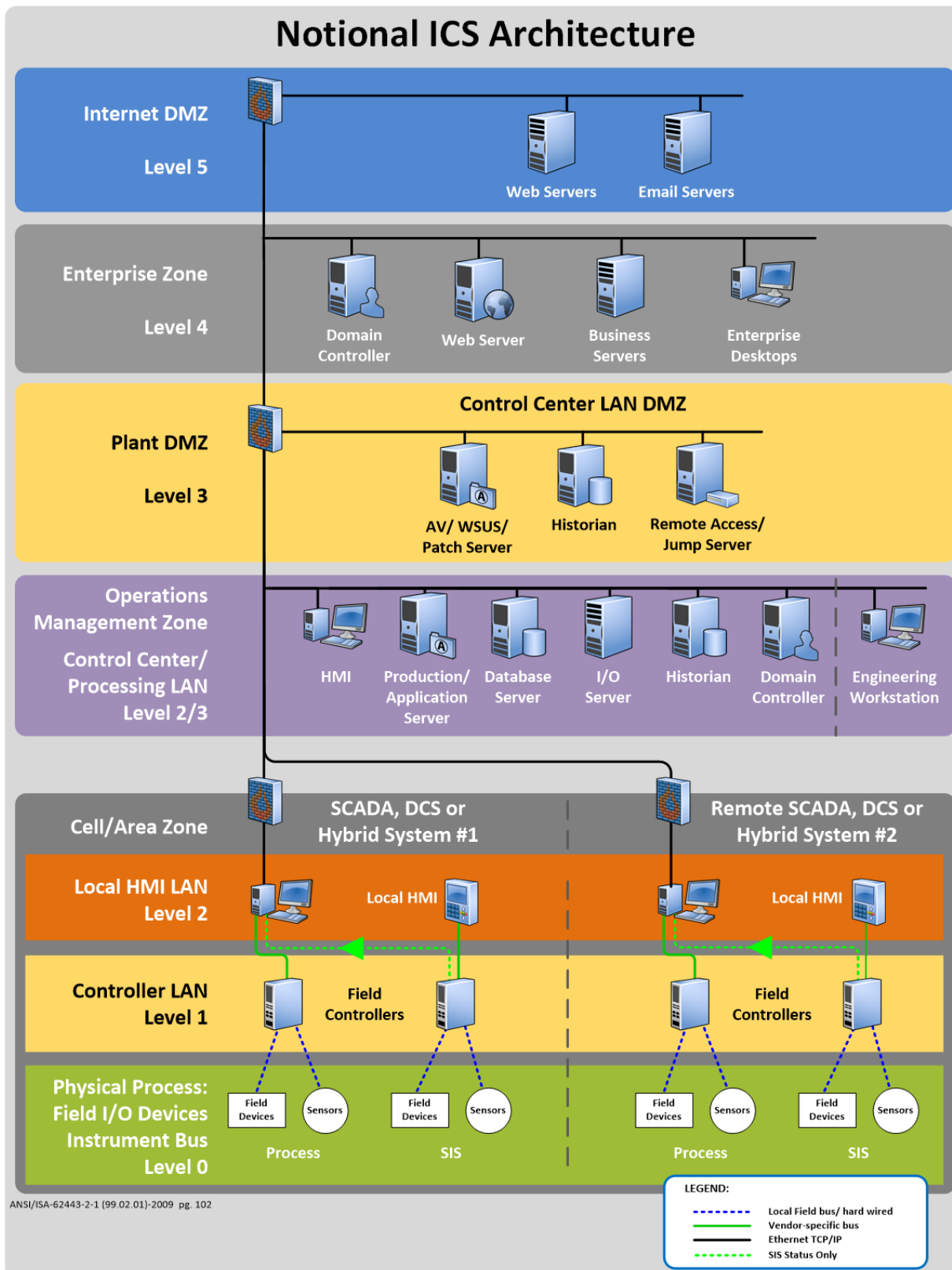


Figure 2.1 Purdue Model for Control Hierarchy

External Network

The external network layer within an ICS facility is needed to communicate with external entities such as customers, suppliers, external management, and other businesses. The internet is a giant connected web, and bad actors around the globe can communicate and coordinate attacks against ICSs through this external facing network, making this layer the single largest threat to an ICS facility. Many protections are put in place to protect and isolate the internal facility networks from the external network.

The most typical protection put in place is known as a demilitarized zone (DMZ). Figure 2.2 depicts the typical DMZ architecture and what services tend to reside in the zone. The DMZ is a subnetwork that stands between the external network and the internal network to filter out and verify all traffic entering the facility's network domain [5, 13, 14]. Firewalls, intrusion detection systems, email and web servers, web services, and other forms of network security are implemented inside the DMZ to serve as the first line of defense.

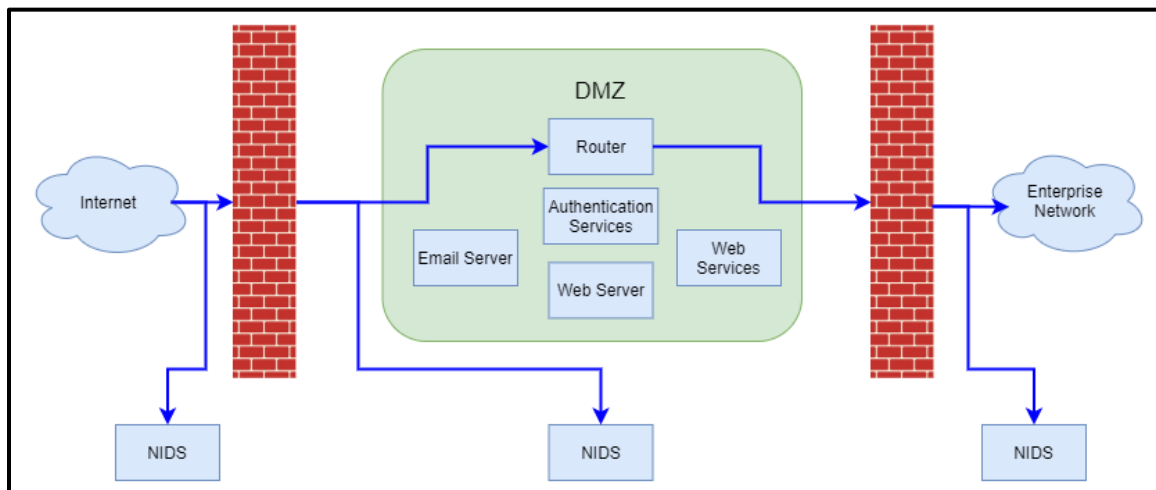


Figure 2.2 DMZ Architecture

Enterprise Layer

The enterprise layer or enterprise network is where all the information technology (IT) infrastructure for the management and corporate side of an industrial control facility resides. Typically, this enterprise network has access to the internet to make sales, contact suppliers, and other types of communication necessary to keep the facility running. Inside the enterprise layer, IT systems are leveraged to schedule production, manage inventory, schedule OT maintenance, and other corporate tasks.

Due to its external network connection, this layer's connection to the internet has the largest attack surface and poses the largest cyber threat. For this reason, there is also a DMZ isolating network traffic into the supervisory and other lower network layers of the ICS facility. In addition to network isolation, there are a myriad of different security mechanisms that are implemented on the network and inside of all hosts and internet-connected devices that reside on the network. Much of these topics will be explained in further detail in the section titled Best ICS Security Practices. The best approach involves addressing the attack surfaces that exist to implement mitigation and protection techniques, thereby providing the greatest amount of coverage. The goal is to leave attackers with minimal to zero available options.

Supervisory Layer

As the exterior layers of the ICS facility are peeled back, the shift from IT to OT technology occurs. The bridge between IT and OT technology mainly exists within the supervisory layer. IT is needed to receive scheduling, resource, and maintenance information from the enterprise layer. IT also reports production related statuses back to the enterprise layer, the OT is properly driven and capable of controlling processes inside

of the field layer. Lastly, IT gathers ICS status information and employs various methods of ICS security such as intrusion detection.

One of the main OT systems used today is SCADA. As the name implies, SCADA retrieves status information about resource assets which may be scattered throughout a facility or even different geographic locations. In addition to resources location, SCADA systems keep track of the flow of resources and waste in and out of the industrial control facility. SCADA is responsible for the maintaining the processes of resource importing and waste removal that keep the facility afloat. By knowing both where resources are and how they are flowing, SCADA can oversee and operate control processes. However, SCADA systems are not responsible for the control processes which act upon the input materials to produce the outputs.

The OT systems in the supervisory layer which handle the controlling of the control processes are DCSs. DCSs are responsible for controlling one to multiple control processes which consist of actuators, sensors, and one or more PLCs. DCSs are the entities responsible for reporting feedback information between dependent control systems, with the objective of aiding in the control of complex and interdependent control processes. To meet the needs of the different control process stages that occur throughout production, DCSs can reconfigure the PLCs and change their response characteristics.

As with the enterprise layer, network intrusions may take place, therefore it is of utmost importance to isolate the field layer from the supervisory layer. Isolation is employed to stop intrusions from propagating further down the control hierarchy. A final DMZ is placed between these two layers to achieve isolation of the control network.

Field Layer

The field layer of the industrial control facility is where control processes physically take place. All the PLCs, actuators, sensors, and transport mechanisms exist here. This layer receives OT information from the supervisory layer and orchestrates necessary control processes to produce output products. Normally, multiple dependent control processes are required to produce an output product. Each ICS and its subsystems are controlled by at least one PLC that is programmed and supervised by a DCS in the supervisory layer. The PLC relays process status information to maintain the flow of production between interdependent processes. Within the field layer exists the final layer of the control hierarchy, the control process layer.

Control Process Layer

The control process layer consists of all the physical devices involved in the production at the facility. Analog electrical signals, whether it be current or voltage, are generated to control actuators such as motors, and report control status information from sensors such as a thermometer. At this layer of the control hierarchy, few solutions exist for ensuring the physical security besides guns, guards, and gates. Additionally, very few systems can even detect degradation or infiltration.

Considering the apparent control security deficiency, this thesis proposes that there is vital information within the physical control inputs and outputs that may be captured outside of the PLC. The physical signals can be captured and monitored by an out-of-band data acquisition system to detect when mechanisms in the control process layer have deteriorated or been compromised. Chapters 5 through 7 will detail the design and implementation of such a system.

Advanced Persistent Threats

APTs are highly sophisticated and spanning networks of organized adversaries seeking to infiltrate the computer networks and systems of government, critical infrastructure, corporations, medical and academic institutions, etc. APTs tend to be either highly organized criminal groups or nation states who have plenty of time and resources to find ways into these targets [19]. APTs bring the advent of zero-day attacks that the world has never seen. The feature that separates APTs from other advanced attackers is their ability to remain unnoticed and undetected for long periods of time before making an attack.

APTs will use all available resources at their disposal to intrude into a network, including social engineering attacks on humans, manufacturing or program defects in hardware or software, common features in operating systems, compromising supply chains, etc. Not only do these attacks require a plethora of manpower, resources, and sophistication, but each APT has a distinct mode of operation that is documented. APTs can reduce system complexity of large systems by strategically targeting the smaller and less complex functions within the system. These subsystems are broken down, understood, and leveraged in advanced attacks. The best way to vet against this pervasive threat is by taking on the perspective of an outside intruder and analyzing the system to find each element that may be corrupted, and therefore must be secured. In addition to securing the system elements, it is necessary to verify the security and integrity of third-party vendors, supplies, and their networks.

ICS Cyber-Attacks

Stuxnet

The most notorious and discussed attacks leveraged against ICSs around the world is known as Stuxnet. Stuxnet is a dangerous computer worm that was discovered during a cyber-attack against the Iranian nuclear program in 2010 [7]. The worm enabled attackers to take control of multiple PLCs that were controlling nuclear centrifuges. The worm was designed to settle itself inside the host computer, and after a designated period, begin propagating to other hosts on the network in search of PLCs running the Siemens Step7 software. It then released its payload, altering the firmware running in a PLC. Once discovered, Stuxnet was found to have infected over 200,000 computers and destroyed a large portion of the Iranian nuclear program's centrifuges.

Stuxnet specifically targeted Windows based systems and exploited four zero-day operating system vulnerabilities [7]. Stuxnet has many propagation techniques, allowing it to spread laterally on a large computer network despite established securities. One method for self-replication was through taking advantage of an exploit allowing for auto-execution of removable drives plugged in anywhere. Alternatively, the worm spread through local area network (LAN) connections by leveraging an exploit of the Windows Print Spooler which stores print jobs on the LAN. Another exploit Stuxnet leveraged was through the server-wide filesystem protocol known as server message block (SMB), which allowed Stuxnet to copy itself to other hosts. Lastly, Stuxnet copied and executed itself through network shared resources or SCADA database server connections.

Once Stuxnet was on a host, it could begin searching for the Siemens Step7 software, responsible for programming and running PLCs. If found, Stuxnet would copy

itself into Step7 projects to automatically execute when a project was loaded into a PLC. Once inside of the PLC, Stuxnet utilized peer-to-peer LAN connections to update the global status of the worm across all machines infected. Additionally, the virus took advantage of privilege escalation vulnerabilities which allowed the PLC to contact command and control servers and execute code. From there host information about the PLCs and computers could be sent across a hypertext transfer protocol (HTTP) connection to a server controlled by the attacker [7].

Stuxnet proved to be devastating to the field of industrial control, because of its ability to leverage multiple vulnerabilities to replicate, execute code, modify programs, and report back status information, all while hiding itself and bypassing security.

Ukrainian Electrical Grid

Another great example of an attack against ICS environments was the cyber-attack on the Ukrainian Electrical Grid in 2015. In late December of 2015, power outages across large regions of Ukraine were reported as multiple 100kV and 23.35kV substations had been compromised and shut down due to an attack on the power company's SCADA systems. The outage lasted several hours before the power company had to assume manual control of each of these substations, but at that point, the outages had affected approximately 225,000 customers.

Like Stuxnet, the attacker was sophisticated and organized in their attack, leveraging multiple exploits and vulnerabilities to infiltrate and maneuver the internal systems of the power company. The attacker gained access through spear phishing emails and malware known as BlackEnergy 3, which were embedded inside of Microsoft Word documents that unsuspecting employees downloaded and viewed. Once the attacker had a foothold

inside of the IT portion of the power company, they were able to extract high-privilege credentials to gain access to the SCADA and ICS networks [10].

Having the valid credentials gave the attacker a foothold. They then used a virtual private network (VPN) connection including the valid credentials to enter the SCADA and ICS networks. Once on the network, they either leveraged existing tools in the environment or accessed commands from the human to machine interfaces (HMIs) located at each of the substations. From there they were able to alter then delete system records and logs to hide exact details on how the attack was performed. Lastly, the attacker was able to shut down the electrical load systems all while performing a telephone denial of service (DoS) attack against the call centers, stopping outage reports from being received and acknowledged [10].

Best ICS Cybersecurity Practices

Industrial security, or security in general, requires an understanding of attack vectors and attack surfaces. An attack vector is an avenue an attacker may take to gain access to a computer network or host, or deliver a malicious payload. Systems may have multiple attack vectors at different regions of the systems, such as network endpoints, computer hosts, or even unsuspecting users. The assortment of all the system attack vectors is known as an attack surface. The attack surface is always changing as system components are modified and as new attacks are created and discovered. The defenses include the disabling of unused features, data integrity mechanisms, authentication and authorization systems, secure communication protocols, intrusion detection systems (IDS), and network enclave. This chapter will cover some common attack vectors and their defenses that comprise the typical ICS attack surface.

Removal of Unnecessary Features

The most basic form of security used today is the removal of attack vectors from a system entirely. Doing so reduces the attack surface available and can be accomplished by removing or deactivating unused features, processes, ports, or connections of PLCs, computer networks, computer hardware systems, or software applications. Figure 2.3 depicts the disabling of unused features in an IDC environment. In a PLC, there may be input and output ports that are not being used but are still enabled, thereby introducing an avoidable attack vector into the control system. In a computer network, there may be unused network internet protocol (IP) addresses, unused physical or virtual network ports, or unnecessary subnetwork connections in place. In a computer system, there may be several elements not in use, including operating system services, network or peripheral ports, or even needless applications. In a software application, there may be entire libraries linked into the program space when only a few functions are being used. For each of these circumstances, the best practice is to remove unused mechanisms from the system as they provide avenues for an adversary to gain access and control of critical systems and infrastructure. This simple practice can drastically reduce the attack surface of a system.

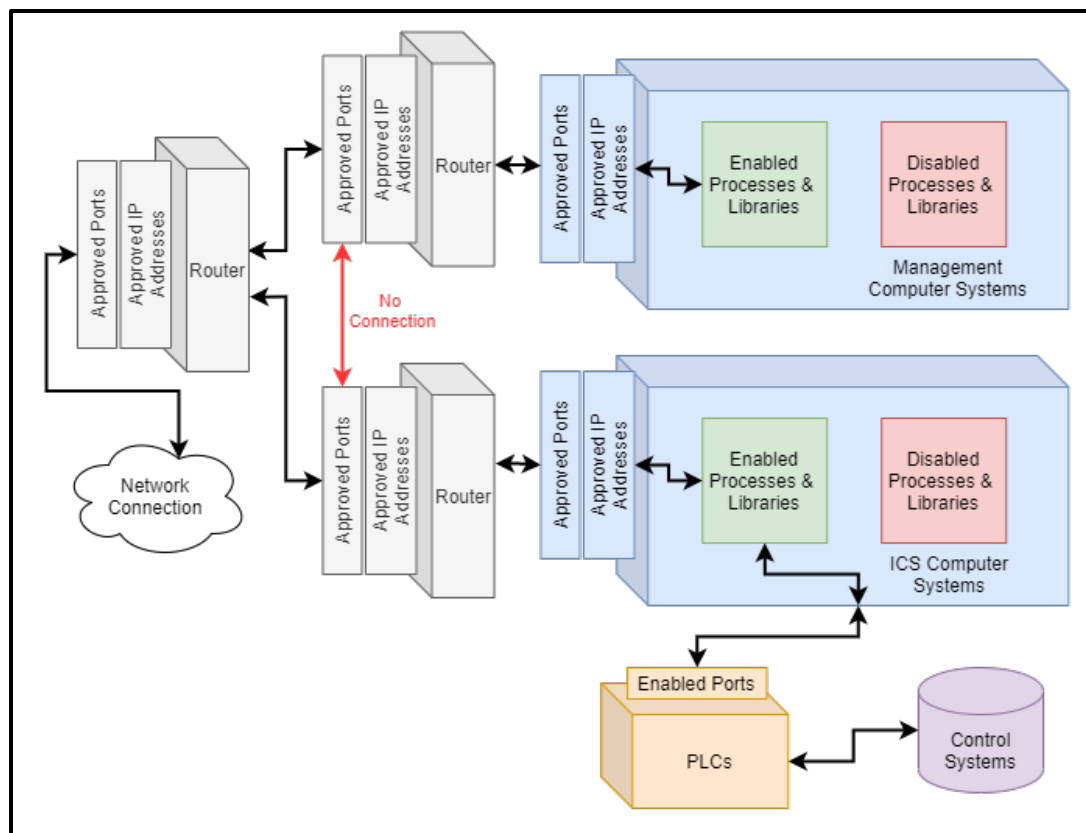


Figure 2.3 Disabling of Unused Features

Data Integrity

There are many attack vectors into hardware and software systems that involve the unauthorized alteration of the data being used. From memory attacks to data corruption, data integrity violations may be catastrophic to a computer system. In an ICS facility, this may lead to attacker intrusion, program execution, or IP extortion. Therefore, data integrity should be applied to every system and subsystem of the industrial control system [17]. Data integrity practices are implemented in firmware, software, hardware, memory storage, data communication, or file systems and operating systems.

Data integrity assurance functions within three scopes: integrity violation avoidance, detection, and correction [17]. Avoidance is a practice of preventative action meant to avoid certain types of data loss. Detection, the most common method of

integrity assurance, identifies when there have been modifications to data. Correction is the act of repairing data which has been lost, modified, or damaged. By itself correction is useless and requires some form of detection to know when it is necessary to repair the damaged data. Normally, detection and correction go hand in hand, and avoidance is an extra precaution to form a well-rounded data integrity assurance policy within a system.

Avoidance

There are several commonly used methods of data integrity violation avoidance, each providing a unique guarantee of the integrity of data being stored or transmitted. The first and most utilized method is read-only memory [17]. By preventing certain portions of memory from being written over by programs or users, data integrity violations may be prevented. However, this method does not prevent software or hardware glitches from damaging data and bypassing these built-in operating system protections.

Another robust method of avoiding data integrity violations is the implementation of cryptographic filesystems [17]. By maintaining confidentiality in system data, attackers have no feasible or predictable way of causing real damage without any knowledge of the data being modified. Without a key to the encrypted data, it is infeasible to start overwriting random portions of data in the hopes of causing a system crash, failure, or security breach. Additionally, authentication can be performed on the encrypted data via a hash function to check the integrity of the data. However, like read only memory, these memory protections are not immune to software and hardware glitches or failures which may cause losses in data or storage in incorrect locations.

A more modern approach to protecting file systems through data integrity avoidance is the usage of transactional file systems. This method journals every system action as a transaction which is logged to always maintain the system in a recoverable state [17]. One great example of a transactional file system is known as New Technology File Systems (NTFS). In NTFS, several mechanisms besides journaling are employed to protect the system from unexpected data integrity violations, whether they be caused by a user or malicious attacker, or by software or hardware glitches.

Detection

Data integrity violations may be avoided, but some of causes of data integrity losses are unavoidable in computer systems [17]. While detecting cannot repair or recover lost data, it is necessary to first detect integrity violations before they may be corrected. Mirroring or checksumming the data are the most trivial ways of detecting integrity violations. To mirror, a complete copy of the data is made, and the pieces of data are compared byte by byte. To run a checksum, the data is served as an input to some one-way function, and the output is compared to a precomputed result stored at the end of a section of data. If any discrepancies arise, then there was a modification to the data somewhere. However, not only are these methods inefficient, but they can only indicate whether there is an integrity violation, and not which piece of data is corrupted. Another common method used in error detection today is known as Cyclic Redundancy Checks (CRC) where blocks of data are protected by the appending of extra sequences of data. The CRC performs Frame Check Sequences (FRC) to detect errors on a frame-by-frame basis. This method is used in determining which piece of data has been corrupted

without copying entire portions of data. It is much more efficient in terms of both time and memory usage [17].

Integrity violation detection is also critical for network communications. If messages are intercepted by an attacker, they may modify the data and perform a man-in-the-middle (MITM) attack. A MITM attack is when an attacker gets between two endpoints of communication to secretly steal information, and possibly relay false information to the receiver. One common way of securing communications against altered information is to use Error Correction Codes (ECC). ECCs are additional pieces of redundant information about packets of data which are used to tell whether modifications or errors have appeared in the received data [17]. While these methods are good, they are not fool-proof, and attackers can very precisely modify packets of data to trick ECCs into thinking they have received unmodified data. A more secure, but more resource intensive way of securing communications is to use some form of encryption and/or authentication. By encrypting the data being sent, you not only make the data confidential, but the data can also be proven to be authentic and unmodified. Cryptographic hashing functions may be used to produce a message authentication code (MAC) which is then appended to a message being sent. When the message is received, the MAC can be regenerated by the party which has the key and compared to the MAC that was sent. If the MACs are different, an integrity violation in the transmitted data has occurred [17].

Correction

Half of the battle in correcting integrity violations when they occur is the detection, while the other half of the battle is correcting the data that has been damaged. There are methods in which correction is performed, and there are also methods which

employ both detection and correction in the same algorithm. One of the most common forms of integrity violation correction is achieved by the many implementations of redundant arrays of inexpensive disks (RAID). RAID may be used to ensure the integrity of a filesystem upon system bootup, but also ensures that the memory can be recreated if one of the disks is attacked and vital data is lost. This method is known as RAID parity, where additional parity codes are stored in each of the RAID disk, thus allowing reconstruction of lost data when a violation is detected [17].

Another method of correcting and detecting data at the same time are ECCs, which may be implemented in more than just communication protocols. Stored in ECCs, there are at least three bits for every byte of information stored. While not memory efficient, this method is able to detect as little as two-bit errors in each byte of data, and is able to correct single-bit errors in the data. ECCs are commonly used in servers and critical applications that cannot afford to lose large portions of data. Normally, to be able to detect and correct N bits of information, the $\log_2(N)$ bits are required to perform the correction. There are many classes of ECCs, the most common being parity and hamming-correction codes.

Forward Error Correction (FEC) is a form of ECC used on smaller portions of information that may be used in memory, but are often used in lower-level communication applications. In FEC, check bits are calculated on the transmitter side of communication and are appended to a message to be checked on the receiver side. If any differences are detected, the lost data can be reconstructed. FEC in its simplest form sends redundant copies of packets across a channel. The most common forms of FEC are hamming codes and Reed Solomon codes [17].

ICS Data Integrity

Within an industrial control facility there are many computer systems, network connected devices, and PLCs which require many forms of data integrity checking. Data integrity protection on these systems may be divided into three categories: firmware or OS integrity, program and memory integrity, and communication integrity.

The first category of ICS system data integrity involves the firmware, or OS of ICS systems. If any data is manipulated within the firmware on these systems, the system could crash, vital information could be altered or damaged, or escalated permissions could be given to an adversary. To avoid this, it is recommended to frequently verify the integrity of firmware by using a verification tool that confirms the integrity by recreating and verifying a digital signature which is only re-creatable by the owners of the device.

The second category of ICS system data integrity involves the programs and memory storage of the systems running within the ICS. The functionality and reliability of the PLC software depend on the integrity of the program and instruction data of the program being run. If improper programming techniques are used, programs may become weak to integrity violation attacks, which give attackers an avenue of control within the systems. A sophisticated attack may be able to change program instructions or data, causing improper behavior of the ICS system. To avoid this, many forms of integrity checks are recommended. To protect the integrity of data being used by the system, integrity checkers which verify integrity of the program before execution, and online integrity checkers which scan before every critical read or write operation are recommended.

The third area of ICS data integrity is the integrity of transient values being sent to and received by an ICS system, which are used to make live decisions. This is more challenging because the data is foreign to these systems. Integrity may be accomplished by using secure communication protocols and live data integrity checking mechanisms. These checks for data integrity and authenticity ensure the data being received is legitimate and has not been tampered with.

By focusing on the data integrity of every system within an ICS, the system boundaries become much stronger and it becomes much more difficult for a system to be broken when its data has been altered by an adversary. Figure 2.4 depicts a simple example of a computer system that performs data integrity checking on network communications, data storage to external memory, PLC data and program integrity checking, and lastly filesystem, integrity checking for the operating system.

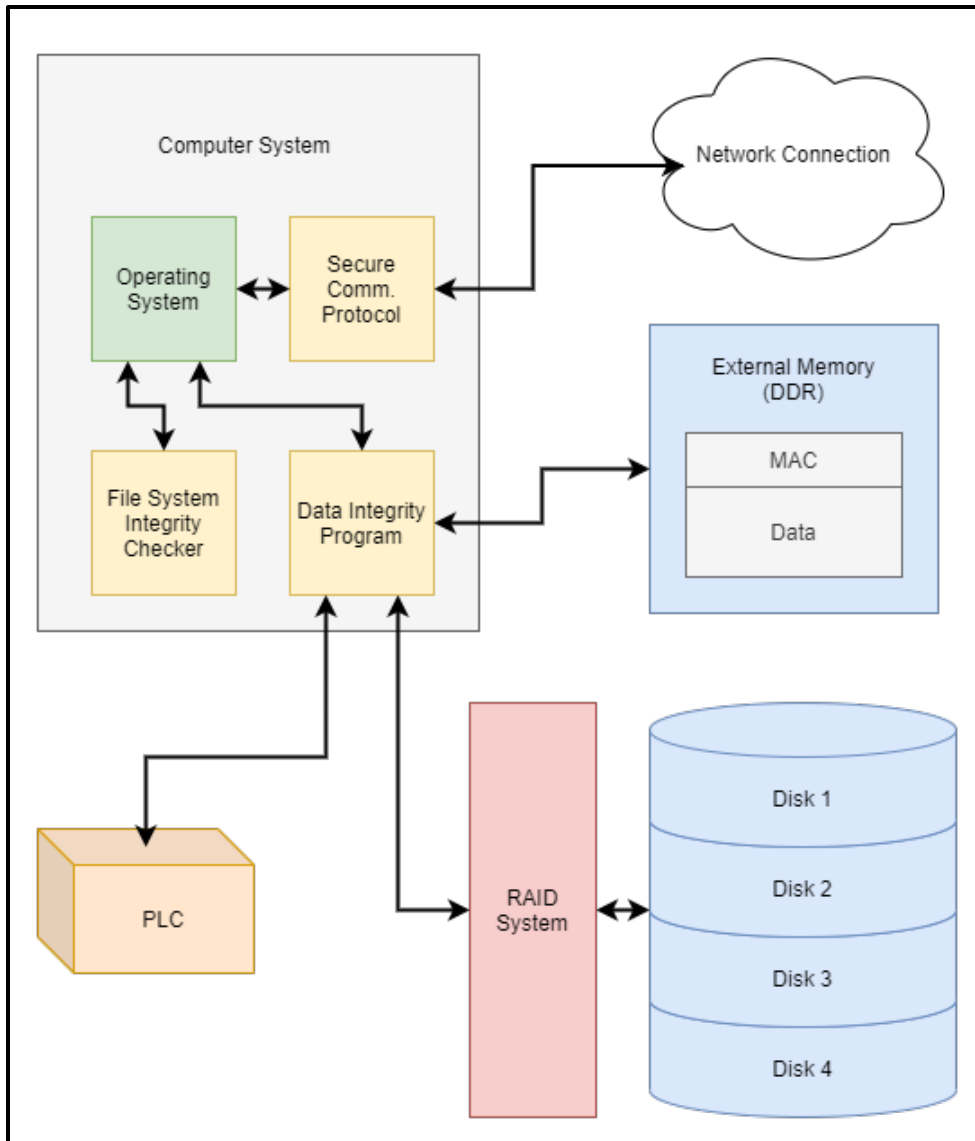


Figure 2.4 Data Integrity System

Authentication & Authorization

Another vital piece of system security and resiliency is understood by asking the questions: “Who are you?” and “Are you allowed this resource?”. These questions encapsulate the critical security aspects of authentication and authorization.

Authentication is the act of verifying the identity of the origin of data or commands sent and received inside and between system boundaries. Authorization is the act of checking and verifying whether the origin of the data or commands may have access to information

or system functions. By controlling these security mechanisms, system boundaries become much more secure and difficult to be breached by an attacker without necessary credentials [5, 9, 13]. Authentication methods not only verify the origin of the data or commands transmitted and received, but they also may be used to verify the integrity of data that has been stored and later retrieved. Authorization also provides layered access to systems so that only users with a need of access are granted it, reducing the overall attack surface substantially. Together, authentication and authorization allow for secure access to industrial control facility resources by legitimate users who should be using those resources.

Authentication

The many different forms, methods, and applications of authentication are often coupled with some form of confidentiality and data integrity. The origin of the data being transmitted and seeking authentication is typically referred to as the claimant.

Authentication verifies the identity of the claimant by using one or many different authentication factors [21]. The factors frequently used to authenticate the claimant are the following:

1. Something the user knows: A password, PIN, or other piece of private information held by the user.
2. Something the user owns: A hardware authenticator, a cellphone, software certificate, or smartcard.
3. Something that qualifies the user: Something biologically unique to the user such as a fingerprint, DNA sequence, or voice recognition.

4. Something the user can do: Used less commonly in digital systems, a signature or physical gesture.

Added strength of authentication comes from combining the methods of authentication and the strength of the underlying authentication methods themselves. Using more than one form of authentication is often referred to as Two-Factor Authentication (TFA). For example, if someone stole a password, but does not have the relevant hardware authentication device such as a smartcard, then with TFA the attacker is denied access. However, while passwords may be easy to implement in a system, passwords have a few vulnerabilities such as being leaked, stolen, or brute forced. Since passwords may be leaked, stolen, or brute forced, one workaround to this vulnerability is to only use the password once. This type of authentication is known as a one-time password (OTP).

In addition to logging into networks or gaining access to services on a network via password authentication, there are also secure authentication protocols which may allow the private communication of authentication credentials over an unsecure medium such as the internet. One of the oldest authentication protocols, known as Password Authentication Protocol (PAP), is not a secure authentication protocol because user credentials are sent as unmodified plaintext over the network. An upgrade to PAP is known as the Challenge Handshake Authentication Protocol (CHAP), where only username credentials are sent over the network connection, then a unique challenge message is generated and sent as a response to the client. The challenge message is then appended to the password and sent through some form of hashing function, preferably a keyed hashing function, to generate a MAC or hashed-based MAC (HMAC) which is

then sent back to the server. The server performs the same hash function to generate the MAC/HMAC and compares it to the received message. If they are both the same, the user has the correct password that was sent to the server in a secure fashion.

These primitive authentication protocols are insecure by today's standards. For this reason, other advanced forms of authentication protocols have been developed, such as Extensible Authentication Protocol (EAP), which are embedded into different cryptographic communication protocols such as Secure Shell (SSH), Transport Layer Security (TLS), and WiFi-Protected Access (WPA). EAP is an authentication framework that provides many different authentication methods for different communication protocols. Additionally, EAP may be used on many different communication layers of the OSI model, whether they be the data link layer using network switches, the network layer using routers for a LAN, the transport layer over the internet using TCP/IP, or any of the other further abstracted layers. EAP provides different forms of authentication for wired and wireless communications from the client to a network device such as a router, and then finally to the server. EAP supports MAC or message digests authentication such as CHAP, certificate-based authentication methods such as TLS which require secure certificates to be installed on the client, or even OTP authentication methods. Figure 2.5 depicts the authentication and establishment of secure communication channels between a peer and a server via the utilization of EAP Pre-Shared Key (PSK). Both the peer and the server have a PSK, which they use to create MACs from randomly generated numbers which are used to authenticate each other, and subsequently establish a secure communication channel.

Whatever the method taken, it is important to consider and weigh all the authentication options for an ICS facility since the facility combines many different systems together via network connections and different communication protocols.

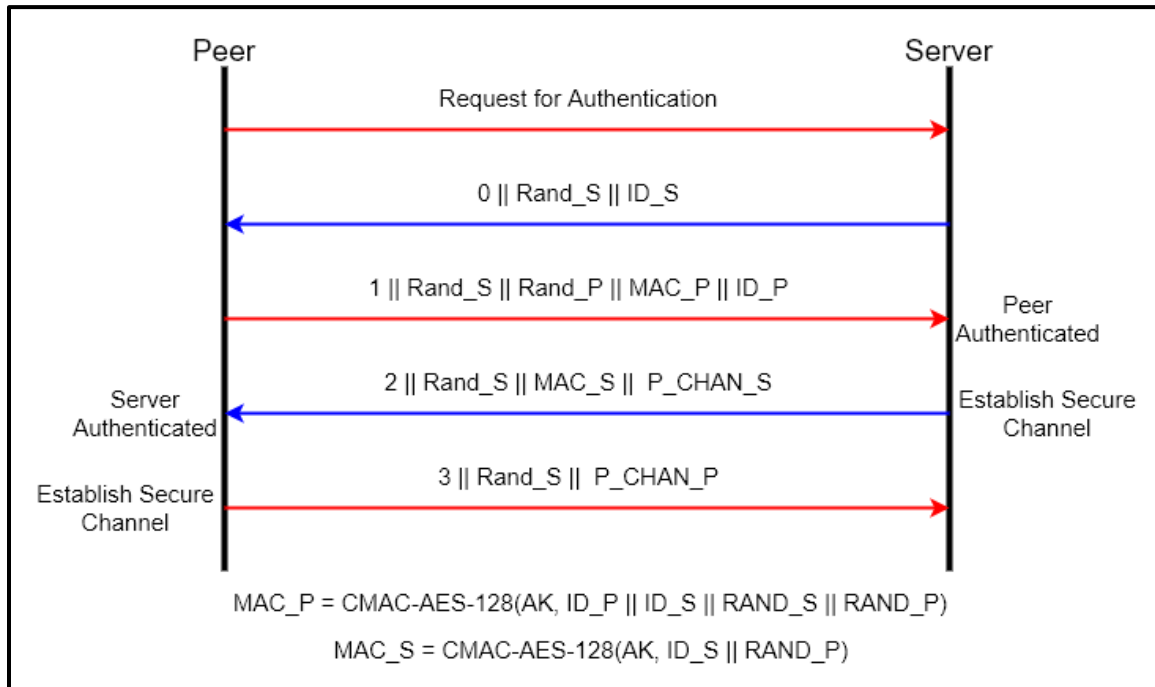


Figure 2.5 EAP - Pre-Shared Key

Authorization

Once the requesting client or incoming data has been authenticated, the process of authorization and access control may begin. Authorization and access control are the process of determining whether the authenticated client or data may gain access or complete transmission to an endpoint within the ICS. Certain clients and sources of data are not authorized to reach different endpoints in an ICS to reduce the overall risk associated with that endpoint. If the endpoint is a controller in a critical application, limited access is also critical. By doing this, attackers who have managed to gather low access credentials cannot have access to critical resources. By limiting access to all

endpoints within an ICS, a stronger security posture may be achieved, and risk associated with attacks from within may also be mitigated [5, 9, 13].

Secure Communication Protocols

As mentioned previously, data integrity and authentication are crucial parts of securing clients and data on the ICS network. In addition to these two cornerstones of secure communications, data confidentiality is the final cornerstone. All three of these cornerstones can be achieved by utilizing the many different secure communication protocols used in the field of cybersecurity today.

A key aspect of secure communications is choosing the right secure communication protocols for given applications. Some forms of communication are duplex, some simplex, and others using some form of broadcasting to multiple clients. Securing communications involves many different components that aid in authentication, encryption, and data integrity. To be able to authenticate and encrypt data, some form of cryptographic key is needed, whether symmetric, asymmetric, public, or private. In addition to normal communications, there might be the need to create and distribute public-private key pairs in a secure way. All these capabilities may be achieved by utilizing one or many different secure communication protocols. As with most features of cybersecurity, there is a performance tradeoff due to the computational overhead of cryptographic calculations. This can be a challenge in ICS environments where data transfer is time sensitive, and a balance is usually found somewhere in the middle.

Computer network communication is a layered approach, consisting of seven different layers referred to as the Open Systems Interconnection (OSI) Model. The seven layers of this model are network traffic abstractions starting from physical hardware and

building up into applications and their interface to a network. The seven layers are the physical, data link, network, transport, session, presentation, and application layers. The physical layer is the hardware layer where information is passed via some physical media. The data link layer is where physical addressing, such as MAC addressing, occurs. The network layer is where routing between computers on a LAN occurs via IP addresses. The transport layer is where communication between LAN networks occur forming wide area networks (WAN) via transport protocols such as TCP/IP. The session layer is where computer hosts establish communication sessions with endpoints on a network. The presentation layer is data encoding/decoding, encryption/decryption, and compression/decompression occur. Lastly, the application layer is where applications interface with the network via the other six layers. This section will be covering communication protocols associated with different layers of the OSI model.

When considering the layered model of networking, it is important to note that higher abstracted layers cannot control what takes place in lower layers. For example, the transport layer cannot affect or secure communications at the data link layer. Due to this, transport layer security mechanisms would have no effect on an attack that took place at the data link layer. For this reason, secure communication protocols have been introduced at nearly every layer of the OSI model to employ a layered security approach in tandem with a layered networking model.

IEEE 802.1X

One of the most common forms of network security across all networks is data link layer security. This type of security occurs between hosts on a local area network. Since data link layer security is implemented on the networking hardware, it is the lowest

possible level of network security achievable. One implementation of this is IEEE 802.1X, which is a data link layer access control protocol. This protocol uses EAP to provide authentication mechanisms between physical hosts on a LAN. The protocol works for both LAN and wireless LAN (WLAN) connections on a network. The protocol offers optional data encryption. Data encryption does not often occur on the data link layer as it causes intense overhead on networking hardware, so encryption occurs on layers higher up on the OSI model.

IPSec

By itself, internet protocol (IP), which exists at the network layer, does not provide any security features, and was built for the raw functionality of transmitting information between networks. If a third party were to intercept the packet, they could retrieve critical information from the raw data, modify the data, or replay the data again later. To avoid this, Internet Protocol Security (IPSec) was developed by the Internet Engineering Task Force (IETF) in 1998 [4]. IPSec is a secure network layer security framework which encapsulates two protocols called Authentication Header (AH) and Encapsulated Security Payload (ESP). These two protocols perform authentication to verify authenticity and integrity of the data received and performs encryption to keep data confidential. IPSec is frequently used in many different network settings, whether network to network, host to host, or network to host, and is frequently used in establishing virtual private networks (VPNs) [4]. Additionally, IPSec has some features built into it that protect communications against replay attacks so that confidential data cannot be intercepted and sent multiple times to replay the same message.

TLS

When communicating simple data or commands between servers and clients, a widely used communication protocol used to achieve this is known as Transport Layer Security (TLS). Originally known as Secure Sockets Layer (SSL), TLS was derived to secure TCP/IP communications over transport layer network communications. As suggested by the name, TLS is an example of a transport layer security protocol. TLS seeks to achieve confidentiality and authenticity in communications to prevent eavesdropping or tampering [16]. This is done by the utilization of public asymmetric key exchange to establish a private key configuration used in encryption and authentication. TLS is highly configurable and can use many different public key exchange methods for the establishing of a session, and many different private key encryption methods for the secure communication across networks.

Figure 2.6 depicts the typical TLS connection handshaking sequence. The client requests a connection to a server in which cryptographic details about the client are exchanged. The server then sends its public certificate, and the cipher suite is utilized. The client then authenticates and verifies the server certificate. The client then sends its own public certificate along with its private key. Once both the client and server are finished verifying, a secure and authenticated communication channel is established in which all messages are encrypted and authenticated.

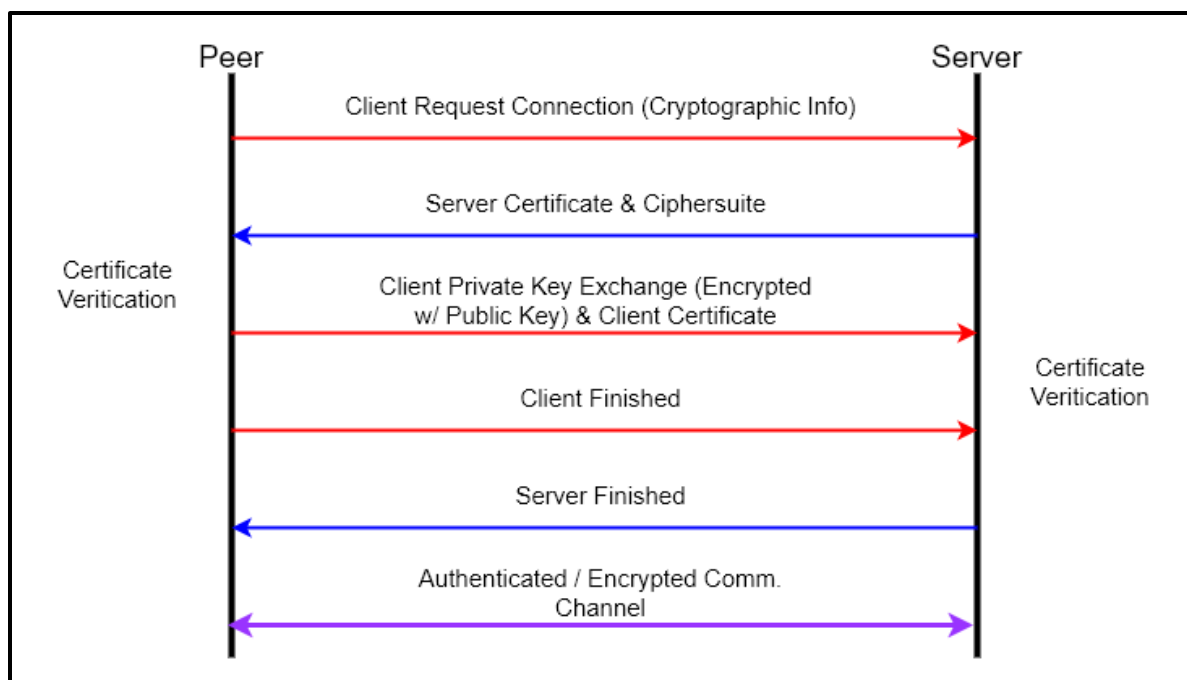


Figure 2.6 TLS Handshake

SSH

In an ICS, there may arise the need for embedded systems and computers to communicate with one another for the access of a terminal, file transfers, or application-level transmission. This brings up grave security concerns because if these connections were to be intercepted and compromised, the control of the entire system could be at risk. To avoid this, Secure Shell (SSH) was developed in 1995 to protect against these types of attacks. SSH is an application layer security protocol that is used for remote system logins, command executions, and file transfers. Instead of sending login credentials, commands, and files over the network in plaintext, all information sent and received is encrypted, decrypted, and authenticated. This is achieved over TCP/IP protocol which handles the initial public asymmetric key exchange and sets up the symmetric key encryption and authentication configurations. This process is repeated periodically to keep keys fresh and brute force resistant. Once this has been established, all further

information is encrypted, decrypted, and authenticated using the symmetric key to prevent man-in-the-middle and brute force attacks.

HTTPS

If there is ever a need to communicate with websites hosted on the internet, security is a huge concern because all communications are now leaving the ICS and entering the wild west of the internet. However, while this should be avoided in most cases, there are many different methods to securing internet communications. The most widely used secure internet communication protocol is known as Hypertext Transfer Protocol Secure (HTTPS). HTTPS is an example of an application layer security protocol with which internet applications communicate. This is desirable for an ICS that is controlled and governed by a central control facility that does not exist on the local ICS network. HTTPS is secured by TLS, meaning the establishing of secure communications, encryption, and authentication are handled by TLS in the transport layer, and then presented to the application layer by HTTPS at endpoints of networks.

Intrusion Detection Systems

While protecting the communications and access into ICS technologies is of extreme importance, it cannot be the only solution to protecting the system. Since cybersecurity is a constant game of cat vs. mouse, the capabilities of industries and attackers alike are becoming more sophisticated. To keep up with this game of cat vs. mouse, it is obviously important to keep ICS security protocols and policies up to date. It is also important to realize that ICS are never completely invulnerable to attacks and intrusions. Extreme efforts have been made toward identifying abnormalities in network communications and ICS behavior to detect intrusions when they are occurring, or if they

have already happened. These systems are typically known as Intrusion Detection Systems (IDSs). These systems scan network traffic and internal system functions to detect and prevent intrusions. Intrusion detection systems on the network side of an ICS are known as Network-based IDS (NIDS) [8, 12, 14, 18]. Intrusion detection systems that work on computers to scan system calls and resources are known as Host-based IDS (HIDS) [8, 12, 14].

IDS Overview

For intrusion detection systems, there are two dimensions in which they are classified. The first dimension is the medium in which the IDSs are implemented. The second dimension is via the strategic implementation of intrusion detection. Within the first dimension, IDSs are implemented on computer networks and computer hosts to form network-based intrusion detection systems (NIDS) and host-based intrusion detection systems (HIDS). The second dimension divides IDSs in terms of their implementation, in which there are signature-based IDSs and anomaly-based IDSs [8]. In signature based IDSs, sets of governing security policies are applied to all network traffic or system function calls to log all behavior of users inside of the governed systems. If these users breach the security policies based off known attack patterns, the activity is halted, and system administrators are notified of the activity [18]. In anomaly based IDSs, the system also scans and logs all network traffic and system function calls and usually, but not always, employs some form of artificial intelligence or machine learning to detect anomalies in the behaviors of certain network connections or users on the system [2]. If these behaviors stray too far from a deemed normal behavior, the activity is labeled as anomalous and activity is halted, and system administrators are notified. While there are

distinct differences in the different methods of intrusion detection, these systems can be integrated into a single IDS for more coverage [20].

While traditional firewalls solutions can prevent most unwanted network traffic into an ICS, it does not block all traffic as attackers have found many ways to get around IDSs. IDSs are a great compliment to the overall security of an ICS as they provide additional layers of protection and security policy governance. However, it is much more difficult to achieve such a task in an ICS environment [2, 12]. ICSs have more tailored needs due to the nature of the critical tasks being performed on site. Because ICS are controlling critical and sometimes destructive processes, it is vital that any IDSs employed by the ICS should be real-time and have the capability to detect intrusions as they occur to prevent any damage or loss of information from occurring. Additionally, since many of the devices running in an ICS are embedded devices, there are limited computational resources available to perform IDS measures, and therefore must be made lightweight on top of rigorous. Since ICS have a myriad of legacy hardware and software technologies, they are weaker to attacks, harder to update, can break easier, and are operated by insecure protocols. Due to these flaws of the legacy nature of ICSs, it is of utmost importance to have highly configurable or tailored IDSs that are capable of melding into current technologies and are capable of protecting older technologies. Lastly, since ICS IDSs exist at the lowest level of the control hierarchy, it is more difficult to employ a system with proper visibility to the physical control signals [2].

There are many different IDSs in existence today and choosing the right IDS is critical to securing relevant systems depending on the application. Some IDSs are very

focused on one type of detection, while others attempt to fill the role of “jack of all trades”.

Zeek

One of the more widely used IDSs in existence today is known as Zeek, which has recently changed its name from Bro. Zeek is an open-source IDS that runs on Linux OS. It specializes in sniffing and analyzing network traffic, classifying it as a NIDS. However, it does not focus on just one type of intrusion detection scheme, as it is able to employ both signature and anomaly-based detection techniques. Zeek, once deployed onto a computer system or network server, can perform both offline and online analysis of network traffic and is capable of being scaled out to very large or simple at home networks, making it a perfect candidate for industrial applications. Zeek employs logging of all activity of network traffic for analysis or network administrator forensics and supports analysis on many different application layer protocols including File Transfer Protocol (FTP), Domain Name System (DNS), HTTPS, TLS, and SSH. Above and beyond scanning the protocols, Zeek has the capability of analyzing all file contents introduced into the system via cryptographic hash computation for fingerprinting. In addition to all these features, it has a lot of documentation and support for integrating into whichever application you are seeking to protect.

Figure 2.7 depicts a NIDS and where it fits into a layout of a secured network. Normally, the NIDS sits on the front-facing side of a network and scans all incoming and outgoing traffic for signature and anomaly-based intrusions. When an intrusion sets off any configured rules, an alert is generated for security administrators to review, and if necessary, sever any connections with a potentially malicious source.

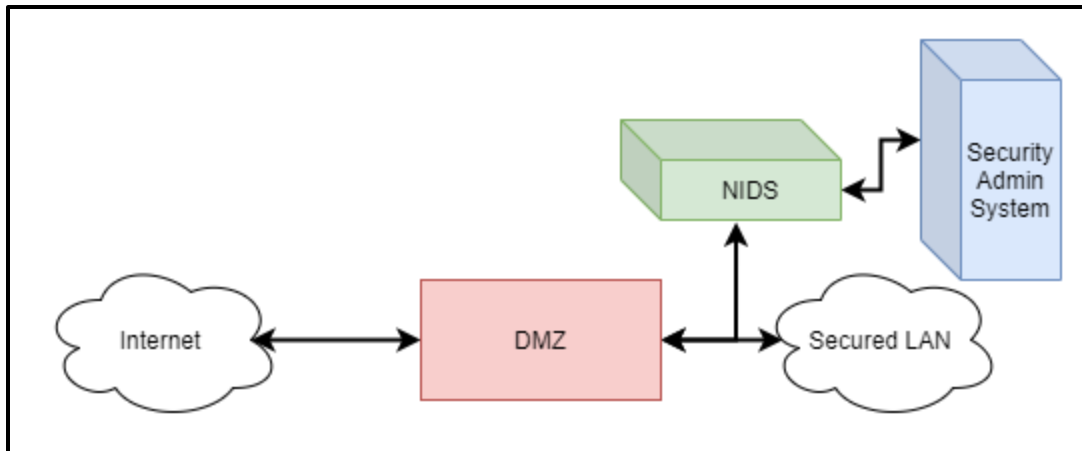


Figure 2.7 Network-Based IDS

Snort

Many open-source intrusion detection systems are combinations of smaller open-source engines tailored to accomplish specific tasks. Snort is one of these smaller IDS engines used for packet sniffing. Snort is a NIDS that excels at sniffing network traffic and performing analysis and packet logging. Snort can be configured in three different modes, Sniffer mode, Packet Logger mode, and NIDS mode. Sniffer mode reads network packets and displays them to the user, while Packet Logger mode logs the packets to disk. In NIDS mode, Snort will monitor and analyze network traffic based on a rule set defined by the network administrator. Based on the rules, different actions are taken depending on what threat has been identified. In NIDS mode, signature-based, protocol-based, and anomaly-based analysis is performed on traffic coming into or leaving the network. Custom rules can be configured by administrators to aid in traffic analysis and search for specific patterns their adversary may use. This is particularly useful for ICS security because very specific information should be moving between endpoints on a network, and Snort allows for alerts or actions to be taken when one of these facility custom rules has been breached.

OSSEC

In addition to integrating NIDS to protect networks, it is also vital to protect host systems and applications with some form of HIDS. OSSEC is a widely used HIDS that is open source and is compatible with many platforms. OSSEC is a HIDS comprised of configurable components. OSSEC is managed from a central server inside the host machine that can monitor agents which could be small programs or collections of multiple programs running on the host. OSSEC monitors many different aspects of the agents, such as their function calls to system functions, resource utilization, or interactions with other processes. OSSEC also intercepts network traffic moving to and from the host. System calls, resource utilization, network traffic, and inter-process communications between agents on the host are logged into a central database to be analyzed.

In addition to the monitoring of agents on the system, OSSEC can be configured to periodically inspect files that agents are interacting with via a data integrity checking process known as Syscheck. OSSEC also has the capability of scanning the host system for any presence of a rootkit using a process called Rootcheck. A rootkit is a collection of computer software designed to run silently on a computer and slowly cause damage and/or leak information from a system. Using all the collected information, OSSEC generates alerts via administrator-tailored rules and policies and helps to efficiently police a system to detect and stop unwanted process behavior.

Figure 2.8 depicts a HIDS and where it fits into the scope of a computer host system. The HIDS, like the NIDS, scans all communications going into and out of the host. Additionally, the HIDS scans processes running on the host and ensures that inter-

process communications are not performing system calls they should not be or partaking in any anomalous behavior. If so, the HIDS reports the behavior to a system administrator who can in turn tell the host to shut down the potentially malicious processes or external connections to the host.

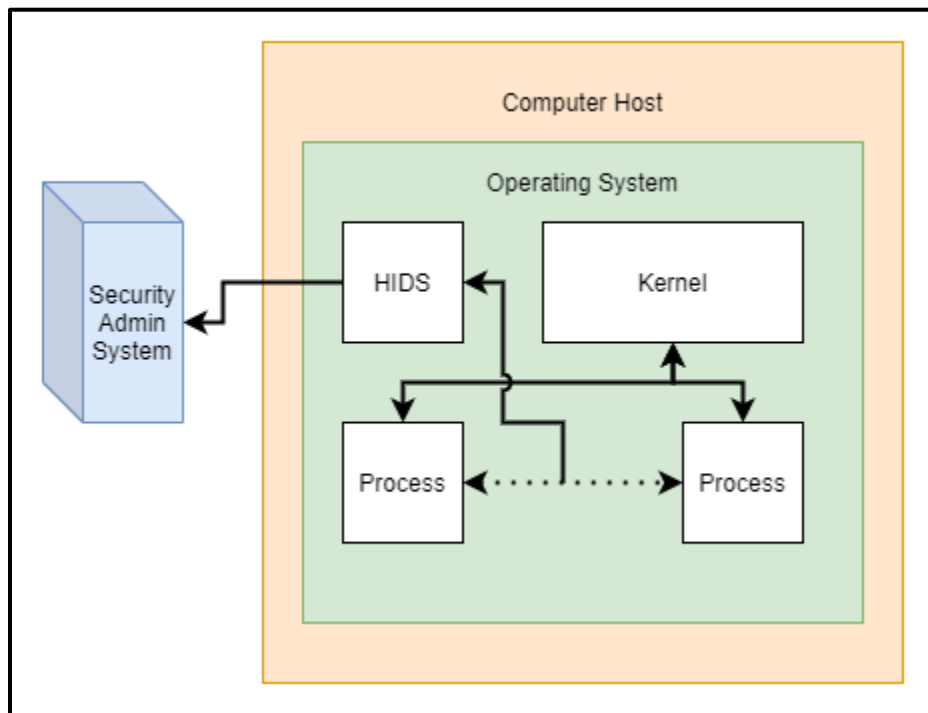


Figure 2.8 Host-Based IDS

Security Onion

IDSs come in many flavors and configurations. It can be difficult for security teams to set up and integrate the features of many different IDSs. In this circumstance, it may be desirable for security teams to adopt an IDS security suite which houses and integrates many different IDSs. One example of an IDS security suite used for protecting an entire network is known as Security Onion, a Linux-based OS security platform. Instead of installing and configuring each security tool one by one, Security Onion offers different IDS security components that can be enabled, configured, and integrated

together seamlessly. Security Onion's tools have the capability to analyze all the network traffic moving on the network it is attached to. Security Onions can analyze the traffic for intrusions, generate security alerts, perform corrective actions in response to alerts, and log all actions into a database to profile each network endpoint. Security Onion offers a couple NIDS to sniff all traffic moving onto the network to which it is attached, such as Zeek and Snort. Additionally, Security Onion offers analyst tools that can be used by network administrators to view the profiles of connections, network packets, transmitted files, etc. Security Onion also can serve as a central point of communication for HIDS running on each host on the network. Using HIDS such as OSSEC, alerts and logs can be stored and sent to the central Security Onion server to alert system administrators of suspicious behavior or intrusions occurring on hosts connected to the network.

Network Enclave

As seen in the section titled Typical ICS Facility Structure, each of the different regions of the industrial control facility have isolated enterprise, supervisory, and field networks. These network subregions are known as network enclaves. Network enclaves are meant to restrict access to certain portions of the network to increase security and reduce traffic on the same network channels. Each of these network enclaves have different functional purposes and connectivity requirements. Network enclaves allow for the separation of network resources and tailoring of what connectivity they have to the rest of the facility [9, 13].

Network enclaves also allow for network resources to be distributed without any interaction to unrelated enclaves. This improves network performance as network traffic is distributed along branches of the network as opposed to all traffic moving through a

main branch. As described in the section titled Removal of Unnecessary Resources, the removal of unnecessary resources is critical to reducing the attack surface of the system being protected. By removing unnecessary connectivity via network enclaves, the attack surface is reduced because attackers do not have as much facility to move laterally within a large spanning network [9, 13]. Since the network traffic is distributed, attackers also do not have the capability to listen to all traffic moving on the network, and only within the enclave to which they have access.

While the large network enclaves have DMZs between them, such as the enterprise layer and the supervisory layer, the network can be further divided into smaller network enclave subregions within each of these larger layers. For example, the management layer may have an accounting division, a resource planning division, a marketing division, etc. Figure 2.9 depicts an example of a network enclave structure in an ICS environment.

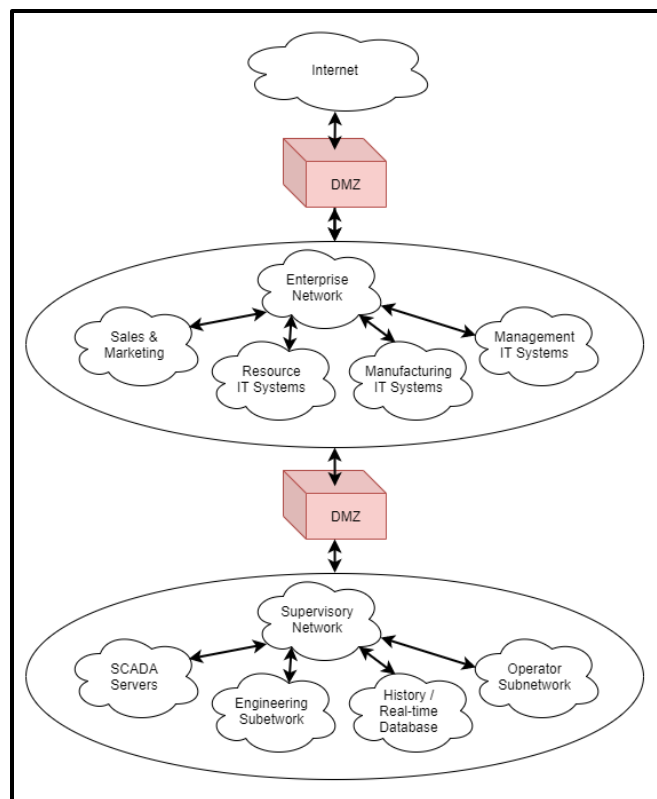


Figure 2.9 ICS Network Enclave

Threat Modeling & Intelligence

Due to the nearly infinite possible attack vectors known and unknown, the real challenge to ICS security is knowing where to apply these security protections in a way that makes a difference. Without any knowledge of what the attack surface looks like, implementing secure communication protocols or integrating an IDS into a network may result in nothing at all. Only by understanding the attack surface and threats to an ICS, can protections be applied and make a difference. By bringing all the security elements together, ICS network administrators are left with an abundance of tools and information, but perhaps lack the knowledge and/or perspective of how to assemble, configure, and maintain the security system. Security teams may be left without somewhere to build an understanding of their attack surface, and subsequently the forming of a fortified security posture.

However, there are many tools that have been developed for enterprise, corporate, and industrial environments to combat against small, or potentially very large threats such as APTs. Some of these tools might seem very intimidating to a smaller ICS facility comprised of a few individuals, without a network team, who are responsible for securing the facility to protect investments and intellectual property. An organization that does a phenomenal job at distilling this information is known as the MITRE Corporation. The MITRE ATT&CK Framework is an organized database that provides threat modeling, detection, and mitigation capabilities for many different industries. The framework details common APTs, their tactics, techniques, and procedures (TTP). The framework also details methods of detecting and mitigating the impact these TTPs have in the event of an attack. APTs and their TTPs have been recorded and publicly posted so that the world can learn from their actions, and how to protect against them in the future. With an abundance of threats, attacks, and databases of logged attacker behavior, the MITRE ATT&CK Framework presents all the information necessary for the protection of ICSs with smaller network teams to large ICS facilities with large and sophisticated network teams dedicated to attack, analysis, and defense [15].

MITRE ATT&CK Framework

The MITRE ATT&CK Framework was created in 2013 as an open-source tool to document attacker tactics and methodologies, whether small or large. The Framework offers a large database of 12 different tactics that APTs perform to infiltrate networks and perform actions once on the network [15]. The framework categorizes the tactics into 12 categories:

1. Initial Access – getting into a network
2. Execution – running malicious code
3. Persistence – maintaining a foothold
4. Privilege Escalation – attaining higher level permissions
5. Defense Evasion – avoiding detection
6. Credential Access – stealing account names and passwords
7. Discovery – understanding the environment
8. Lateral Movement – moving within the environment
9. Collection – gathering data of interest to meet their goal
10. Command and Control – communicating with/controlling compromised systems
11. Exfiltration – stealing critical data
12. Impact – manipulating/interrupting/destroying systems and data

Within the MITRE ATT&CK Database, there are a plethora of attack methods within each category. Each type of industry, whether medical, industrial, or corporate, has APTs that persistently attack them. Within each industry, the 12 categories contain the TTPs used by ATPs and suggestions for mitigation techniques are made. With this information, building an attack surface for the desired application becomes streamlined. With tools such as their ATT&CK Navigator, a categorical matrix representation of an attack surface can be visualized [15]. The levels of risk and security capability associated with each threat can be used to generate heat maps of the attack surface to depict where weaknesses lie. Figure 2.10 [28] depicts an example of a generated heat map.

There are four key uses cases that the framework offers in building a holistic defense system [15]. The four main use cases of the Framework are:

1. Threat Intelligence – what threats and methodologies are present for the application
2. Detection and Analysis – how to build detections and analysis tools for known threats
3. Emulation – how to mimic known threats to increase extensiveness of detection
4. Assessments – how to measure defenses and enable improvements in the future

Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Collection	Command And Control	Exfiltration	Impact
11 items	27 items	42 items	21 items	57 items	16 items	22 items	15 items	13 items	21 items	9 items	14 items
Spearphishing Attachment	PowerShell	Registry Run Keys / Startup Folder	Valid Accounts	Routing	Chromium Dumping	Process Discovery	Remote File Copy	Data from Information Repositories	Remote File Copy	Data Compressed	Disk Structure Wipe
Valid Accounts	User Execution	Scheduled Task	Valid Accounts	Obfuscated Files or Information	Input Capture	File and Directory Discovery	Remote Desktop Protocol	T1105	Layer	Data Encrypted	Data Destruction
Spearphishing Link	Command-Line Interface	Scheduled Task	Process Injection	Valid Accounts	File Deletion	System Information Discovery	Windows Admin Shares	-Groups: Lazarus Group, APT28, APT29, Bansec, MuddyWater, Gorgon, Group, Eldenwood, APT3, Magis House, APT33, FIN10, APT37, PLATINUM, Threat Group-3390, Lapsus\$, FIN6, APT32, Dragonfly 2.0, Turk, Gamaredon Group, BRONZE BUTLER, Pathwork, FIN7, Cobalt Group, APT14, MenuPass, Ollig	Used	Exfiltration Over Command and Control Channel	Stored Data Manipulation
Drive-by Compromise	Scheduled Task	External Remote Services	Exploitation for Privilege Escalation	File Deletion	Account Manipulation	System Network Configuration Discovery	Windows Admin Shares	Remote Services	Used	Exfiltration Over Command and Control Channel	Data Encrypted for Impact
External Remote Services	Exploitation for Client Execution	New Service	Escalation	Doobfuscate/Decode Files or Information	Credentials in Files	System Owner/User Discovery	Remote Services	Pass the Hash	Used	Exfiltration Over Alternative Protocol	Disk Content Wipe
Exploit Public-Facing Application	Shortcut Modification	New Service	Process Injection	Network Sniffing	Forced Authentication	Remote System Discovery	Pass the Ticket	Logon Scripts	Used	Automated Exfiltration	Resource Hijacking
Replication Through Removable Media	Dynamic Data Exchange	Redundant Access	Web Shell	Code Signing	Hooking	Account Discovery	Logon Scripts	Shared Drive	Connection Proxy	Automated Exfiltration	Runtime Data Manipulation
Spearphishing via Service	Rundll32	Hidden Files and Directories	Accessibility Features	Code Signing	Input Prompt	System Network Connections Discovery	Replication Through Removable Media	Data from Information Repositories	Custom Command and Control Protocol	Data Transfer Size Limits	Service Stop
Trusted Relationship	Regsvr32	Account Manipulation	Accessibility Features	Modify Registry	Credentials in Registry	Network Service Scanning	Application Deployment Software	Data from Removable Media	Remote Access Tools	Exfiltration Over Other Network Medium	Defacement
Supply Chain Compromise	Service Execution	Account Manipulation	Access Token Manipulation	Bypass User Account Control	Exploitation for Credential Access	Query Registry	Distributed Component Object Model	Clipboard Data	Standard Non-Application Layer Protocol	Exfiltration Over Physical Medium	Endpoint Denial of Service
Hardware Additions	Compiled HTML File	Create Account	DLL Search Order Hijacking	Disabling Security Tools	Indicator Removal on Host	Security Software Discovery	Audio Capture	Video Capture	Man in the Browser	Scheduled Transfer	Firmware Corruption
Execution through API	Mhta	Modify Existing Service	AppCert DLLs	Redundant Access	LLMNR/NBT-NS Poisoning and Relay	System Service Discovery	Permission Groups	Man in the Browser	Standard Non-Application Layer Protocol	Over Physical Medium	Endpoint Denial of Service
Signed Binary Proxy Execution	CMSTP	Application Shimming	Application Shimming	Redundant Access	LLMNR/NBT-NS Poisoning and Relay	System Service Discovery	Permission Groups	Man in the Browser	Standard Non-Application Layer Protocol	Scheduled Transfer	Firmware Corruption
Graphical User Interface	Execution through API	BITS Jobs	Hooking	DLL Side-Loading	Password Filter DLL	Network Share Discovery	Component Object Model	Man in the Browser	Standard Non-Application Layer Protocol	Scheduled Transfer	Firmware Corruption
Signed Script Proxy Execution	Office Application Startup	DLL Search Order Hijacking	Image File Execution Options Injection	Indicator Removal from Tools	Private Keys	Network Sniffing	Application Window Management	Man in the Browser	Standard Non-Application Layer Protocol	Scheduled Transfer	Firmware Corruption
Third-party Software	Logon Scripts	Port Monitors	Regsvr32	Binary Padding	Two-Factor Authentication Interception	Peripheral Device Discovery	System Time Discovery	Man in the Browser	Standard Non-Application Layer Protocol	Scheduled Transfer	Firmware Corruption
Windows Remote Management	Office Application Startup	AppCert DLLs	Hidden Files and Directories	Regsvr32	Hidden Files and Directories	System Time Discovery	Application Window Management	Man in the Browser	Standard Non-Application Layer Protocol	Scheduled Transfer	Firmware Corruption
XSL Script Processing	Windows Management Instrumentation	Extra Window Memory Injection	Template Injection	Timestamp	Compiled HTML File	Virtualization/Sandbox Evasion	Password Policy Discovery	Man in the Browser	Standard Non-Application Layer Protocol	Scheduled Transfer	Firmware Corruption
Control Panel Items	Winlogon Helper DLL	AppCert DLLs	Path Interception	Process Hollowing	Access Token Manipulation	Browser Bookmark Discovery	Domain Trust Discovery	Man in the Browser	Standard Non-Application Layer Protocol	Scheduled Transfer	Firmware Corruption
Execution through Module Load	Browser Extensions	Weakness	MShta	Process Hollowing	Access Token Manipulation	Browser Bookmark Discovery	Domain Trust Discovery	Man in the Browser	Standard Non-Application Layer Protocol	Scheduled Transfer	Firmware Corruption
InstallUtil	LSASS Driver	Component Firmware	SID-History Injection	DLL Search Order Hijacking	Execution Guardrails	Rootkit	Signed Binary Proxy Execution	Man in the Browser	Standard Non-Application Layer Protocol	Scheduled Transfer	Firmware Corruption
LSASS Driver	Regsvcs/Regasm	Component Firmware	SID-History Injection	DLL Search Order Hijacking	Execution Guardrails	Rootkit	Signed Binary Proxy Execution	Man in the Browser	Standard Non-Application Layer Protocol	Scheduled Transfer	Firmware Corruption
Trusted Developer Utilities	Hooking	Image File Execution Options Injection	Port Monitors	AppCert DLLs	Component Object Model Hijacking	AppCert DLLs	Component Object Model Hijacking	Man in the Browser	Standard Non-Application Layer Protocol	Scheduled Transfer	Firmware Corruption

Figure 2.10 ATT&CK Navigator Heat Map

Threat Intelligence

Applying defenses for every type of threat or attack can be time consuming for an organization, especially those with large amounts of resources such as an ICS facility. In addition to the time required to address every type of threat, designing IDSs with rules and policies for every single existing threat will result in a slow security system. If an IDS applies each rule/policy for all network traffic, the system will be slow, inefficient, and incapable of monitoring all traffic. For this reason, the ATT&CK Framework is well suited for any organization who wants to employ a threat-informed defense where they approach their defensive posture from a pragmatic and well-informed point of view. The ATT&CK Framework enables organizations of any size to build their security tools around threat intelligence on relevant threats and their TTPs [15].

Within the ATT&CK database, queries can be made into specific industries and their systems to build an attack surface based on the recorded TTPs of APTs in that area. In the case of ICSs, MITRE has an entire website dedicated to threat intelligence for ICSs. From their website you can view the relevant APTs and the TTPs they have employed against ICS facilities worldwide.

Detection & Analysis

After performing threat intelligence, the next step is to choose and configure the IDSs in a facility network. Each system has its own configuration techniques and understanding the syntax and approaches of TTPs is necessary in putting up the detection mechanisms. There are many different open-source databases with the internet registries of these attacks to be studied and understood. MITRE themselves has their own ATT&CK Framework database for all submitted attacks under their recognized APTs.

Using this information, custom rules and policies can be applied to the IDSs within an organization [15].

In addition to configuring IDSs, MITRE offers suggestions on other types of detection and mitigation techniques for each TTP. For example, in the category of Impact, there exists the known technique of Firmware Corruption. The framework offers resources on how this has been performed and offers the detection mechanism of logging and monitoring all attempts to read/write to the BIOS. Within this TTP, there are three different mitigation techniques suggested: Boot Integrity, Privileged Account Management, and Software Updates.

For each relevant category, the framework can be utilized to gather detection and mitigation techniques to build up the security mechanisms of an organization. With this information, organizations may begin to configure rules that define the behavior of their IDSs and form mitigation techniques to protect against the ATPs that exist in their industry. Once protections have been put in place, the IDSs and mitigation techniques can be developed further by red teams and blue teams within the organization [15]. Over time, a strong and holistic security posture can be formed.

Emulation

While ATPs behavior is documented and strong detection and mitigation techniques can be formed, ATPs are sophisticated and adaptive. ATPs often form new tactics to circumnavigate protections put in place to ward against their previous efforts. For this reason, it is important to get in the mind of an adversary and attempt to emulate their types of attacks. Red teams and blue teams can attempt to breach and defend their organizations systems to test the comprehensiveness of defenses and expose any

weaknesses. Without emulation, it is difficult to determine whether the detection and mitigation techniques are effective.

While it is much more difficult to cover a lot of different emulations in a single organization, there exist tools to help the automation process of adversary emulation. An example of emulation tools that map well into the ATT&CK Framework is known as Atomic Red Team. It offers different tests which can execute automated attacks against an organization's network to test the effectiveness of detection and mitigation techniques put in place. While they only focus on a singular attack technique, hence the name atomic, these tests are set up in such a way that they can be tested in sequence cover a wider area. Over time, network security teams can expand their covered attack surface greatly by taking a targeted approach towards emulating different attack techniques made by the APTs that are the biggest threats to their organization [15].

Assessments

The final major use case for the MITRE ATT&CK Framework is utilizing the framework to assess on an organization's security posture. By doing an assessment on defenses, one may find areas that are lacking in security and make improvements. Assessments, like any of the other use cases for the Framework, can be utilized by any size of organization seeking to bolster security defenses. MITRE suggests assessments be taken in 3 incremental steps:

1. Assess the coverage of defenses using the ATT&CK Navigator
2. Identify the most vital gaps that exist in defense coverage
3. Modify defenses to fill those security gaps

This process of repeated threat intelligence gathering and assessment should be periodically repeated because the capabilities of attackers are always evolving and growing stronger, and so should an organization's defenses. By doing this, a heat map can be updated frequently to show improvements, and where general weaknesses lie in an organization's security coverage. Once coverage has been assessed, organizations can begin creating priorities for implementing change [15]. Now that the organization knows where change should be implemented, detection and mitigation techniques can be created, and the cycle continues.

CHAPTER 3: RESILIENT ICS DATA

To understand how this thesis fits in relation to ICS security, a larger perspective is required. Several concepts form the foundation of this thesis: system functions and interactions, adversaries and threats to industrial control, the thought process of adversaries, and modern ICS security practices. Each of these elements come together to form a more complete view of ICS system security, cyber resiliency, system/data-level resiliency, and how each can be addressed. This chapter proposes a data acquisition system with the main purpose of increasing resiliency, including cyber, system, and data.

Trust and vulnerability are common struggles for modern ICS security engineers. Vetting data for trustworthiness is critical, as illegitimate data may be presented to IDS/IPSs by adversaries or because of degrading devices. Perhaps an ATP has compromised equipment in the field that is responsible for reporting information to an IDS. This device could be a PLC with corrupted firmware, or a compromised or a physically degraded or broken sensor or actuator. Anything that can report false information may cause cascading side effects for functionality and system security. This thesis seeks to provide a relevant solution to the pervasive issue of false information within the control hierarchy.

There are other avenues of research that align with what this thesis proposes [24]. One such avenue is the application of additive layer manufacturing (ALM), where circuit features and functionality are applied on top of one another, rather than the typical subtractive silicon approaches. However, circuit modifications via 3D printing

technologies run the risk of adversarial interference. In the case where an adversary has compromised an ALM control system, the adversary may have the ability to change circuit topology and behavior. This is accomplished by interfering with the control process, leading to cascading security concerns post-manufacturing. Dawson pursues the formation of an IDS via an out-of-band data acquisition unit in a test environment [24]. This thesis expands on the research proposed by Dawson et al. [24] by designing and demonstrating a data acquisition unit compatible with general ICS applications.

The Proposed System

This thesis provides the missing link in modern ICS security systems and attempts to implement a device that provides increased visibility at the lowest level of the control hierarchy. This device is a new type of data acquisition unit that may be deployed within control processes for an out-of-band data channel control process I/O. The proposed system monitors, records, and verifies control/sensor voltages/currents to verify the legitimacy of data being reported back from the control system. If any of the control process components are damaged or compromised, the data may not reflect reality, and the proposed system addresses these concerns. For example, a damaged sensor may report inaccurate data, or a compromised PLC may purposefully report false or misleading data.

To address these issues the proposed data acquisition unit converts the physical electrical signals of the control process out-of-band from the PLC. It then reports the information to a network endpoint that contains an IDS/IPS as a form of verification of in-band control process I/O values. The out-of-band and in-band data can be synthesized and analyzed by the IDS/IPS to build trust in ICS data sources. If there are any

discrepancies in the data being collected, then the sources may be degrading, damaged, or compromised. The IDS may alert security and system engineers to shut down systems and address the discrepancy. Figure 3.1 depicts the proposed data acquisition system and how it fits into the ICS model.

One application of this data acquisition system may have been of benefit during the devastating ICS data subversion attack, Stuxnet. Compromised firmware within the PLCs were forcing Iranian centrifuges to accelerate but were reporting false RPM back to servers. If this data acquisition unit were installed at the electrical contacts of a centrifuge control system, the visibility on the physical state of the system may have enabled an IDS to detect the attack. By noticing a discrepancy in the data being reported to the IDS, an alert could have been generated and security engineers could have halted systems before the severe damage occurred.

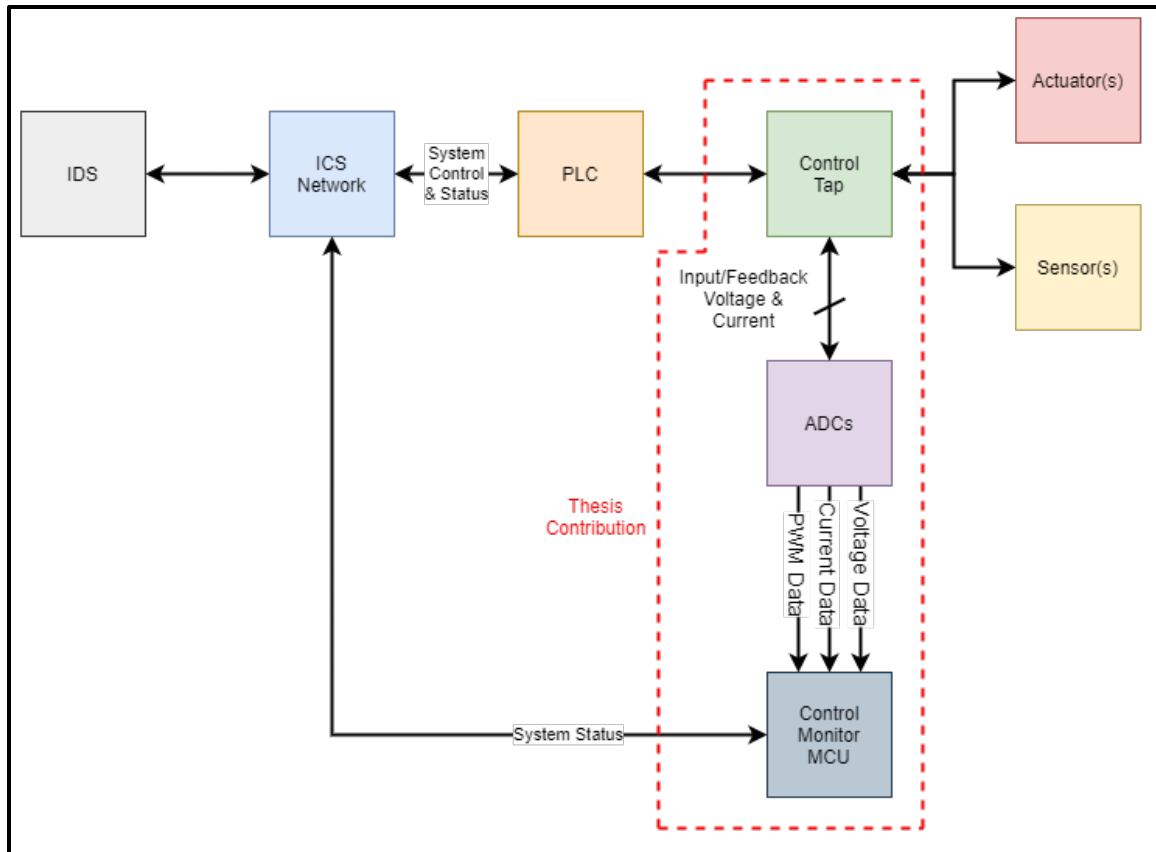


Figure 3.1 Proposed Thesis Contribution

An additional application of this data acquisition system is for verification of control process data used as training data for machine learning security algorithms. Some machine learning algorithms are trained to understand normal control process behavior to detect anomalies. If the systems used for training data are already compromised, the algorithm will be taught the wrong lessons about reported typical behavior. Incorrect training would render the AI system incapable of detecting anomalous behavior, causing security problems in the future. Avoidance of compromised training data can be achieved by verifying the legitimacy of the data through the cross reference of data acquired by the proposed system.

There may be other data science applications for this data acquisition system. However, the main purpose of this thesis is to propose a new tactic, technique, and procedure for detecting anomalous or illegitimate data due to system degradation and infiltration. In chapters 6 and 7, the electrical hardware and software design will be discussed further to demonstrate how such a system is created, tested, and integrated into modern ICS security systems.

CHAPTER 4: CYBER RESILIENCY IN SYSTEMS & DATA

Through observation of current security practices implemented in ICS facilities, it is clear there is an extensive variety of cybersecurity technologies to thwart the efforts of adversaries. All the protections are put in place to secure something within the ICS facility whether it be credentials, intellectual property, or physical systems. In any case, security engineers must fully understand the systems they are attempting to fortify as well as the attack surfaces that exist. Much study, effort, and priority has been put into securing and fortifying existing IT systems in an effort to prevent attackers from getting into ICS networks initially. However, because of cases like Stuxnet where attackers have managed to infiltrate a network, detecting the adversary, and protecting internal systems becomes more difficult. This is an overarching relationship present between attackers and defenders of ICS networks; with each new successful attack, stronger and more resilient defenses must be devised.

For this reason, there can never be a completely fortified ICS network, and it must be assumed that IT and OT systems are always at risk. The perceived attack surface might be covered, but the growing interconnectedness of IT/OT systems always provides opportunities for attackers to seep through the cracks. This chapter discusses cybersecurity elements at a high level to exfoliate issues found within modern ICS security systems. To begin, system thinking is used in threat intelligence considerations and will be expounded upon, revealing its role in proper modern cybersecurity system implementations [11]. The second portion of this chapter covers concepts of cyber

resiliency. Next, a discussion on system resiliency reveals broader implications related to cyber resiliency and modern IT/OT security practices. Finally, this chapter discusses a lacking element of cyber/system resiliency considerations in ICS security, data resiliency.

System Thinking

In the face of APTs and their relentless crusade against many industries, cybersecurity personnel must think in terms of systems to have a more complete understanding of the attack surface of IT/OT systems. One method of inspection which offers an advantageous perspective of attack surfaces within whole systems is known as “System Thinking”. System thinking requires a deconstruction of the system design into its primary components, behavior, and purpose. When a system is broken down into its fundamental elements, understanding their relation to the whole system becomes clearer. This process aids in creating cybersecurity solutions that fit well into the system and do not increase the attack surface elsewhere. This can be a time-consuming task as complex systems are generally interconnected, and a complex system can have feedback where the entire system interacts with itself [11]. Therefore, small changes to systems or security may cause cascading consequences throughout the rest of the system. For example, a new functionality may be introduced to an ICS that initially allows the system to perform better or provide more security. However, unknown consequences may arise due to the feature’s connections with other components of the system, possibly providing the attacker a new avenue of entry into the system.

Every technological system has hardware and software components, hardware-defined and software-defined behaviors, and an overall functionality goal. The components may consist of processors, transducers, routers, programs, operating systems,

etc. The behaviors originate from how the software and hardware components are connected, integrated, or programmed. Components and behaviors come together to complete the system and to achieve an ultimate outcome whether it be maintaining the pressure in a tank, calculating credit score, or sending messages across the world.

Every system has an overarching structure which is known as the system hierarchy; the system may have subsystems, and those systems may have subsystems of their own [11]. Viewing the system as hierarchical offers an advantageous perspective over the system when trying to identify and protect the attack surface of a system. This is especially true because this method of functional decomposition is exactly how ATPs find vulnerabilities within large and complex systems. Every layer of the hierarchy, whether it be the main system or the most simple and low-level subsystem, has an attack vector associated with it. By increasing the complexity, breadth, or depth of the system hierarchy, more attack vectors are exposed ultimately growing the attack surface.

The process of observing and addressing security at the hierarchical level is known as “security in depth”. Every piece of the system may offer an avenue for attackers to enter the system, so it is critical that every layer of the system be addressed for attack vectors and suitable corresponding security protections [9, 12, 13]. The security in depth approach addresses a wide range of security concerns, from malware to educating system operators and users. Chapter 2 describes different types of protections and security practices for IT/OT computer systems and networks in depth.

Unfortunately, modern ICS practices are often lacking in protection and verification of the physical systems processes that exist at the lowest level of the control system hierarchy. The concepts of cyber, system, and data resiliency can address these

concerns. There are many definitions of resiliency with respect to systems and their applications. Researchers have often used multiple attributes to encompass what “resilient” means with respect to the given application. Some sources claim that resiliency is the ability to rebound back from a failure, while others claim resiliency is the ability to withstand the forces that cause system failure [1, 5]. For the concept of resiliency, more definitions provide a broader context and encourage a well-planned cybersecurity approach.

Cyber Resiliency

A high-level concept of cyber resiliency provides a framework for breaking down resiliency in the cybersecurity environment. NIST defines cyber resiliency as “the ability to anticipate, withstand, recover, and adapt from adverse conditions, stress, attacks, or compromises on systems that use or are enabled by cyber resources” [22]. The goals of cyber resilient systems are the following [22]:

1. Anticipate: Maintain a state of informed preparedness for adversity.
2. Withstand: Continue essential mission or business functions despite adversity.
3. Recover: Restore mission or business functions during and after adversity.
4. Adapt: Modify mission or business functions and/or supporting capabilities to predicted changes in the technical, operational, or threat environments.

Within the context of these goals, here are eight objectives listed by NIST that aid in the forming of cyber resilient systems, and they are the following [22]:

1. Prevent: Preclude the successful execution of an attack or the realization of adverse conditions.
2. Prepare: Maintain a set of realistic courses of action that address predicted or anticipated adversity.
3. Continue: Maximize the duration and viability of essential mission or business functions during adversity.
4. Constrain: Limit damage from adversity.
5. Reconstitute: Restore as much mission or business functionality as possible after adversity.
6. Understand: Maintain useful representations of mission and business dependencies and the status of resources with respect to possible adversity.
7. Transform: Modify mission or business functions and supporting processes to handle adversity and address environmental changes more effectively.
8. Re-Architect: Modify architectures to handle adversity and address environmental changes more effectively.

Within these definitions are a few objectives that pertain to the proposed data acquisition system introduced in Chapter 3. First and foremost, the data acquisition system's main goal is to aid in the prevention (1) and constraining (4) of adversity via degradation and infiltration of ICS OT. Prevention and constraining of adversity are achieved through an understanding (6) of system level signals through increased visibility of the electrical behavior of ICS processes. This understanding, achieved through increased visibility, stems from the re-architecture (8) of the ICS environment. This re-

architecture is what this thesis proposes, the integration of an out-of-band data acquisition system at the control process level.

The specific implementation of cyber resiliency differs between applications, and along with the defined objectives NIST additionally lists techniques that can be interpreted and applied. The listed techniques in [22] are Adaptive Response, Analytic Monitoring, Contextual Awareness, Coordinated Protection, Deception, Diversity, Dynamic Positioning, Non-Persistence, Privilege Restriction, Realignment, Redundancy, Segmentation, Substantiated Integrity, and Unpredictability [22].

The cyber resiliency techniques that apply to the data acquisition system are the following [22]:

1. Analytic Monitoring: Monitor and analyze a wide range of properties and behaviors on an ongoing basis and in a coordinated way.
2. Contextual Awareness: Construct and maintain current representations of the posture of missions or business functions considering threat events and courses of action.
3. Substantiated Integrity: Ascertain whether critical system elements have been corrupted.

These three techniques apply directly to the proposed data acquisition system. The data acquisition system can be integrated into existing ICS analytic monitoring systems such as IDSs, widening the range of system signal inputs. In addition to widening the range of inputs, the data acquisition system provides out-of-band data channels which give both contextual awareness and increased visibility to the electrical behavior of control processes. By integrating the data acquisition system into existing analytic

monitoring systems, and gaining increased contextual awareness via increased visibility, a greater level of data integrity can be achieved. By comparing in-band and out-of-band channels of ICS data, discrepancies may point to degraded or corrupted system components.

System Resiliency

In the context of ICSs, a framework known as the R4 framework of resilience provides description of what resiliency is in the context of systems [3]. The four attributes listed are robustness, redundancy, resourcefulness, and rapidity.

1. **Robustness:** the ability of systems, system elements, and other units of analysis to withstand disaster forces without significant degradation or loss of performance
2. **Redundancy:** the extent to which subsystems, system elements, or other units are sustainable, that is, capable of satisfying functional requirements, if significant degradation or loss of functionality occurs
3. **Resourcefulness:** the ability to diagnose and prioritize problems and to initiate solutions by identifying and mobilizing material, monetary, informational, technological, and human resources
4. **Rapidity:** the capacity to restore functionality in a timely way, containing losses and avoiding disruptions

The R4 framework tends to be thoroughly implemented in IT environments.

Robustness is achieved via the implementation of systems which employ authentication and authorization to ensure system resources are given to entrusted users. Those systems also ensure a network enclave that separates and isolates network regions and provides

data integrity methods to ensure the transmission and storage of untampered data. Redundancy is achieved via backup servers and hosts which maintain service in the event of system crashes, and through storage of redundant data as backups. Resourcefulness is achieved via the implementation of intrusion detection systems which can analyze network traffic for bad actors and allow network administrators to shut down or deploy resources when problems arise. As a result of proper implementation of the first three attributes, rapidity can be achieved by network administrators, allowing timely isolation where problems arise and containing the problem locally to avoid cascading problems throughout the facility.

Data Resiliency

IDSs and mitigation systems are implemented through many different means to meet the diverse demands of infrastructure. Some examples previously mentioned include the monitoring of network and host behavior, reduction of access, cryptographic methods, etc. However, the same considerations cannot be directly applied to OT where physical processes take place. In typical ICS security applications, emphasis has been placed on the fortification of systems in IT and the availability of OT information [5]. As a result, there is less focus on OT security and OT system/data resiliency. However, if an attacker has already infiltrated both IT and OT systems in examples such as Stuxnet, it becomes difficult to detect an attack due to compounding data subversion and obfuscation techniques [7, 10]. These types of attacks successfully obfuscate the reality of what is happening in the physical domain of systems.

For modern ICS environments, the system and its operation are only as resilient as the data they use. From the IT perspective, the common cybersecurity practices do their

best to keep transmitted and stored data resilient. However, from the OT perspective, there is usually trust placed in the data returned from the PLCs. System engineers can frequently check the integrity of the programs and data within PLCs and may conclude that these devices are not compromised. Misplaced trust can cause substantial problems, as in the case of Stuxnet. The malware was stored in a rootkit, where program integrity checking does not work and therefore the data cannot be trusted. For this reason, it is important to apply the R4 framework to OT in the same capacity as IT [3].

Of the four attributes to the R4 framework, redundancy is scarce in modern OT security practices. Significant trust is placed in field devices, which may have already been compromised and may be returning illegitimate data. Unfortunately, PLCs may be infected, or sensors and actuators may be compromised unknowingly during manufacturing by a third-party. This also feeds directly into the issue of contextual awareness deficiencies described in [22]. Without context of the data being received, a complete picture of the behavior of ICS OT cannot be formed.

To regain trust lost by incomplete visibility in ICS field devices, a new method must be devised which can address the areas of deficiency. This thesis proposes a new type of out-of-band ICS data acquisition system that aids in the verification of in-band data being processed by control system and security technology. The data acquisition system is designed to offer increased control process visibility via an additional perspective on the physical processes being controlled. This additional data can be used to verify the legitimacy of ICS data and build trust with the data sources in the ICS environment. Without any visibility of ICS data available to check on the physical

environment, data subversion and data poisoning attacks will remain a large threat to the protection and operation of the systems which are the life and blood of ICS facilities.

CHAPTER 5: DATA ACQUISITION SYSTEM

To demonstrate the proposed data acquisition system, an example design was created, assembled, and integrated into an existing control system. The acquisition unit was tapped into the example design's electrical signals to gather, record, and monitor the behavior of the system. This chapter will cover the layout of the example design, some system failures due to degradation or infiltration, the layout of the data acquisition system, and the integration of the data acquisition system into the control system.

Example Design

For this thesis, a simple four-tank water flow control system was utilized to demonstrate the capabilities of a data acquisition system. For the purposes of demonstration, only one of the four tanks was used. Figure 5.1 is a diagram of the water tank control system. The system actuator is a water valve which controls the flow rate of water into a water tank. The system sensor is a pressure sensor that returns the water pressure inside the tank back to the PLC. The water valve is controlled by the PLC and opens or closes the water valve to control the flow rate of water being poured into the water tank. The bottom of the water tank has a fixed size drain where water empties into a reservoir that houses the water pumps. The flow rate out of the drain is proportional to the water pressure inside of the tank, which is proportional to the water level of the tank. The PLC firmware was designed to control the level inside of the water tank via the control feedback loop. The value of water pressure inside of the tank and the desired water level inside of the tank are used by the PLC to drive the water valve to a position.

While the system is simple, it is enough to demonstrate the different types of sensors, actuators, and control mechanisms that can be monitored for deterioration or compromised behavior. Figure 5.2 depicts the control loop for the water tank system.

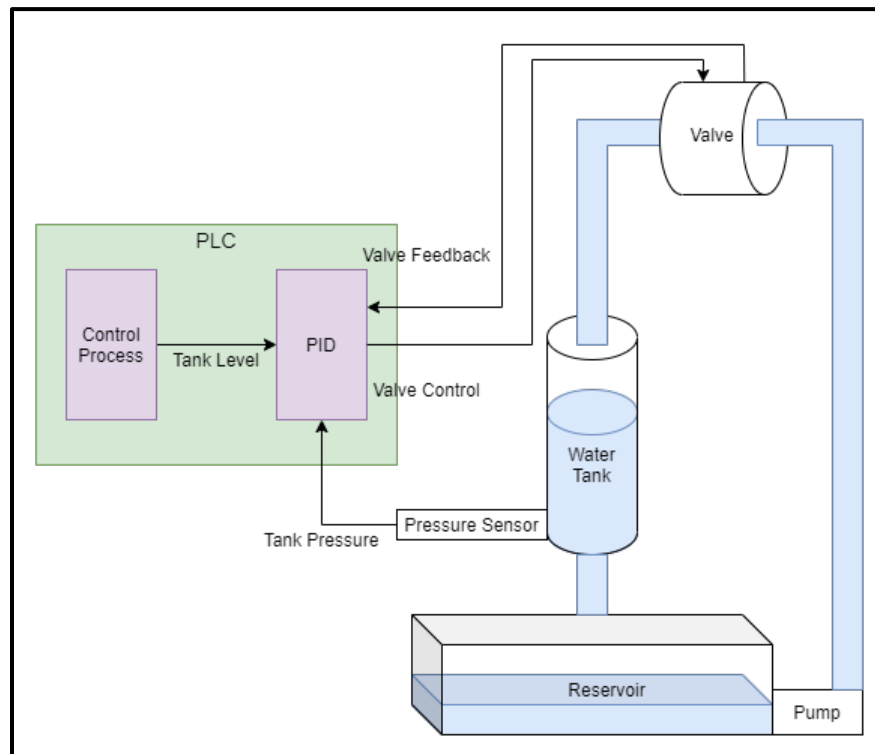


Figure 5.1 Water Tank Control System

The control mechanisms are implemented within the PLC via a proportional, integral, and derivative (PID) controller. The PID controller accepts the desired tank level and the current tank pressure as inputs and is responsible for driving the position of the water valve whether fully closed, fully open, or somewhere in between. By controlling the water valve position, the PLC also controls the flow rate of water into the tank. The entire control system is implemented digitally. This means the PID controller is based in software, and all analog inputs are digitized and stored in the internal PLC register space. Analog I/O conversion is performed by analog-to-digital converters (ADCs) and digital-to-analog converters (DACs) inside of the PLC.

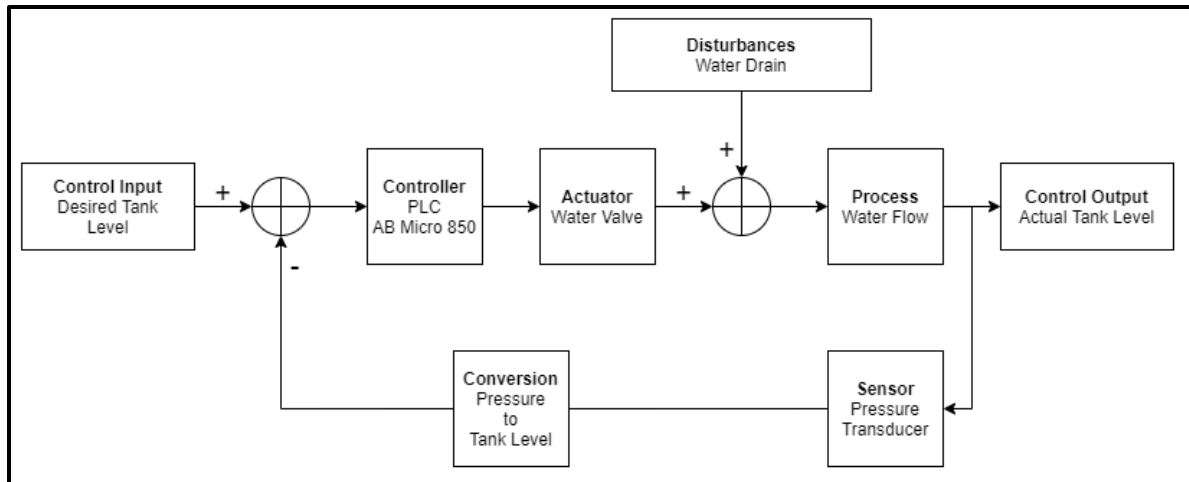


Figure 5.2 Water Tank Control Loop

The control system input, the water tank level, was designed to be an oscillating process in which the water tank level oscillates from 25% to 75% full over the period of an hour. To get the control system to track this water tank level with reasonable accuracy, the PID controller underwent a PID tuning process in which the different proportional, integral, and derivative gains are incrementally adjusted to meet the control system response requirements. By tuning the gains on the controller, a responsive water tank system was derived that is capable of closely tracking the desired water tank level. Figure 5.3 depicts the recorded sinusoidal control process over the course of approximately two hours. This data was recorded via the data acquisition system and depicts the behavior of the water tank control output responding to the sinusoidal input.

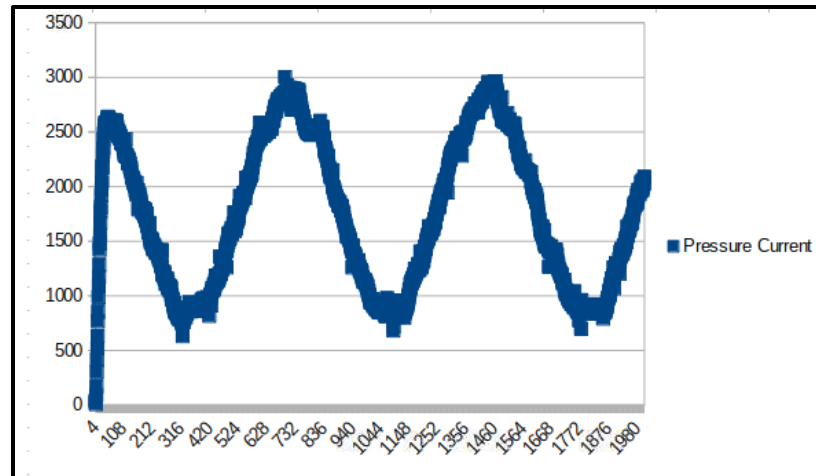


Figure 5.3 Recorded Control Process

Degradation & Infiltration

In the age of cybersecurity, it can be easy to attribute failures that arise in the control system environment to the infiltration and attacks of an adversary. While this is an important concern, there also exists the possibility that the control system is degrading and breaking down. This section will detail the challenges in identifying and diagnosing failures when and after they occur and will give perspective into how the proposed data acquisition system may aid in this effort. With IDSs employed today, degradation and infiltration have become incredibly difficult to detect once introduced to the system. This thesis proposes an addition to the lowest level of the control hierarchy within this data acquisition system. This is done with the goal of offering increased visibility to physical devices that may be degrading or compromised.

Degradation

Within any control system, there are physical parts that electrically operate and perform work. In these environments, physical components are prone to degradation. Degradation may lead to undefined behavior that is difficult to anticipate and identify. If

a degrading component is not repaired, the system may sustain damages or fail entirely. The reason identification of degradation is such a challenge for system engineers is related to how control status signals are reported in the control environment. A PLC reports digital status information to the control network. Actuators and sensors return analog status information to the PLC which digitizes it. Due to the transient nature of this data transfer, if degradation interferes with status reporting anywhere along the chain of information flow, it may be difficult for system engineers to identify if and where a problem exists.

Manufacturers tend to perform extensive degradation analysis on their products. Their findings are generally published in a reliability report or datasheet which details the common causes of failure and provides a predicted life expectancy of the device. Detailed in this report are many different statistical figures which combined form a reliability model. One of the key figures in this report is known as the Mean Time Between Failure (MTBF), which essentially gives the predicted duration of time between failures of a device. Another key figure is the Mean Time to Repair (MTTR). These figures come together to form the calculation for availability which is defined as:

$$A = MTBF / (MTBF + MTTR)$$

A: Availability; MTBF: Mean Time Between Failure; MTTR: Mean Time to Repair

Devices are designed so their availability is very close to one but are never perfect. For example, in Allen Bradley ControlLogix controllers, their anticipated MTBF is predicted to be at least 1 million hours and is often greater. With an advertised MTTR of 10 hours, that results in a high availability of 0.99999.

While devices are designed to have very high availability, these figures are calculated based on typical operating environments. If any of the devices are operating in anything outside of a typical environment, the predicted MTBF is going to be smaller, and failures should be prepared for and anticipated.

Infiltration

If an adversary has compromised any of the devices in the control hierarchy, data being reported in the chain of information flow may be illegitimate and hiding failures as an attack is in progress. While the example design is simple and houses a small number of components, there are still a few attack vectors present.

The first method of infiltration compromises either the firmware or the hardware of the PLC. The firmware can be altered via a network intrusion into the ICS facility in which malicious code is introduced to the PLC. Introduced malicious code may give the adversary control over the device and data reporting, visibility over the control process, the ability to move laterally on the control network. The PLC may have been compromised during the manufacturing process or in transit, where an adversary may have infiltrated the third-party vendor to introduce malicious code and/or hardware modifications to the PLC.

Once the PLC has been compromised, the adversary may gain complete control over the device. With complete control, the attacker may force devices to report false control status information and perform a data subversion attack. A data subversion attack is a circumstance where false status information is reported while control processes are driven to unstable operating conditions to cause failures and damage. This type of attack was performed via the Stuxnet computer worm against the Iranian nuclear program.

Another type of intrusion would be the introduction of malicious hardware modifications into the actuators and sensors sold by third-party vendors [25]. If the attackers were to introduce malicious hardware into the sensors or actuators, then the adversary might be able to perform a data subversion attack.

System Integration

The addition of a data acquisition system at the physical layer of the control hierarchy will offer visibility to combat the challenges that device degradation and compromise introduce. The proposed system is designed to be installed between any PLC and the sensors/actuators. The data acquisition system converts the electrical I/O signals, current or voltage, into digital values that are returned to the control network. This additional out-of-band channel of data can be used as a cross-reference to the in-band control data being reported via the standard flow of information from the PLC. If a discrepancy exists in any of the control signal data channels, such as the water valve position or water tank level, system engineers can halt the process and investigate the inconsistency before damage occurs.

Chapter 6 details the electrical design of the data acquisition system. The function of the data acquisition system is to convert arbitrary control I/O signals into an appropriate range that can be digitized by an ADC. Once digitized, the values can be retrieved via a serial data interface on a microcontroller.

Control processes are sensitive and designed around a system's physics, therefore interfering with the control system's electrical behavior can be devastating. The proposed data acquisition system is also designed to have negligible impact on the electrical behavior via the implementation of a high impedance amplifier circuit. The high

impedance is designed to stop any control system power from leaking into the data acquisition system. Additionally, the data acquisition system is designed to be highly configurable and compatible with different forms of control signals, whether they be current controlled, voltage controlled, or PWM controlled.

The ADC circuit is the second portion of the data acquisition system. The ADC circuit is designed to have a microcontroller interface to select and retrieve the digitized value of an I/O signal. The software running on the microcontroller asserts a signal to select which control I/O signal it wants to retrieve, and the ADC circuit returns the specified data. Both the high impedance amplifier circuit and the ADC circuit were designed and manufactured onto a single printed circuit board (PCB). This PCB can be mounted onto a microcontroller for easy integration.

The final portion of the data acquisition system is the microcontroller and its software, which are used to extract real-time I/O data. An example software design was created to demonstrate the capabilities of the data acquisition hardware. Chapter 7 details the software design of the hardware device driver and a simple data monitoring system. The hardware device driver reads the control I/O signal values via a serial interface on the control tap circuit.

For purposes of demonstration, a simple data monitoring system was implemented that compares the control tap values with values retrieved from the PLC. Realistically, the data acquisition from the system and PLC would be transmitted to a network endpoint containing an IDS to aid in the detection of anomalous behavior. If any discrepancies are detected, alerts could be generated to halt production and investigate

the source of the error. Figure 5.4 depicts where the proposed control monitor would fit into the example control system design.

If any alerts are generated, system administrators and engineers could investigate the cause and backtrack to determine the cause of the system deviation. This gives the administrators and engineers an additional point of reference to understand where the problem lies: whether it is a compromised hardware device and/or program, or simply a degrading device in the field.

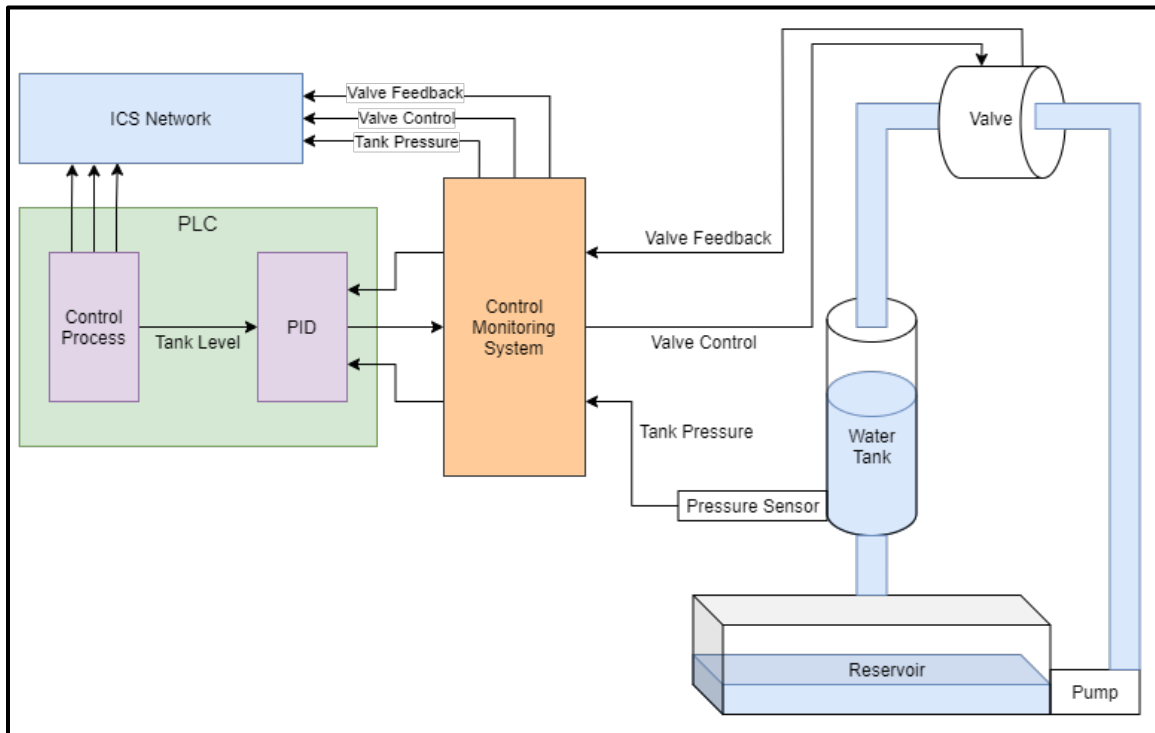


Figure 5.4 Water Tank Control System with Monitor

CHAPTER 6: HARDWARE DESIGN

Control Tap Overview

Monitoring and tracking of the analog I/O signals from sensors and actuators are normally performed by the PLC. However, a beneficial feature the control tap circuit provides is an out-of-band avenue of data at the physical layer of the control hierarchy. By obtaining the analog signals on an out-of-band channel from the PLC and its network connection, security administrators have a second point of reference with which to compare in-band I/O values. This comparison may enable security administrators and engineers to detect discrepancies between the channels, which could be a sign of system degradation or infiltration.

To record the analog signals and use them in the digital world, the signals must be converted from analog to digital values. While this may seem like a trivial task, the electrical design of the data acquisition unit must not interfere with the control system. This chapter describes the design process of the data acquisition circuitry that produces a digital value which can be utilized by data scientists and security teams. The electrical design consists of a few amplifier circuits that condition the control I/O electrical signals within a range of voltages that can be converted to a digital value by an ADC. Each amplifier circuit will contain different electrical and amplification stages. The electrical and amplification stages are linked in series and the final amplification stage is presented to the input terminal of an ADC. All amplification stages, ADCs, and power supplies

come together to form an amplifier sensor circuit. The flow of information from ICS control I/O to out-of-band data is presented in Figure 6.1.

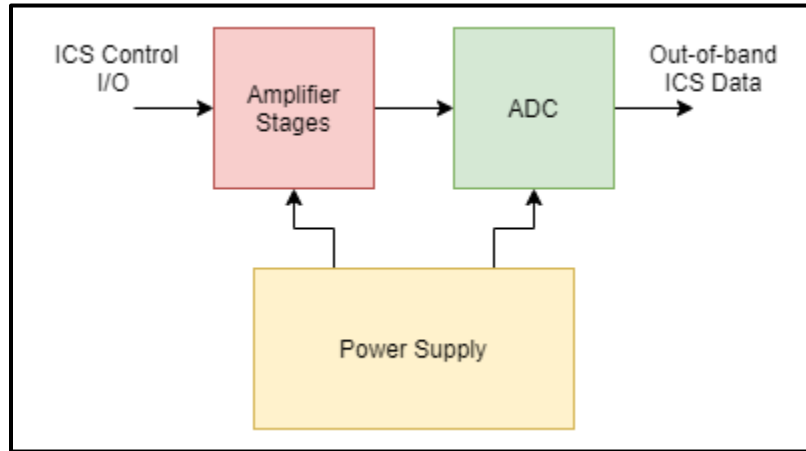


Figure 6.1 Amplifier Sensor Circuit Block Diagram

One challenge in the design of the amplifier circuitry is preventing electrical interference with the physical control process. Each of the actuators and sensors send and receive electrical signals to perform the control and command of the physical process. If these electrical signals are modified in transit to their destinations, then the control process will be disrupted. Control processes are designed around the physics of the control system and must not change how the system electrically behaves. To prevent this, the amplifier circuits were designed to have high input impedance. By doing so, negligible current (and subsequently power) leaks into the control tap circuit.

Amplifier Sensor Circuits

There are a few different amplifier circuits that may be required, depending on the type of electrical I/O signals utilized by the control system application. A control system may have current controlled, voltage controlled, or pulse-width modulation (PWM)

controlled I/O. For this reason, the data acquisition system features amplifier circuits compatible with each of these different forms of electrical control.

Typically, there are industry-accepted ranges of electrical signals used in ICSs. The first and most common type of electrical control signal used in the ICS industry is low power current, typically ranging from either 4 mA to 20 mA or 0 mA to 20 mA. The second most common type of electrical control signal used is voltage which can typically range from -10V to 10V, 0V to 5V, 0V to 10V, etc. The final type of electrical control signal is low power PWM which typically has voltage levels of 0V and 5V with a varying duty cycle.

ADCs will convert an analog signal to digital value based on a reference voltage, V_{REF} . If an ADC produces digital values with N bits, then numbers from 0 to $2^N - 1$ will represent the input analog voltage from 0V to V_{REF} . The number 0 would represent 0V and the number $2^N - 1$ would represent the reference voltage V_{REF} . The analog to digital conversion process is presented in Figure 6.2 [27], where the analog signal $u(t)$ is converted to a digital signal D with N bits.

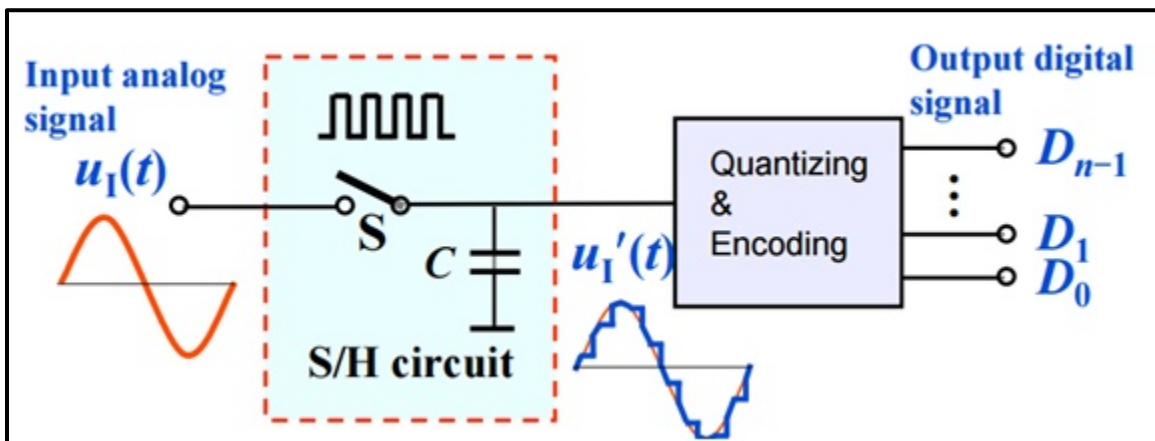


Figure 6.2 Analog to Digital Conversion Process

Sometimes a control system may not utilize the full electrical range when controlling a device. If the output of the amplifier circuit utilizes a portion of the voltage range of 0V to V_{REF} , then the output of the ADC will not utilize the full digital resolution of N bits. This is suboptimal, and cases like this occur frequently. For example, a heater may only require a 3V to 6V range while the entire control system has a 0V to 10V supply. In this case, only 30% of the digital space would be utilized by the ADC. For this reason, each amplifier circuit on the control tap is designed to have modular amplification stages to fine tune the output voltage based on the input criterium of the ICS application.

Current Sensor Circuit

Before the current inputs and outputs of a control system can be converted to a digital value and measured, they must be converted to a voltage. To achieve this, a resistor can be placed in series with the I/O signal to measure the voltage drop across it. By using Ohm's law $V = IR$, current can be calculated by modifying the equation to $I = V/R$. However, placing a resistor in series with the load may alter the electrical behavior of the control system, which is unacceptable. To avoid this, using a very small resistor in comparison to the load will add negligible change to the total resistance while still allowing measurement of the voltage across the resistor.

Amplifying the differential voltage across the resistor will require a differential amplifier circuit to produce an output voltage referenced to ground. This setup could convert a 0 mA to 20 mA signal into a 0 V to 4V output signal. Figure 6.3 depicts a differential amplifier circuit and its gain equation. If the control system is using a smaller range of the total current range, for example 10 mA to 15 mA the of 0 mA to 20 mA, then a second differential amplifier circuit is required produce the same 0V to 4V output

signal. This is done via a second differential amplification stage. In this second stage, the lower end of the voltage range is subtracted from the input and then amplified. The circuit was designed to produce a final output that is within the acceptable range of voltages for the ADC. Figure 6.4 depicts the current amplification stages performed by the control tap circuit before producing a digitized value.

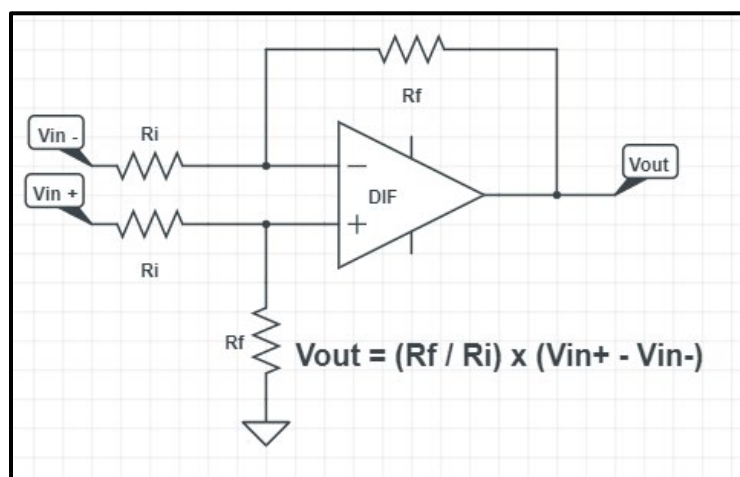


Figure 6.3 Differential Amplifier Circuit

Figure 6.5 depicts each of the sub circuits of the current sensing circuit. The first electrical circuit is the sensing resistor in series with the control system load. The second electrical circuit is a voltage divider network that divides the voltages on either side of the sensing resistor. This voltage division is performed due to the electrical limits of amplifiers. Amplifiers are rated for maximum voltages, and because the ICS application may be using voltages that exceed these limits, voltage division may be required. Additionally, the voltage divider network is implemented using high impedance resistors to maintain the high impedance of the control tap circuit. After the voltage divider network, the divided voltages are input into buffer amplifier circuits. These amplifiers are electrically isolating due to their naturally large impedance and ability to buffer the

control system from the sensing circuit. The outputs of the buffer amplifier circuits are the positive and negative inputs to the first differential amplifier stage. The output of the first amplifier stage is the positive input to the second amplifier stage, and the negative input is an adjustable voltage supply that subtracts the low ended voltage before amplification. Finally, the second stage differential amplifier output is connected to an ADC.

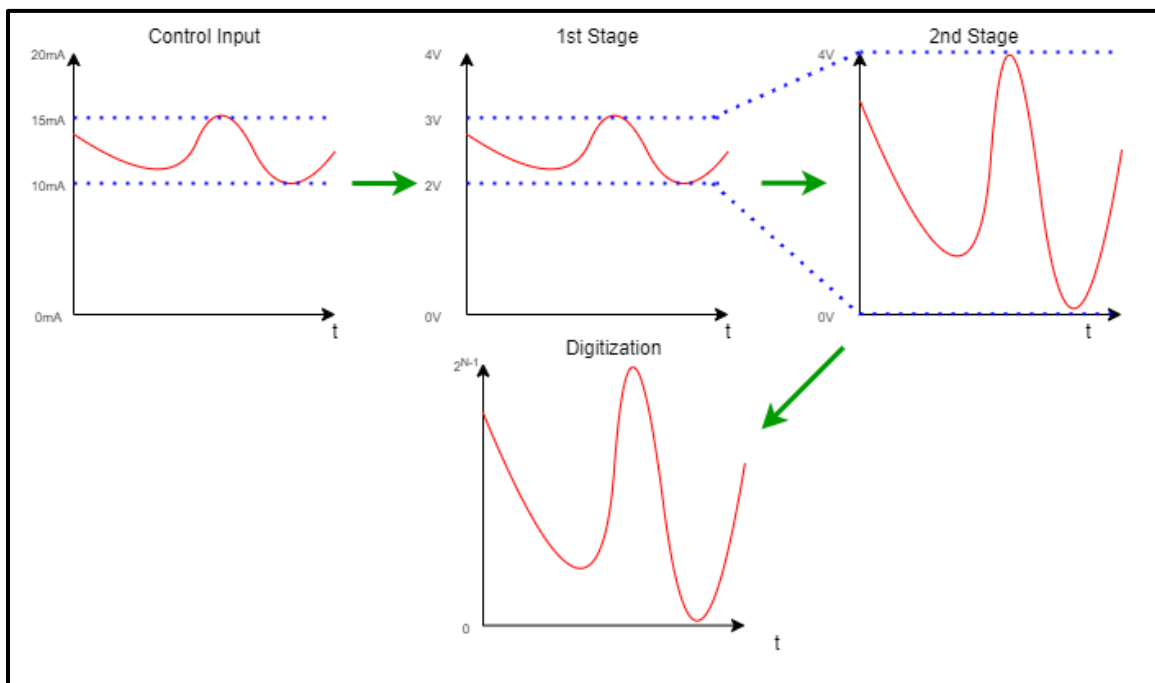


Figure 6.4 Current Amplification Stages

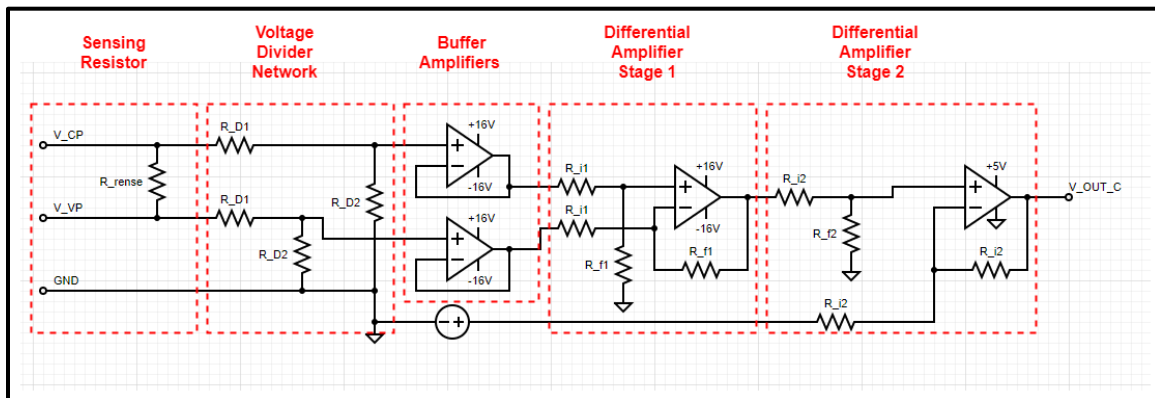


Figure 6.5 Current Sensor Amplifier Circuit

In summary, there are five different electrical or amplification stages used in the current sensing circuit to produce an output voltage within an acceptable range for an ADC. This circuit was designed to be configurable for different ICS applications without impacting the control process due to high input impedance.

Voltage Sensor Circuit

The voltage sensing circuit follows the same principles as the current sensing circuit. The main difference between the two circuits is that there is no sensing resistor or buffer amplifiers to measure a voltage across the resistor. Instead, a voltage is already provided, and the first stage of amplification is a standard single-ended amplifier. The voltage divider network and second amplification stage of the voltage sensing circuit is identical to the current sensing circuit. Figure 6.6 depicts the visibly simpler voltage sensing circuit.

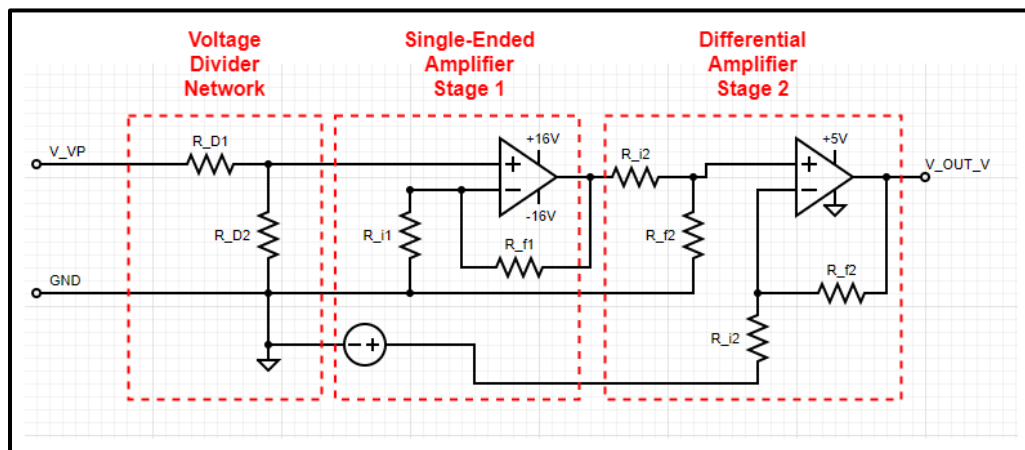


Figure 6.6 Voltage Sensing Amplifier Circuit

PWM Sensor Circuit

The PWM sensing circuit follows the same principles as the prior two circuits. The PWM signal needs to be averaged before conversion to an analog representation of the duty cycle. Because of this, there is a distinct difference in the PWM sensing circuit

from the other two circuits. Instead of a first stage of amplification, the PWM signal goes through a second order low-pass filter to integrate the signal to an average value. Figure 6.7 depicts how a PWM signal and its varying duty cycles can be averaged to a stable output voltage.

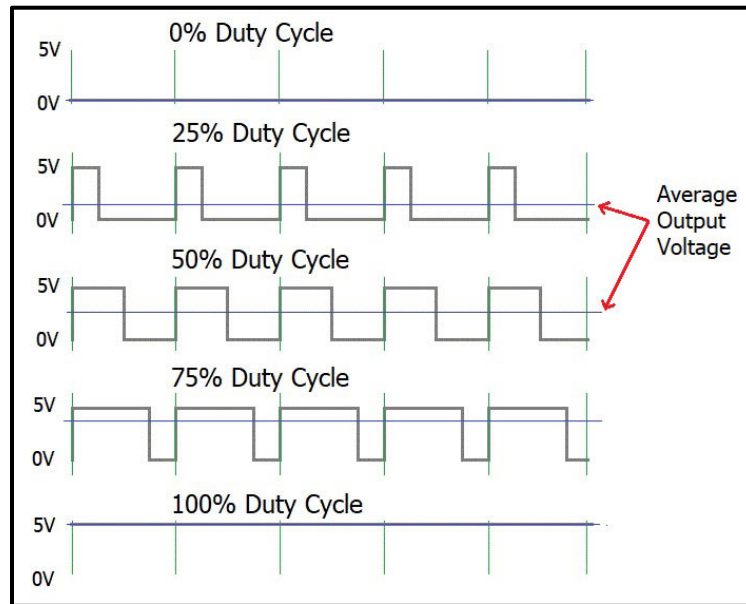


Figure 6.7 Pulse Width Modulation

Once the voltage has been averaged out, the output of the filter can be sent into the second stage of amplification. The voltage divider network and second stage of the PWM sensing circuit is identical to the prior two circuits. Figure 6.8 depicts the PWM sensing circuit who's first amplification stage is notably different than the previous two circuits.

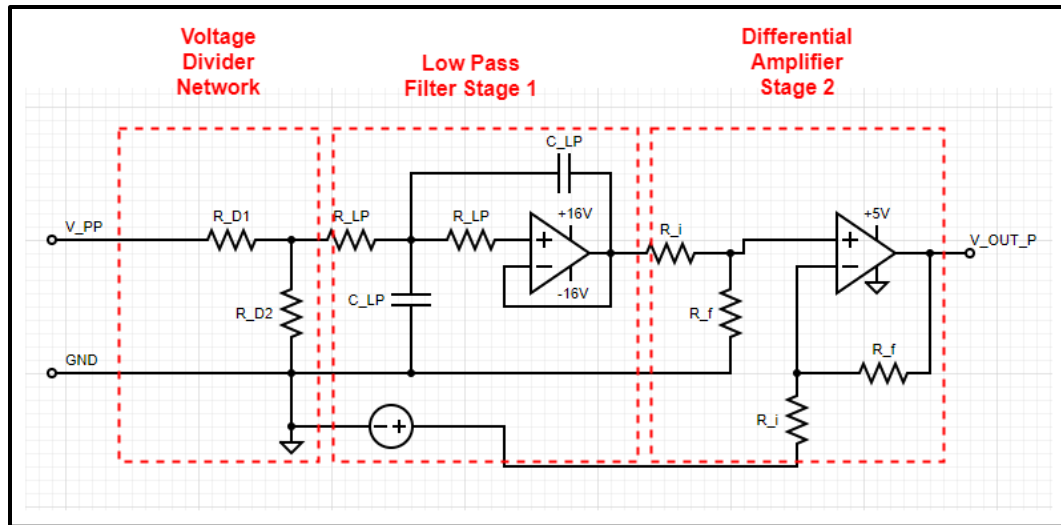


Figure 6.8 PWM Sensing Amplifier Circuit

ADC Interface

Once the desired ranges of voltages are received from the chosen sensor amplifier circuit, the values can be digitized by an ADC circuit and retrieved by a microprocessor. The chosen microcontroller for the thesis proposal was a Raspberry Pi 4, which has a 40-pin general purpose input/output (GPIO) interface for serial communication and general-purpose digital control. The Raspberry Pi 4 also has a dedicated serial peripheral interface (SPI) hardware unit which can be configured and controlled through software via a hardware driver. The microcontroller can assert selection pins to select which ADC to read from via a chip selecting circuit. All the circuitry, from sensing circuits to ADCs and their chip selecting circuit, was designed and manufactured onto a printed circuit board (PCB). Once assembled, the PCB can be mounted onto the Raspberry Pi's 40-pin GPIO interface for easy integration. This 40-pin interface and the pinout functionality is presented in Figure 6.9 [29].

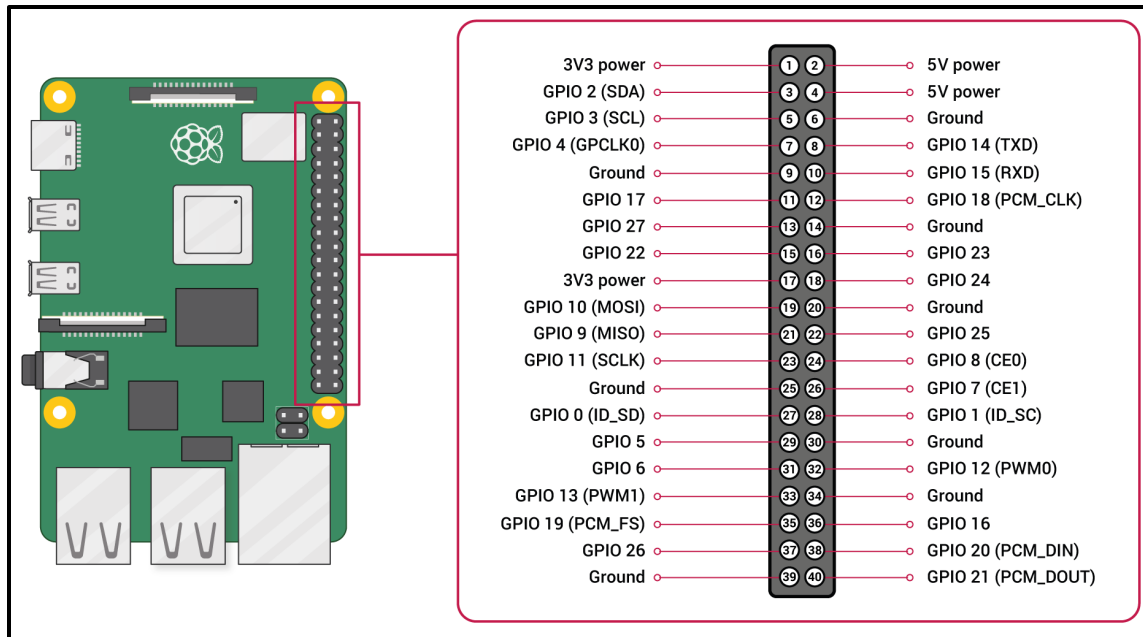


Figure 6.9 Raspberry Pi GPIO Interface

SPI protocol is a serial communication protocol in which a master unit can communicate with one or many slave devices to read from or write to. Figure 6.10 depicts the common structure of an SPI master unit connected to SPI slave(s). SPI has 4 different digital inputs and outputs: the system clock (SCLK), slave select (SS), master output / slave input (MOSI), master input / slave output (MISO). The clock signal drives the protocol and serves as the heartbeat to time each SPI transaction. The slave select is used to activate the slave with which the master is communicating. Lastly, the MISO/MOSI lines are used for duplex communication between the master and the slave.

The only difference in between the SPI structure depicted in Figure 6.10 and the Raspberry Pi 4's SPI interface is that the SPI interface on the Raspberry Pi 4 only has a singular slave select pin, meaning it is only capable of selecting a singular slave. For this reason, the ADC interface must also have a slave selecting circuit that can select as many slaves as it needs. The PCB is equipped with 4 of each type of sensing circuit, meaning

there are 12 ADC slaves on the board. To accommodate the need of 12 different slave-select pins, a chip selecting circuit was devised.

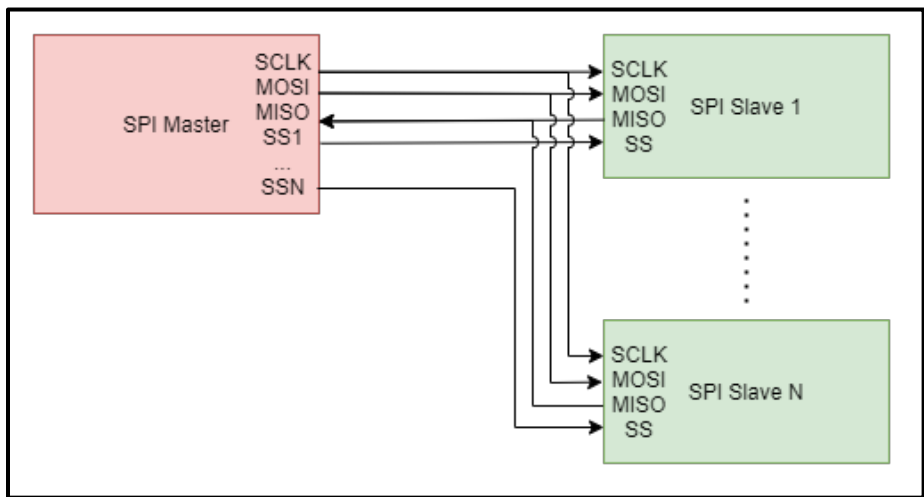


Figure 6.10 Typical SPI Structure

This chip selecting circuit takes in 4 GPIO select signals, and a 4x16 decoder allows for the selecting of all 12 ADCs. The ADCs being used have active LOW slave select pins, meaning when the slave select line is 0V, the slave is active. The 4x16 decoder uses active LOW selecting to meet these requirements. Figure 6.11 depicts the layout of the designed ADC chip selecting circuit.

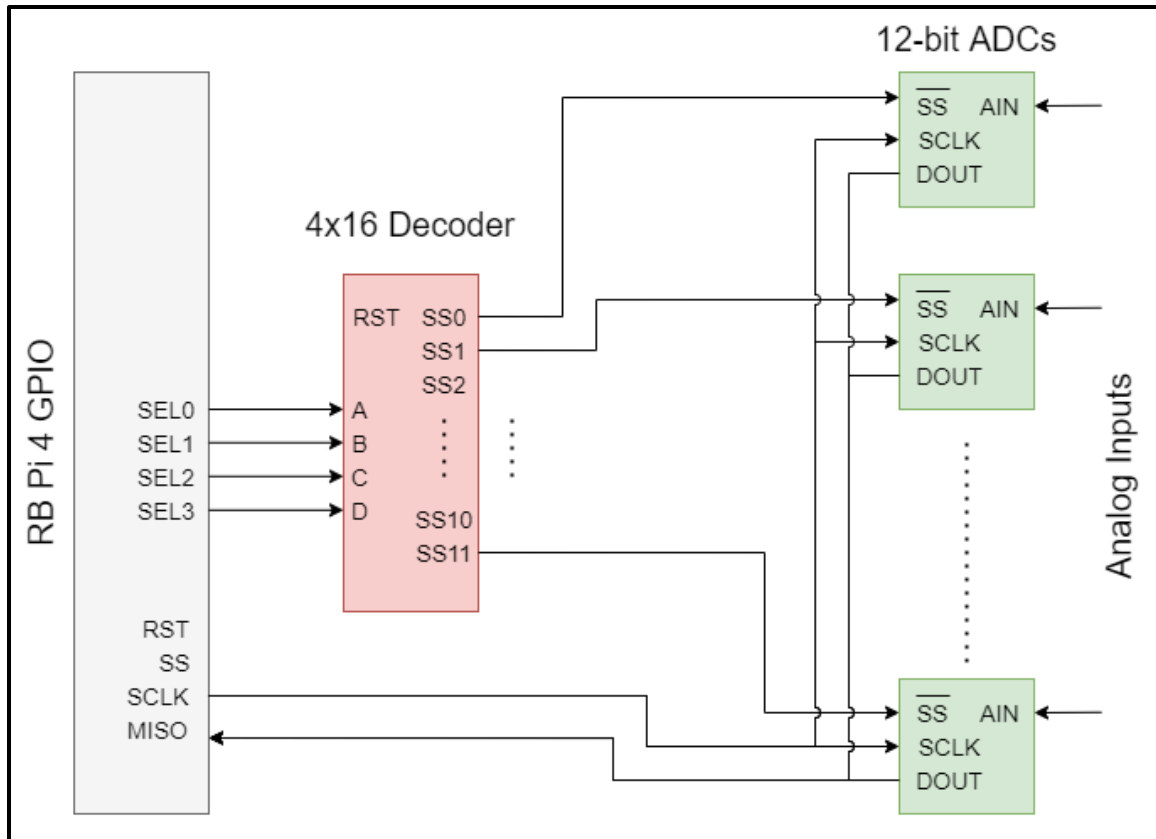


Figure 6.11 ADC Chip Selecting Circuit

This chip selecting circuit only allows for a single ADC to be controlling the MISO data bus at any given moment. When the master wants data from a slave, it will assert the necessary selecting pins and supply the SCLK to the slave devices. When this occurs, the activated slave will return its digitized value to the master over the MISO line one bit at a time. For example, if the Raspberry Pi 4 wants data from ADC number 7, it will assert the slave select pins $\{\text{SEL3, SEL2, SEL1, SEL0}\} = '0\ 1\ 1\ 1'$ (7 in binary) and send the SCLK signal until the transaction has been completed.

The front and back of the PCB described in this chapter are presented on the next two pages in Figures 6.12 and 6.13. The schematics of the PCB design are included in Appendix A.

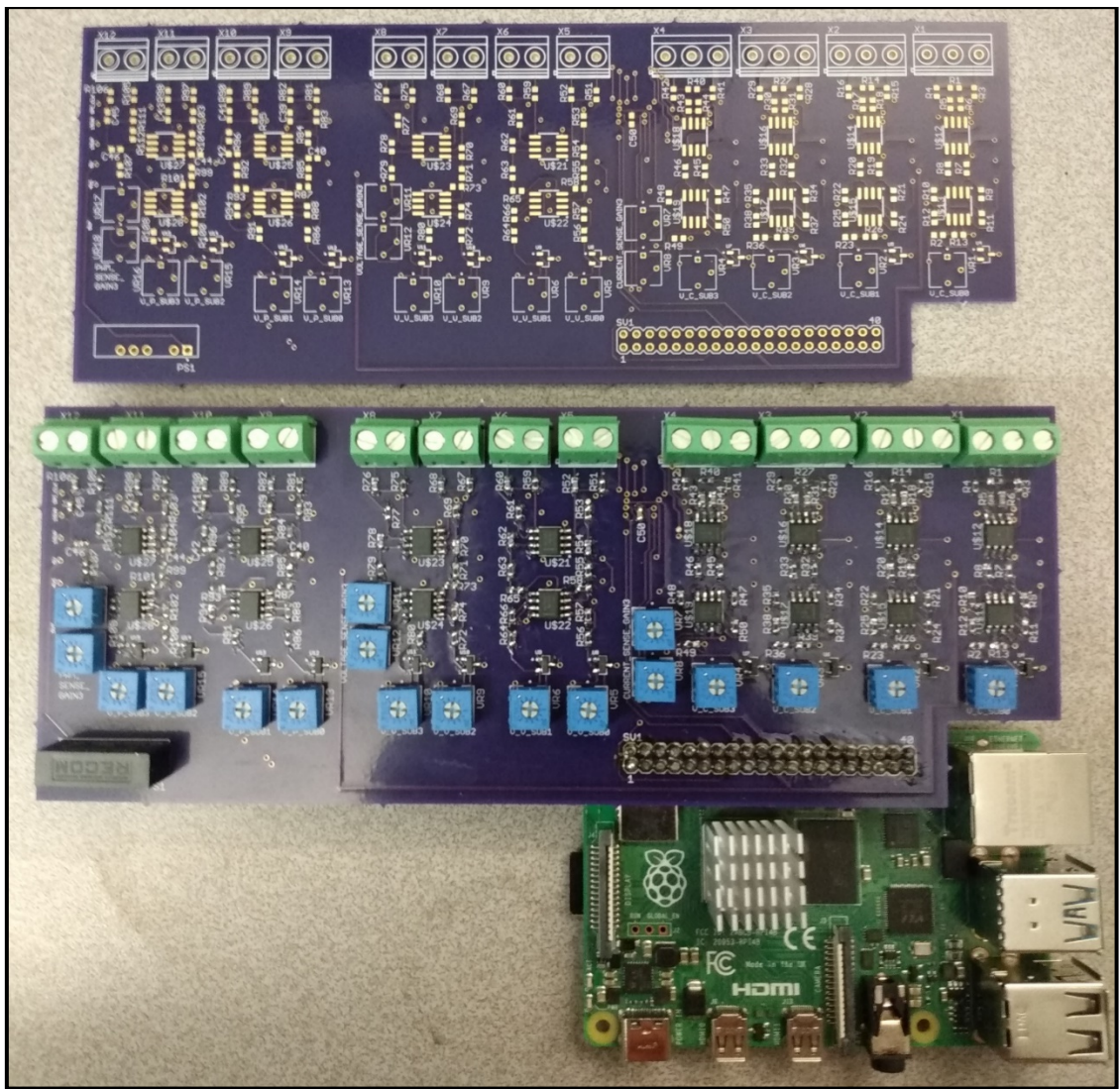


Figure 6.12 PCB and Raspberry Pi (front)

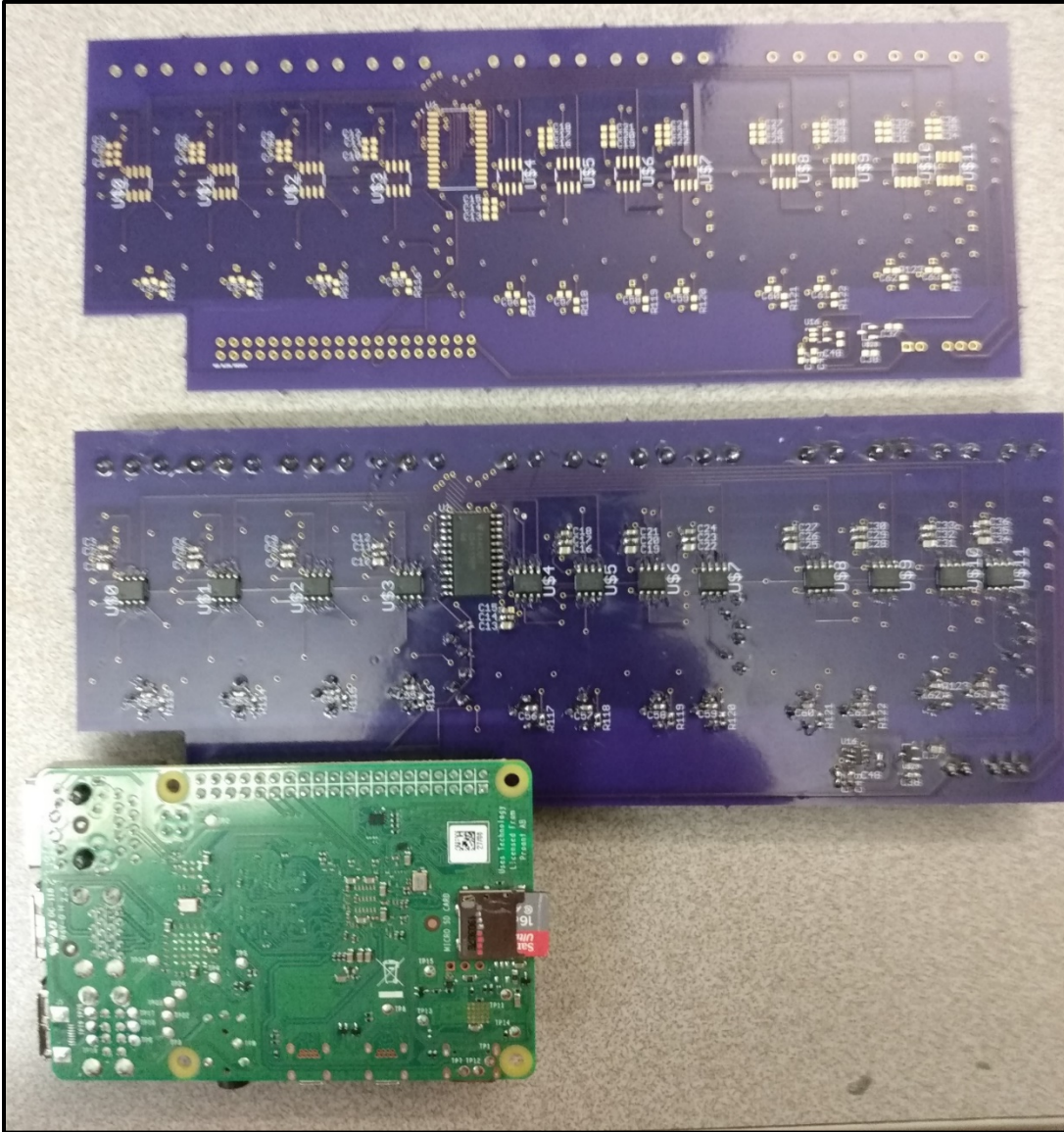


Figure 6.13 PCB and Raspberry Pi (back)

CHAPTER 7: SOFTWARE DESIGN

While the front end of the system is based in hardware, the remainder of the proposed data acquisition system is configured in software. For purposes of demonstration, a mock IDS process was designed to exhibit the capabilities of the data acquisition system. There are 3 main processes that are responsible for performing the data acquisition, monitoring, and detection of anomalous behavior.

The first process reads and formats the serial data retrieved from an ADC before handing it off to the next process. A hardware driver communicates with the hardware via an SPI interface which is configured before any transactions take place. This serves as an application program interface (API) to any process that wants to retrieve out-of-band data from the hardware via requests and responses.

The second process performs in-band data reads from a PLC via the ModBus protocol and has a very similar function to the ADC hardware driver. The ModBus reader process connects to a PLC over a TCP/IP network connection and reads register data from the PLC's internal register space. This process serves as an API to any process that wants to retrieve in-band data from the PLC via requests and responses.

The third process is a control monitoring process that interfaces with the two aforementioned processes. This process interfaces with both APIs to make read requests from designated ADCs or PLC registers. The process receives the responses of incoming data from the two APIs and performs simple control monitoring by comparing the data to each other. The control monitoring contains a file of a recorded normal process behavior. The process performs comparisons on the ADC data, PLC data, and known process

behavior data. In comparing these channels of data, discrepancies may arise. This process has settings to configure what level of discrepancies to trigger the generation of security alerts. Figure 7.1 depicts the control process flow and paths for inter process communication.

In a real ICS environment, the IDSs employed are sophisticated and interconnected. The main purpose of this thesis is to understand the importance of, and to demonstrate increased control system hardware visibility via the data acquisition system. If data scientists should find the data acquisition system of benefit for their systems, they must determine how the data acquisition system can be integrated into their existing IDS infrastructure.

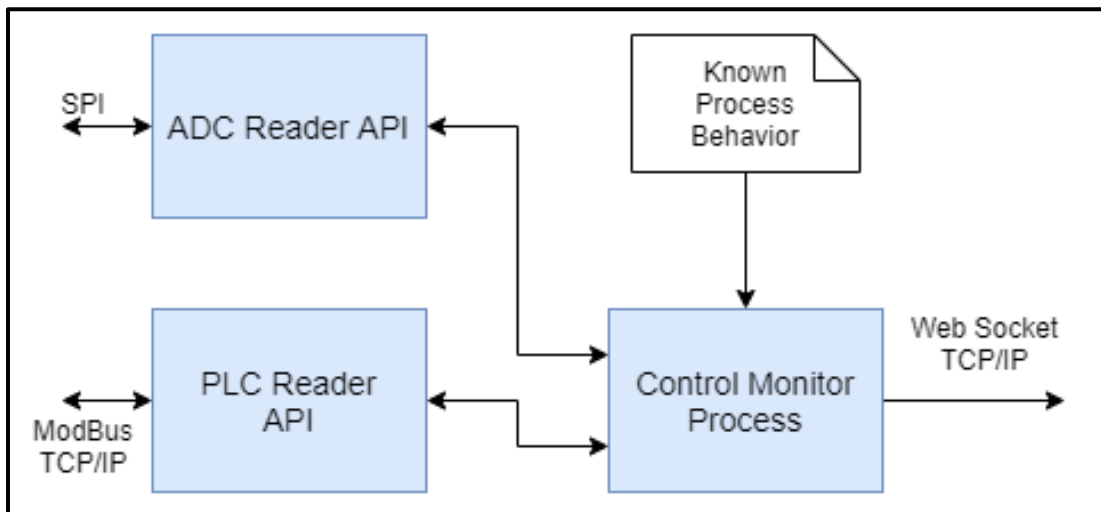


Figure 7.1 IDS Software Flow

The source code for each of the processes described in the next sections are included in Appendix B.

ADC Reader API

The Raspberry Pi 4 comes equipped with a system on a chip (SoC), the BCM2835, which has many different integrated subsystems. Figure 7.2 depicts the BCM2835 SoC block diagram which details the CPU, GPU, memory, and some key peripherals typically used in embedded environments. The ADC reader API was written in C to access the SPI hardware driver libraries written for the BCM2835 SoC. This library is open source and can configure the settings of the SPI hardware peripheral. Once configured the library can read from and write to slaves via the designated SPI pins on the 40-pin GPIO header.

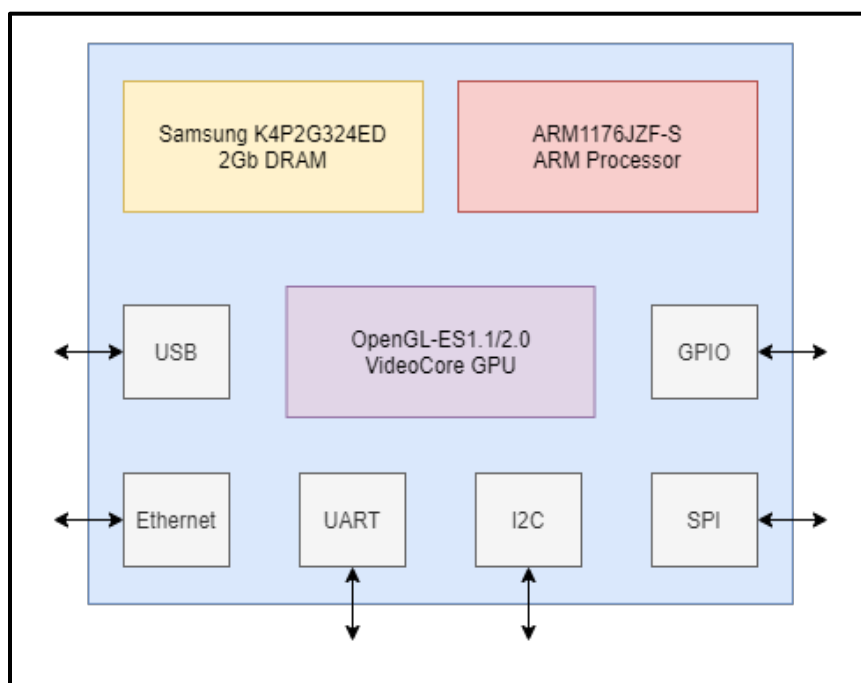


Figure 7.2 BCM2835 SoC Block Diagram

First, the API configures all the SPI settings. The process initializes all BCM2835 peripherals such as the SPI module and performs error handling if any of these initialization steps were to fail. Second, the process performs the configuration of the initialized SPI module. Once SPI has been configured, the process begins a polling mode

where it waits for SPI transaction requests from another process. When a request is received, the ADC Reader API fulfills the request by reading from the appropriate ADC and responding with the formatted data.

The ADC reader retrieved data from the appropriate ADC by asserting the GPIO selection pins associated with the ADC. Once asserted, the SPI driver is activated to read the raw serial data from the ADC over the MISO line. The ADCs are 12 bits, but the data that is received from the ADCs is contained in two 8-bit packets. The ADCs used were MCP3201s, and their datasheet details the format in which the data is received [23]. To make use of the received data, it must be modified to extract the 12 bits from the combined 16-bit data packet. Figure 7.3 depicts the formatting of the 16-bit data packet obtained from the MCP3201 datasheet.

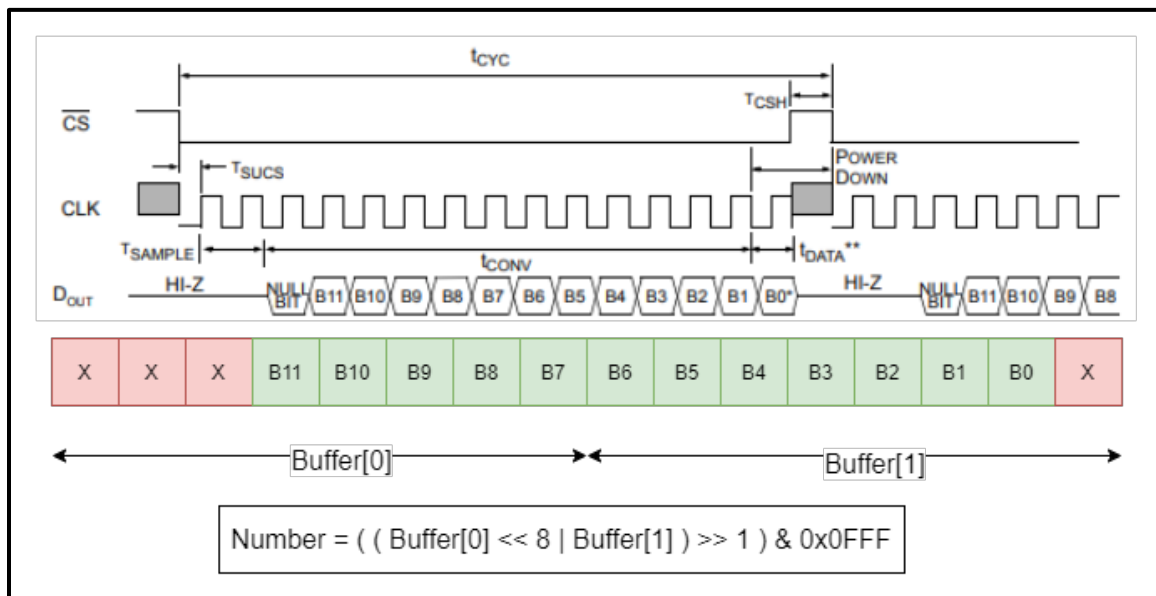


Figure 7.3 ADC SPI Transaction Format

Additionally, the SPI drivers are only capable of performing 8-bit SPI transactions, meaning two separate SPI transactions must occur to read from the full buffer inside the ADC. Based on this format, the first 8-bit buffer, buffer[0], will contain

the upper byte coming from the ADC. The second 8-bit buffer, `buffer[1]`, will contain the lower byte coming from the ADC. Once the data has been retrieved, it needs to be converted to the 12-bit number that was originally stored inside the ADC. The bottom of Figure 7.3 depicts the logical operations used to perform this conversion ($\ll N$ is logical shift left by N bits. $\gg N$ is logical shift right by N bits. $|$ is bitwise OR. $\&$ is bitwise AND.)

PLC Reader API

The PLC used in the test control system environment is the Allen-Bradley (AB) Micro 850. The PLC and its internal control process were configured and programmed using the free to use Connected Components Workbench developed by Rockwell Automation. Besides configuring the control process, the user can configure where different control process constants, variables, and control I/O is stored in its internal register space. The PLC Reader API was written to fulfill read requests to the PLC by retrieving register data via the ModBus protocol. Since ModBus is the industry standard protocol in the ICS environment for communicating with different control entities, it is also the protocol that defines the register space that the user can configure.

The ModBus register space region is defined as a range of registers typically starting from address 0 and counting to 65,535. The first 10,000 (00001 – 09999) registers are write-only registers that are allocated for output driver switches which are digital outputs that activate and deactivate different physical components such as motors, relays, or valves. The second 10,000 (10001 – 19999) registers are read-only registers that are allocated for digital input contacts that store status information of different physical components. The third 10,000 (20001 – 29999) registers are read-only registers

that are allocated for analog inputs which hold the digitized values of feedback information being used for control, such as pressure level, speed, temperature, etc. The fourth 10,000 (30001 – 39999) registers are write-only registers that are allocated for analog outputs that drive the control process to perform physical actions such as operating a motor or increasing the temperature inside of a vessel.

Since the Raspberry Pi 4 has ethernet and wireless communication abilities, it can be connected to the same network as the PLC. The PLC Reader process uses this network connection to read the internal register space and fulfill read requests in a similar fashion as the ADC reader process. The PLC reader process is configured with the static IP address of the PLC and the proper register address range for the control signals inside of the PLC.

The PLC Reader process is simple, was written in Python, and makes use of a library called “uModBus”. The PLC reader process configures and establishes a network socket connection to the PLC. Once connected, it then enters its main program loop where it waits for a Modbus read request that it can initiate to the PLC. Once receiving the data back from the PLC, the process fulfills the request by responding with the internal register data.

Control Monitoring Process

The control monitoring process compares the PLC control process data, the data acquisition system’s data, and a record known control process. This is accomplished in real-time by requesting data from the PLC and ADC APIs. Periodically, the process requests data from both the PLC and out-of-band data acquisition unit and then records the data. Using this data, the process stores a constantly updated array of data recorded

from the PLC and data acquisition unit. The width of this array, and a few other parameters such as error tolerance and error count tolerance are configurable and can be used to tighten or loosen conditions required to generate an alert.

The control monitoring process starts by priming the sliding time window with data from each of the input locations including: the control tap, the PLC, and the known control process. Once the window of time has been filled, the control monitoring process begins by comparing each of the time-series inputs with one another. In the case of the example control system, the primary control variable is the water level of the tank. The control process compares the both the control tap and PLC values to each other with the known control process. If any of the compared values exceed the tolerance boundaries, an alert is generated and handed off to the security alert process.

Figure 7.4 depicts normal control process behavior without any disturbances to the control signals. Figure 7.5 depicts the normal control process behavior with an added disturbance in the system which is picked up by the data acquisition unit and PLC. In this case, the control monitoring system generates alerts signaling to the administrator alert process that both the data acquisition unit and PLC have returned control signals that deviate from the normal control process behavior.

In addition to normal and disturbed process behavior, the data acquisition unit was able to detect instances where the PLC was reporting false information. Figure 7.6 depicts an instance where a harsh disturbance was applied to the control system, yet the PLC reported normal behavior. Figure 7.7 depicts an instance where the PLC reported a completely different behavior than the physical control process behavior. In this case, the PLC reported a sinusoidal process while the control system followed a sawtooth

behavior. In both cases, the data acquisition system was able to detect a discrepancy in the false information presented by the PLC and the physical system behavior.

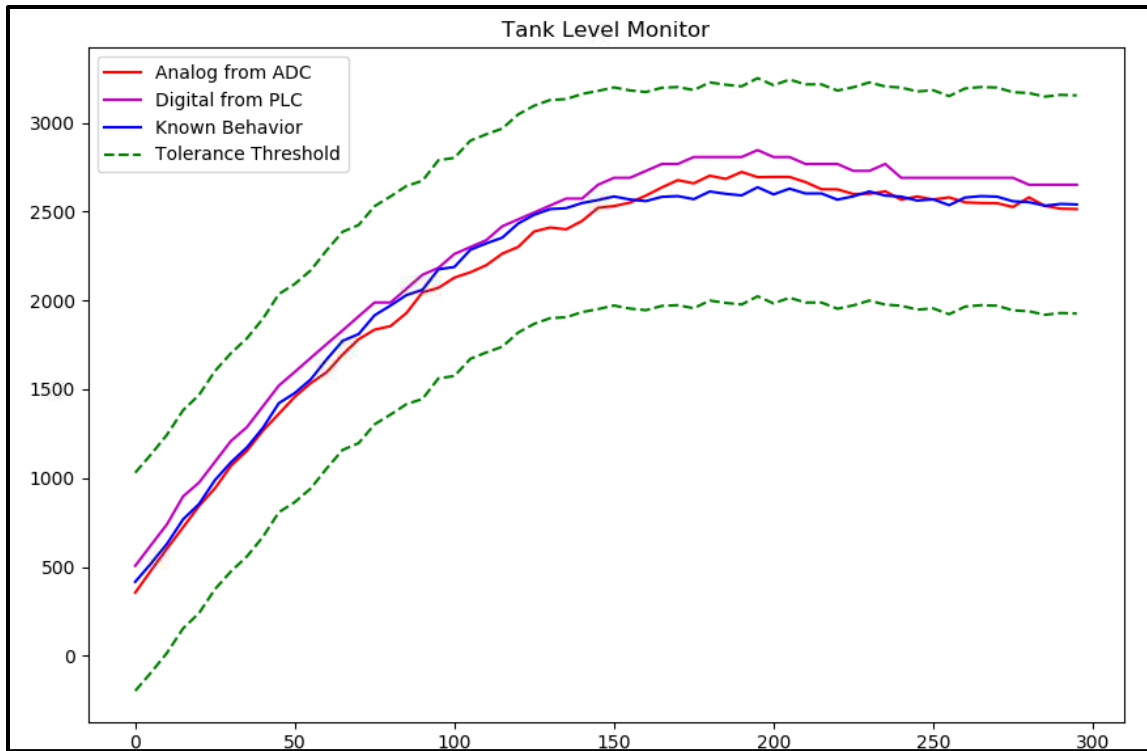


Figure 7.4 Normal Control Process Behavior

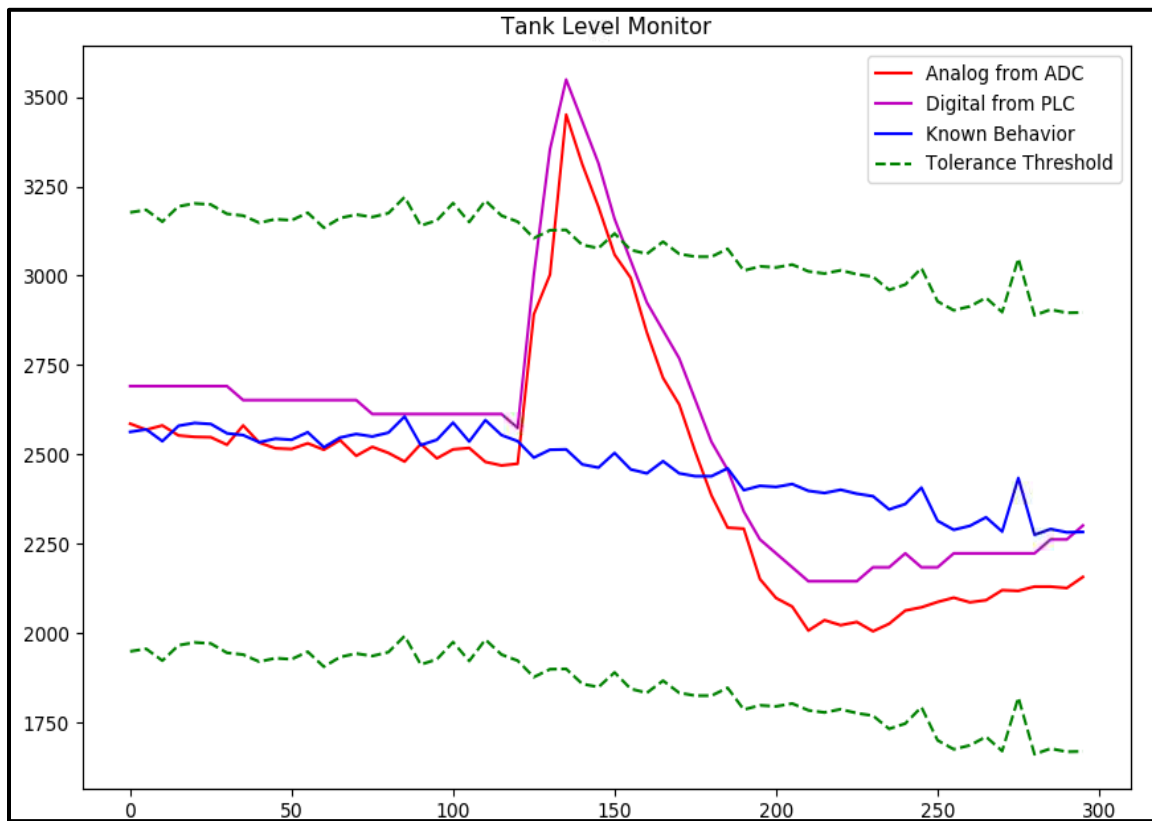


Figure 7.5 Disturbed Control Process Behavior

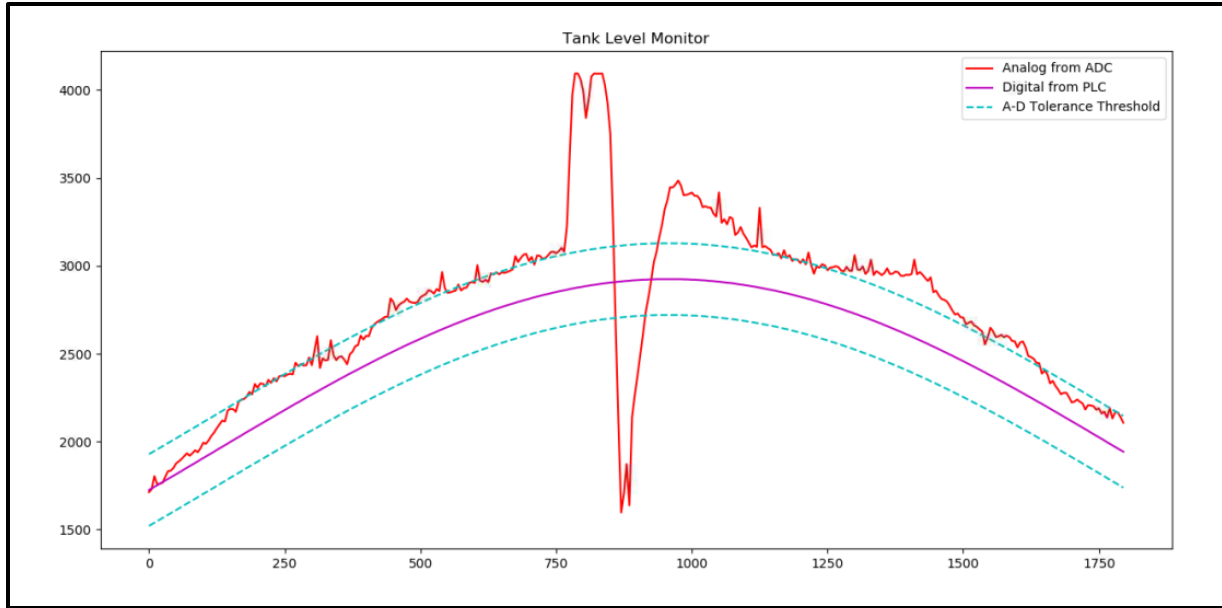


Figure 7.6 Undetected Disturbance

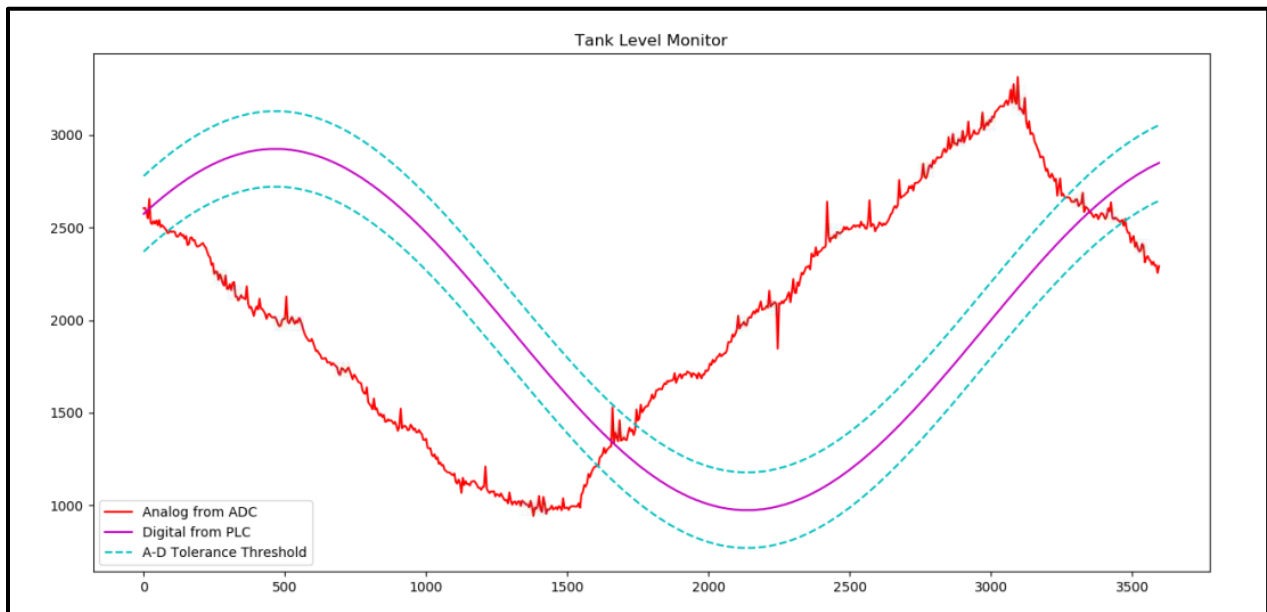


Figure 7.7 False Control Process

CHAPTER 8: FUTURE RESEARCH

Artificial Intelligence & Machine Learning

The main purpose of this Masters' thesis is to propose and demonstrate a new data acquisition system at the lowest level of the control hierarchy. Integrating this data acquisition into a larger and more sophisticated IDS is outside the scope of this thesis. The control monitoring software implemented in this thesis is rudimentary and for demonstrative purposes. Realistically, industrial control facilities have more sophisticated real-time IDSs which involve databases of historic behavior, rule sets, and some form of machine learning (ML) with artificial intelligence (AI). By training some form of AI algorithm with known and expected behavior, industrial control facilities can detect and prevent attacks when they are occurring in real time by detecting anomalous behavior.

IDSs are frequently used in signature-based environments such as networks or on operating systems. However, for the purposes of protecting the physical control processes, there have been studies and implementations of IDSs that train AI to detect anomalous behavior at the control process level. The control process signals are digitized inputs and outputs that are being controlled by PLCs in the Field Layer. By training AI with many instances of this control process data, the AI can detect small to large deviations from the expected behavior.

Unfortunately, this intrusion detection model begins to deteriorate when trust in the training data and real-time data is compromised. In attacks such as data poisoning or data subversion, control process data being returned from the PLCs can be illegitimate

and cause problems in the training and detection phases of the AI/ML IDS model. If the data is compromised, it may cause the AI/ML algorithms to misclassify unwanted behavior. This can cause detection problems because the AI within the IDS has been trained to classify improper behavior as expected and normal. In addition to data poisoning during the training phase, component degradation and data subversion attacks during the detection phase may cause data to be illegitimate and appear normal while failures occur. In this circumstance, the IDS receiving illegitimate data may not be able to detect the unwanted behavior.

For this reason, the work of this thesis may be of use in the field of AI/ML [3]. This out-of-band channel of control process information recorded at the control process layer may be leveraged as an additional input to these AI/ML algorithms. By having in-band and out-of-band data acquisition channels, increased visibility may enable AI/ML algorithms to detect small discrepancies in the channels of data, signaling component degradation or a data subversion attack. Additionally, the data acquisition system can be used to verify the legitimacy of the data used in the training of AI/ML algorithms. By integrating the proposed data acquisition system into existing ICS IDSs, ICS security may be bolstered increasing system visibility and data resiliency.

FPGA Data Acquisition

The proposed data acquisition system introduces new vulnerabilities to the ICS network. The microcontroller performing data acquisition is running software on an operating system or bare metal resources. This exposes avenues on the ICS network by which attackers may inject code to compromise the out-of-band data acquisition system. An approach that would fix this problem would be to design the entire data acquisition

system in hardware. In an architecture entirely comprised of hardware, there would be no avenues by which the attacker could inject malware into the data acquisition system.

A versatile method of implementing a hardware data acquisition unit is via the utilization of a field programmable gate array (FPGA). FPGAs are reconfigurable hardware units that are commonly used in hardware prototyping or hardware systems that require functional flexibility and low cost. FPGAs are equipped with many different types of resources that are used to implement custom hardware logic and computational solutions repeatedly on the same chip. Within FPGA's there are configurable logic blocks (CLBs), I/O blocks, memory blocks, interconnect lines, etc. Configurable logic blocks are used to implement user logic. I/O blocks are used to transport data in and off the FPGA chip. Memory blocks are used for data storage. Lastly, interconnect lines are used to connect and route these resources together to form a complete hardware design. Figure 8.1 highlights these commonly used resources.

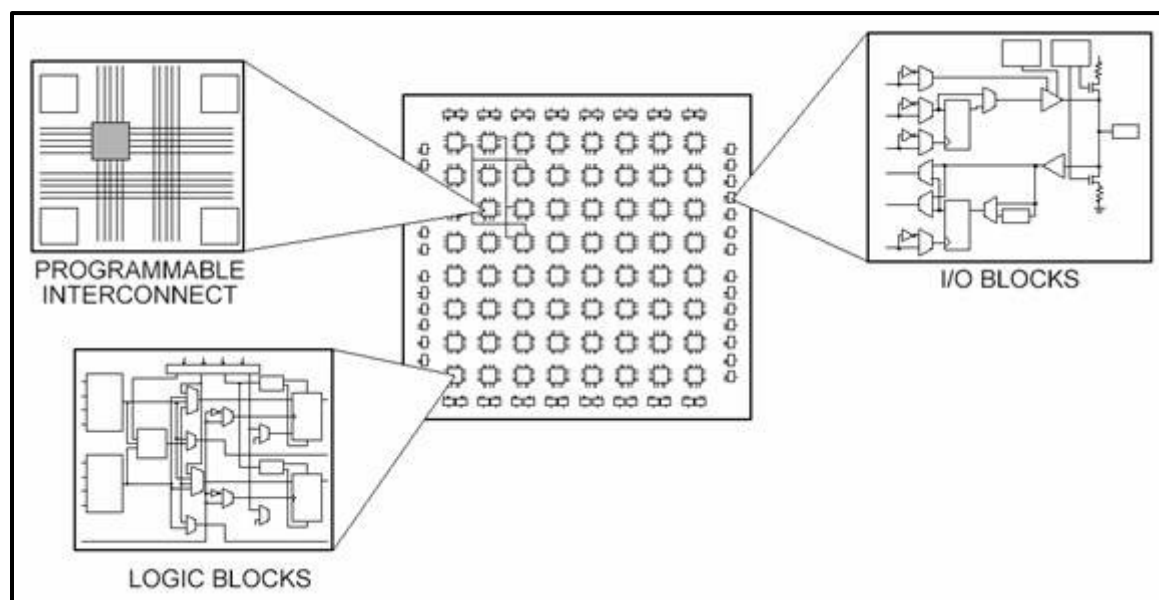


Figure 8.1 FPGA Resources

For purposes of demonstration, a simple FPGA data acquisition unit was designed and integrated with the PCB described in Chapter 6. The FPGA board used in this demonstration was the Digilent Nexys4 DDR FPGA board. The FPGA hardware design was specifically created to replace the data acquisition functionality of the Raspberry Pi microcontroller. In the example design, a serial universal asynchronous receive/transmit (UART) connection is made through the FPGA. This connection is used to send data packets containing the desired number of the ADC. Once the packet is received by the FPGA, the FPGA logic asserts the necessary chip selecting signals to activate the desired ADC. Once the ADC is selected, the FPGA design uses the SPI protocol to retrieve the data packet from the ADC. Once the data is acquired, it is returned to the PC host over the serial UART connection. Figure 8.2 depicts the layout of the FPGA data acquisition system. Figure 8.3 depicts the block design programmed into the FPGA.

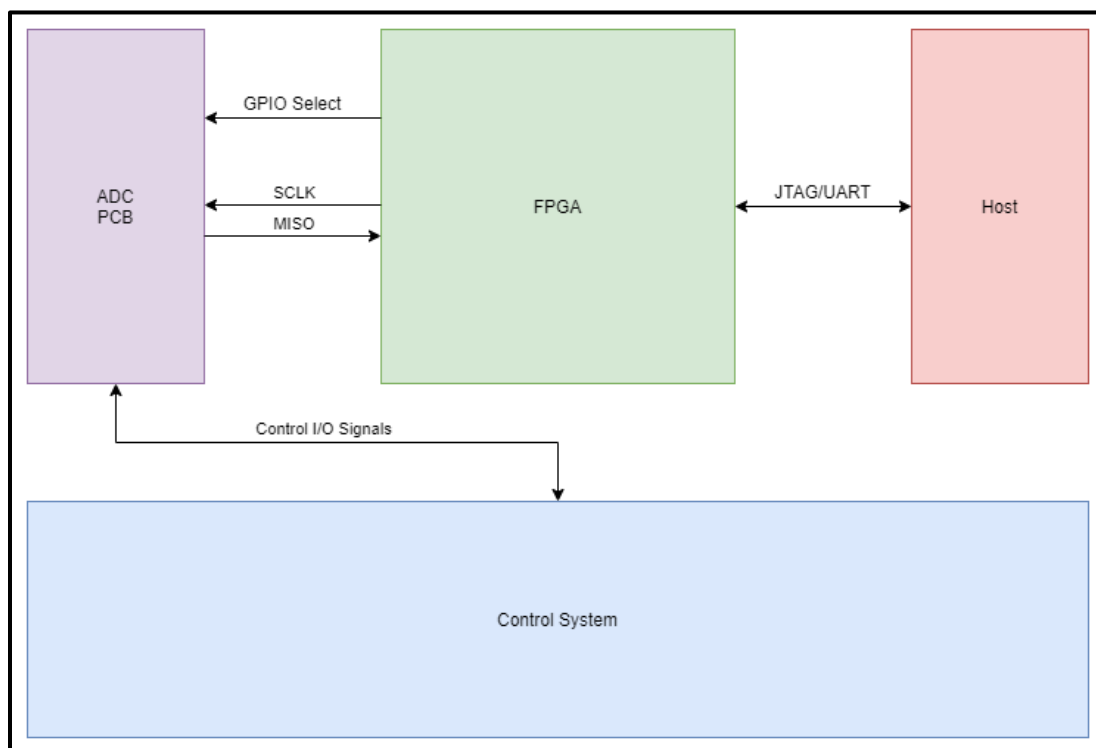


Figure 8.2 FPGA Data Acquisition System

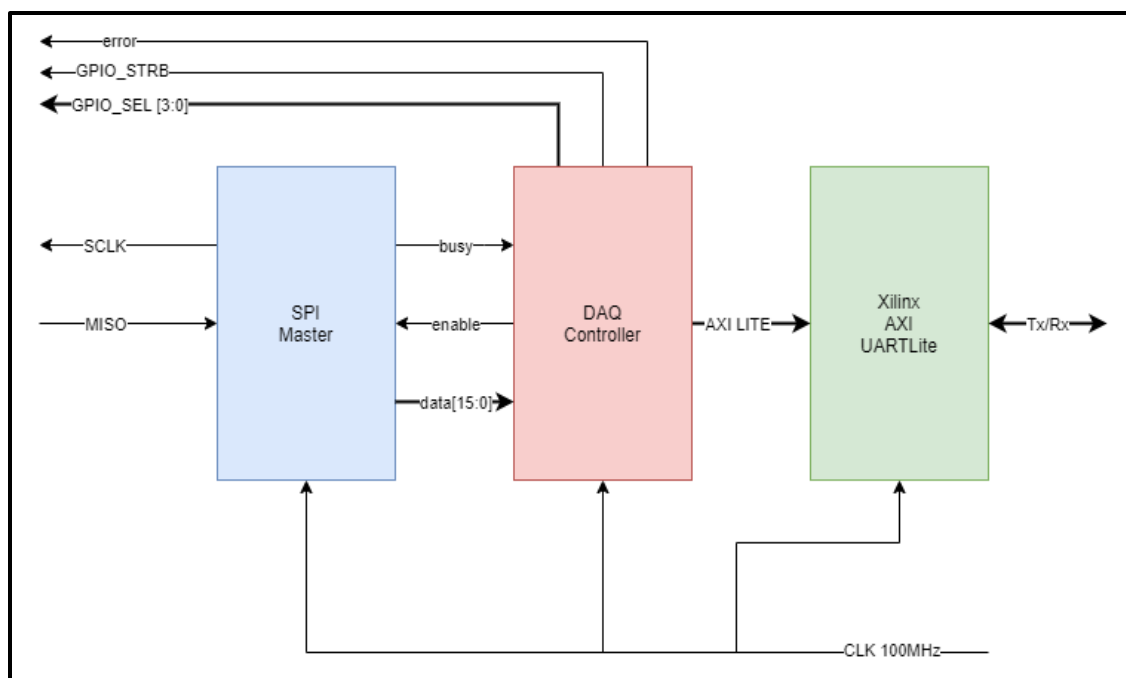


Figure 8.3 FPGA Block Design

Figure 8.4 shows the connections made between the PC, the FPGA board, and the PCB. The USB UART connection is on the left of the green FPGA board. The GPIO and SPI signals were connected from the FPGA board to the PCB using jumper cables.

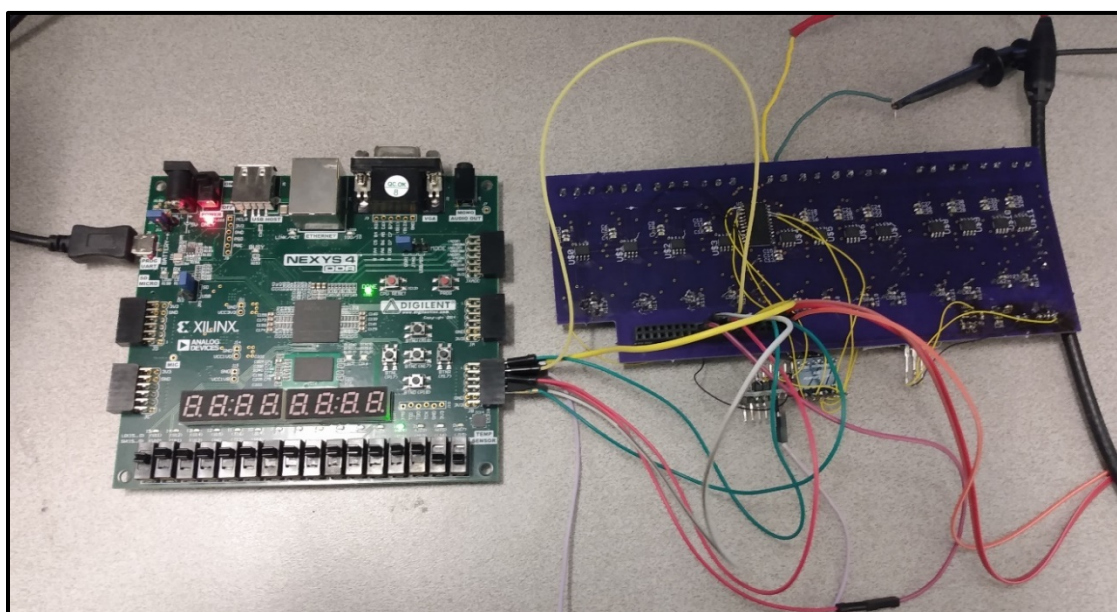


Figure 8.4 FPGA / PCB Connection

Figure 8.5 depicts an oscilloscope capture of the SPI clock (yellow), and the SPI data packet (blue). Figure 8.6 depicts the data being returned to the PC via the serial connection and displayed in a terminal. While the design is very simple, the main purpose of this demonstration was to show how an FPGA could replace the microcontroller in the data acquisition system. In doing so there is no operating system or software running in the entire design.

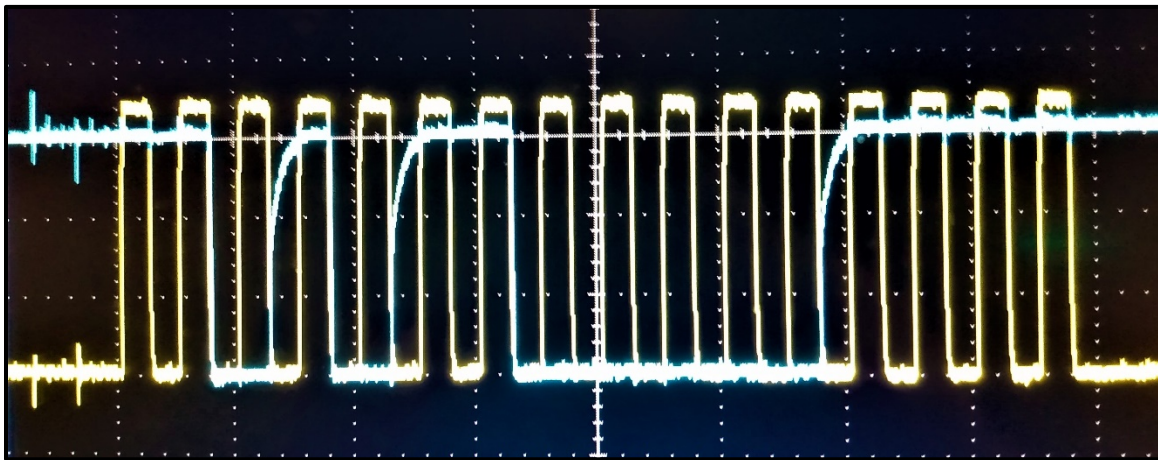


Figure 8.5 SPI Waveform

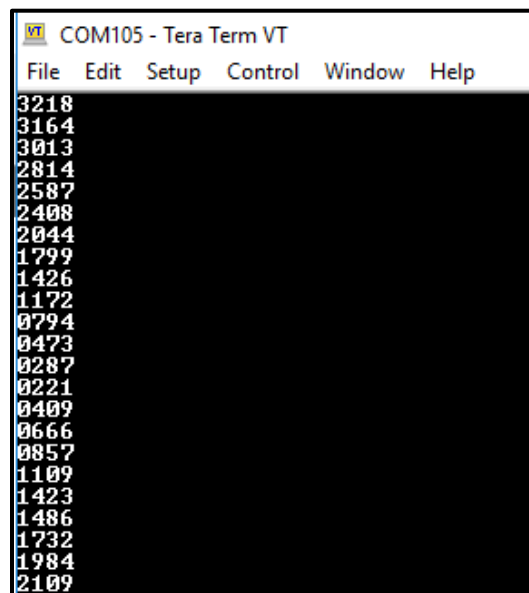


Figure 8.6 Serial Terminal Data Responses

However, there are many improvements that can be made to this hardware design. The FPGA development board has unnecessary peripheral interfaces and resources equipped that could potentially expose vulnerabilities to the design. This design can be improved by integrating the FPGA chip onto the PCB to keep the data acquisition unit tightly coupled to a single board. Lastly, since ICS environments and all their components exist on an ICS network, the final improvement would be modifying the FPGA design to make it an ethernet capable device. This modification would allow IDSs to query data from a secure out-of-band data acquisition source. The FPGA design, referred to as a bitstream, could be stored on a non-volatile memory source on the PCB allowing the FPGA to be loaded with the secure hardware design when the device is powered up. Using this approach, the entire data acquisition system would exist on a single board that requires a single 5V power supply.

CHAPTER 9: CONCLUSION

Industrial Control Systems (ICSs) play a large role in modern civilization, whether it be through the manufacturing of products, or the maintenance of critical infrastructure. ICS and the facilities that house them are complex and interconnected, allowing for sophisticated control of processes that keep civilization moving forward. Due to the significant investments of both capital and intelligence to form ICSs and their facilities, protection from external forces is required. These external forces range from unavoidable erosions of system components over time to the malicious actions of APTs. The fortification of ICS facilities and their internal systems has been an ongoing effort since the conception of the ICS. ICS cybersecurity is a topic that appears limitless in depth, with new developments made every year to increase the strength and resiliency of cybersecurity systems that protect ICS technology.

Over the course of decades, best cybersecurity practices have been established. Each practice aims at addressing an existing attack vector, and together the best practices seek to fortify the entire attack surface of ICSs and the facilities that house them. This thesis covered various best practices, such as the removal of unnecessary features, data integrity, authentication & authorization, secure communication protocols, intrusion detection systems (IDSs), and network enclave. In the face of an abundance of cybersecurity practices, finding the best path forward based on the given application can be intimidating and time-consuming. The MITRE ATT&CK Framework was devised to

give industries a way to perform threat intelligence, threat mitigation, and assessments on their fortifications.

However, despite the intense efforts to protect ICSs and their systems from intrusion, APTs manage to seep through the cracks with near limitless time and money to support their endeavors. In cyber-attacks such as Stuxnet, the ICS community was faced with new challenges because existing cybersecurity practices were inadequate, and the adversary remained undetected even when the attack was taking place. Since cybersecurity is no silver bullet, flaws in defenses will always exist, and a focus on making ICS systems resilient has become an increasingly important topic in recent years.

Addressing the topic of resiliency is an ongoing effort, as the term resiliency is defined differently depending on the given application. Researchers have attempted to define resiliency in many ways, such as cyber resiliency and system resiliency. Within each definition there exists different goals, objectives, and techniques by which resiliency is achieved. Within modern ICS cybersecurity practices, many of these goals and objectives are achieved by the given techniques. However, one challenge the ICS community is currently faced with relates to system visibility and contextual awareness.

Current ICS cybersecurity practices place emphasis on fortifying information technology (IT) around ICSs. Actively scanning for internal threats within ICS environments requires an availability of control status information from operational technology (OT). However, if operational technology has degraded over time, or been infiltrated by an adversary, control status information may be skewed or outright false. Since the protection mechanisms put in place require information from potentially degrading or corrupted sources, a paradox arises.

This thesis investigated the topics of system thinking and data resiliency to give context into what might be done to break this paradox. A mechanism that can provide visibility and contextual awareness to physical control process information is proposed. The introduction of an out-of-band data acquisition system residing at the physical control process level may solve some of the issues relating to system visibility. By acquiring data from an out-of-band channel, a direct view of the system level behavior can be utilized by modern ICS cybersecurity solutions such as IDSs. The introduction of a data acquisition system allows for the verification of in-band sources of ICS data, such as data returned from programmable logic controllers (PLCs). Due to this, trust is not inherently placed in devices that are subject to degradation and infiltration.

This thesis detailed the electrical design of the data acquisition unit. The unit was designed to be applied to different types of control signals (e.g. current-controlled, voltage-controlled, and PWM-controlled). This thesis also demonstrated the software design and system integration of the data acquisition system in a simplified testbed consisting of a water tank control system. The application of the proposed data acquisition unit is up to security engineers in ICS applications.

By integrating the proposed system into modern ICS cybersecurity applications, cyber, system, and data resiliency may be achieved. The cybersecurity application of real-time anomaly-based IDSs may benefit from an out-of-band data source. This out-of-band data source can be a step of verification and validation of data used in artificial intelligence (AI) and machine learning (ML) algorithms within IDSs.

The integration of a data acquisition such as the one proposed in this thesis could bolster security for OT within the physical environment, which is something that is

lacking in modern ICS security. With increased visibility, computer worms such as Stuxnet would have a much more difficult time slipping under the radar as the electrical signals are closely monitored within the physical environment. Malware infections within PLCs that enable data subversion and data poisoning attacks could be stifled and nearly impossible to perform as analog electrical signals cannot be hacked like bits and bytes. In order to combat the seemingly bottomless resources of APTs, it is imperative that the ICS community adopts creativity with its cybersecurity approaches. The idea proposed within this thesis breaches the normal digital approach to solve a digital problem. In doing so, this thesis synthesizes the analog world with the digital world to form a cohesive security posture within the ICS environment.

By itself, the data acquisition unit is just a hunk of silicon and metal. The power of such a unit lies in the hands of the engineers and scientists. For those who wish to bolster the security of the infrastructure that fuels the forward motion of modern civilization, the work within this thesis offers an avenue forward, and hopefully it inspires future developments in the field of ICS security.

REFERENCES

- [1] R. Anderson, J. Benjamin, V. Wright, and L. Quinones, “Cyber-Informed Engineering,” Idaho Falls, Idaho, 2017.
- [2] D. Bhamare, M. Zolanvari, A. Erbad, R. Jain, K. Khan, and N. Meskin, “Cybersecurity for industrial control systems: A survey,” *arXiv*, no. November, 2020.
- [3] M. Bishop, C. Sample, S.M. Loo, “Resilient Data,” *2020 Resil. Week*, pp. 1–10, 2020.
- [4] B. K. Chawla, O. P. Gupta, and B. K. Sawhney, “A Review on IPsec and SSL VPN,” *Int. J. Sci. Eng. Res.*, vol. 5, no. 11, pp. 21–24, 2014.
- [5] D. Dolezilek, D. Gammel, and W. Fernandes, “Complete IEC 61850 Protection and Control System Cybersecurity Is So Much More Than Device Features Based on IEC 62351 and IEC 62443,” *10th Annu. Prot. Autom. Control World Conf.*, no. June, 2019.
- [6] M. Emadadeen, “Process Control in the Chemical Industries, Distributed Control Systems,” pp. 132–145, 2002.
- [7] N. Falliere, L. O. Murchu, and E. Chien, “W32. stuxnet dossier,” *Symantec Secur. Response*, vol. 14, no. February, pp. 1–69, 2011.
- [8] FSabahi and AMovaghar, “Intrusion detection: A survey,” *Proc. - 3rd Int. Conf. Syst. Networks Commun. ICSNC 2008 - Incl. I-CENTRIC 2008 Int. Conf. Adv. Human-Oriented Pers. Mech. Technol. Serv.*, no. May, pp. 23–26, 2008, doi: 10.1109/ICSNC.2008.44.
- [9] C. Hale, “Security in depth,” *ISACA J.*, vol. 3, pp. 45–51, 2018.

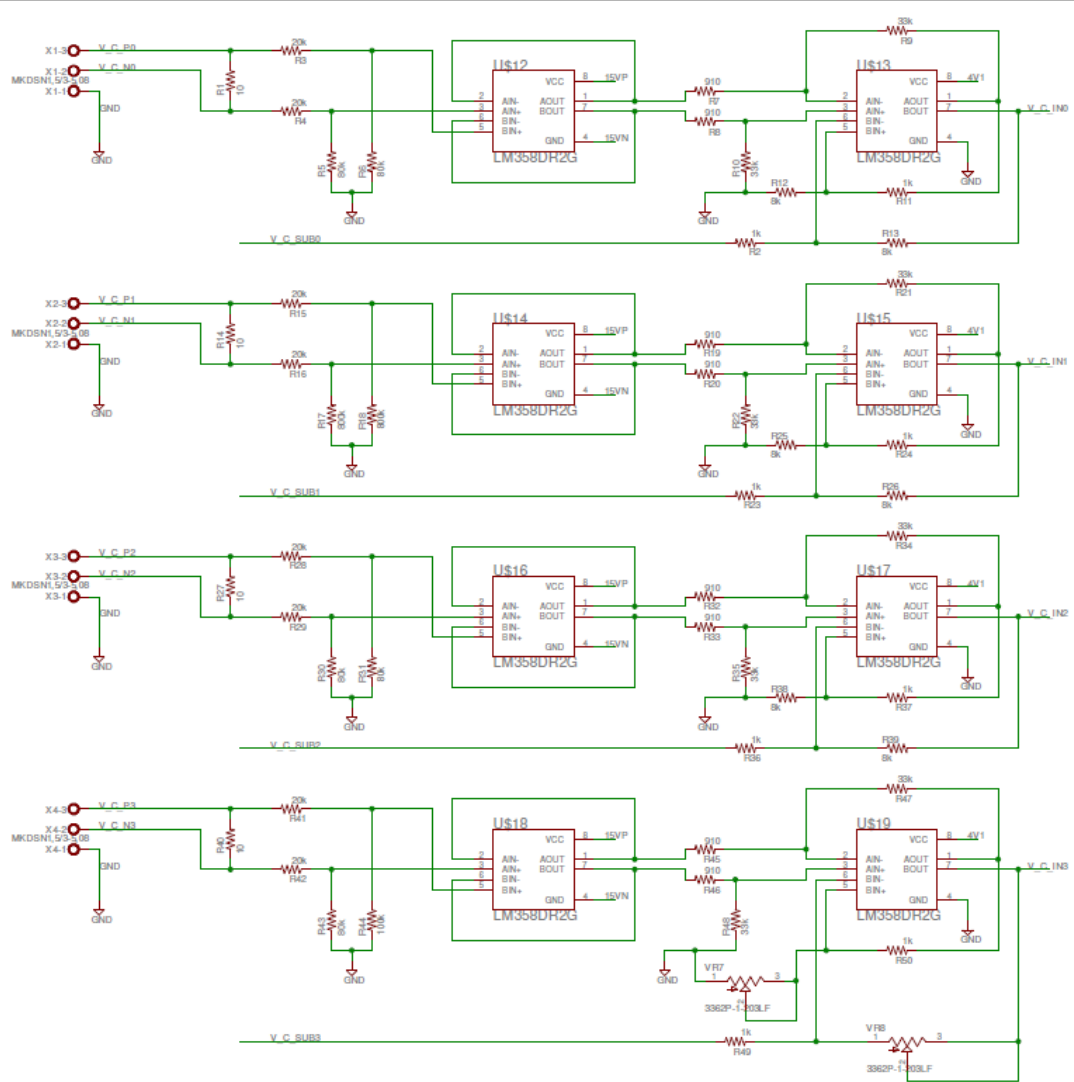
- [10] R. M. Lee, M. J. Assante, and T. Conway, “Analysis of the Cyber Attack on the Ukrainian Power Grid Defense Use Case,” *Electr. Inf. Shar. Anal. Cent.*, p. 36, 2016.
- [11] D. H. Meadows, *Thinking in Systems*. 2009.
- [12] Michael Horkan, “Challenges for IDS / IPS Deployment in Industrial,” *Inf. Secur. Read. Room*, 2015.
- [13] L. Obregon, “Information Security Reading Room Secure Architecture for Industrial Control Systems,” 2020.
- [14] Pacific Northwest National Laboratory, “Secure data transfer guidance for industrial control and SCADA systems,” *Rep. to US Dep. Energy, PNNL-20776*, no. September, 2011.
- [15] A. Pennington, A. Applbaum, K. Nickels, T. Schulz, B. Strom, and J. Wunder, “Getting Started with ATT&CK,” pp. 1–45, 2019.
- [16] T. Salman and R. Jain, “A survey of protocols and standards for internet of things,” *arXiv*, vol. 1, no. 1, 2019, doi: 10.34048/2017.1.f3.
- [17] G. Sivathanu, C. P. Wright, and E. Zadok, “Ensuring data integrity in storage: Techniques and applications,” *StorageSS’05 - Proc. 2005 ACM Work. Storage Secur. Surviv.*, pp. 26–36, 2005, doi: 10.1145/1103780.1103784.
- [18] R. Srivastava, “Survey of Current Network Intrusion Detection Techniques,” vol. 3, no. 6, pp. 27–33, 2013.
- [19] K. Stouffer, J. Falco, and K. Scarfone, “Guide to Industrial Control Systems (ICS) Security Recommendations of the National Institute of Standards and Technology,” *NIST Spec. Publ.*, pp. 1–157, 2007.
- [20] W. Yu, Y. Wang, and L. Song, “A two stage intrusion detection system for industrial control networks based on ethernet/IP,” *Electron.*, vol. 8, no. 12, 2019, doi: 10.3390/electronics8121545.

- [21] S. Zulkarnain, S. Idrus, E. Cherrier, and C. Rosenberger, “A Review on Authentication Methods,” *Aust. J. Basic Appl. Sci.*, no. August 2015, pp. 95–107, 2013.
- [22] Ross, Ron, et al. “Developing Cyber Resilient Systems: A Systems Security Engineering Approach.” *NIST Special Publication 800-160*, vol. 2, Nov. 2019, doi:10.6028/NIST.SP.800-160v2.
- [23] Microchip Corporation, “MCP3201 - 2.7V 12-Bit A/D Converter with SPI Serial Interface,” MCP3201 datasheet, September 1998 [Revised August 2011].
- [24] Dawson, Joel, et al. “Control-Theory-Informed Feature Selection for Detecting Malicious Tampering in Additive Layer Manufacturing Processes” *Proceedings of the 16th International Conference on Cyber Warfare and Security*, February 2021, pp. 55–64 doi:10.34190/IWS.21.036.
- [25] The Big Hack: How China Used a Tiny Chip to Infiltrate U.S. Companies. (n.d.). Retrieved April 26, 2021, from <https://www.bloomberg.com/news/features/2018-10-04/the-big-hack-how-china-used-a-tiny-chip-to-infiltrate-america-s-top-companies>
- [26] Distributed Control System - Basic Elements & Features of DCS. (2020, May 11). Retrieved April 26, 2021, from <https://www.elprocus.com/distributed-control-system-features-and-elements/>
- [27] Analog to digital converter - How ADC Works and Types? (2021, March 09). Retrieved April 26, 2021, from <https://microcontrollerslab.com/analog-to-digital-adc-converter-working/>
- [28] DeTT&CT: Mapping your blue team to MITRE ATT&CK™. (2021, February 22). Retrieved April 26, 2021, from <https://www.mbsecure.nl/blog/2019/5/dettact-mapping-your-blue-team-to-mitre-attack>
- [29] GPIO. (n.d.). Retrieved April 26, 2021, from <https://www.raspberrypi.org/documentation/usage/gpio/>

- [30] Hurd, C., & McCarty, M. (2017, June 12). INL - A Survey of Security Tools for the Industrial Control System Environment. Retrieved April 26, 2021, from <https://www.osti.gov/servlets/purl/1376870>

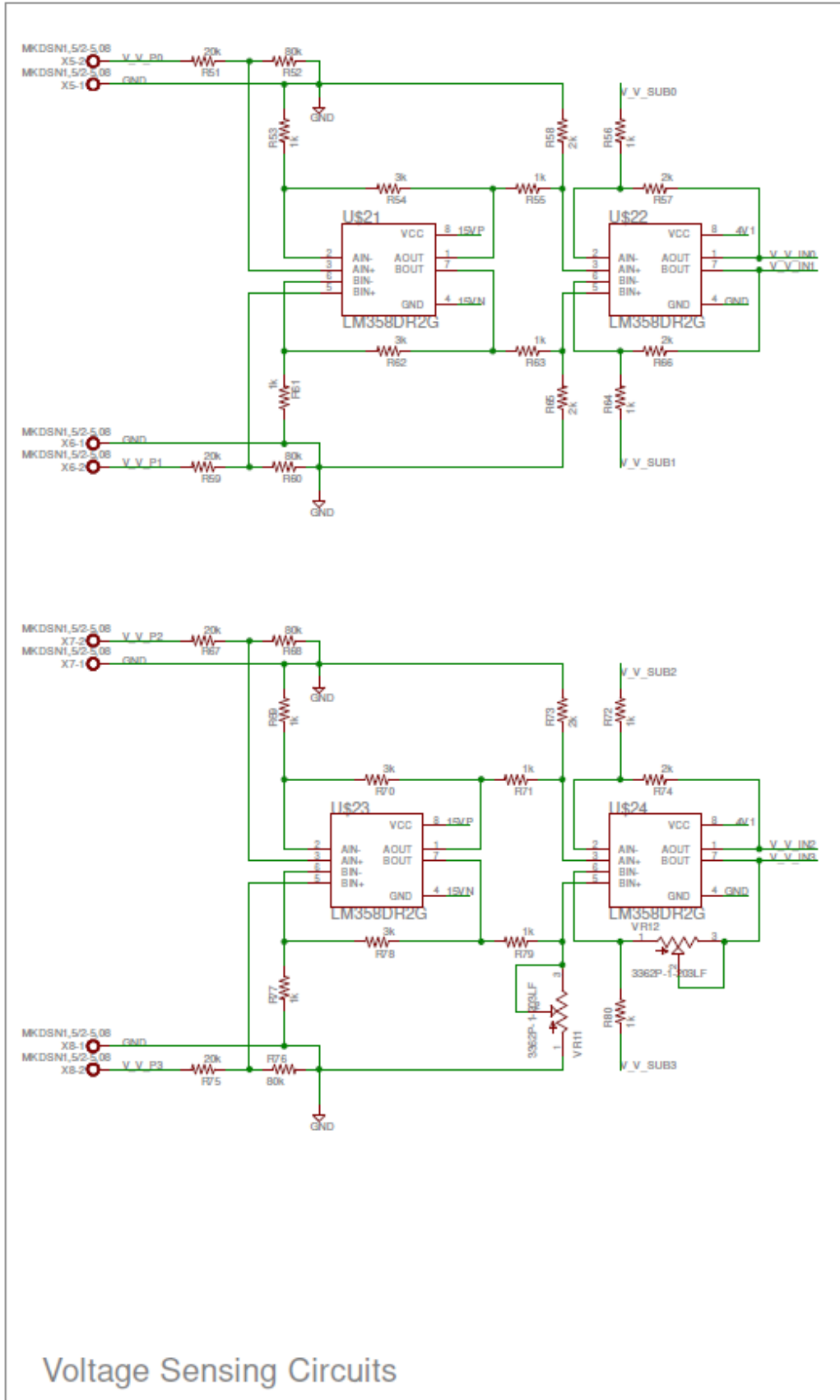
APPENDIX A

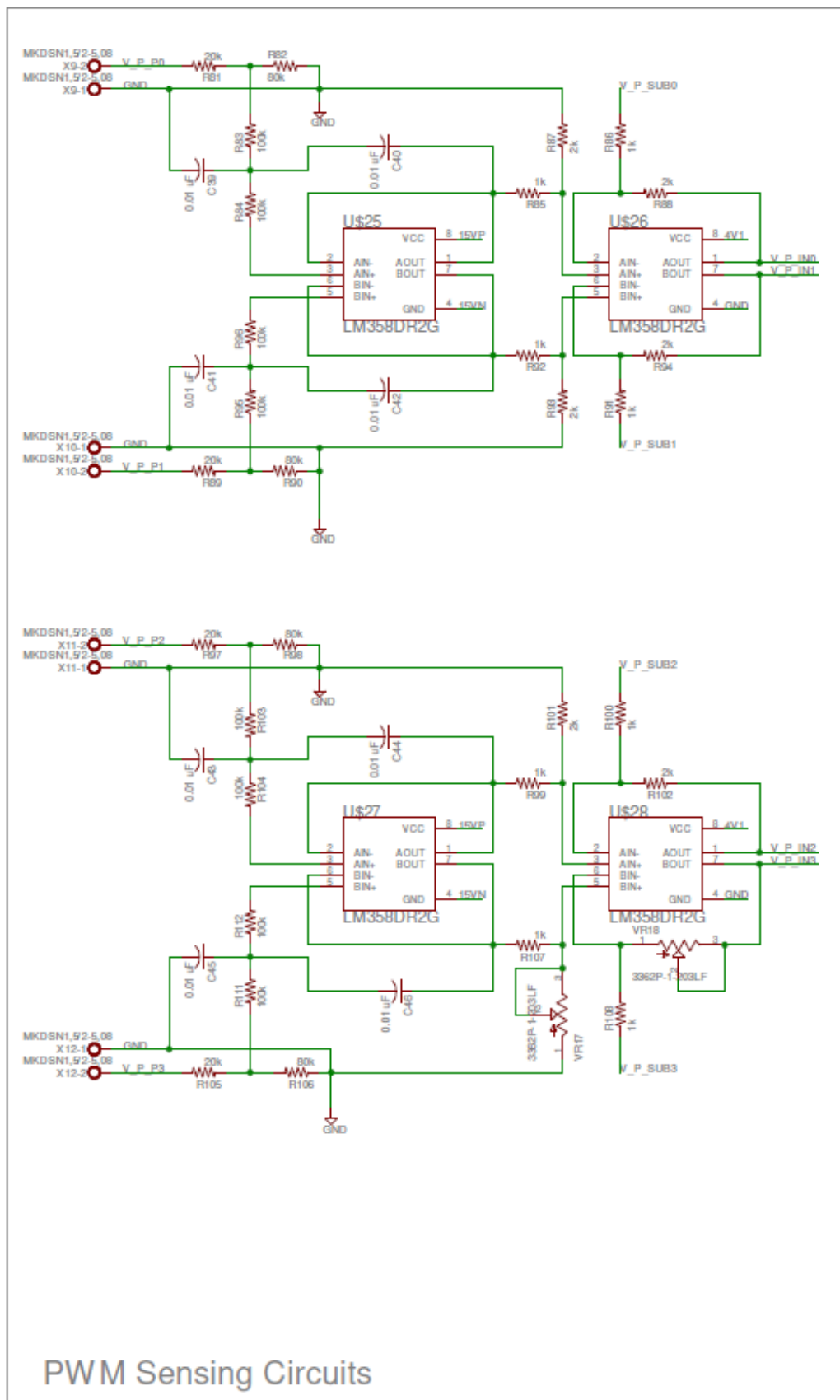
PCB Schematic

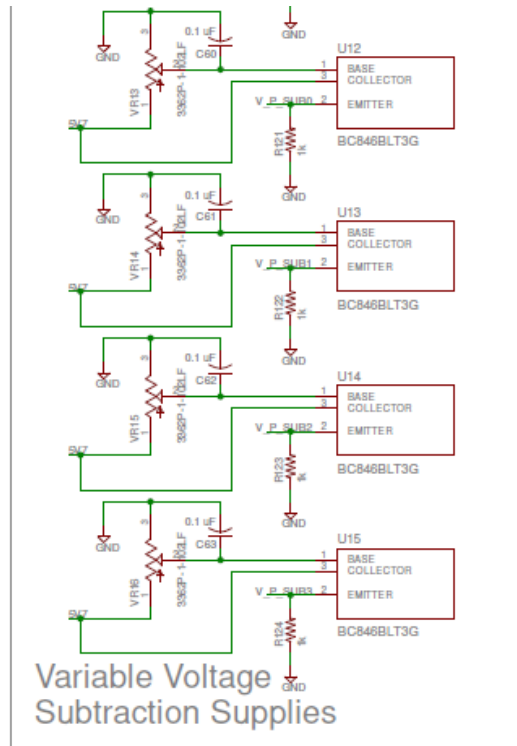
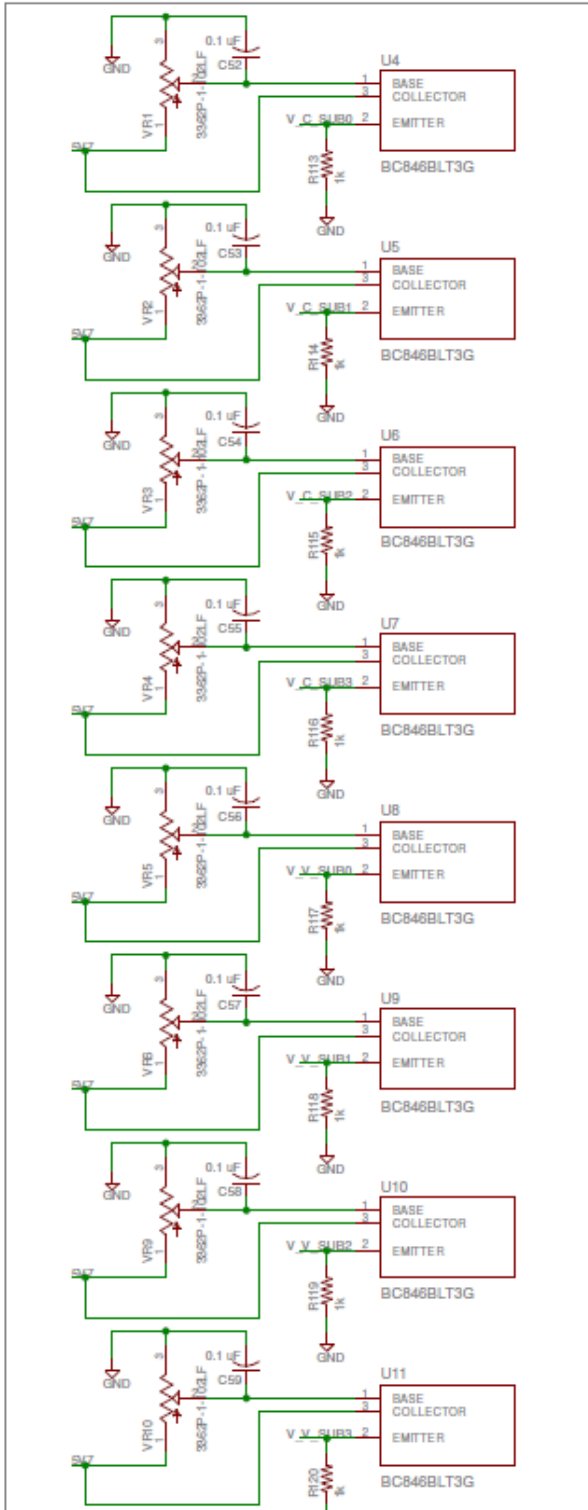


Current Sensing Circuits

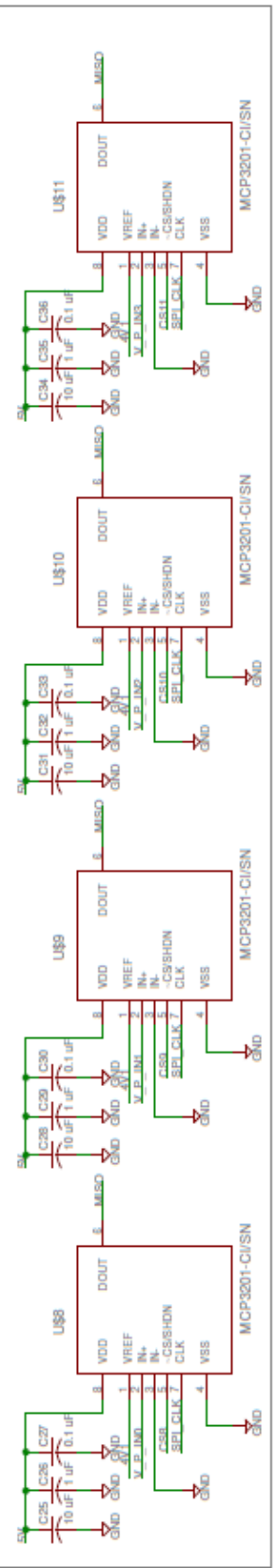
LMV358 is Rail-to-Rail and is used instead of legacy LM358 chips



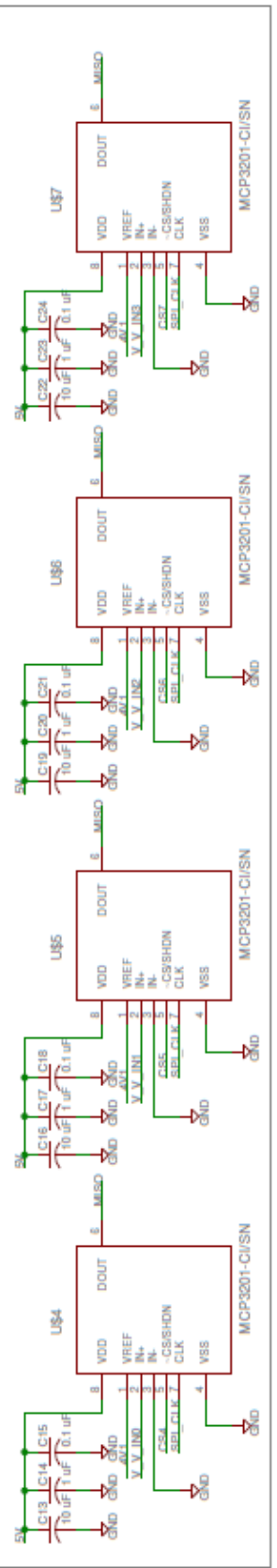




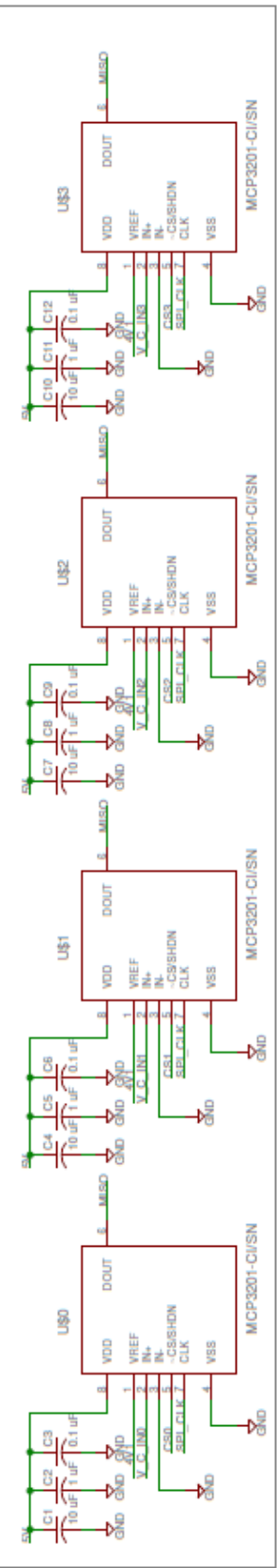
PWM Sensing ADCs



Voltage Sensing ADCs



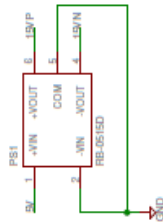
Current Sensing ADCs



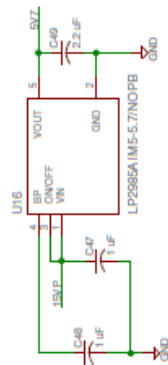
4.096 Voltage Reference



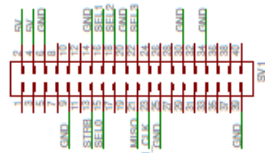
+/- 15V Internal Supplies



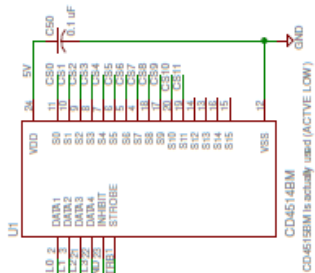
5.7V Internal Supply



GPIO Connector



GPIO Chip Select



APPENDIX B

Source Code

ADC Reader Program

```

/*
 * Description: adc_reader.c
 *             This program reads and returns 12-bit values from 12 ADCs.
 *
 *             The program first initializes the SPI module on the BCM2835
processor on the Raspberry Pi.
 *             The main program loop waits for stdin unsigned integer (0 - 11),
and returns selected ADC
 *             value over stdout.
 *
 * Author:
 *         Daniel Bovard ...      (danielbovard@u.boisestate.edu)
 */

#include "../deps/bcm2835/bcm2835.h"
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <signal.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>

#define SEL0 22
#define SEL1 23
#define SEL2 24
#define SEL3 25
#define STRB 27

/* Signal interrupt declarations */
int terminate = 0;
int status;

/* In/Out pipe declarations */
int fifoFd;
int bytesRead;
const char* adcFifo = "tmp/adcFifo";
char inStr[5], outStr[5];

/* ADC transfer declarations */
uint8_t sel;
uint8_t* buf;
uint16_t value;

/* Function prototypes */
void init_device();
void handle_sigint(int signal_number);
int check_sigint();
void chip_select(uint8_t sel);

int main()
{
    /* Initialize SIGINT (ctrl+c) handler */
    struct sigaction sa;
    memset(&sa, 0, sizeof(struct sigaction));
    sa.sa_handler = handle_sigint;
    sigaction(SIGINT, &sa, NULL);

    /* Initialize device */
    init_device();

    // Allocate memory space for ADC data
    buf = malloc(2*sizeof(uint8_t));

    // Main program loop
    while(1)
    {

```



```

// Clear input buffer
memset(&inStr, 0, sizeof(inStr));

sleep(1);

// Open named FIFO pipe 'adcFifo' in read mode
fifoFd = open(adcFifo, O_RDONLY);
if (fifoFd == -1)
{
    printf("Failed to open adcFifo.\n");
    return -1;
}

do
{
    // Read selection number from input pipe
    bytesRead = read(fifoFd, inStr, sizeof(inStr));

    status = check_sigint();
    if(status) return status;
}
while (bytesRead == 0); // Exit loop when data is read

// Close pipe
close(fifoFd);

// Open named FIFO pipe 'adcFifo' in write mode
fifoFd = open(adcFifo, O_WRONLY);
if (fifoFd == -1)
{
    printf("Failed to open adcFifo.\n");
    return -1;
}

// Select chip
sel = atoi(inStr);
chip_select(sel);

// SPI read from selected ADC
bcm2835_spi_transfern(buf, 2);

// Deassert Chip Select
chip_select(15);

// Extract 12 bits from 16 bit packet
value = (((buf[0] << 8) | (buf[1])) >> 1) & 0xFFFF;
//printf("ADC %d data: %d\n", sel, value);
snprintf((char*)&outStr, sizeof(outStr), "%d", value);

// Write ADC data to output pipe
write(fifoFd, outStr, strlen(outStr)+1);

// Close pipe
close(fifoFd);

status = check_sigint();
if(status) return status;
}
return 1;
}

void init_device()
{
    // Initialize low-level BCM2835 functionality
    if(!bcm2835_init())
    {
        printf("Error initializing BCM2835.\n");
        return;
    }
}

```

```

// Initialize BCM2835 SPI modules
if(!bcm2835_spi_begin())
{
    printf("Error intializing BCM2835 SPI module.\n");
    return;
}

// Configure SPI0 Channel
bcm2835_spi_setBitOrder(BCM2835_SPI_BIT_ORDER_MSBFIRST); // MSB
first is only mode supported by SPI0
bcm2835_spi_setDataMode(BCM2835_SPI_MODE0); //
Rising edge enabled clock, samples at middle of data bit
bcm2835_spi_setClockDivider(BCM2835_SPI_CLOCK_DIVIDER_16384); // 16384 =
15.25878906kHz on Rpi2, 24.4140625kHz on RPI3
bcm2835_spi_chipSelect(BCM2835_SPI_CS0); // Using CS0
as chip select
bcm2835_spi_setChipSelectPolarity(BCM2835_SPI_CS0, 1); // CS0 is
active HIGH

// Init GPIO signals
bcm2835_gpio_fsel(SEL0, BCM2835_GPIO_FSEL_OUTP);
bcm2835_gpio_set_pud(SEL0, BCM2835_GPIO_PUD_UP);
bcm2835_gpio_fsel(SEL1, BCM2835_GPIO_FSEL_OUTP);
bcm2835_gpio_set_pud(SEL1, BCM2835_GPIO_PUD_UP);
bcm2835_gpio_fsel(SEL2, BCM2835_GPIO_FSEL_OUTP);
bcm2835_gpio_set_pud(SEL2, BCM2835_GPIO_PUD_UP);
bcm2835_gpio_fsel(SEL3, BCM2835_GPIO_FSEL_OUTP);
bcm2835_gpio_set_pud(SEL3, BCM2835_GPIO_PUD_UP);
bcm2835_gpio_fsel(STRB, BCM2835_GPIO_FSEL_OUTP);
bcm2835_gpio_set_pud(STRB, BCM2835_GPIO_PUD_UP);
chip_select(15);
}

void chip_select(uint8_t sel)
{
    // Assert GPIO signals per input select
    if (sel & 0x01) bcm2835_gpio_set(SEL0);
    else bcm2835_gpio_clr(SEL0);
    if (sel & 0x02) bcm2835_gpio_set(SEL1);
    else bcm2835_gpio_clr(SEL1);
    if (sel & 0x04) bcm2835_gpio_set(SEL2);
    else bcm2835_gpio_clr(SEL2);
    if (sel & 0x08) bcm2835_gpio_set(SEL3);
    else bcm2835_gpio_clr(SEL3);
    // Generous set up time
    usleep(1);
    // Assert Strobe for input latching
    bcm2835_gpio_set(STRB);
    // Strobe Pulse
    usleep(1);
    // Deassert Strobe for input latching
    bcm2835_gpio_clr(STRB);
}

void handle_sigint(int signal_number)
{
    if (signal_number == SIGINT)
    {
        terminate = 1;
    }
}

int check_sigint()
{
    if (terminate)
    {
        bcm2835_spi_end();
        return 1;
    }
    else return 0;
}

```

PLC Reader Program

```

import socket
import numpy as np
from umodbus import conf
from umodbus.client import tcp
from time import sleep
import csv

PLC_ADDR = '192.168.1.101'
PLC_PORT = 502

conf.SIGNED_VALUES = True
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect((PLC_ADDR, PLC_PORT))

plcFifo = "tmp/plcFifo"

while(True):

    fifoFd = open(plcFifo, 'r')
    plcAddr = fifoFd.read()
    fifoFd.close()

    # Perform modbus request
    try:
        message = tcp.read_input_registers(slave_id=0,
starting_address=int(plcAddr), quantity=1)
        response = tcp.send_message(message, sock)

        # Stdout transfer response packet

        fifoFd = open(plcFifo, 'w')
        plcAddr = fifoFd.write(str(int(response[0])))
        fifoFd.close()

    except (KeyboardInterrupt, SystemExit):
        print()
        sock.close()
        break

```

Control Monitor Process

```

import numpy as np
import os
import csv
import matplotlib.pyplot as plt
import signal
import sys
from time import sleep

SAMPLE_TIME = 5
MINUTE = 60
WINDOW_TIME = MINUTE*60
WINDOW_SIZE = int(WINDOW_TIME/SAMPLE_TIME)
MAX = 4095
PROC_VAL_TOL = int(0.10 * MAX)
A2D_VAL_TOL = int(0.05 * MAX)
PLC_MODE = True

PROCESS_PATH = "data/sinewave_process.csv"
TANK_ADC_ADDR = 0
TANK_PLC_ADDR = 25

adcFifo = "tmp/adcFifo"
plcFifo = "tmp/plcFifo"

adc_tank_list = []
plc_tank_list = []

def signal_handler(sig, frame):
    print("Process terminated.")
    print("Performing cleanup...")

    os.remove("tmp/adcFifo")
    if PLC_MODE:
        os.remove("tmp/plcFifo")

    global adc_tank_list

    adc_tank_csv = np.array(adc_tank_list).reshape(len(adc_tank_list), 1)

    print(adc_tank_csv)

    with open("data/adc_process.csv", 'w') as file:

        writer = csv.writer(file)

        writer.writerows(adc_tank_csv)

        file.close()

    sys.exit(0)

def create_pipes():
    try:
        os.mkfifo(adcFifo, mode=0o666)
    except OSError:
        print("Failed to create ADC FIFO")

    if PLC_MODE:
        try:
            os.mkfifo(plcFifo, mode=0o666)
        except OSError:
            print("Failed to create PLC FIFO")

def read_adc_value(select):

    fifoFd = open(adcFifo, 'w')
    fifoFd.write(str(int(select)))
    fifoFd.close()

```

```

    fifoFd = open(adcFifo, 'r')
    adcVal = fifoFd.read()
    fifoFd.close()

    return int(adcVal.rstrip('\x00'))

def read_plc_value(addr):

    fifoFd = open(plcFifo, 'w')
    fifoFd.write(str(int(addr)))
    fifoFd.close()

    fifoFd = open(plcFifo, 'r')
    plcVal = fifoFd.read()
    fifoFd.close()

    return int(plcVal)

def update_list(val_list, val):
    val_list.append(val)
    if(len(val_list) > WINDOW_SIZE):
        val_list.pop(0)

def linear_fit(y_coords):
    length = len(y_coords)
    x_coords = np.arange(0, length*SAMPLE_TIME, SAMPLE_TIME)
    coeffs = np.polyfit(x_coords, y_coords, 1)
    return coeffs

def plot_samples(fig, axw1, axs2, axs3, y_coords1, y_coords2, y_coords3, lin_fit):
    length = len(y_coords1)
    x_coords = np.arange(0, length*SAMPLE_TIME, SAMPLE_TIME)

    axw1.cla()
    axs2.cla()
    axs3.cla()
    axw1.scatter(x_coords, y_coords1)
    axs2.scatter(x_coords, y_coords2)
    axs3.scatter(x_coords, y_coords3)

    if(lin_fit):
        coeffs1 = linear_fit(y_coords1)
        coeffs2 = linear_fit(y_coords2)
        coeffs3 = linear_fit(y_coords3)

        lin_fit1 = np.zeros(length)
        lin_fit2 = np.zeros(length)
        lin_fit3 = np.zeros(length)

        for i in range(length):
            lin_fit1[i] = coeffs1[1] + i*SAMPLE_TIME*coeffs1[0]
            lin_fit2[i] = coeffs2[1] + i*SAMPLE_TIME*coeffs2[0]
            lin_fit3[i] = coeffs3[1] + i*SAMPLE_TIME*coeffs3[0]

        axw1.plot(x_coords, lin_fit1, 'r')
        axs2.plot(x_coords, lin_fit2, 'r')
        axs3.plot(x_coords, lin_fit3, 'r')

    fig.canvas.draw()

def upload_process(path):
    control_process = []
    i = 0
    with open(path, 'r') as ipc_file:
        ipc_reader = csv.reader(ipc_file, delimiter=',')
        for row in ipc_reader:
            if i != 0:
                control_process.append(int(row[1]))

```

```

        i+=1

    return control_process

def monitor_signals(path):

    fig, axs = plt.subplots(1,1, figsize=(15,7))
    fig.show()
    fig.canvas.draw()

    control_process = upload_process(path)

    print(WINDOW_SIZE)
    j = 0

    while(True):

        update_list(adc_tank_list, read_adc_value(TANK_ADC_ADDR))
        if PLC_MODE:
            update_list(plc_tank_list, read_plc_value(TANK_PLC_ADDR))

        sleep(SAMPLE_TIME)
        # Plot control process
        plot_process(fig, axs, adc_tank_list, plc_tank_list) #,
control_process[j:j+WINDOW_SIZE])

        j += 1

    print("Process detection complete.")
    return

def plot_process(fig, axs, analog, digital): #, expected):
    length = len(analog)

    a2d_low = np.zeros(len(analog))
    a2d_high = np.zeros(len(analog))
    x_coords = np.arange(0, length*SAMPLE_TIME, SAMPLE_TIME)

    for i in range(len(analog)):
        a2d_low[i] = digital[i] - A2D_VAL_TOL
        a2d_high[i] = digital[i] + A2D_VAL_TOL

    axs.cla()
    plt.title("Tank Level Monitor")
    axs.plot(x_coords, analog, 'r', label="Analog from ADC")
    axs.plot(x_coords, digital, 'm', label="Digital from PLC")
    axs.plot(x_coords, a2d_low, 'c--', label="A-D Tolerance Threshold")
    axs.plot(x_coords, a2d_high, 'c--')
    axs.legend()

    fig.canvas.draw()

def main_process():

    create_pipes()

    signal.signal(signal.SIGINT, signal_handler)

    monitor_signals(PROCESS_PATH)

# Main Process
main_process()

```

APPENDIX C

FPGA Design

Data Acquisition Top

```
-----
-- Engineer: Daniel Bovard
--
-- Create Date: 04/11/2021 11:52:01 AM
-- Design Name: Data Acquisition System
-- Module Name: daq_top - rtl
-- Target Devices: XC7A100T-CSG324
-- Tool Versions: Vivado 2019.1
-- Description:
--   Data acquisition unit for IDS ICS Thesis.
--
--   Can receive serial UART packets and parse hexadecimal numbers from them.
--   Based on received number, GPIO pins are strobed to set a 4x16 decoder
--   external to the FPGA. Once GPIO is set, SPI data packets are retrieved
--   from a selected external ADC. Data packet is parsed and sent back over
--   the UART serial connection.
--
-----
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
```

```
entity daq_top is
  port (
    gpio_sel      : out std_logic_vector(3 downto 0);
    gpio_strb     : out std_logic;

    miso          : in  std_logic;
    sclk          : out std_logic;

    rx            : in  std_logic;
    tx            : out std_logic;

    gpio_led      : out std_logic_vector(3 downto 0);
    flash_led     : out std_logic_vector(3 downto 0);

    err          : out std_logic;

    clk           : in  std_logic;
    rst_n        : in  std_logic
  );
end daq_top;
```

```
architecture rtl of daq_top is
```

```
  component daq_controller is
    generic (
      C_AXI_AWIDTH : integer := 4;
      C_AXI_DWIDTH : integer := 32;
      C_SPI_DWIDTH : integer := 16
    );
    port (
      m_axi_uart_araddr : out std_logic_vector(C_AXI_AWIDTH - 1 downto 0);
      m_axi_uart_arready : in  std_logic;
      m_axi_uart_arvalid : out std_logic;
      m_axi_uart_awaddr : out std_logic_vector(C_AXI_AWIDTH - 1 downto 0);
      m_axi_uart_awready : in  std_logic;
      m_axi_uart_awvalid : out std_logic;
      m_axi_uart_bresp  : in  std_logic_vector(1 downto 0);
      m_axi_uart_bready : out std_logic;
      m_axi_uart_bvalid : in  std_logic;
      m_axi_uart_rdata  : in  std_logic_vector(C_AXI_DWIDTH - 1 downto 0);
      m_axi_uart_rready : out std_logic;
      m_axi_uart_rresp  : in  std_logic_vector(1 downto 0);
      m_axi_uart_rvalid : in  std_logic;
      m_axi_uart_wdata  : out std_logic_vector(C_AXI_DWIDTH - 1 downto 0);
      m_axi_uart_wready : in  std_logic;
```



```

m_axi_uart_wstrb : out std_logic_vector((C_AXI_DWIDTH/8) - 1 downto 0);
m_axi_uart_wvalid : out std_logic;

gpio_sel        : out std_logic_vector(3 downto 0);
gpio_strb       : out std_logic;

spi_enable      : out std_logic;
spi_busy       : in  std_logic;
spi_rxdata     : in  std_logic_vector(C_SPI_DWIDTH - 1 downto 0);

err             : out std_logic;

clk             : in  std_logic;
rst_n          : in  std_logic
);
end component;

component spi_master is
generic(
  slaves : integer := 4; --number of spi slaves
  d_width : integer := 2); --data bus width
port(
  clock      : IN      STD_LOGIC;           --system clock
  reset_n    : IN      STD_LOGIC;          --asynchronous reset
  enable     : IN      STD_LOGIC;          --initiate transaction
  cpol      : IN      STD_LOGIC;          --spi clock polarity
  cpha      : IN      STD_LOGIC;          --spi clock phase
  clk_div    : IN      INTEGER;           --system clock cycles
  per 1/2 period of sclk
  miso       : IN      STD_LOGIC;          --master in, slave out
  sclk       : OUT     STD_LOGIC;          --spi clock
  busy       : OUT     STD_LOGIC;          --busy / data ready
signal
  rx_data : OUT     STD_LOGIC_VECTOR(d_width-1 DOWNT0 0)); --data received
end component;

component axi_uartlite_0
port (
  s_axi_aclk : in std_logic;
  s_axi_aresetn : in std_logic;
  interrupt : out std_logic;
  s_axi_awaddr : in std_logic_vector(3 downto 0);
  s_axi_awvalid : in std_logic;
  s_axi_awready : out std_logic;
  s_axi_wdata : in std_logic_vector(31 downto 0);
  s_axi_wstrb : in std_logic_vector(3 downto 0);
  s_axi_wvalid : in std_logic;
  s_axi_wready : out std_logic;
  s_axi_bresp : out std_logic_vector(1 downto 0);
  s_axi_bvalid : out std_logic;
  s_axi_bready : in std_logic;
  s_axi_araddr : in std_logic_vector(3 downto 0);
  s_axi_arvalid : in std_logic;
  s_axi_arready : out std_logic;
  s_axi_rdata : out std_logic_vector(31 downto 0);
  s_axi_rresp : out std_logic_vector(1 downto 0);
  s_axi_rvalid : out std_logic;
  s_axi_rready : in std_logic;
  rx : in std_logic;
  tx : out std_logic
);
end component;

component BUFG is
port (
  I : in std_logic;
  O : out std_logic
);
end component;

component OBUF is

```

```

port (
    I : in std_logic;
    O : out std_logic
);
end component;

constant C_AXI_AWIDTH : integer := 4;
constant C_AXI_DWIDTH : integer := 32;
constant C_SPI_DWIDTH : integer := 16;

signal m_axi_uart_araddr  : std_logic_vector(C_AXI_AWIDTH - 1 downto 0);
signal m_axi_uart_arready : std_logic;
signal m_axi_uart_arvalid : std_logic;
signal m_axi_uart_awaddr  : std_logic_vector(C_AXI_AWIDTH - 1 downto 0);
signal m_axi_uart_awready : std_logic;
signal m_axi_uart_awvalid : std_logic;
signal m_axi_uart_bresp   : std_logic_vector(1 downto 0);
signal m_axi_uart_bready  : std_logic;
signal m_axi_uart_bvalid  : std_logic;
signal m_axi_uart_rdata   : std_logic_vector(C_AXI_DWIDTH - 1 downto 0);
signal m_axi_uart_rready  : std_logic;
signal m_axi_uart_rresp   : std_logic_vector(1 downto 0);
signal m_axi_uart_rvalid  : std_logic;
signal m_axi_uart_wdata   : std_logic_vector(C_AXI_DWIDTH - 1 downto 0);
signal m_axi_uart_wready  : std_logic;
signal m_axi_uart_wstrb   : std_logic_vector((C_AXI_DWIDTH/8) - 1 downto 0);
signal m_axi_uart_wvalid  : std_logic;

signal spi_enable         : std_logic;
signal spi_busy          : std_logic;
signal spi_rxdata        : std_logic_vector(C_SPI_DWIDTH - 1 downto 0);

signal gpio_sel_i : std_logic_vector(3 downto 0);
signal flash_count : unsigned(31 downto 0) := (others => '0');
signal flash_led_i : std_logic_vector(3 downto 0);

signal sclk_i : std_logic;

begin

flash_led_p : process (clk)
begin
    if (rising_edge(clk)) then
        flash_count <= flash_count + 1;
    end if;
end process;

flash_led(3) <= flash_count(26);
flash_led(2) <= flash_count(25);
flash_led(1) <= flash_count(24);
flash_led(0) <= flash_count(23);

daq_controller_i : daq_controller
generic map (
    C_AXI_AWIDTH => C_AXI_AWIDTH,
    C_AXI_DWIDTH => C_AXI_DWIDTH,
    C_SPI_DWIDTH => C_SPI_DWIDTH
)
port map(
    m_axi_uart_araddr => m_axi_uart_araddr,
    m_axi_uart_arready => m_axi_uart_arready,
    m_axi_uart_arvalid => m_axi_uart_arvalid,
    m_axi_uart_awaddr => m_axi_uart_awaddr,
    m_axi_uart_awready => m_axi_uart_awready,
    m_axi_uart_awvalid => m_axi_uart_awvalid,
    m_axi_uart_bresp => m_axi_uart_bresp,
    m_axi_uart_bready => m_axi_uart_bready,
    m_axi_uart_bvalid => m_axi_uart_bvalid,
    m_axi_uart_rdata => m_axi_uart_rdata,
    m_axi_uart_rready => m_axi_uart_rready,
    m_axi_uart_rresp => m_axi_uart_rresp,

```

```

    m_axi_uart_rvalid => m_axi_uart_rvalid,
    m_axi_uart_wdata  => m_axi_uart_wdata,
    m_axi_uart_wready => m_axi_uart_wready,
    m_axi_uart_wstrb  => m_axi_uart_wstrb,
    m_axi_uart_wvalid => m_axi_uart_wvalid,

    gpio_sel          => gpio_sel_i,
    gpio_strb         => gpio_strb,

    spi_enable        => spi_enable,
    spi_busy          => spi_busy,
    spi_rxddata       => spi_rxddata,

    err               => err,

    clk               => clk,
    rst_n             => rst_n
);

spi_master_i : spi_master
generic map (
    slaves => 1,
    d_width => 16
)
port map (
    clock    => clk,
    reset_n  => '1',
    enable   => spi_enable,
    cpol     => '0',
    cpha     => '0',
    clk_div  => 100,
    miso     => miso,
    sclk     => sclk_i,
    busy     => spi_busy,
    rx_data  => spi_rxddata
);

axi_uartlite_0_i : axi_uartlite_0
PORT MAP (
    s_axi_aclk    => clk,
    s_axi_aresetn => rst_n,
    interrupt     => open,
    s_axi_awaddr  => m_axi_uart_awaddr,
    s_axi_awvalid => m_axi_uart_awvalid,
    s_axi_awready => m_axi_uart_awready,
    s_axi_wdata   => m_axi_uart_wdata,
    s_axi_wstrb   => m_axi_uart_wstrb,
    s_axi_wvalid  => m_axi_uart_wvalid,
    s_axi_wready  => m_axi_uart_wready,
    s_axi_bresp   => m_axi_uart_bresp,
    s_axi_bvalid  => m_axi_uart_bvalid,
    s_axi_bready  => m_axi_uart_bready,
    s_axi_araddr  => m_axi_uart_araddr,
    s_axi_arvalid => m_axi_uart_arvalid,
    s_axi_arready => m_axi_uart_arready,
    s_axi_rdata   => m_axi_uart_rdata,
    s_axi_rresp   => m_axi_uart_rresp,
    s_axi_rvalid  => m_axi_uart_rvalid,
    s_axi_rready  => m_axi_uart_rready,
    rx            => rx,
    tx            => tx
);

gpio_led0_obuf_i : OBUF
port map (
    I => gpio_sel_i(0),
    O => gpio_led(0)
);

gpio_led1_obuf_i : OBUF
port map (

```

```
    I => gpio_sel_i(1),
    O => gpio_led(1)
);

gpio_led2_obuf_i : OBUF
port map (
    I => gpio_sel_i(2),
    O => gpio_led(2)
);

gpio_led3_obuf_i : OBUF
port map (
    I => gpio_sel_i(3),
    O => gpio_led(3)
);

gpio_sel0_obuf_i : OBUF
port map (
    I => gpio_sel_i(0),
    O => gpio_sel(0)
);

gpio_sel1_obuf_i : OBUF
port map (
    I => gpio_sel_i(1),
    O => gpio_sel(1)
);

gpio_sel2_obuf_i : OBUF
port map (
    I => gpio_sel_i(2),
    O => gpio_sel(2)
);

gpio_sel3_obuf_i : OBUF
port map (
    I => gpio_sel_i(3),
    O => gpio_sel(3)
);

sclk_obuf_i : OBUF
port map (
    I => sclk_i,
    O => sclk
);
end rtl;
```

Data Acquisition Controller

```

-----
-- Engineer: Daniel Bovard
--
-- Create Date: 04/11/2021 11:52:01 AM
-- Design Name: Data Acquisition System
-- Module Name: daq_controller - rtl
-- Target Devices: XC7A100T-CSG324
-- Tool Versions: Vivado 2019.1
-- Description:
--   Data acquisition controller for IDS ICS Thesis.
--
--   The controller orchestrates the entire data acquisition process.
--   First the controller polls the Xilinx AXI UARTLite LogiCORE IP
--   for valid data packets. From there it parses the packets, asserts
--   the GPIO, retrieves the SPI data packet, and returns the data
--   back to the PC via the UART LogiCORE IP.
--
-----

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity daq_controller is
  generic (
    C_AXI_AWIDTH : integer := 4;
    C_AXI_DWIDTH : integer := 32;
    C_SPI_DWIDTH : integer := 16
  );
  port (
    m_axi_uart_araddr : out std_logic_vector(C_AXI_AWIDTH - 1 downto 0);
    m_axi_uart_arready : in std_logic;
    m_axi_uart_arvalid : out std_logic;
    m_axi_uart_awaddr : out std_logic_vector(C_AXI_AWIDTH - 1 downto 0);
    m_axi_uart_awready : in std_logic;
    m_axi_uart_awvalid : out std_logic;
    m_axi_uart_bresp : in std_logic_vector(1 downto 0);
    m_axi_uart_bready : out std_logic;
    m_axi_uart_bvalid : in std_logic;
    m_axi_uart_rdata : in std_logic_vector(C_AXI_DWIDTH - 1 downto 0);
    m_axi_uart_rready : out std_logic;
    m_axi_uart_rresp : in std_logic_vector(1 downto 0);
    m_axi_uart_rvalid : in std_logic;
    m_axi_uart_wdata : out std_logic_vector(C_AXI_DWIDTH - 1 downto 0);
    m_axi_uart_wready : in std_logic;
    m_axi_uart_wstrb : out std_logic_vector((C_AXI_DWIDTH/8) - 1 downto 0);
    m_axi_uart_wvalid : out std_logic;

    gpio_sel : out std_logic_vector(3 downto 0);
    gpio_strb : out std_logic;

    spi_enable : out std_logic;
    spi_busy : in std_logic;
    spi_rxdata : in std_logic_vector(C_SPI_DWIDTH - 1 downto 0);

    err : out std_logic;

    clk : in std_logic;
    rst_n : in std_logic
  );
end daq_controller;

architecture rtl of daq_controller is

  constant C_UART_RX_ADDR : std_logic_vector(C_AXI_AWIDTH - 1 downto 0) := "0000";
  constant C_UART_TX_ADDR : std_logic_vector(C_AXI_AWIDTH - 1 downto 0) := "0100";
  constant C_UART_SR_ADDR : std_logic_vector(C_AXI_AWIDTH - 1 downto 0) := "1000";

```

```

constant C_UART_CR_ADDR : std_logic_vector(C_AXI_AWIDTH - 1 downto 0) := "1100";

-- ASCII to Hex parser for numbers 0 through F
function parse_rx_uart (data: std_logic_vector(7 downto 0)) return
std_logic_vector is
begin
  case data is
    when x"30" => return x"00";
    when x"31" => return x"01";
    when x"32" => return x"02";
    when x"33" => return x"03";
    when x"34" => return x"04";
    when x"35" => return x"05";
    when x"36" => return x"06";
    when x"37" => return x"07";
    when x"38" => return x"08";
    when x"39" => return x"09";
    when x"41" => return x"0A";
    when x"61" => return x"0A";
    when x"42" => return x"0B";
    when x"62" => return x"0B";
    when x"43" => return x"0C";
    when x"63" => return x"0C";
    when x"44" => return x"0D";
    when x"64" => return x"0D";
    when x"45" => return x"0E";
    when x"65" => return x"0E";
    when x"46" => return x"0F";
    when x"66" => return x"0F";
    when others => return x"FF";
  end case;
end parse_rx_uart;

type digits_t is
  record
    d3 : std_logic_vector(3 downto 0);
    d2 : std_logic_vector(3 downto 0);
    d1 : std_logic_vector(3 downto 0);
    d0 : std_logic_vector(3 downto 0);
  end record;

function parse_digits (data: std_logic_vector(11 downto 0)) return digits_t is
  variable remainder : integer;
  variable temp      : integer;
  variable digits    : digits_t;
begin
  temp := to_integer(unsigned(data)) / 1000;
  remainder := to_integer(unsigned(data)) mod 1000;
  digits.d3 := std_logic_vector(to_unsigned(temp, digits.d3'length));
  temp := remainder / 100;
  remainder := remainder mod 100;
  digits.d2 := std_logic_vector(to_unsigned(temp, digits.d2'length));
  temp := remainder / 10;
  remainder := remainder mod 10;
  digits.d1 := std_logic_vector(to_unsigned(temp, digits.d1'length));
  digits.d0 := std_logic_vector(to_unsigned(remainder, digits.d0'length));

  return digits;
end parse_digits;

type state_t is (INIT,
  CLR_RX,
  CLR_TX,
  RX_POLL, -- Poll for valid Rx Data
  RX_VALID,
  RX_READ,
  RX_DATA,
  GPIO_SET,
  GPIO_STROBE_H,
  GPIO_STROBE_L,
  SPI_READ,

```

```

        SPI_WAIT,
        TX_DATA,
        ERROR);

type reg_t is
    record
        state      : state_t;
        uart_rx_data : std_logic_vector(7 downto 0);
        gpio_clr   : boolean;
        digit_count : integer;
        strobe_count : integer;
        aw         : boolean;
        w         : boolean;
        b         : boolean;
    end record;

constant INIT_REG : reg_t := (state => INIT,
                              uart_rx_data => (others => '0'),
                              gpio_clr => false,
                              digit_count => 5,
                              strobe_count => 0,
                              aw => false,
                              w => false,
                              b => false);

signal r, r_in : reg_t := INIT_REG;
signal gpio_sel_i : std_logic_vector(3 downto 0);
signal digits : digits_t;

begin

m_axi_uart_wstrb <= (others => '1');

gpio_sel <= gpio_sel_i;

comb_p : process ( m_axi_uart_arready, --
                  m_axi_uart_awready,--
                  m_axi_uart_bvalid,--
                  m_axi_uart_rvalid,--
                  m_axi_uart_wready,--
                  spi_busy,
                  rst_n,
                  r) is
    --
    variable v : reg_t;
begin

    -- Output defaults
    v := r;
    m_axi_uart_araddr <= (others => '0');
    m_axi_uart_arvalid <= '0';
    m_axi_uart_awaddr <= (others => '0');
    m_axi_uart_awvalid <= '0';
    m_axi_uart_bready <= '0';
    m_axi_uart_rready <= '0';
    m_axi_uart_wdata <= (others => '0');
    m_axi_uart_wvalid <= '0';
    gpio_strb <= '0';
    spi_enable <= '0';
    err <= '0';
    gpio_sel_i <= gpio_sel_i;

    case r.state is
        when INIT =>
            v := INIT_REG;
            v.state := CLR_RX;

        when CLR_RX =>
            m_axi_uart_awaddr <= C_UART_CR_ADDR;
            m_axi_uart_wdata <= x"00000002"; -- Rst RX FIFO

            if (r.aw = false) then

```

```

    m_axi_uart_awvalid <= '1';
end if;
if (r.w = false) then
    m_axi_uart_wvalid <= '1';
end if;
if (r.b = false) then
    m_axi_uart_bready <= '1';
end if;

if (m_axi_uart_awready = '1') then
    v.aw := true;
end if;
if (m_axi_uart_wready = '1') then
    v.w := true;
end if;
if (m_axi_uart_bvalid = '1') then
    v.b := true;
    if (m_axi_uart_bresp /= "00") then
        v.state := ERROR;
    end if;
end if;

if ((r.aw = true) and (r.w = true) and (r.b = true)) then
    v.state := CLR_TX;
    v.aw := false;
    v.w := false;
    v.b := false;
end if;

when CLR_TX =>
    m_axi_uart_awaddr <= C_UART_CR_ADDR;
    m_axi_uart_wdata <= x"00000001"; -- Rst TX FIFO

    if (r.aw = false) then
        m_axi_uart_awvalid <= '1';
    end if;
    if (r.w = false) then
        m_axi_uart_wvalid <= '1';
    end if;
    if (r.b = false) then
        m_axi_uart_bready <= '1';
    end if;

    if (m_axi_uart_awready = '1') then
        v.aw := true;
    end if;
    if (m_axi_uart_wready = '1') then
        v.w := true;
    end if;
    if (m_axi_uart_bvalid = '1') then
        v.b := true;
        if (m_axi_uart_bresp /= "00") then
            v.state := ERROR;
        end if;
    end if;

    if ((r.aw = true) and (r.w = true) and (r.b = true)) then
        v.state := RX_POLL;
        v.aw := false;
        v.w := false;
        v.b := false;
    end if;

when RX_POLL =>
    m_axi_uart_araddr <= C_UART_SR_ADDR;
    m_axi_uart_arvalid <= '1';
    if (m_axi_uart_arready = '1') then
        v.state := RX_VALID;
    end if;

when RX_VALID =>

```



```

m_axi_uart_rready <= '1';
if (m_axi_uart_rvalid = '1') then
  if (m_axi_uart_rresp /= "00") then
    v.state := ERROR;
  elsif (m_axi_uart_rdata(7 downto 4) /= "0000") then
    v.state := ERROR;
  elsif (m_axi_uart_rdata(0) = '1') then
    v.state := RX_READ;
  else
    v.state := RX_POLL;
  end if;
end if;

when RX_READ =>
  m_axi_uart_araddr <= C_UART_RX_ADDR;
  m_axi_uart_arvalid <= '1';
  if (m_axi_uart_arready = '1') then
    v.state := RX_DATA;
  end if;

when RX_DATA =>
  m_axi_uart_rready <= '1';
  if (m_axi_uart_rvalid = '1') then
    v.uart_rx_data := parse_rx_uart(m_axi_uart_rdata(7 downto 0));
    v.state := GPIO_SET;
  end if;

when GPIO_SET =>
  if (r.gpio_clr = false) then
    gpio_sel_i <= x"F";
  else
    gpio_sel_i <= r.uart_rx_data(3 downto 0);
  end if;

  if(r.strobe_count /= 50) then -- let gpio signals settle
    v.strobe_count := r.strobe_count + 1;
  else
    v.strobe_count := 0;
    v.state := GPIO_STROBE_H;
  end if;

  when GPIO_STROBE_H =>
    gpio_strb <= '1';
    -- Decoder chip requires a strobe of at least 250ns (25 clks) in width
    if(r.strobe_count /= 50) then
      v.strobe_count := r.strobe_count + 1;
    else
      v.strobe_count := 0;
      v.state := GPIO_STROBE_L;
    end if;

when GPIO_STROBE_L =>

  if(r.strobe_count /= 50) then
    v.strobe_count := r.strobe_count + 1;
  else
    v.strobe_count := 0;
    if (r.gpio_clr = false) then
      v.gpio_clr := true;
      v.state := GPIO_SET;
    else
      v.gpio_clr := false;
      v.state := SPI_READ;
    end if;
  end if;

when SPI_READ =>
  spi_enable <= '1';
  if (spi_busy = '1') then
    v.state := SPI_WAIT;
  end if;

```

```

end if;

when SPI_WAIT =>
  if (spi_busy = '0') then
    v.state := TX_DATA;
    digits <= parse_digits(spi_rxddata(12 downto 1));
  end if;

when TX_DATA =>
  m_axi_uart_awaddr <= C_UART_TX_ADDR;
  if (r.digit_count = 5) then
    m_axi_uart_wdata(7 downto 0) <= x"3" & digits.d3;
  elsif (r.digit_count = 4) then
    m_axi_uart_wdata(7 downto 0) <= x"3" & digits.d2;
  elsif (r.digit_count = 3) then
    m_axi_uart_wdata(7 downto 0) <= x"3" & digits.d1;
  elsif (r.digit_count = 2) then
    m_axi_uart_wdata(7 downto 0) <= x"3" & digits.d0;
  elsif (r.digit_count = 1) then
    m_axi_uart_wdata(7 downto 0) <= x"0A"; -- Newline
  else
    m_axi_uart_wdata(7 downto 0) <= x"0D"; -- Carriage Return
  end if;

  if (r.aw = false) then
    m_axi_uart_awvalid <= '1';
  end if;
  if (r.w = false) then
    m_axi_uart_wvalid <= '1';
  end if;
  if (r.b = false) then
    m_axi_uart_bready <= '1';
  end if;
  if (m_axi_uart_awready = '1') then
    v.aw := true;
  end if;
  if (m_axi_uart_wready = '1') then
    v.w := true;
  end if;
  if (m_axi_uart_bvalid = '1') then
    v.b := true;
    if (m_axi_uart_bresp /= "00") then
      v.state := ERROR;
    elsif (r.digit_count /= 0) then
      v.digit_count := r.digit_count - 1;
    else
      v.state := RX_POLL;
      v.digit_count := 5;
    end if;
  end if;

  if ((r.aw = true) and (r.w = true) and (r.b = true)) then
    v.aw := false;
    v.w := false;
    v.b := false;
  end if;

when ERROR =>
  v.state := ERROR;
  err <= '1';

when others =>
  v.state := INIT;

end case;

if (rst_n = '0') then
  v.state := INIT;
end if;

r_in <= v;

```

```
end process;

sync_p : process(clk)
begin
    if (rising_edge(clk)) then
        r <= r_in;
    end if;
end process;

end rtl;
```

SPI Master

```

-----
-- Engineer: Daniel Bovard
--
-- Create Date: 04/11/2021 11:52:01 AM
-- Design Name: Data Acquisition System
-- Module Name: spi_master - rtl
-- Target Devices: XC7A100T-CSG324
-- Tool Versions: Vivado 2019.1
-- Description:
--     SPI Master module for IDS ICS Thesis.
--     This module was derived from the open source model provided by Digikey:
--     https://forum.digikey.com/t/spi-master-vhdl/12717
--
--     This SPI Master is responsible for providing a slave with an SPI SCLK
--     and retrieving a data packet over the MISO line. Since slave selecting
--     is performed via an external 4x16 decoder, the slave select functionality
--     was removed from this module.
-----

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity spi_master is
    generic(
        slaves : integer := 4; --number of spi slaves
        d_width : integer := 2); --data bus width
    port(
        clock : in      std_logic;           --system clock
        reset_n : in    std_logic;          --asynchronous reset
        enable : in     std_logic;          --initiate transaction
        cpol : in      std_logic;           --spi clock polarity
        cpha : in      std_logic;           --spi clock phase
        clk_div : in    integer;            --system clock cycles
per 1/2 period of sclk
        miso : in      std_logic;           --master in, slave out
        sclk : out     std_logic;           --spi clock
        busy : out     std_logic;           --busy / data ready
    signal
        rx_data : out  std_logic_vector(d_width-1 downto 0)); --data received
    end spi_master;

    architecture logic of spi_master is
        type machine is (ready, execute);           --state machine data
    type
        signal state : machine;                     --current state
        signal clk_ratio : integer;                 --current clk_div
        signal count : integer;                     --counter to trigger
    sclk from system clock
        signal clk_toggles : integer range 0 to d_width*2 + 1; --count spi clock
    toggles
        signal rx_buffer : std_logic_vector(d_width-1 downto 0); --receive data buffer
        signal last_bit_rx : integer range 0 to d_width*2;      --last rx data bit
    location
        signal sclk_i : std_logic;

    begin

        sclk <= sclk_i;

        process(clock, reset_n)
        begin

            if(reset_n = '0') then                --reset system
                busy <= '1';                       --set busy signal
                rx_data <= (others => '0'); --clear receive data port

```

```

state <= ready;          --go to ready state when reset is exited

elsif(clock'event and clock = '1') then
  case state is          --state machine

    when ready =>
      busy <= '0';          --clock out not busy signal
      rx_buffer <= (others => '0');
      if(clk_div = 0) then  --check for valid spi speed
        clk_ratio <= 1;    --set to maximum speed if zero
        count <= 1;       --initiate system-to-spi clock counter
      else
        clk_ratio <= clk_div; --set to input selection if valid
        count <= clk_div;  --initiate system-to-spi clock counter
      end if;
      sclk_i <= cpol;      --set spi clock polarity
      clk_toggles <= 0;   --initiate clock toggle counter
      last_bit_rx <= d_width*2 + conv_integer(cpha) - 1; --set last rx data

      bit

      --user input to initiate transaction
      if(enable = '1') then
        busy <= '1';      --set busy signal
        state <= execute; --proceed to execute state
      else
        state <= ready;  --remain in ready state
      end if;

    when execute =>
      busy <= '1';      --set busy signal

      --system clock to sclk ratio is met
      if(count = clk_ratio) then
        count <= 1;          --reset system-to-spi clock counter
        if(clk_toggles = d_width*2 + 1) then
          clk_toggles <= 0;  --reset spi clock toggles counter
        else
          clk_toggles <= clk_toggles + 1; --increment spi clock toggles

          counter

        end if;

        --spi clock toggle needed
        if(clk_toggles <= d_width*2) then
          if (sclk_i = '0') then
            sclk_i <= '1';
          else
            sclk_i <= '0';
          end if;
        end if;

        --receive spi clock toggle
        if((clk_toggles < last_bit_rx + 1) and (sclk_i = '0')) then
          rx_buffer <= rx_buffer(d_width-2 downto 0) & miso; --shift in

          received bit

        end if;

        --end of transaction
        if(clk_toggles = d_width*2 - 1) then
          busy <= '0';      --clock out not busy signal
          rx_data <= rx_buffer; --clock out received data to output port
          state <= ready;   --return to ready state
        else
          state <= execute; --not end of transaction
        end if;

        else
          --system clock to sclk ratio not met
          count <= count + 1; --increment counter
          state <= execute;  --remain in execute state
        end if;

      end case;
    end if;
  end if;

```

```
    end process;  
end logic;
```

Xilinx AXI UARTLite LogiCORE IP Configuration

Re-customize IP

AXI Uartlite (2.0)

Documentation IP Location Switch to Defaults

Show disabled ports

Component Name: axi_uartlite_0

Board IP Configuration

AXI CLK Frequency: 100 [10-300]MHz

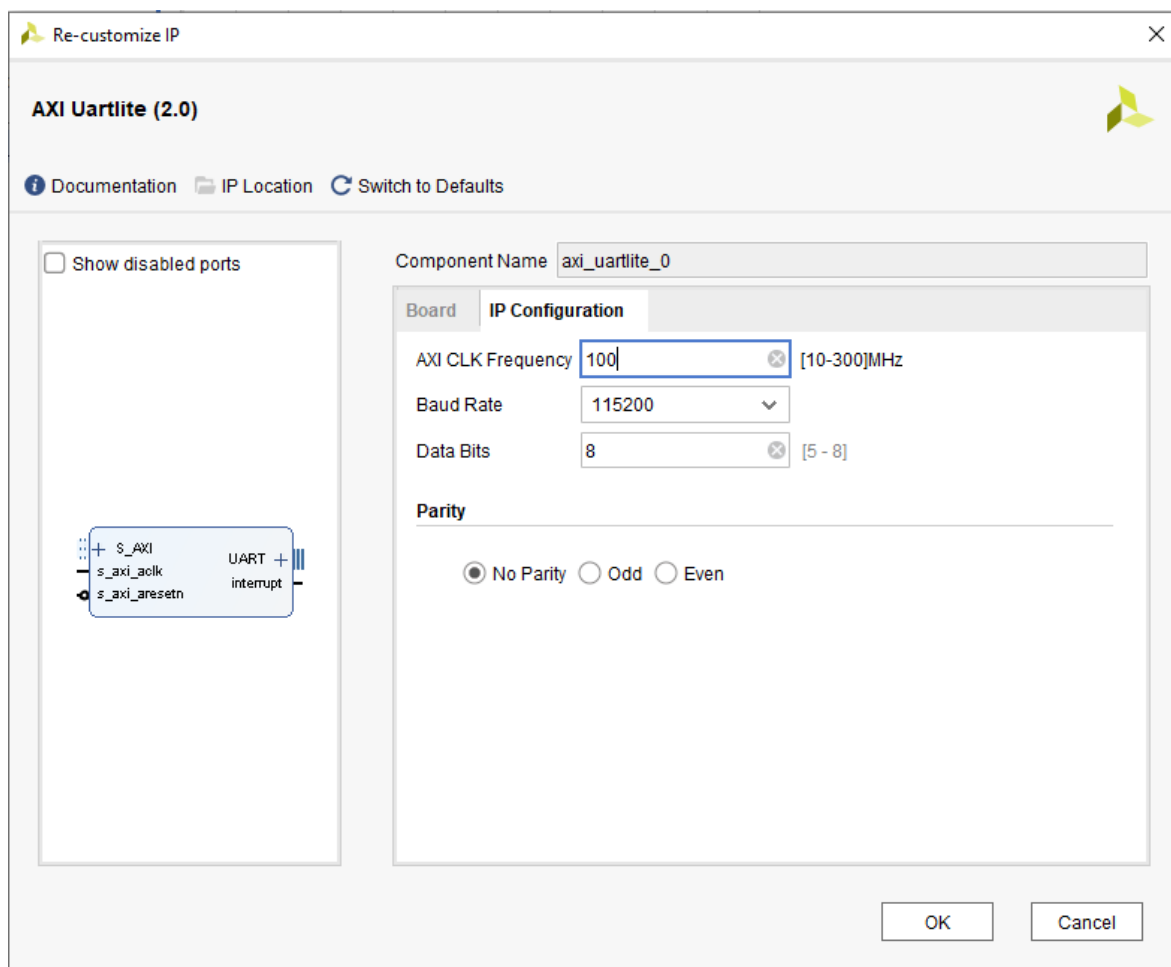
Baud Rate: 115200

Data Bits: 8 [5 - 8]

Parity

No Parity Odd Even

OK Cancel



The image shows the 'Re-customize IP' dialog box for the AXI UARTLite (2.0) component. The window title is 'Re-customize IP' and it has a close button in the top right corner. Below the title bar, the component name 'AXI Uartlite (2.0)' is displayed. There are three links: 'Documentation', 'IP Location', and 'Switch to Defaults'. On the left side, there is a checkbox labeled 'Show disabled ports' which is currently unchecked. Below this checkbox is a schematic diagram of the component's ports: 's_axi' (input), 's_axi_aclk' (input), 's_axi_aresetn' (input), 'UART' (output), and 'interrupt' (output). The main configuration area is titled 'Component Name' and contains a text field with 'axi_uartlite_0'. Below this are two tabs: 'Board' and 'IP Configuration', with 'IP Configuration' selected. Under the 'IP Configuration' tab, there are three input fields: 'AXI CLK Frequency' with a value of '100' and a range of '[10-300]MHz'; 'Baud Rate' with a value of '115200'; and 'Data Bits' with a value of '8' and a range of '[5 - 8]'. Below these fields is a section for 'Parity' with three radio buttons: 'No Parity' (selected), 'Odd', and 'Even'. At the bottom right of the dialog are 'OK' and 'Cancel' buttons.

Project Constraint File

```
## -----  
-----  
## ICS IDS Nexys4 DDR Project Constraint file  
## -----  
-----  
  
# Clock signal  
set_property -dict {PACKAGE_PIN E3 IOSTANDARD LVCMOS33} [get_ports clk]  
create_clock -period 10.000 -name sys_clk -waveform {0.000 5.000} -add [get_ports  
clk]  
  
## LEDs  
  
set_property -dict {PACKAGE_PIN H17 IOSTANDARD LVCMOS33} [get_ports err]  
  
set_property -dict {PACKAGE_PIN J13 IOSTANDARD LVCMOS33} [get_ports  
{flash_led[0]}]  
set_property -dict {PACKAGE_PIN N14 IOSTANDARD LVCMOS33} [get_ports  
{flash_led[1]}]  
set_property -dict {PACKAGE_PIN R18 IOSTANDARD LVCMOS33} [get_ports  
{flash_led[2]}]  
set_property -dict {PACKAGE_PIN V17 IOSTANDARD LVCMOS33} [get_ports  
{flash_led[3]}]  
  
set_property -dict {PACKAGE_PIN V15 IOSTANDARD LVCMOS33} [get_ports {gpio_led[0]}]  
set_property -dict {PACKAGE_PIN V14 IOSTANDARD LVCMOS33} [get_ports {gpio_led[1]}]  
set_property -dict {PACKAGE_PIN V12 IOSTANDARD LVCMOS33} [get_ports {gpio_led[2]}]  
set_property -dict {PACKAGE_PIN V11 IOSTANDARD LVCMOS33} [get_ports {gpio_led[3]}]  
  
##Buttons  
  
set_property -dict {PACKAGE_PIN C12 IOSTANDARD LVCMOS33} [get_ports rst_n]  
  
##Pmod Headers
```



```
##Pmod Header JB
```

```
set_property -dict {PACKAGE_PIN D14 IOSTANDARD LVCMOS33} [get_ports {gpio_sel[0]}]
set_property -dict {PACKAGE_PIN F16 IOSTANDARD LVCMOS33} [get_ports {gpio_sel[1]}]
set_property -dict {PACKAGE_PIN G16 IOSTANDARD LVCMOS33} [get_ports {gpio_sel[2]}]
set_property -dict {PACKAGE_PIN H14 IOSTANDARD LVCMOS33} [get_ports {gpio_sel[3]}]
set_property -dict {PACKAGE_PIN E16 IOSTANDARD LVCMOS33} [get_ports gpio_strb]
set_property -dict {PACKAGE_PIN F13 IOSTANDARD LVCMOS33} [get_ports miso]
set_property -dict {PACKAGE_PIN G13 IOSTANDARD LVCMOS33} [get_ports sclk]
```

```
##USB-RS232 Interface
```

```
set_property -dict {PACKAGE_PIN C4 IOSTANDARD LVCMOS33} [get_ports rx]
set_property -dict {PACKAGE_PIN D4 IOSTANDARD LVCMOS33} [get_ports tx]
```