

University of Arkansas, Fayetteville

**ScholarWorks@UARK**

---

Theses and Dissertations

---

7-2020

## Development of a Reference Design for a Cyber-Physical System

Nicholas Paul Blair

*University of Arkansas, Fayetteville*

Follow this and additional works at: <https://scholarworks.uark.edu/etd>



Part of the [Electrical and Electronics Commons](#), [Power and Energy Commons](#), and the [Systems and Communications Commons](#)

---

### Citation

Blair, N. P. (2020). Development of a Reference Design for a Cyber-Physical System. *Theses and Dissertations* Retrieved from <https://scholarworks.uark.edu/etd/3821>

This Thesis is brought to you for free and open access by ScholarWorks@UARK. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of ScholarWorks@UARK. For more information, please contact [ccmiddle@uark.edu](mailto:ccmiddle@uark.edu).

Development of a Reference Design for a Cyber-Physical System

A thesis submitted in partial fulfillment  
of the requirements for the degree of  
Master of Science in Electrical Engineering

by

Nicholas Paul Blair  
University of Arkansas  
Bachelor of Science in Engineering, 2018

July 2020  
University of Arkansas

This thesis is approved for recommendation to the Graduate Council.

---

H. Alan Mantooth, Ph.D.  
Thesis Director

---

Roy A. McCann, Ph.D.  
Committee Member

---

Jia Di, Ph.D.  
Committee Member

---

Chris Farnell  
Committee Member

## **ABSTRACT**

The purpose of this thesis is to develop a reference design to assist in the selection of security practices in power electronics design. A prototype will be developed from this reference design for evaluation. This evaluation will include a brief cost/benefit analysis to gauge the efficacy of implementing each layer of security throughout the power electronics design process. This thesis will also describe the obstacles and effectiveness of integrating a Trusted Platform Module (TPM) into a cyber-hardened grid-connected device. The TPM device is a secured crypto processor that assists in generating, storing, and restricting the use of cryptographic keys. The emphasis of this research is to establish integrity, authenticity, and confidentiality within a system by providing a baseline of security concerns for segments of the system. This research considers communication, control, and hardware level securities. The scope of this thesis will review the necessary security methods as well as consider the effects these methods have on the embedded system, to assess the desired security to responsiveness trade off. Applying this approach to a design process will alleviate various unknowns of appending security to a power electronics design.

This thesis describes the specific vulnerabilities introduced within this grid-edge environment, and how the liabilities within the system can be mitigated. Initially, common security techniques will be considered to establish a guideline to benchmark performance and resource costs of the system. The foundation will be a non-hardened power electronic system platform with industry standard communication protocols. Several security techniques and attack vectors will then be evaluated to contribute to the base level platform. Other fail-safe features take place to gauge progress of the selected approach, non-inclusive to the TPM. Collectively, this investigation will determine a valid experiment by appraising and categorizing resource

allocation, performance overhead, and monetary cost analysis results into a reference design. The prototype will then demonstrate methods to relieve common threats that are purposefully implemented into the design.

©2020 by Nicholas Paul Blair  
All Rights Reserved

## **ACKNOWLEDGMENTS**

My gratitude goes to my advisor, Dr. H. Alan Mantooth, who has been my guide through my graduate education providing a deeper understanding of cybersecurity as well as many other opportunities. I would also like to thank Dr. Jia Di, Dr. McCann, and Chris Farnell for being on my advising committee, directing me along the way. It has also been a pleasure to work with Joe Moquin, Ammar Khan, Estefano Soria, and many other students in the cyber security group who have helped me progress.

## TABLE OF CONTENTS

Chapter 1 .....	1
Introduction .....	1
1.1 Motivation .....	1
1.2 Security Principles .....	2
1.3 Thesis Objectives .....	7
1.4 Thesis Organization .....	8
1.5 References .....	8
Chapter 2 .....	10
Background .....	10
2.1 Introduction .....	10
2.2 Cyber-Physical System .....	10
2.3 Trusted Platform Module .....	22
2.4 References .....	26
Chapter 3 .....	28
Test Setup and Experimental Results .....	28
3.1 Introduction .....	28
3.2 Communication Layer Assessment and Results .....	28
3.3 Control Layer Assessment and Results .....	38
3.4 Hardware Layer Assessment and Results .....	41
3.5 References .....	45
Chapter 4 .....	47
Cost/Benefit Analysis of Mitigation Strategies .....	47
4.1 Introduction .....	47
4.2 Overhead and Computation Costs .....	48
4.3 Economical Costs .....	54
4.4 References .....	58
Chapter 5 .....	60
Conclusions and Recommendations .....	60
5.1 Summary of Conclusions .....	60
5.2 Recommendations and Future Work .....	61
Appendices .....	62
Appendix A: Code .....	62

A-1 DSP C Code.....	62
A-2 CPLD VHDL Code.....	93
A-3 Web Server Python Code.....	109



## LIST OF FIGURES

Figure 1.1. U.S. annual installed DER power capacity additions by DER technology [4] .....	2
Figure 2.1. Reference design architecture.....	11
Figure 2.2. Physical device architecture recognition .....	12
Figure 2.3. Communication layer topology .....	13
Figure 2.4. Raspberry Pi 4 Model B Board Layout .....	14
Figure 2.5. Lantronix XPort embedded ethernet module .....	16
Figure 2.6. Optiga TPM SLI 9670 development board .....	17
Figure 2.7. Control layer topology.....	18
Figure 2.8. Reference design controller board.....	20
Figure 2.9. Hardware layer topology .....	21
Figure 2.10. PE-EVAL board .....	22
Figure 2.11.Components of the TPM [1].....	23
Figure 3.1. Communication path diagram .....	30
Figure 3.2. CSPR web server user interface .....	33
Figure 3.3. HTTP packet Wireshark monitor .....	35
Figure 3.4. TPM 2.0 key wrapping [5] .....	36
Figure 3.5. OpenSSL client/server session flow [5] .....	37
Figure 3.6. HTTPS packet Wireshark monitor .....	38
Figure 3.7. Phase-A voltage cutoff .....	40
Figure 3.8. Phase-A low voltage output switching.....	41
Figure 3.9. Basic 3-phase inverter topology .....	42
Figure 3.10. Dead time effect .....	43

Figure 3.11. Accurate on/off delay wave forms .....	43
Figure 3.12. CPLD shoot-through protection simulation .....	45
Figure 4.1. Performance Breakdown of TLS Accesses [2] .....	50
Figure 4.2. Performance Breakdown by Percentage [2] .....	51
Figure 4.3. Connection Time of Successive Clients [2] .....	51
Figure 4.4. Control code Assembly breakdown.....	53
Figure 4.5. Vilros stock limitation .....	56
Figure 4.6. MicroCenter order upcharge.....	57
Figure 4.7. TPM Economy of scale example.....	58

## LIST OF TABLES

Table 1.1. Distributed energy resources by state [5] .....	4
Table 1.2. STRIDE Threat Model.....	6

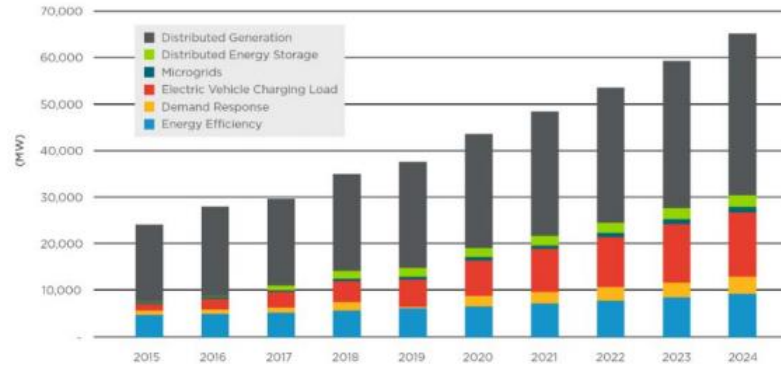
## **CHAPTER 1**

### **INTRODUCTION**

#### **1.1 Motivation**

In recent years, the residential market has seen substantial penetration of sophisticated power electronics devices which expose a considerable attack surface, particularly for the power grid. Predominantly, these grid-connected devices such as solar inverters and battery energy storage systems (BESS) have little to no security. Too often, security is an afterthought of the design process, provided it is even a consideration to begin with. With rapidly developing grid technologies and smart-grid communities, the demand for smarter distribution systems is increasing leading to the necessity of secure power electronic systems. Electric transmission/distribution systems in which we deploy new distributed energy resources must still meet changing consumer needs while constantly battling new regulations and being aged [1]. With these demanding considerations in mind, security requirements must be extensive to protect new systems being put in place and to avoid interfering with other issues with maintaining a reliable grid, which were previously discussed.

Heightened grid complexity has a negative connotation because of increasing points of failure; however, smart power transmission systems provide more efficient and effective environments. Intelligent devices within the grid provide more robust situational awareness which yields quicker reaction times for dynamic systems. Even so, concerns with operation security and reliability are drawbacks to consider [2]. An underlying goal of this research is to bridge the gap between the more secure methods of information technology (IT) and the newer technologies being introduced to the operational technology (OT) spectrum.



*Figure 1.1. U.S. annual installed DER power capacity additions by DER technology [4].*

This reference development will allow further innovation with less constraint and hasten technological developments within the power industry; all while meeting requirements to ensure the safe execution and management of all user data. This is particularly important to the field of Distributed Energy Resources (DER), considering the vast spread of these devices across a rapidly expanding market. As shown in Figure 1.1, the market will expand as much as 50% in the next five years. The existing complexity of modern system design can be egregious when considering all DER process variables. The desire of this work is to provide a design process that allows for rapid development in this field, as well as deliver different security options throughout the system design.

## 1.2 Security Principles

Security is a measurement of safety. This implies that if a system is thoroughly secure, then no undesired influence may have any impact on the system. That said, no security is absolute, no matter the intricacies of the solution. Therefore, not only is it critical to protect against outsider threats, but also to understand that an intrusion event is imminent, and one must have the capability to mitigate any adverse effects an intrusion may have on the system. Simply

put, a system must not exclusively consider if an event will happen, but when it will happen and how to stop the event from causing harm.

In a cyber-physical system, the approach to security is relatively unique. Apart from being vulnerable to component failures, these systems are often predisposed to malicious attacks due to a direct opening to the once ignorant physical system created by the increase in communication and data flow now broadcasted to and from the device [3]. The proposed research investigates security from a layered perspective to the cyber-physical system. A layered approach considers security throughout the design phase rather than injecting security into a complete system. This aspect is critical to this approach due to the immature nature of merging IT and OT techniques to ensure protection at vital points of contact created using this method. Though immature and exposed, the operations coupling IT and OT will only become more prominent in the coming years [5]. This can be compared to any technological scramble to market; take smart phones for example. Like smart phones, this relatively new concept is rapidly pressing the market of distributed energy resources and expanding quickly, but without forethought on how to administer security to this field, these devices will succumb to the same setbacks. Many security exploitations became relevant to the attack surface, that is the smart phone platform, since this area had yet to be explored. Smart phones faced many new challenges with obstacles that could have been avoided with slightly more consideration to the security and the data that was to be protected. Challenges akin to infection of hostile, intrusive, or otherwise malicious software become far more significant to those whose livelihoods that depend on these resources [6, 7]. The residential and commercial markets both have substantial DER presence reported by the DOE and the More Than Smart working group: more than 42,500 MW of total distributed energy resources already established nationwide. Table 1.1. shows this on a state by

state breakdown and this is not even concerning those to come. Though slightly dated, this table alone shows the urgency to secure future devices

*Table 1.1. Distributed Energy Resources by State [5]*

STATE	TOTAL DER <sup>a</sup> (MW)	DISTRIBUTED <sup>a</sup> GENERATION (MW)	STORAGE <sup>a</sup> (MW)	ENERGY <sup>a</sup> EFFICIENCY T (MW)	DEMAND <sup>a</sup> RESPONSE (MW)	CHP <sup>a</sup> INSTALLATIONS (MW)	# ELECTRIC <sup>a</sup> VEHICLES
AK	391	9	0	0	22	360	155
AL	1,263	522	0	22	610	109	773
AR	715	299	0	88	230	99	374
AZ	1,136	646	0	228	226	35	4,361
CA	8,387	3,149	5	584	2,998	1,651	126,283
CO	436	277	0	99	29	32	4,001
CT	520	119	0	47	88	267	2,476
DC	62	20	0	8	20	14	493
DE	190	54	0	0	129	7	383
FL	1,052	428	0	105	347	173	10,383
GA	1,212	775	0	102	197	139	15,551
HI	636	527	0	28	3	78	3,050
IA	463	38	0	143	127	155	928
ID	860	70	0	252	472	66	409
IL	533	33	0	12	54	434	6,694
IN	1,110	33	0	548	401	127	1,697
KS	141	10	0	2	45	85	750
KY	565	64	0	96	358	47	701
LA	825	520	0	80	0	226	527
MA	1,088	570	0	184	16	319	4,612
MD	597	183	0	124	191	99	5,028
ME	604	504	0	19	0	81	695
MI	659	181	1	160	28	290	8,844
MN	1,604	226	0	460	730	188	2,775
MO	302	100	0	65	29	107	1,859
MS	999	274	0	23	655	47	201
MT	63	15	0	37	0	11	362
NC	1,886	384	0	701	625	175	3,384
ND	373	11	0	4	311	46	91
NE	496	6	0	46	409	35	579
NH	97	41	0	9	0	47	761
NJ	1,635	1,146	0	86	74	329	6,021
NM	213	70	0	70	43	30	637
NV	301	83	0	36	158	24	1,509
NY	1,256	441	0	214	37	564	11,278
OH	874	155	0	371	198	150	3,814
OK	336	76	0	73	123	65	806
OR	557	301	0	11	35	211	5,681
PA	936	343	0	120	101	371	4,540
RI	115	18	0	39	n/a <sup>a</sup>	58	417
SC	1,546	353	0	268	753	172	1,056
SD	271	1	0	162	100	8	160
TN	1,054	228	0	50	703	73	2,730
TX	1,457	287	0	243	543	384	9,925
UT	273	35	0	53	120	64	1,565
VA	701	398	0	17	142	145	3,628
VT	69	33	0	12	1	24	840
WA	772	304	0	302	15	151	12,291
WI	677	224	0	106	99	248	2,429
WV	215	106	0	4	84	22	271
WY	44	2	0	7	4	31	73
<b>TOTAL</b>	<b>42,569</b>	<b>14,691</b>	<b>6</b>	<b>6,517</b>	<b>12,683</b>	<b>8,672</b>	<b>278,851</b>

in this area and helps to better understand the need for this research. The need for energy storage and electric vehicles is also scaling quickly, and the same security principles can be applied to these areas. Diligence is important in a rapidly growing industry such as this, and any complacency could be detrimental to the residential market and those who inhabit it.

### **1.2.1 Security Modeling**

There are many ways to approach security to determine vulnerabilities within a system. Most common methods involve an arrangement of considerations relating to the threats and risks of vital system operations. These arrangements are called models which help to guide policies or distinguish potential risks to a system; generally, a good starting point when assessing probable points of interest during the design process. The significance behind this, regardless of what method is chosen, is the visibility into the device and the resources being used. More precisely, this gives foresight on what the system is committed to doing, while mapping out interconnections and communications within a design process before truly assessing threats.

Different situations require specific measures when appending a security model, therefore various security assessments exist to help provide for the individual needs of most scenarios. Example standardized models include: CIA, STRIDE, DREAD, and Attack Trees. These models each focus on distinct aspects of exposure to a system, such as the STRIDE model; made as a mnemonic model by the Microsoft Corporation to identify computer security threats. The acronym defined in Table 1.2 considers threats associated with this model. Various other models exist to help define the risks and liabilities of cyber systems. Each model defines the approach differently, but the goals are all the same; to provide a proactive solution to understanding the vulnerabilities within a system and define countermeasures to nullify these exploits.



*Table 1.2. STRIDE Threat Model*

<b>S</b>	Spoofing Identity
<b>T</b>	Tampering with Data
<b>R</b>	Repudiation
<b>I</b>	Information Disclosure
<b>D</b>	Denial of Service
<b>E</b>	Elevation of Privilege

### 1.2.2 Modeled with CIA Triad

The model that will be considered for this research is the CIA model; sometimes identified as the AIC model as to avoid confusion with the Central Intelligence Agency. This model was designed specifically for organizations to guide policies for information security. Typically represented with a triangle, hence the name triad, emphasis within this research is focused on three key points in the acronym: Confidentiality, Integrity, and Availability. Many standards and third parties have partnered to provide users with the ability to manage their own privacy, durability, and legitimacy based on this model [8]. Subsequently, this research has defined the terms to represent vulnerabilities to be addressed with a security-in-depth approach. Each term is described here and explained in further detail in the following sections of the paper.

Confidentiality is considered a method of security intended to control the access to information. This means to ensure sensitive data is only available to those with authorized privilege can access this data. Two-way encryption contributes to confidentiality by mapping symbols, letters, and figures arranged to represent plain text messages and is commonly used to

prevent unauthorized access to concealed data. Hashing, like encryption, is another algorithm used to arrange information into a fixed length format for confidentiality among other uses.

Data integrity aims to protect the data from modification or removal. To do so, the data must only be modified by means that would not lead to system failure or as a baseline be recoverable. The idea is that one can trust the data that is in the system. If authentication is successful, the user must not be able to manipulate or interpret the data in a way that would be harmful to the user or system.

Lastly, availability is the most essential element in determining authorized users and the level of accessibility to the system. This component of the triad is focused on targeting hardware failures, power failures, and failovers to ensure that the data is attainable. Recovery is essential for the availability of data to stay accessible. In the event of data corruption, hardware failure, or malicious attack, the user and the system must still retain the ability to monitor and use the data as long as it can be trusted. A method to ensure the availability is to consistently backup critical data in a secured location secluded from the system. This guarantees that previous data can be accessed without worry of system level influence or interaction.

### **1.3 Thesis Objectives**

The purpose of this thesis is to investigate methods in which to incorporate security into a cyber-physical system, as well as perform a cost/benefit analysis on the described system to portray the effectiveness of each layer of security and categorize the costs associated with each addition. From this analysis, the expectation is to realize a reference design that provides support in securing the power electronics design process. This reference design will demonstrate what base level functionality within a system may look like, different layers of the system, and how to approach security at each layer. A portion of this thesis will focus on the benefits of utilizing the

Trusted Platform Module, one way it is specifically used within this system, and further capabilities of this module pertaining to security and system overhead. Experimental results are based on the mitigation techniques used in this analysis to determine the functionality of each method used. Other system design considerations will be described to explain the reason for choosing certain system components, attack scenarios, and mitigation techniques.

## **1.4 Thesis Organization**

The composition of this thesis is arranged sequentially by chapter as listed:

- Chapter 1 serves as an introduction to the concepts associated with this research such as basic security principles, security models, the chosen approach, and objectives.
- Chapter 2 provides an explanation of the background needed to understand the functionality of the cyber-physical system. This section also considers the integration of the TPM and how to further utilize the available resources for future development.
- Chapter 3 details the remainder of the system with experimental setup to depict the mitigation strategies that can be applied to a cyber-physical system.
- Chapter 4 presents a cost/benefit analysis of the mitigation techniques.
- Chapter 5 defines the summarized conclusion and introduces potential future work.

## **1.5 References**

- [1] D. Tan and D. Novosel, "Energy challenge, power electronics & systems (PEAS) technology and grid modernization," in CPSS Transactions on Power Electronics and Applications, vol. 2, no. 1, pp. 3-11, 2017.
- [2] Y. Jia, Z. Xu, L. L. Lai and K. P. Wong, "Risk-Based Power System Security Analysis Considering Cascading Outages," in IEEE Transactions on Industrial Informatics, vol. 12, no. 2, pp. 872-882, April 2016.
- [3] F. Pasqualetti, F. Dörfler and F. Bullo, "Attack Detection and Identification in Cyber-Physical Systems," in IEEE Transactions on Automatic Control, vol. 58, no. 11, pp. 2715-2729, Nov. 2013.

- [4] Federal Energy Regulatory Commission (FERC), "Distributed Energy Resources: Technical Considerations for the Bulk Power System," FERC, 2018.
- [5] P. Martini, T. Brunello, and A. Howley, "Planning for More Distributed Energy Resources on the Grid: A Summary for Policy-Makers on the Walk-Jog-Run Model",
- [6] M. La Polla, F. Martinelli and D. Sgandurra, "A Survey on Security for Mobile Devices," in IEEE Communications Surveys & Tutorials, vol. 15, no. 1, pp. 446-471, First Quarter 2013.
- [7] P. Srikantha and D. Kundur, "A DER Attack-Mitigation Differential Game for Smart Grid Security Analysis," in IEEE Transactions on Smart Grid, vol. 7, no. 3, pp. 1476-1485, May 2016.
- [8] Microsoft, The STRIDE Threat Model, Nov 2009. [Online] Available: [https://docs.microsoft.com/en-us/previous-versions/commerce-server/ee823878\(v=cs.20\)?redirectedfrom=MSDN](https://docs.microsoft.com/en-us/previous-versions/commerce-server/ee823878(v=cs.20)?redirectedfrom=MSDN)
- [9] S. Deepika and P. Pandiaraja, "Ensuring CIA triad for user data using collaborative filtering mechanism," 2013 International Conference on Information Communication and Embedded Systems (ICICES), Chennai, 2013, pp. 925-928.

## **CHAPTER 2**

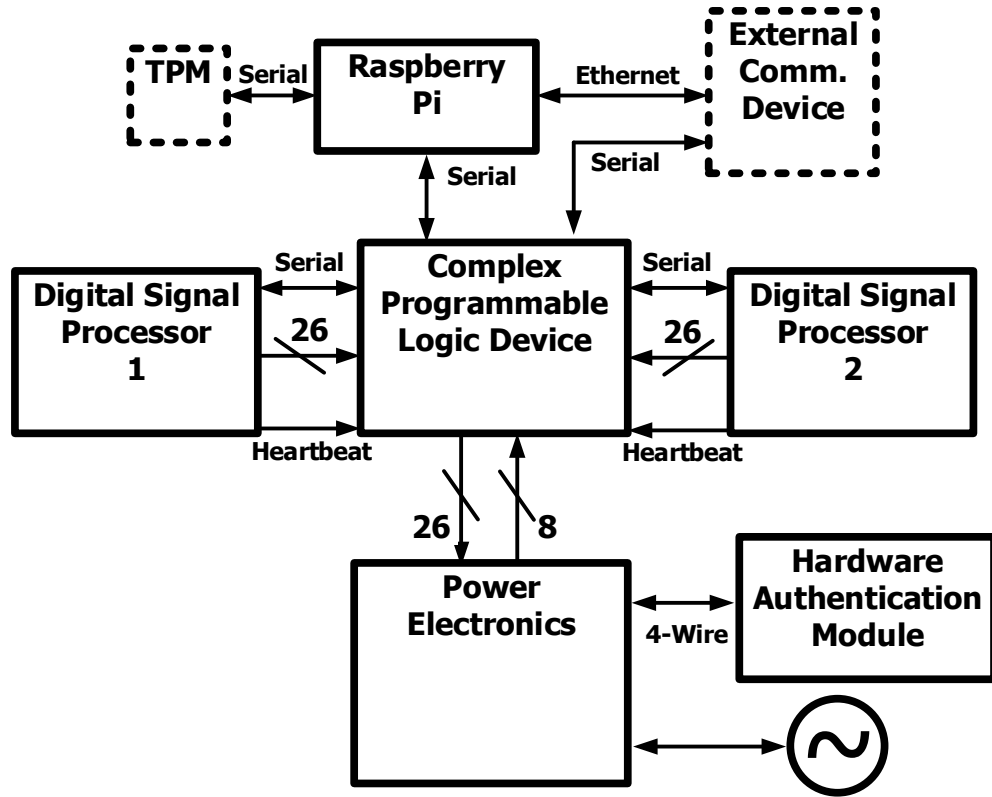
### **BACKGROUND**

#### **2.1 Introduction**

From Chapter 1, the motivation for this research is to implement layers of security within a cyber-physical device that will act as a model reference design. The device considered in this research is readying for a preproduction prototype, meaning that the controls and overhead are likely to become much more resource intensive due to complex secondary and tertiary control algorithms as well as protection schemes. To add, having each layer of security implemented in a singular location rather than placed throughout the architecture is not beneficial for reasons concerning basic security principles. These flaws in design display the need for a multi-level security architecture. The approach for the reference design architecture defines these layers and applies example securities that demonstrate protection against realistic threats.

#### **2.2 Cyber-Physical System**

A cyber-physical system is defined as the integration of computation, communication, and physical processes in order to complete various defined tasks. Analogous to the Internet of Things (IoT), specifically the Industrial Internet of Things (IIOT), these systems utilize a combination of actuators and sensors that communicate with control systems to operate certain physical procedures. Embedded control systems further consult with an elevated computing environment that monitors critical points within the system. There are considerable challenges with integrating these systems particularly due to the qualitative safety and reliability



*Figure 2.1. Reference design architecture.*

requirements of the physical environment are fundamentally different from those same stipulations in computing [9]. Regardless of the challenges, the demand for such devices has grown exponentially as technology advances, and the importance for future growth is evident. For this precise reason, security will play an important role in cyber-physical systems, and this displays the need for implementing a reference design.

In relation to the field of distributed energy resources, this research realizes a base architecture of the critical layers to consider within a device. These layers to be examined are labeled the communication layer, control layer, and the physical or hardware layer. This approach uses components from Figure 2.1 and structures these interconnected pieces as displayed in Figure 2.2. This method appeals to the need for a reference design by describing the

vital components in the device, recognizing various risks in each layer, and how to address the foremost risks; all to define an outline for addressing vulnerabilities within a relatable system.

For a proper understanding to recognize the physical makeup of the reference design architecture, the components within the system are displayed in Figure 2.2. The following sections will reference Figure 2.2 to help realize the physical attributes of each device as they are explained in further detail below. The components are numbered chronologically to show where each component resides in the setup in the following order: Raspberry Pi, Complex Programmable Logic Device, Digital Signal Processor 1, Digital Signal Processor 2, Power Electronics Hardware, Hardware Authentication Module, and the Trusted Platform Module (TPM). Each element listed will be described below when detailing the various layers.

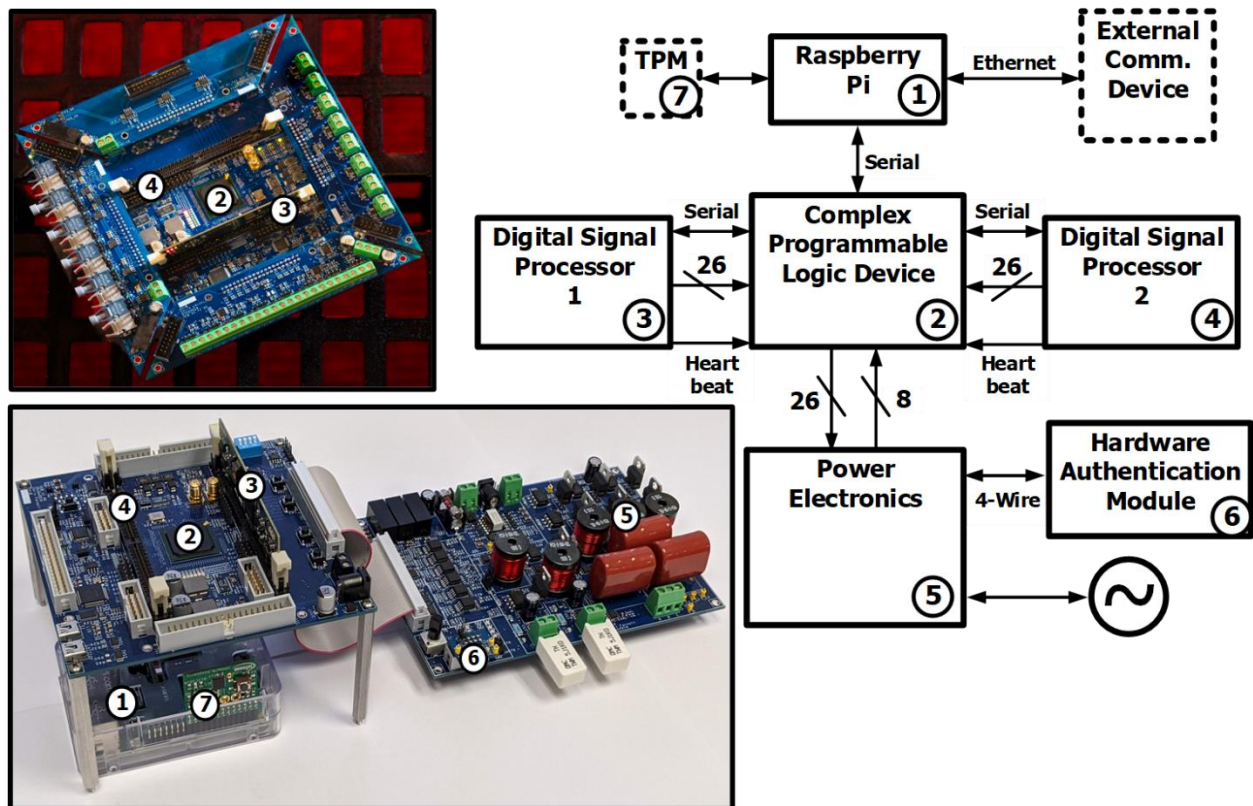


Figure 2.2. Physical device architecture recognition.

### 2.2.1 Communication Layer

Within a cyber-physical system, some form of communication typically exists in order to communicate acquired data out of the system to be processed and utilized. It is necessary to verify that the data reported from within the system is correct and realistic. This data provides growth for companies within the power industry and supports the development of a robust smart grid [10]. This happens by monitoring the data received from the system to address any needs or concerns from the system. Issues observed within the system can then be documented and counteracted by either correcting the issue in the current system with an update or the development of a new revision to correct the design will also suffice. Collecting and monitoring system data through communication allows for better data management, data analysis, visualization, and processing of the data. Figure 2.3 defines a generic communication method chosen to be used in the reference design. Like many other elements of this reference design, the components within each layer can be replaced with similar devices to better address the user's needs. That is why this is considered a reference design. It grants the user the ability to conform to their own needs with the freedom to swap certain components for more suitable functionality.

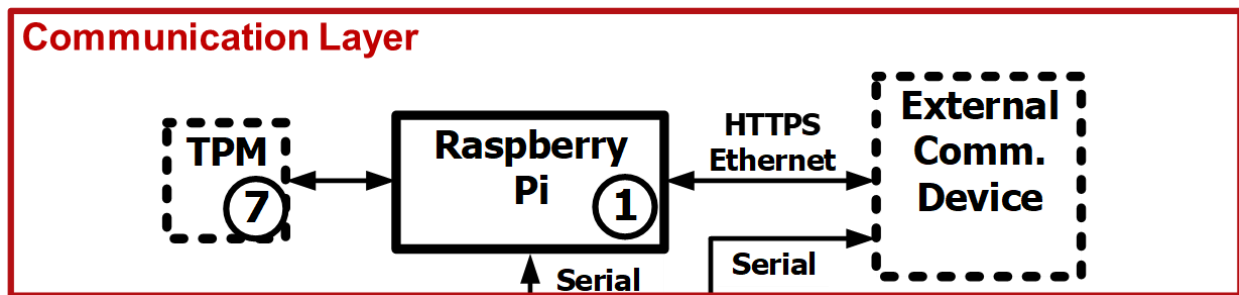


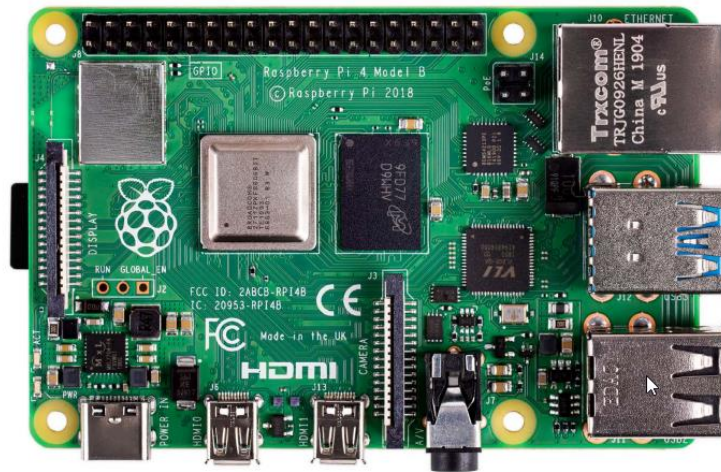
Figure 2.3. Communication layer topology.

The current reference design contains a communication layer consisting of the Trusted Platform Module, a Raspberry Pi, and an external communications device. Also, not shown in the diagram



is one other communication device termed an XPort ethernet module. Each of these components has an important role within the communication layer and will be described in more detail below.

The primary component of the communication layer within the reference design is the Raspberry Pi. The Raspberry Pi is an embedded environment for the Linux operating system featuring multiple inputs and outputs for the user to interface with such as keyboard and mouse inputs for control and a display output, but more importantly direct access to the General Purpose Input/Output (GPIO) directly to the central processor. This provides direct communication with the processor over many standardized protocols in order to connect other devices such as the Trusted Platform Module for instance. For an idea of what the Raspberry Pi looks like, it can be seen in Figure 2.2. For a closer look, see Figure 2.4. The reasons for choosing the Raspberry Pi 4 in this instance is due to the quad core Arm processor that houses the Linux kernel, Gigabit ethernet port, serial interface, and abundant support from documentation and the community.

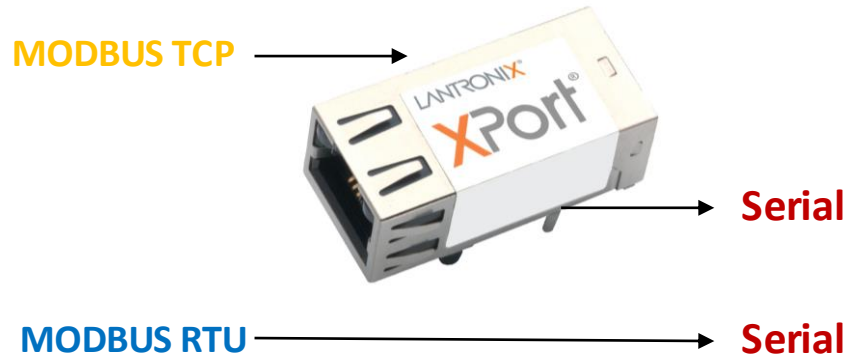


*Figure 2.4. Raspberry Pi 4 Model B board layout.*

Realistically, this portion of the reference design simply depends on any Arm processor platform that can support a Linux kernel. The Linux kernel here is the key component and the

Raspberry Pi architecture is designed for this. To elaborate on this concept, a kernel is the foundation of an operating system whereas an operating system is the software supporting the base computer functionality. For this reference design, a Raspbian operating system Linux kernel is used because it is open source and very well documented in capabilities, versatility, and support; though, any operating system could be used that contains similar functionality.

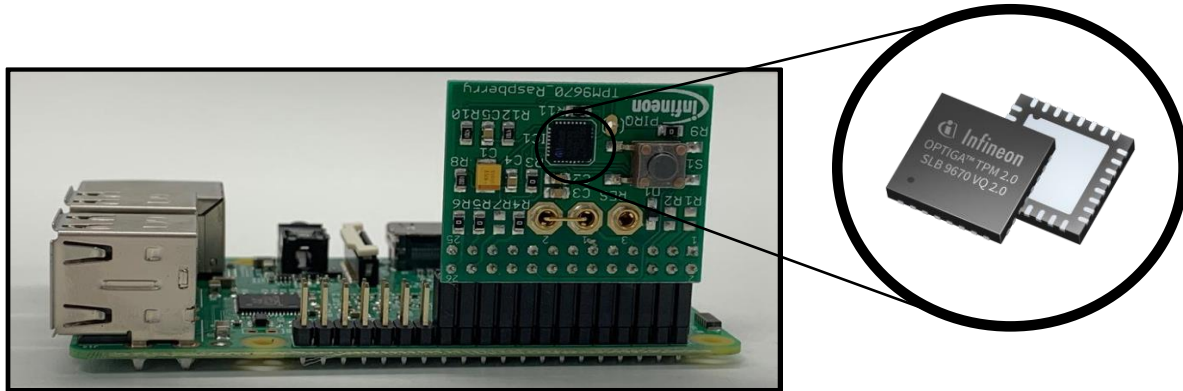
The purpose of this kernel is to accommodate the required communication link between the cyber-physical device and the outside world. This kernel provides an environment in which the system can execute programs for importing commands and exporting data, to and from the system. These programs are simply code that is executed within the device to serve various purposes such as gathering weather data and time of use pricing, hosting a web server, and addressing lower level registers in the system. Before accessing raw data, it must be scaled and formatted to be useful to the user. This formatting is done when addressing the lower level registers in the system. The data is manipulated and managed by a web server running within the system. The webserver acts as a hub to route important data to the appropriate destination. Here, the webserver can display the data in many ways to help the user visualize system states or operation of the system. The webserver also offers ease of access to the device through the external communication device which could be any remote terminal that has access to the network. This device could be any remote terminal connected to the same computing network as the Raspberry Pi. Since the primary access point to the device is through this connection, it is critical for this connection to be secure. The security methods used between the webserver and TPM are further detailed in the integration and testing procedure sections of this thesis.



*Figure 2.5. Lantronix XPort embedded ethernet module.*

Another communication method utilizing the previously mentioned XPORT module is available as an alternative to the webserver communication path. Figure 2.5 details the two communication protocol standards chosen for the reference design. These two are Modbus TCP, which is utilized through the XPort module, and Modbus RTU otherwise. The web server communicates to the lower level control layer via a serial interface using the Modbus RTU protocol while the XPort supports the Modbus TCP protocol to communicate system information to the remote terminal. The remote terminal can also relay system commands back to the XPort module to be converted into Modbus RTU over serial back to the system. This decision on which communication type to depend on is made within the control layer. Even so, the control layer must only standardize on the Modbus RTU protocol and if the access is requested over Modbus TCP, then the XPort will handle the protocol translation.

Lastly, the final element within the communication layer is the Trusted Platform Module, also known as the TPM. The TPM will be reintroduced in section 2.3 to be covered in extensive detail. For now, it is worth noting another key reason for choosing a Raspberry Pi was support for the Optiga TPM SLI 9760 evaluation board manufactured by Infineon. A snippet of the device and its connection to the Raspberry Pi can be seen in Figure 2.6. Choosing this



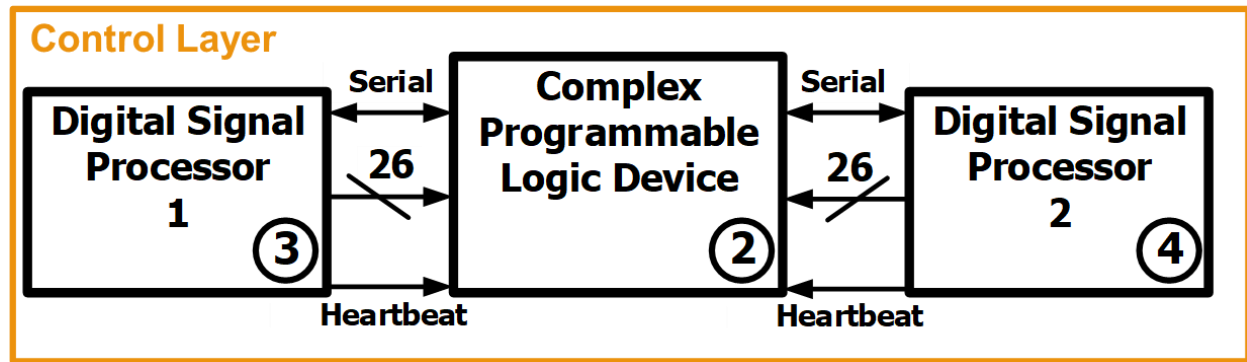
*Figure 2.6. Optiga TPM SLI 9670 development board*

board for the design hastened the development process substantially. Ideally, this TPM would rest on the control board layer to be connected directly to the communication bus to provide support in a full production model, however, this board was used for rapid prototype development with the ability to neglect appending additional components on the control layer. Interfacing with the control layer would take more time to develop for considering a controller would also need to be developed within the CPLD using VHDL to be able to communicate with this device. Also, the Application Programming Interface (API) developed for this board utilized the Serial Peripheral Interface (SPI) protocol for which it is much easier to develop, given the team background.

### **2.2.2 Control Layer**

The control layer is put in place to manage the behavior of the hardware devices within the cyber-physical system. This layer acts as a medium in order to facilitate the communication layer commands to the actuators, which are in the hardware layer. In addition, the control layer

reports sensor data coming from the hardware layer back through the communication layer for



*Figure 2.7. Control layer topology.*

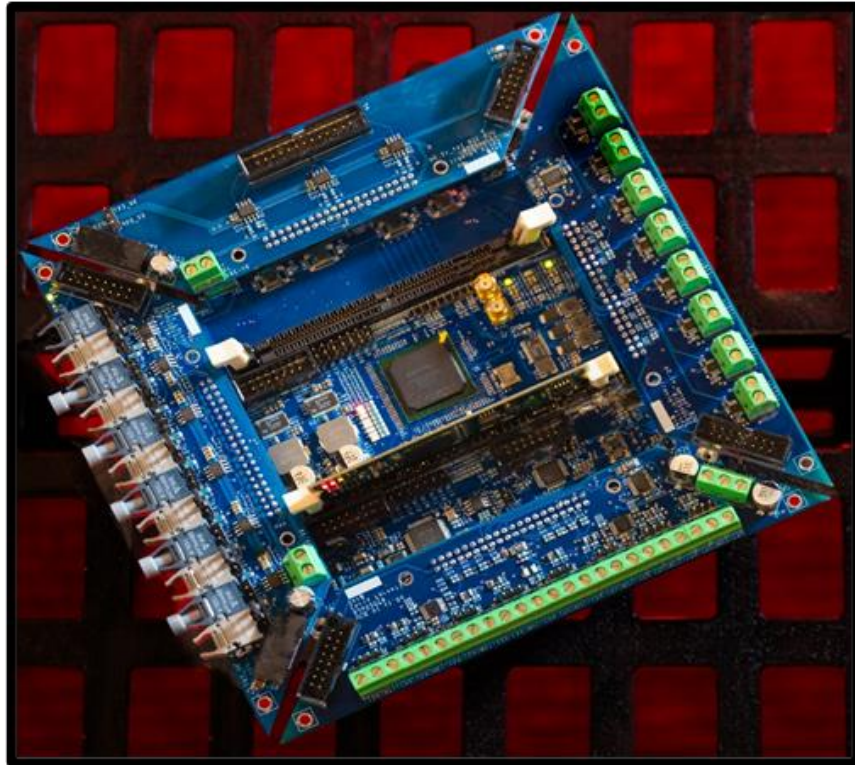
monitoring. The control layer can be completely autonomous or it can actuate from input commands. Regarding the cyber-physical reference design, this layer handles a combination of taking input commands for reference as well as autonomously following the reference through control system feedback. For any complex control system of a physical system, the autonomous control method uses feedback from the sensor data in order to maintain set output values (i.e. voltages and currents). The hardware control methods used within this research relate to power electronics hardware such as ac inverters, dc/dc converters, power regulators, and battery energy storage. To determine how these methods are used, the specific reference design control layer needs to be explained further.

The design illustrates the control layer with one Complex Programmable Logic Device (CPLD) and two Digital Signal Processors (DSP) as seen in Figure 2.7. A closeup of the control board can be seen in Figure 2.8. To further read about these processors, one may read sources [12] and [13]. The role of the DSP is to directly control the behavior of the power electronics. Particularly, for this cyber-physical system, a 3-phase inverter is receiving power from a Direct Current (dc) link and is providing a sinusoidal 3-phase Alternating Current (ac) output. The hardware description will be reintroduced in the next section in more detail. For now, all that

needs to be understood is that the DSP is in control of directing the power flow and reading measurements from the sensors. The DSP receives this sensor data through a control loop. Even though the complexity of the pre-production prototype controls constructed from similar control designs will increase as a result of incorporating secondary and tertiary control algorithms, the power flow will be further optimized. This displays the growth capability within the reference design. There is sufficient processing capability to handle advanced control system models, as long as the appropriate power electronics hardware is considered within the hardware layer of the design.

The CPLD considered in this reference design is an extremely fast integrated circuit that is a derivation of the Field Programmable Gate Array (FPGA). Typically, a CPLD can have anywhere from a thousand to ten thousand gate arrays, whereas FPGAs are known to have tens of thousands to millions of gate arrays on a single chip. Though FPGAs seem to be much larger, CPLDs are more economically efficient, and they can be much faster when considering execution and boot times. FPGAs require more interconnections which can lead to more flexibility in design choices but are more complex to design. CPLDs are architecturally simple compared to the FPGA, which allows for greater speeds. One important role for the CPLD in the reference design is to supply access to the gate array as a routing fabric for the digital signals in the system. In other words, all digital signals being transported through the control layer must travel to the CPLD. The CPLD then decides where to route each signal. Analog signals must be manually routed through the platform, as the CPLD does not support the routing of continuous signals. Routing each digital signal through the CPLD opens the reference design to a wide variety of functions. Some functions currently implemented have been described above. By routing communication signals through the CPLD, the ability to physically reroute and sever

communication signals based on predetermined conditions is available. This functionality is critical to the reference design control layer and will support many more functions which also allows for modularity. In Figure 2.8, the trapezoidal daughter boards serve many functions, and all follow the same standardized pinout. This allows for almost limitless expansion capabilities that would simply require a sensor/actuator daughter board design. That design could then attach to the controller board to enhance functionality and meet the needs of any user.

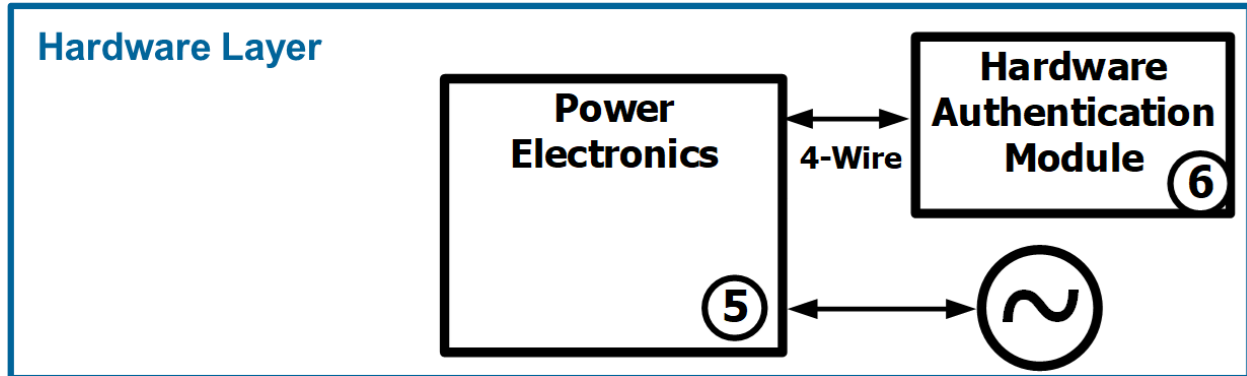


*Figure 2.8. Reference design controller board.*

### **2.2.3 Hardware Layer**

The hardware or physical layer of the device is the power electronics system, composed of all sensors and actuators in contact with the control layer of the cyber-physical system. The sensors within this system are voltage and current sensors set up throughout the power electronics, in order to monitor the system as well as provide feedback to the active control systems in the control layer. Actuators in this context could be the 3-phase motor, acting as a

load, while the power electronics within this platform are responsible for converting the control signals into the necessary output to drive this load. To further explain the hardware, the design of the power electronics evaluation board (PE-EVAL board) will be described in this section.



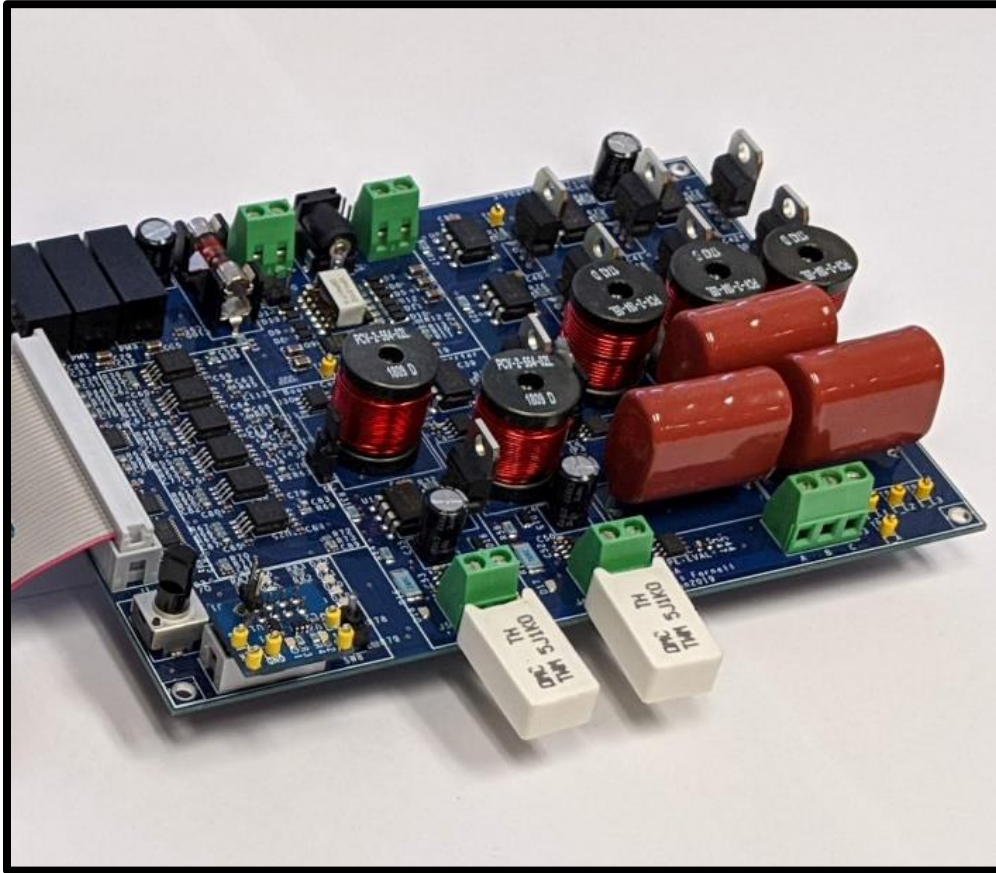
*Figure 2.9. Hardware layer topology.*

The sensors in the hardware layer are also reporting back to the DSP. The DSP has 16 on-board Analog-to-Digital Converter (ADC) inputs that allow for an analog signal, such as a voltage, to be converted into a scaled digital signal. This shows how the sensory data is carried out of the hardware layer. On the output of the 3-phase inverter, the grid connects to the device. This is where more complex control strategies could consider different modes of operation, such as islanded mode, grid feeding mode, and grid forming mode.

The breakdown of the PE-EVAL board has three separate topologies: a buck converter, boost converter, and a 3-phase inverter. For this reference design, only the 3-phase inverter is considered to be the hardware layer. This is for simplicity when denoting attack vectors on the system, however, these security principles can be applied to many power electronic scenarios. The PE-EVAL board can be seen in Figure 2.10. This is an arbitrary hardware topology selected because this design was considered when the controller architecture was designed. The



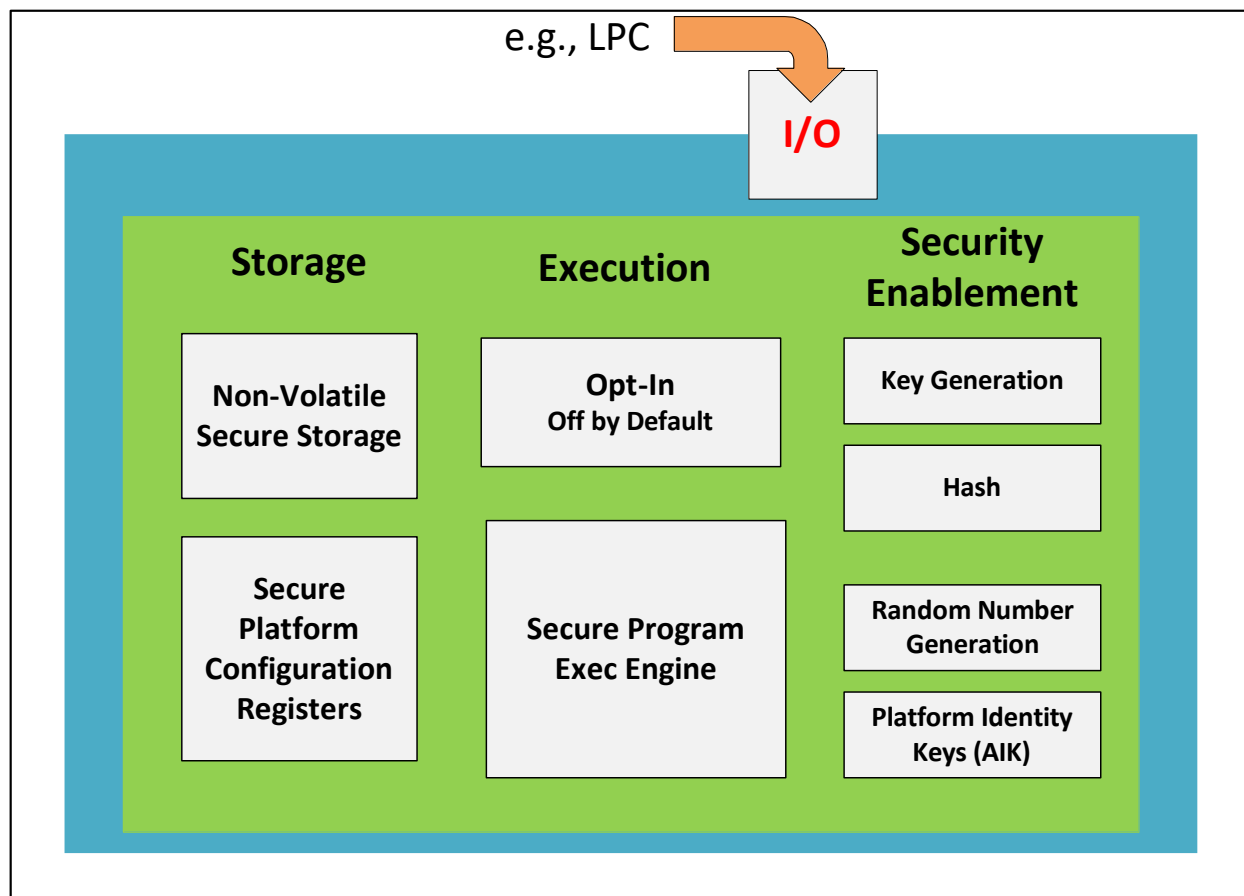
remainder of the power electronics design, that is not directly associated with the attack vectors, will not be discussed to avoid complexity and confusion.



*Figure 2.10. PE-EVAL board.*

### **2.3 Trusted Platform Module**

The Trusted Platform Module, also known as a TPM, is a microcontroller for imparting hardware authentication and tamper protections into a system. This module was created by the Trusted Computing Group (TCG), an international standards body of approximately 140 companies. These companies are involved in creating specifications that define PC TPMs,



*Figure 2.11. Components of the TPM [1].*

trusted infrastructure requirements, APIs, protocols necessary to operate a trusted environment, and trusted modules for other devices [1]. This integrated circuit comes well documented with the TPM 2.0 Library specification that has become an international standard administered by ISO/IEC JTC 1.

The TPM architecture, simplified in Figure 2.11, is comprised of a plethora of security modules such as non-volatile secure storage, secure platform configuration registers, a hardware random-number generator, hash, and isolated key generation and key storage. These functionalities can provide cryptographic keys, prevent private keys from being exported, protect pin values used for authentication, record and anonymously report on software loaded during the

boot process to protect against malware and attacks [1]. Each of these features are effective functions that can be utilized for security purposes before considering that the integrated chip is isolated from the system. Some features are worth noting and are described in more detail below.

Within distributed energy resources, cryptographic keys provide data access to users who possess certain credentials or attributes. In this case, the method for enforcing such policies is to employ a trusted server to store the data and mediate access control between the user and server [2]. Common practice exercises the encryption process and key storage on these mediums.

Although encrypted, these keys still reside on a shared local storage within the system. If the mandating server is compromised or physically jeopardized, a malicious entity could theoretically decipher the keys and decrypt the secured data if given enough time. The TPM improves on this process by physically isolating a portion of the cryptographic key within the hardware cryptographic module. In the case the event that the server is compromised, the TPM ensures that the attacker cannot gain access to the other portion of the encrypted private keys.

Various standard encryption practices for securely preserving keys and decrypted data rely on volatile memory while plaintext portions are loaded into cryptographic algorithms. This means that the private keys are subject to memory disclosure attacks that read unauthorized data from RAM. These attacks can be accomplished through software methods even if the integrity of the system's executable binaries is maintained, the Open SSL Heart Bleed attack for example. Physical attacks are also a potential threat, such as cold-boot attacks on RAM chips, even if the system is completely safe from software vulnerabilities [3]. Secure non-volatile storage helps alleviate this issue by ensuring that entire encryption process, including the storage of the RSA private keys, is completely secluded from the system. If the system is compromised, there is no risk of losing the stored private keys, since they are permanently attached to the TPM, and

cannot be exported, even as cyphertext. Therefore, this method blocks the attacker from gaining access to the keys in any way.

An additional core component required for adequate cryptography is a Random Number Generator (RNG). A random number generator is an instrument used to randomly generate strings of unpredictable numbers and symbols. These devices are not exclusive to cryptography, but are included in simulation environments, statistical sampling, stochastic optimization methods and image authentication watermarking. Within a software environment, a random number generator can be very close to random, but never truly random; namely, a Pseudo Random Number Generator (PSNG). PSNGs generate numbers that may seem random but are procedurally generated and can be recreated if the previous state of the PRNG is determined. This is inadequate for an effective cryptographic processor because the security of cryptographic algorithms depends on generating secret strings of data which are generated by RNGs to create cryptographic keys. To ensure that an RNG is secure, the output must be unpredictable and statistically indistinguishable from a true random sequence [4]. True Random Number Generators (TRNGs) are found to be stochastic and the same output cannot be intentionally reproduced. This is typically done by measuring a physical abnormality that is decidedly random. A trivial example could be electrical noise measured and polled to generate symbols based on the unpredictable noise with theoretically infinite entropy. Utilizing this method to generate cryptographic keys for encryption and secure storage cannot be intentionally recreated from another source, making the TRNG the ideal solution.

Lastly considered, the hardware hash is another distinct component that proves the TPM useful in contrast to a software solution. A hash is a hardware ID that is distinct to the individual device. The hardware ID typically contains information about the system, such as device serial

numbers, hard drive serial numbers, model numbers, and the manufacturer. These are some characteristics that could be described in a hardware hash, but not limited to these variables. The key here is to store information that is indicative of the device and could be used to uniquely identify this device [5]. For this use case, this feature is utilized for what is called remote attestation [6]. Remote attestation specific to this case would require a SHA-256 hash on the known hardware information at the time of startup. This hash is unique, meaning that no other input to this hashing algorithm could resolve to an identical output. Once the hash is complete, it is then compared to the next hash calculated on system startup. If the hash algorithm does not report the exact product as the previous calculation, then it is understood that the hardware has been tampered with. This protection contributes to system integrity in this case. This method is put in place to recognize any changes to the system to protect against any malicious additives.

Due to the increasing system complexity as the prototype develops, it is beneficial to offload portions of the security to help better the future overall utilization of the system, as well as providing a physical isolation provided with these types of hardware protections. As previously mentioned, the TPM provides many hardware security features useful to a full production model while taking a portion of the task handling away from the control system required for basic functionality. Though the current reference design does not utilize the full computing potential of the system, a more rigorous load on a production model could see performance benefits as well as enhanced security.

## **2.4 References**

- [1] TCG, "Trusted Platform Module (TPM) Summary, " [https://trustedcomputinggroup.org/wp-content/uploads/Trusted-Platform-Module-Summary\\_04292008.pdf](https://trustedcomputinggroup.org/wp-content/uploads/Trusted-Platform-Module-Summary_04292008.pdf), April 2008.
- [2] J. Bethencourt, A Sahai and B. Waters, "Ciphertext-Policy Attribute-Based Encryption," 2007 IEEE Symposium on Security and Privacy (SP '07), Berkley, CA, 2007, pp. 321-334

- [3] L. Guan, J. Lin, B. Luo, J. Jing and J. Wang, "Protecting Private Keys against Memory Disclosure Attacks Using Hardware Transactional Memory," 2015 IEEE Symposium on Security and Privacy, San Jose, CA, 2015, pp. 3-19.
- [4] M. E. Yalcin, J. A. K. Suykens and J. Vandewalle, "True random bit generation from a double-scroll attractor," in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 51, no. 7, pp. 1395-1404, July 2004.
- [5] Microsoft, The STRIDE Threat Model, Nov 2009. [Online] Available: <https://docs.microsoft.com/en-us/windows/deployment/windows-autopilot/add-devices>
- [6] Agari Inc., Secure Remote Attestation, Nov 2009. [Online] Available: <https://eprint.iacr.org/2018/031.pdf>
- [7] A. K. Kanuparthi, M. Zahran and R. Karri, "Feasibility study of dynamic Trusted Platform Module," 2010 IEEE International Conference on Computer Design, Amsterdam, 2010, pp. 350-355.
- [8] M. Hutter and R. Toegl, "A Trusted Platform Module for Near Field Communication," 2010 Fifth International Conference on Systems and Networks Communications, Nice, 2010, pp. 136-141.
- [9] E. A. Lee, "Cyber Physical Systems: Design Challenges," 2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC), Orlando, FL, 2008, pp. 363-369.
- [10] J. Zhan, J. Huang, L. Niu, X. Peng, D. Deng and S. Cheng, "Study of the key technologies of electric power big data and its application prospects in smart grid," 2014 IEEE PES Asia-Pacific Power and Energy Engineering Conference (APPEEC), Hong Kong, 2014, pp. 1-4.
- [11] Raspberry Pi Trading Ltd., Raspberry Pi 4 Computer, June 2019. [Online] Available: <https://static.raspberrypi.org/files/product-briefs/200206+Raspberry+Pi+4+1GB+2GB+4GB+Product+Brief+PRINT.pdf>
- [12] Lattice Semiconductor, "MachXO2 Family Handbook," LCMX02-7000HC datasheet, May 2013.
- [13] Texas Instruments, "TMS320F2833x, TMS320F2823x Digital Signal Controllers (DSCs)," TPMS320F28335 datasheet, June 2007.

## **CHAPTER 3**

### **TEST SETUP AND EXPERIMENTAL RESULTS**

#### **3.1 Introduction**

The purpose of this section is to evaluate the vulnerabilities within the cyber-physical reference design and deploy mitigation strategies. The intent is to apply the results of this reference design prototype to a metric that will analyze the importance of each technique to gauge the impact it may have on the system. Before that happens, risks must be addressed within each layer of the design to identify the correct mitigation strategies that could be applied to the available attack surface. The base reference design does not contain any security features, which exposes the system to outsider threats. To protect these openings from malicious intent, the openings must first be discovered by determining which data or operations are valuable to the system. If the data is considered essential for operation, chances are that an attacker will be able to exploit this data against the user. It is important to realize where these weaknesses exist in the system in order to protect against them. Finding the exposures in the system defines the attack surface, where one can establish corrective measures to combat these deficiencies. Particular to the reference design mentioned, the test setup will elaborate on the attack surface considered and the approach taken to alleviate defined concerns. Every layer considered will have specific obstacles that require various levels of attention, which are determined by the significance behind each level being compromised. This will classify considered threats, determine what a malicious attack might look like, and measure, to what degree, it will affect the system.

#### **3.2 Communication Layer Assessment and Results**

As mentioned in the introduction, to prepare any layer for test, an assessment must be performed to understand the system and its vulnerabilities. The communication layer

encompasses all contact dealt outside the system, where much of this interaction occurs within the Raspberry Pi. Logically, this makes sense, considering the intention of installing the Raspberry Pi into the reference design was primarily to disclose information from the system. However, knowing that the attack surface within this layer consists of the Raspberry Pi, the next step is to learn the vulnerabilities of this portion of the system; namely, how important roles in this layer can be at risk of being attacked. Referenced in Chapter 1, the CIA Triad can contribute to discerning what threats are necessary to protect against. The confidentiality of the data is the greatest concern because the data could be used against the user in the event of exposure. Put simply, the confidentiality of the communication layer is the first step into realizing the vulnerabilities brought by disclosing sensitive information.

Cyber-physical systems rely on communications as a large portion supporting the functionality of the system. It is imperative that data traverse in and out of the system layers to function properly. That said, the user must prohibit interaction of this data with an assailant, or the data may be taken and manipulated to be exploited against the user. Once the attack surface is identified, understanding how to protect certain data and communication paths is done by realizing the vulnerabilities of the attack surface. To help determine how the data is at risk, the storage and transmission medium are decided to be an influential point of contact for malicious activity. Namely, how the data is being sent and stored must be protected. Due to the webserver being the main point of contact from outside the system, it was decided to direct the attack vector towards the outgoing communication path of the system, directly from the Raspberry Pi. To enhance visualization of the communication path in the test setup, Figure 3.1 will illustrate the path and protocol connecting these devices together.



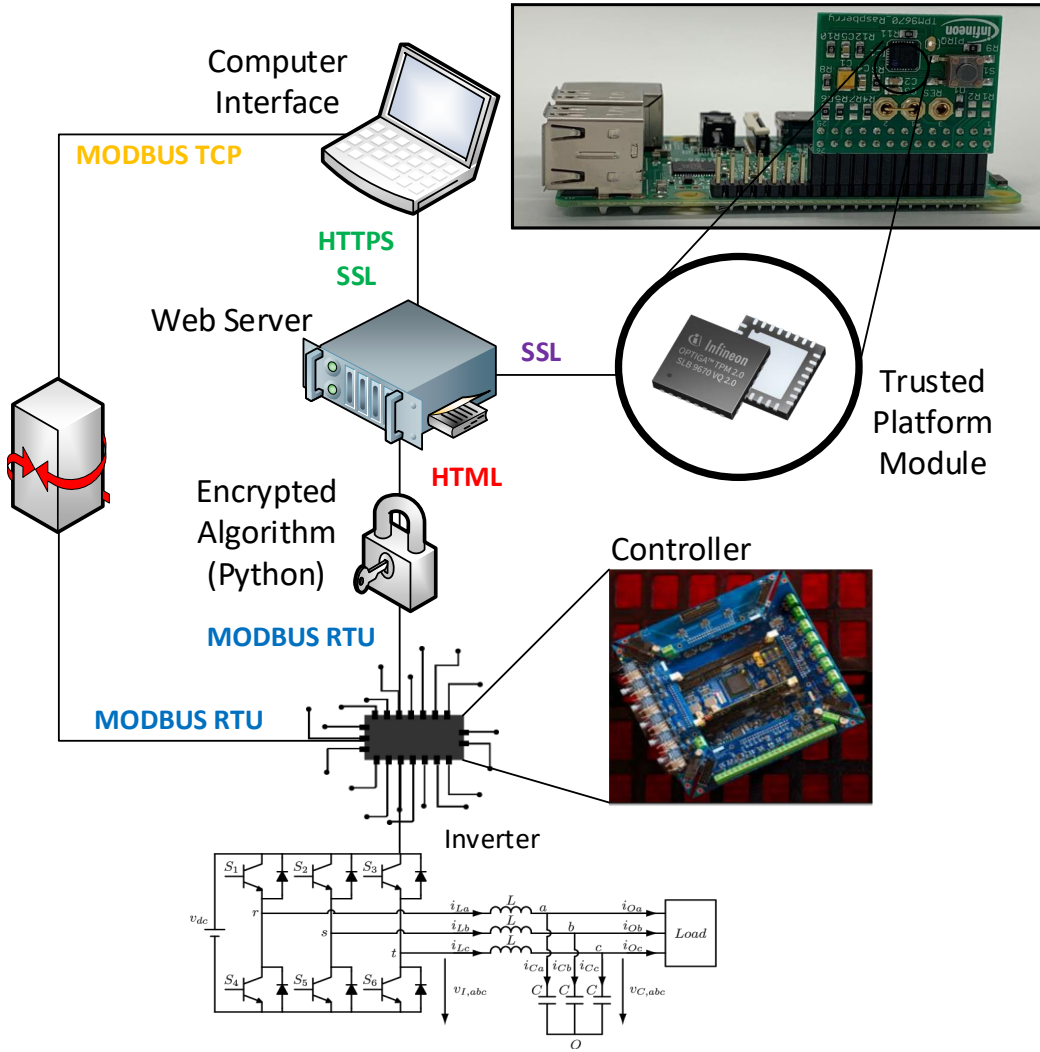


Figure 3.1. Communication path diagram.

Confidentiality is a concern in many areas of communication security and cyber-attack methods that target and identify data in a system. There are many attack scenarios, or attack vectors, to consider; however, this reference design has decided to focus on two critical attack methods for the communication layer. These two attacks are Man-in-the-Middle attacks and Password Attacks, or password cracking. Other possible attack scenarios to be addressed can be considered for other design implementation. These are common attack categories when considering data sharing between one entity and another. In this case, the entities are the web

server and another endpoint device; whether it be a utility/client handled interface or data storage. Man-in-the-Middle attacks typically focus on any data that can be intercepted between the web server and an endpoint [3]. A password hack can take this approach one step further by “sniffing” for specific details (i.e. user credentials) when using a Man-in-the-Middle attack.

Initially, to secure this communication pipeline to the user interface, the SSH or Secure shell protocol was to be used for the user to access the web server from a remote terminal. Once implemented, this created a secure pipeline into the raspberry pi, where the web server is running. The TPM would then oversee encrypting this communication algorithm, while device communication was dormant. This implementation had a couple drawbacks to be considered. First, the SSH tunnel is secured in a similar manner to other transport layer protocols, such as TLS; however, this would grant the end user far too much access to the system outside of the web server. This is not recommended. If the authentication fails, an unauthorized user will have much more access to the system, which is not desired for malicious individuals. Secondly, this is a reference design meant for a power electronics engineer that is looking to develop production hardware, such as a commercial solar inverter. Having to log into the system, to manually run programs required to start the server, would be simple to a developer that is familiar with the backend, but this is not a practical production method. Also, the TPM was not being fully utilized, as it was only being used during down time. No end user would necessarily have the knowledge or the desire to deal with such maintenance overhead with a product. It simply is not enticing as a selling point and has little marketable value.

Alternatively, the web server requires remote access, to multiple users, that does not relinquish administrator access to the entire system. Using SSL, or Secure Sockets Layer, the traffic to and from the Raspberry Pi would be encrypted using TLS v1.3. This method is like

SSH encryption, in that the communication path is completely secure; however, this method only pushes requested data that the web server permits. The SSL link communicates directly with the web browser which can be used on multiple platforms by the end user, to access the data and control the system. Standard websites can be accessed through the web browser by a Hypertext Transfer Protocol (HTTP) link which is strictly an application layer protocol. With SSL, the Hypertext Transfer Protocol Secure (HTTPS) can be utilized to meet the TLS v1.3 standard. Using this method, the user is restricted from any content that the web server does not wish to publish outside of the system.

### **3.2.1 Web Server Implementation Overview**

As mentioned in the previous section, the web server within the communication layer is one way in which the control layer can communicate information to the computer interface. Running the server on a Raspberry Pi 4 was a choice made for easy connectivity to the TPM. This decision was also made to host a Linux kernel, using a Raspberry Pi 4, for connecting an Apache HTTP server, that hosts the back end, with the Django web framework, which is in charge of the higher level web design [3, 4]. The web server interface can be viewed in any modern web browser such as Safari, Firefox, and Google Chrome. A preview of the web server user interface can be seen in Figure 3.2. The data shown in the interface is split into two different types: read and write variables. The read only variables are considered to simply be sensory data. This is data that you would never need to manually manipulate such as voltage and current readings. On the other hand, write variables are the control parameters that are put in place to manage the system in real time. These variables consist of the switching frequency, PI gain parameters, ac amplitude and frequency, and other parameters. Because the use of Django is incredibly versatile, the web interface is very flexible in that the developer has almost limitless



means that anyone with access to that line of communication, another user on the network for example, could interpret this data without any additional difficulty. To show an example of this, a test was initiated to try and obtain the credentials of a user logging in to a web server using a basic HTTP connection. In order to gain access to the web server data, the user must login, with approved credentials, to the main webpage. To capture network data, Wireshark v3.2.4 was used. Wireshark is a widely used network protocol analyzer that allows the user to monitor network activity at a very low level. This enables the network traffic to be monitored down to each individual packet. Since the initial data was not protected by HTTPS, the account username and password were easily discovered in the network traffic. This test can be depicted in Figure 3.3. The highlighted regions show the starting and endpoint destinations as well as a verification of the HTTP protocol being used. Also highlighted below is the HTTP packet payload, which is comprised of the username “admin” and password “password”; each being unencrypted.

To prevent this issue, the Apache server will shift to the HTTPS, or SSL, configuration. In order to utilize the Apache SSL configuration, the server will first need to generate an SSL certificate. An SSL certificate is a file that typically resides on the server, hosting the website. This SSL certificate will also enable the SSL/TLS encryption as a result of holding the public key, and other important information to the web server. One drawback to normal server SSL certificate storage is the placement of the key used for the certificate generation.

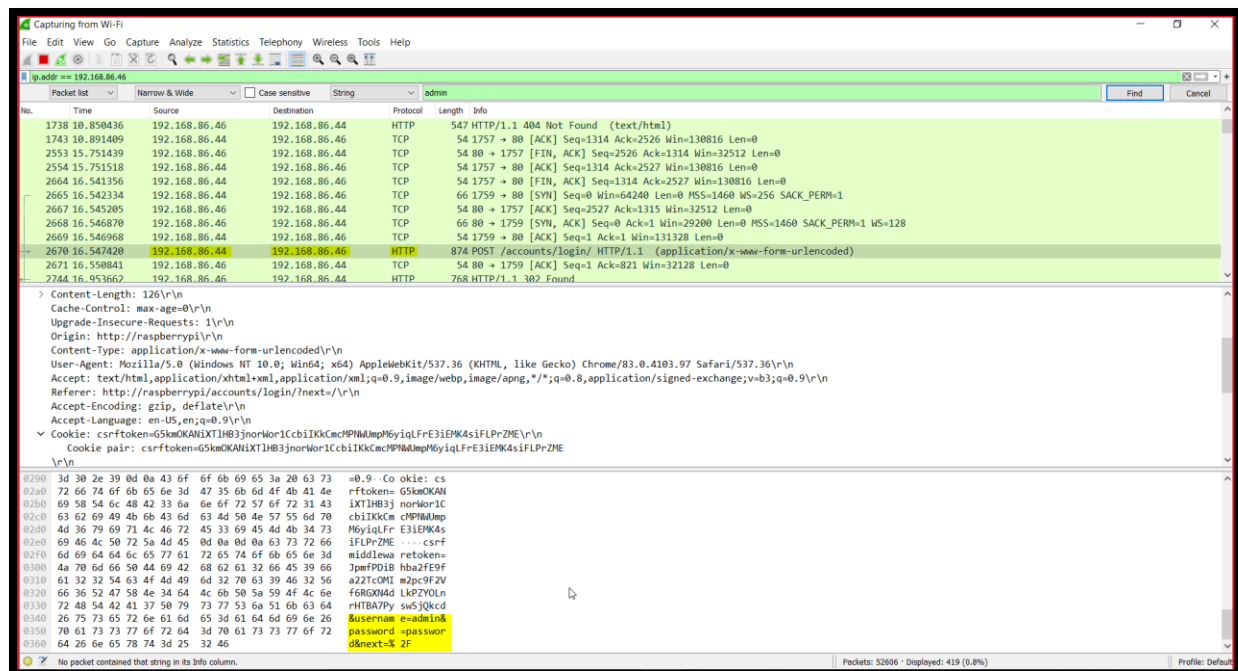


Figure 3.3. HTTP packet Wireshark monitor.

If an assailant were to compromise the server SSL certificate, the attacker may gain unrestricted access to the system, as well as access to other users. To secure this certificate, the certificate can be stored in or protected by the TPM. Infineon claims, the greatest strength to the TPM is to enable an application the use of cryptographic keys, while keeping them safe within the TPM. It can both generate and import externally generated keys. Each key has individual security control, which could be a password or also an enhanced authorization policy. These keys can then be certified by the TPM and used to certify other keys as well. But in order to manage internal memory efficiently, the TPM has the capability to wrap keys (encrypt) within the parent key and store them (the encrypted key) outside the TPM, while maintaining the overall security of the system. When it is time to use the key, the key is loaded back into the TPM. Only the specific TPM used to wrap the key can unwrap and use it. To establish a TLS session between a client and server with OpenSSL and TPM2-TSS engine, one must create an Open SSL configuration

file. This file will have the configuration used to create the Root Certificate Authority (rootCA), server, and client certificates [5]. One can now create a Root CA which would establish a chain of trust to the end user, allowing for the secure connection between certificate authority and end users.

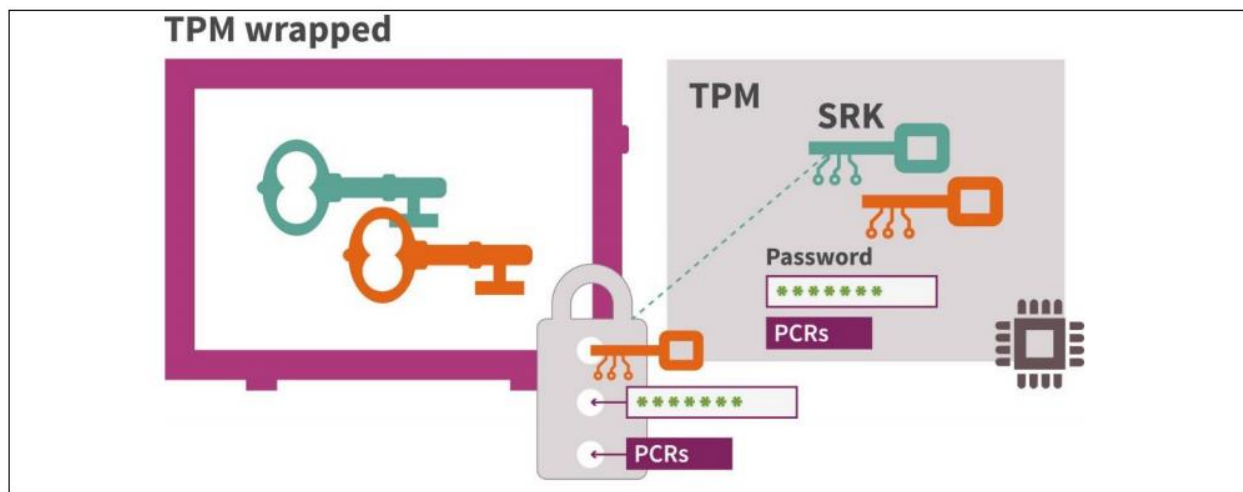


Figure 3.4. TPM 2.0 key wrapping [5].

Once the key has been generated and used to create the OpenSSL server certificate, the server can fully utilize the SSL configuration to host an HTTPS web server. The beginning of the session flow will show client and server greetings and acknowledge the key exchange. This can be viewed in Figure 3.5. Stated previously for this implementation, the focus is less on how the communication happens, but more on the confidentiality of the communication. Here, the TPM is an added layer of security, but not fully necessary to realize basic confidentiality of the HTTP packets. Regardless, the next step is to verify that this method can properly converse data to the end user without revealing information along the transmission path. To test this, a similar setup to the previous HTTP packet sniffing test was enacted, aiming to determine the contents of the packet payload. The same login process being tested was considered for the HTTPS service, as was HTTP. The results remained that each TLS packet was illegible, as expected. Seen in Figure

3.6. HTTPS packet Wireshark monitor, the user and endpoint address locations are the same as before, but now the packet type is TLS v1.3, encrypting the entire packet. The further highlighted section displays the properly encrypted payload with no legible login information confirming confidential communications.

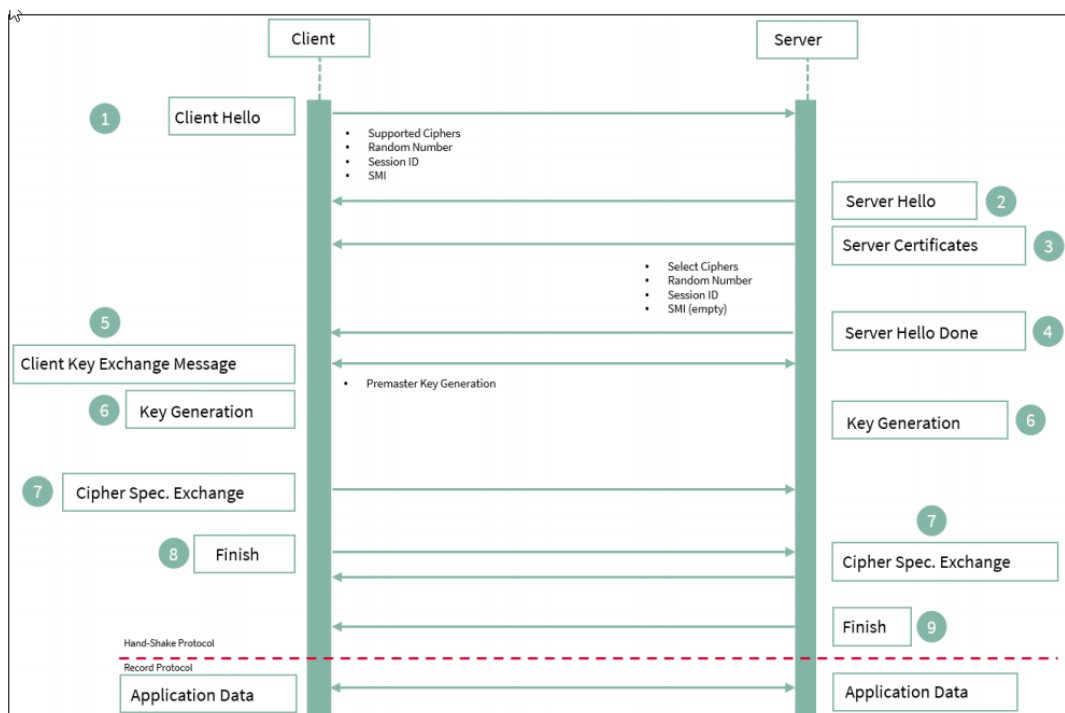


Figure 3.5. OpenSSL client/server session flow [5].



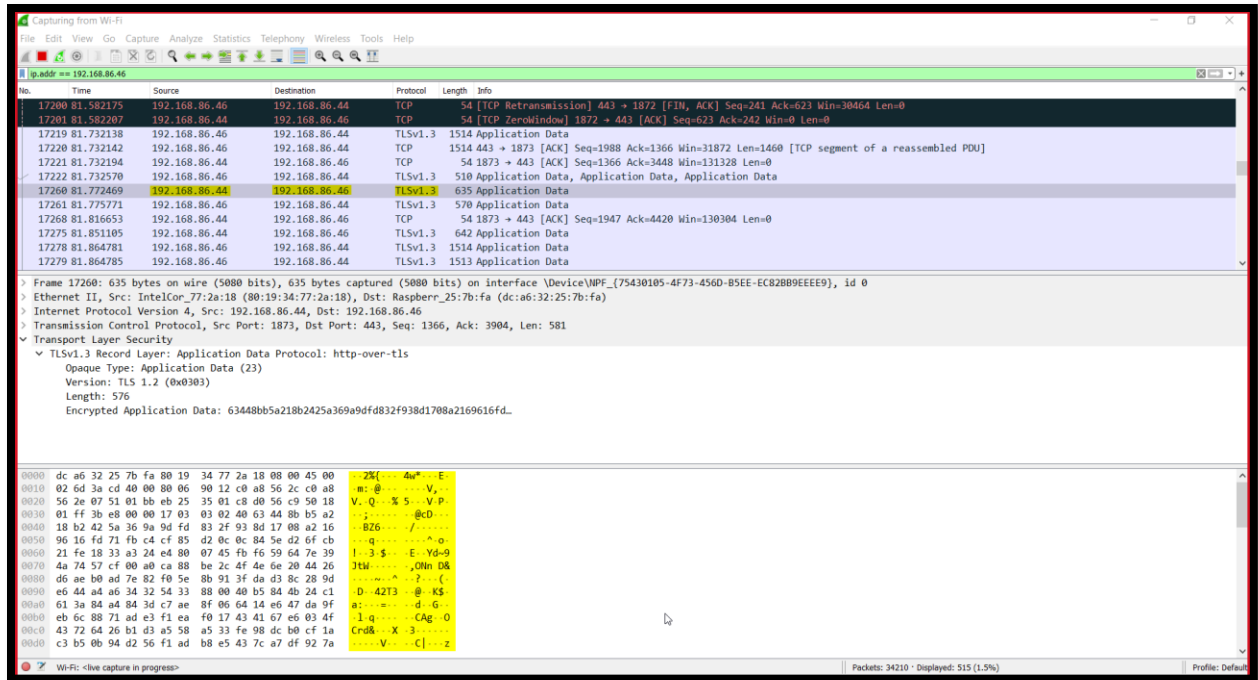


Figure 3.6. HTTPS packet Wireshark monitor.

### 3.3 Control Layer Assessment and Results

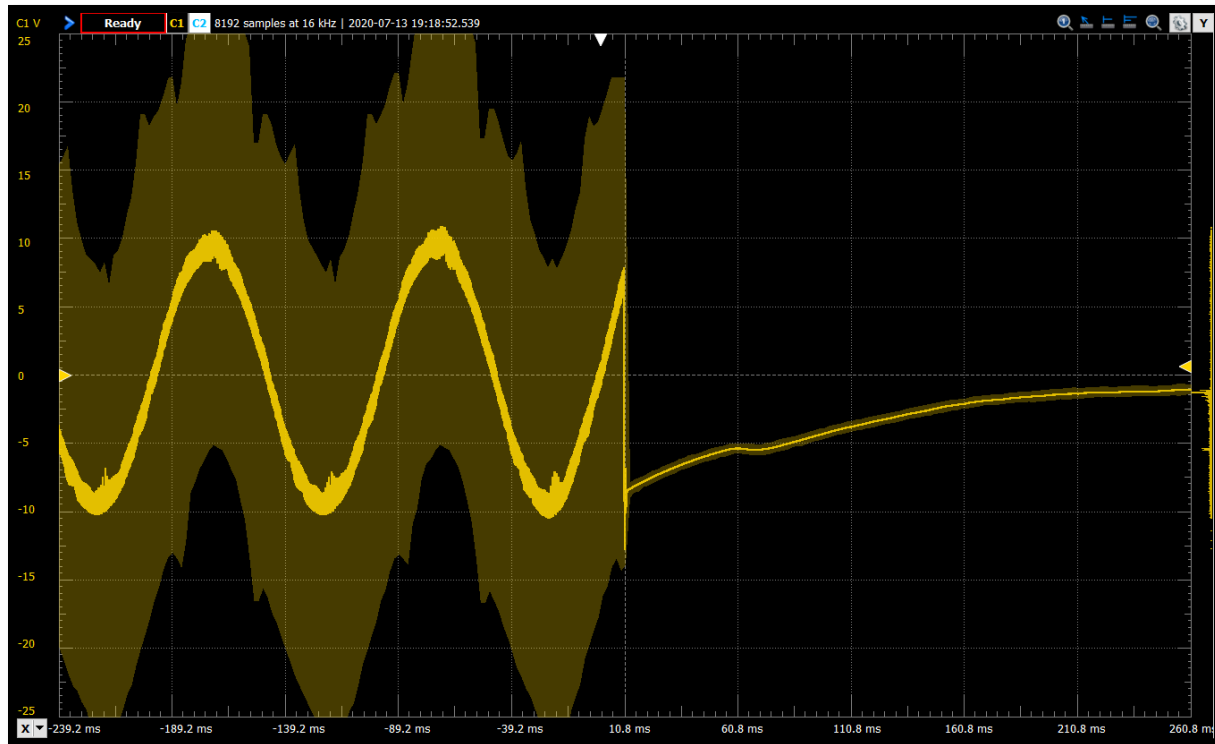
Through this assessment of the control layer, the most critical concept to consider is maintaining the integrity of the control layer throughout the design process. The chosen area of focus is the DSP where all the power electronic controls are physically operating. To maintain this integrity means to keep the data free from corruption. In order to do this, the system must be monitored for what is considered “bad data” and compared to what is believed to be “good data.” The system has current and voltage sensors that are measuring components of the 3-phase inverter in order to monitor the system health. The current and voltage sensors values are delivered to the control system through Analog-to-Digital Converters (ADC). As the name states, the ADC converts the analog voltages from the sensors to a 12-bit binary number stored within registers on the DSP, by successive approximation. More on how successive approximation works in [6]. With this value, the voltages can be compared digitally to one another for a system

health assessment. If the assessment fails to meet expected benchmarks, the system can be disconnected, or turned off, to prevent damage. This is whether the damage is from malicious intent or mechanical hardware failure.

For this test, the phase-A voltage is monitored under normal 3-phase load conditions. To obtain power flow, in this case, a low voltage 3-phase motor is acting as the load to achieve a constant 3-phase output. Once power is flowing, the voltage can be monitored and saved as a known legitimate value for comparing against unwanted measurement conditions. Measurements during test are polled by the ADC rapidly, but only saved periodically when each Modbus packet is transmitted, in order to insert a small delay to account for voltage noise transients. This is done for a couple reasons. The first reason being code simplicity that will require less system interrupts. Secondly, the Modbus transmission time is defined and can change to meet the needs of the system. Once polled, the value is stored in a register until the next cycle. When the measurement reoccurs in during the next cycle, the value is stored in another concurrent register to compare the current and previous voltage values. This is a similar approach to what some consider “de-bouncing.” Essentially, the purpose of this method is to verify that the voltage has maintained it’s value for some period longer than noise transients would typically occur. This technique will help rule out false positive evaluations on poor voltage readings.

The system will rule that a bad condition has occurred when three consecutive hazardous voltage readings have been interpreted. If this occurs, the voltage amplitude will be commanded to zero, for the output of each phase. Following the test, the data was collected and can be observed in Figure 3.7. The beginning of the phase-A voltage output reading is observed as a typical ac sinusoidal waveform. This ac output voltage is directly related to the dc link input voltage of the inverter. Rather than feeding the ADC faulty data, the dc link input voltage was

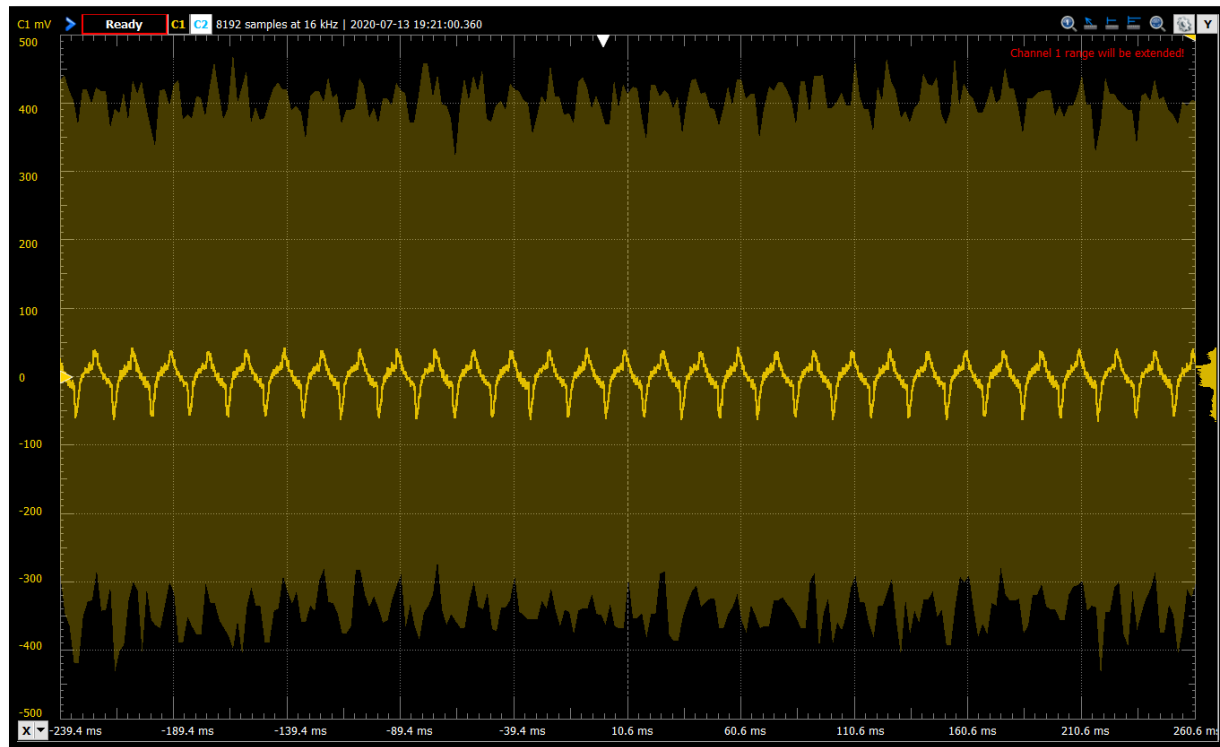
incremented to physically step up the output voltage, and push the ADC reading out of a decidedly valid boundary. The results of this can also be seen in Figure 3.7 where the voltage is commanded to zero on the phase-A voltage leg. Do note that the output capacitor voltage takes time to bleed off into the load, which is the reason there is not an instantaneous jump to zero volts.



*Figure 3.7. Phase-A voltage cutoff.*

During the invalid output condition, it is important to note that the amplitude is simply commanded to zero, rather than the system being completely shut down. This can be verified in Figure 3.8, showing that low voltage switching noise can be seen once the output is commanded to zero. The system is not shut down entirely so that the other essential components can continue to be monitored, or changed, to combat the system failure, malicious or not. Once the output is commanded to zero, corrective measures can be taken to verify the state of the system and the

output. Whereas, if the entire system went without power, the system would possibly need to be isolated from the circuit before evaluation under safe operating conditions.



*Figure 3.8. Phase-A low voltage output switching.*

### 3.4 Hardware Layer Assessment and Results

The leading focus of the hardware layer assessment will be the availability of the hardware. Meaning, that the hardware needs to be safe from defects or compromised controls. The chosen attack vector will target shoot-through currents on the individual legs of the inverter. The mitigation strategy of choice is forced dead band to employ shoot-through protection. Shoot-through occurs when two adjacent switches are active concurrently, resulting in a short circuit across the dc link. This is a key challenge when designing an effective protection approach, because the short circuit loop must be blocked in a very short time period, 3 to 10  $\mu\text{s}$  [7]. A standard 3-phase inverter is shown in Figure 3.9, where the shoot through current can be

observed through leg one of the inverter, possibly destroying switch  $S_4$ . The high current often comes from the dc link capacitor in parallel with the dc voltage source, which has stored energy that is now shorted across the switching leg. Typical shoot through current, seen longer than 10  $\mu\text{s}$ , can be fatal to the switches as they are not made to handle that much current instantaneously.

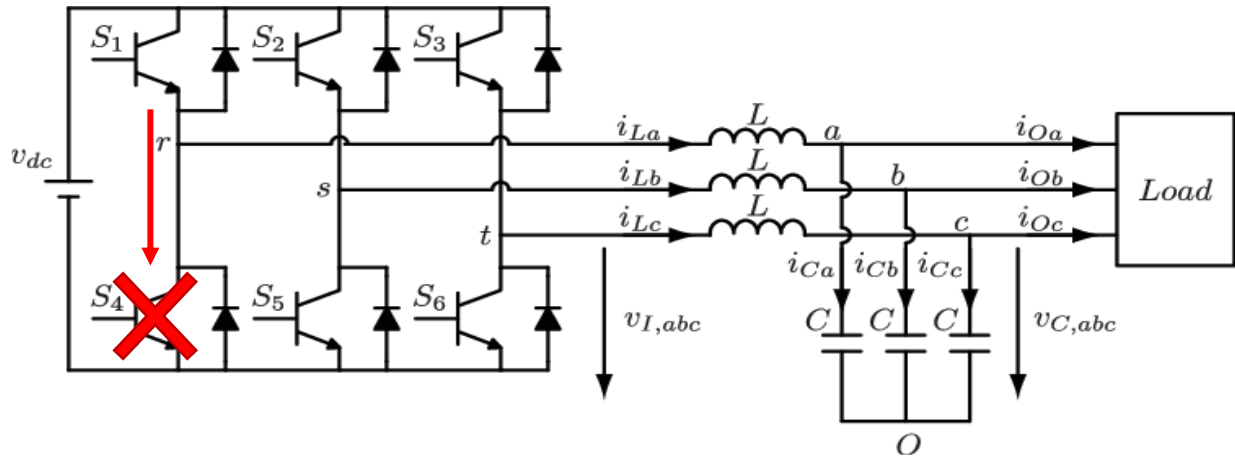


Figure 3.9. Basic 3-phase inverter topology.

To protect against shoot through currents, most control systems implement dead time. Dead time is the time in between switching periods where neither switch is conducting, as to not short circuit the dc bus to ground. Ideally, the switches turn on and off instantaneously, but this is not the case in practice. Ideal dead time effect can be seen in in Figure 3.10. A diagram of more accurate switching period waveforms can be seen in Figure 3.11. To learn more about the dead time effect, [8] does a good job of explaining the fundamentals when applied to a standard 3-phase PWM inverter.

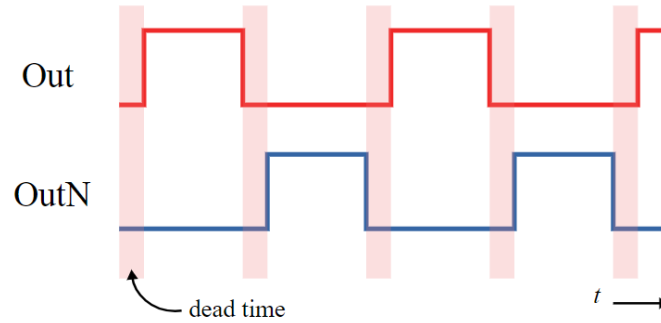


Figure 3.10. Dead time effect.

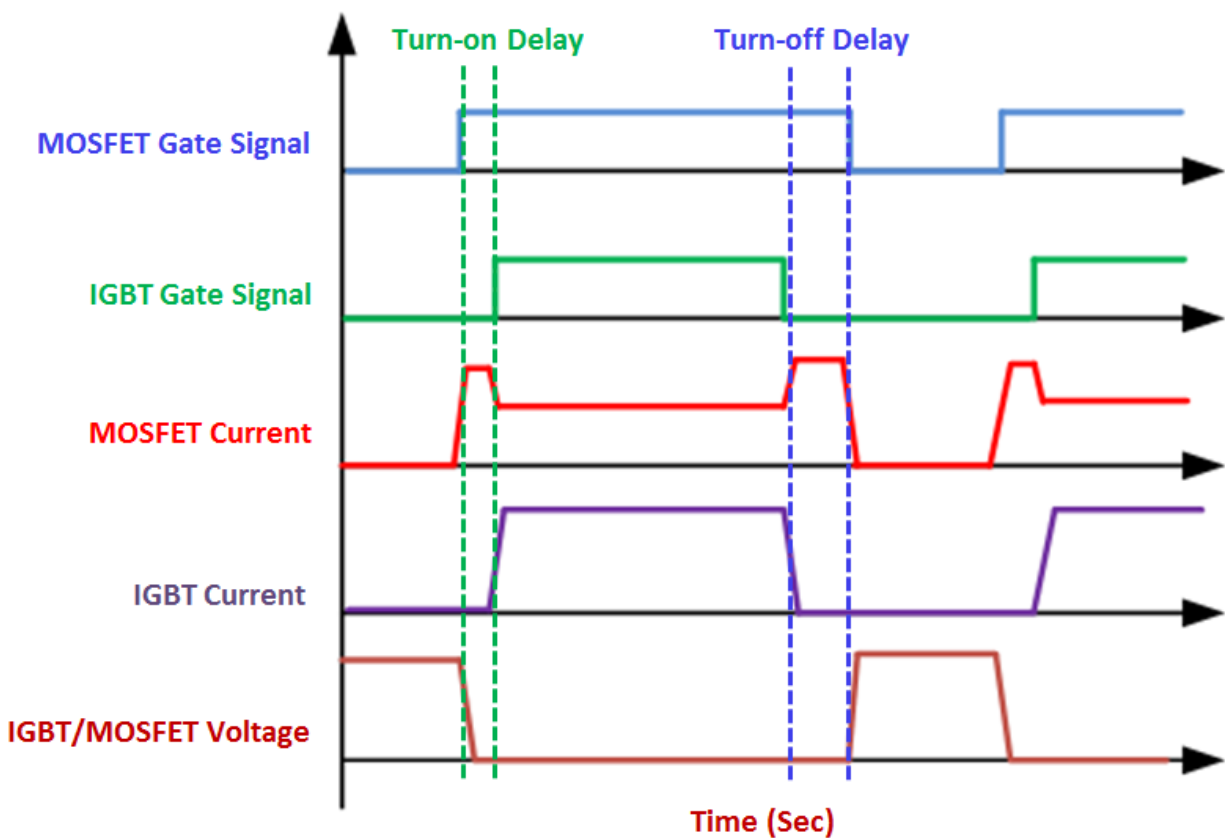


Figure 3.11. Accurate on/off delay wave forms.

Though, programming for dead time is standard practice in control systems, the present focus is to secure the availability of the hardware layer, even if the control layer is compromised. Presume that the control layer is compromised, and the user has malicious intent to remove the deadtime from the control system programming. There is a need for infrastructure in place that

will protect against this problem. The method utilized in this design relies on the CPLD to instantiate hardware interlocks, in order to force the signals opposite from one another. This implies that a malicious user, even removing dead time completely to command all gate signals active, will not be able to force a shoot through current.

To test this method, the CPLD was programmed using VHDL to instantiate the circuit within the CPLD hardware. Stated in Chapter 2, all digital signals to and from the DSP are routed through the CPLD. Rather than having to design a new Printed Circuit Board (PCB) layout whenever the topology changes, the digital signals are routed to their destination through the CPLD architecture. Since obtaining worst case results, as well as verifying the implementation of this method, could result in a shoot through current, it was decided that a test bench should be built to prove this new method. A shoot through current at any operating voltage could be detrimental to the power electronics hardware, and it was decided to avoid any catastrophic consequences to the system. The test bench contains pseudo PWM waveforms that represent the controlled switching signals, sent out to the gate drivers. This test bench was then simulated against the running CPLD code, where the results showed adjacent switches had gone into the same state, and the output sent to the gate drivers was then turned off. These results can be seen in Figure 3.12. This verifies hardware protection by having outputs switch low much faster than 10 us, even without deadtime. This is not protecting against active uptime. This method is more focused on the overall availability of the power electronics by removing the possibility of permanent damage of the system resources.

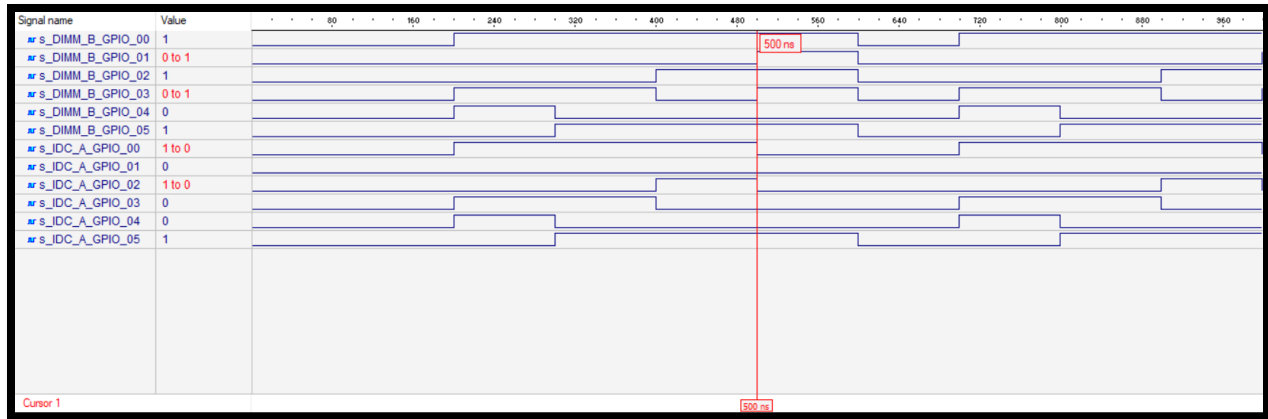


Figure 3.12. CPLD shoot-through protection simulation.

### 3.5 References

- [1] F. Pasqualetti, F. Dörfler and F. Bullo, "Attack Detection and Identification in Cyber-Physical Systems," in *IEEE Transactions on Automatic Control*, vol. 58, no. 11, pp. 2715-2729, Nov. 2013.
- [2] A. M. Shabut, K. T. Lwin and M. A. Hossain, "Cyber attacks, countermeasures, and protection schemes — A state of the art survey," *2016 10th International Conference on Software, Knowledge, Information Management & Applications (SKIMA)*, Chengdu, 2016, pp. 37-44.
- [3] Django Software Foundation, Django Project, July 2005. Accessed on: July 2020. [Online] Available: <https://www.djangoproject.com/>
- [4] The Apache Software Foundation, APACHE HTTPS SERVER PROJECT, April 1995. [Online] Available: <https://httpd.apache.org/>
- [5] Infineon, Optiga TPM SLx9670 Application Note, April 1995. [Online] Available: [https://www.infineon.com/dgdl/Infineon-OPTIGA\\_TPM\\_SLx9670\\_TPM\\_2.0-ApplicationNotes-v01\\_00-EN.pdf?fileId=5546d46271bf4f920171c5598a3a0e7b](https://www.infineon.com/dgdl/Infineon-OPTIGA_TPM_SLx9670_TPM_2.0-ApplicationNotes-v01_00-EN.pdf?fileId=5546d46271bf4f920171c5598a3a0e7b)
- [6] C. E. Luo and L. Zhu, "Jittered random sampling with a successive approximation ADC," *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Florence, 2014, pp. 1798-1802, doi: 10.1109/ICASSP.2014.6853908.
- [7] R. Lai, F. Wang, R. Burgos and D. Boroyevich, "A shoot-through protection scheme for converters built with SiC JFETs," *2009 IEEE Energy Conversion Congress and Exposition*, San Jose, CA, 2009, pp. 2301-2305, doi: 10.1109/ECCE.2009.5316426.
- [8] Jong-Woo Choi and Seung-Ki Sul, "Inverter output voltage synthesis using novel dead



time compensation," in *IEEE Transactions on Power Electronics*, vol. 11, no. 2, pp. 221-227, March 1996, doi: 10.1109/63.486169.

## CHAPTER 4

### COST/BENEFIT ANALYSIS OF MITIGATION STRATEGIES

#### 4.1 Introduction

Cost-benefit analysis is an organized approach to determine the strengths and weaknesses within any process. This method examines features within an initiative to appropriately gauge the expenses required to incorporate these features. Attributes are categorized by economical and performance costs, to realize the advantage of including these attributes, compared to how greatly these features contribute to the efficiency of the operation. In other words, the goal is to determine, to what degree, each proposed strategy will benefit the program, and the cost of each implemented strategy, respectively. The usefulness of this analysis is in its ability to accurately place value on components within a system. In the event that one cannot afford all the elements of a design process due budget constraints, this process allows for one to then choose which features have more value and then neglect those features with benefits that will not outweigh the costs.

Within this reference design, security features will be added to the condensed cyber-physical system to assess the benefits that each security will deliver. The analysis will begin with no securities in place to demonstrate preexisting vulnerabilities within the system. This approach will be monitored on each of the three defined layers of the reference design: communications layer, control layer, and hardware layer. For a more in-depth look at each layer, reference Chapter 2. An example will be given of what each layer, with little to no security features, demonstrates as a base line. From here, the approach will add security techniques to every layer with mitigation for each determined vulnerability. Each layer will then individually appraise the effectiveness of the mitigation strategies implemented to justify the costs of each method.

Another important piece to consider in each layer is the economy of scale. On an initial prototype, the one-time bill of material costs considered with the purchase of the security assets may seem unreasonable for one device, but on an economy of scale, the investments become much more incremental.

## **4.2 Overhead and Computation Costs**

The leading concern most engineers will consider during the design is monetary costs of a product. They will use that as a basis as to whether each examined component is worth implementing. There are more options to consider before deciding the overall value of a device. One important factor is the physical effect these implementations will have on a system. For example, in cyber security, an important measurement to consider is the computational impact of adding more complex tasks and how that effects other operations. If some task takes significant bandwidth and resource allocation, then adding an additional concurrent task could be overbearing on the system. In contrast, if there are too many tasks already limiting the reserves of the system, an alternative element could be included to help supplement the computational load on the system. To estimate computation costs, each scenario must be viewed relative to the environment that the components will be placed in.

### **4.2.1 Appending Communication Security**

The base line for the communication layer is a web server being hosted on the Raspberry Pi 4. As mentioned in Chapter 3, the web server is hosting an HTTPS server for a secure channel to the user interface. The added security in this layer is the use of the HTTPS protocol over the HTTP, which provides the secure communication line as well as certificate authority. HTTP initialization is much faster due to the lack of SSL handshaking between the server and the client. This has been discussed previously in Figure 3.5. OpenSSL client/server session flow [5].

In context of program cycles, the time required for both HTTP and HTTPS to initialize is moderately different with HTTPS taking a bit longer. However, when the system is pushing a large amount of dynamic web server data, for example, the time required to initialize each connection will seem insignificant compared to the overall loading time. Applying HTTPS to the web server communication would appeal to most developers who are more focused on the privacy and authenticity of the connection, rather than those who are focused on the speed of the connection.

The TPM was added as an additional level of security in the communications layer. This is not necessary in order to realize the basic security features of the HTTPS protocol, but this enhances these security features. The TPM does this through a few different techniques, all of which are not implemented, to reduce complexity in the reference design. More on these features and their implementation can be found in Chapter 2 and Chapter 3. The emphasis applied in this reference design prototype is focused on securing the SSL Certificate key. The important resources to consider reside in the Raspberry Pi controller. This architecture is run on a Broadcom BCM2711, 64-bit Quad core Cortex-A72, running at 1.5 GHz, and equipped with 4 GB of RAM. These components are particularly excessive considering the capabilities of this system far exceed the resources required to operate the web server and user interface. The overhead, introduced by the TPM key exchange, will not be significant to the performance of the Raspberry Pi. However, comparing the performance of the SSL key exchange, for this HTTPS web server, to another HTTPS web server that does not have TPM key management, will show differences in performance [2]. Having a TPM manage the Certificate will also lead to increasing connection times of clients attempting to access the server in a short period of time. This is because each client is required to wait for the previous client to finish with the TPM. Whereas, in

a system without a TPM, client connection times are not impacted as much since the processor can use multiple threads and cores to handle these tasks. This can be noted in Figure 4.1. The significance of the performance impact, shown in Figure 4.1. Performance Breakdown of TLS Accesses [2] will be decided by the developer. The decision to implement this method has considerable advantages that can outweigh the cost if security is the main concern. There have also been revisions to the TPM architecture since this publication, possibly lessening the multiple user connection times.

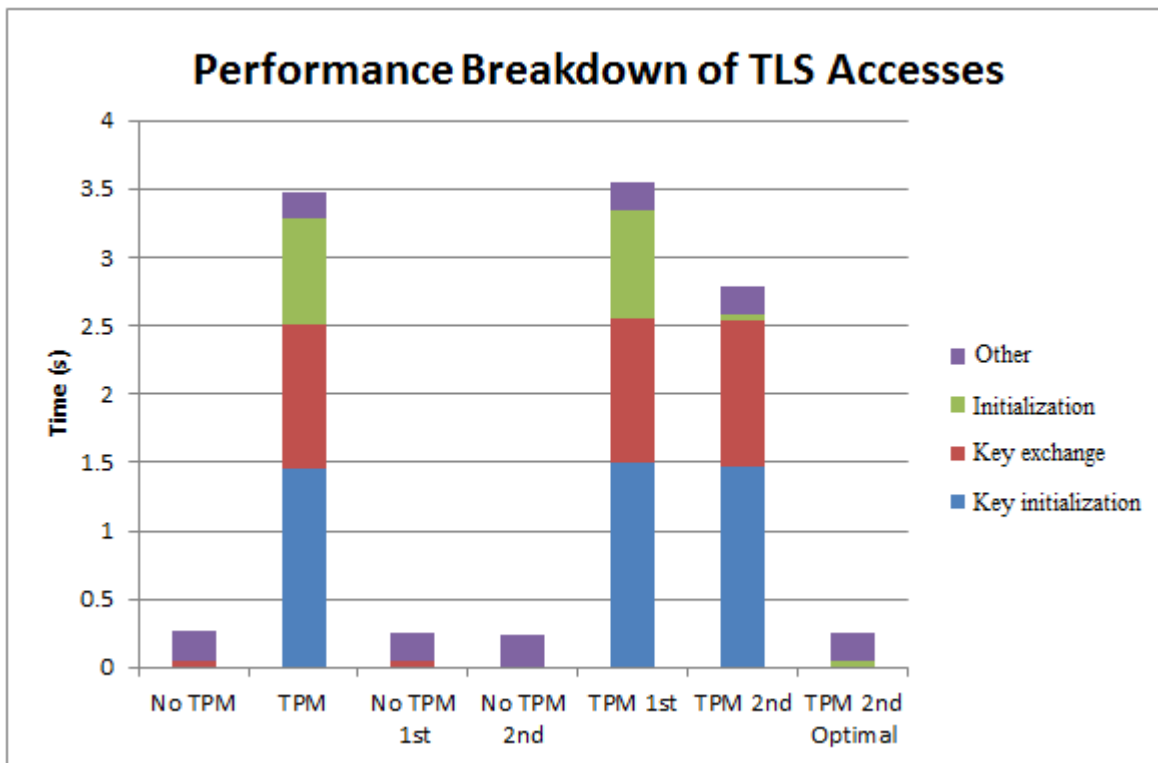


Figure 4.1. Performance Breakdown of TLS Accesses [2].

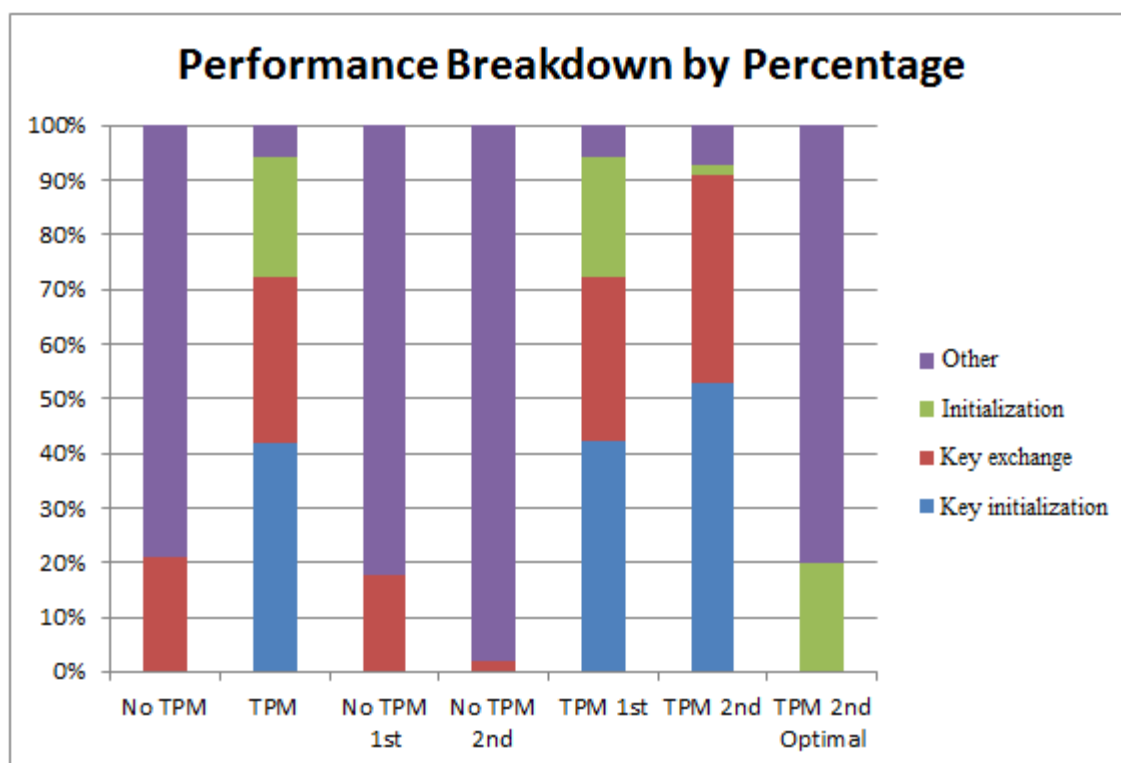


Figure 4.2. Performance Breakdown by Percentage [2].

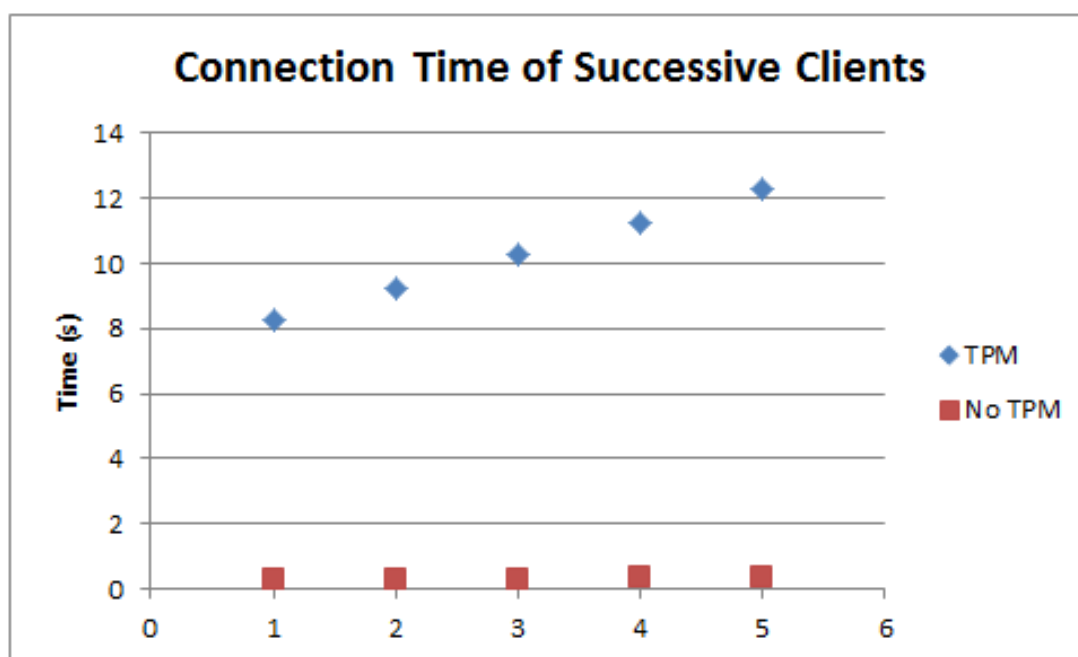


Figure 4.3. Connection Time of Successive Clients [2].

#### 4.2.2 Appending Controller Security

The power electronics controller being used in the reference design prototype is a Texas Instruments Delfino TMS320F28335 digital signal processor. This processor runs at a 150 MHz system clock with a plethora of general-purpose inputs and outputs. The DSP operates the power electronics hardware through various control algorithms. The code running the 3-phase inverter has a simple PI control loop, with a basic PWM inverter control scheme. This device can handle much more than what is being utilized, but the computation costs are relative to the use case of the developer. The security implementation for this design is rather trivial, compared to a complex control scheme, which would infer that the overhead costs would not interfere with the overall operating conditions.

The security technique, applied to the control layer, determines whether the 3-phase output is stable, and will command the output off, if the system detects an instability. This is achieved through additional code that pulls the sensory data after each Modbus packet transmission. The additional code can be found in the Appendix A, under DPS C code. It has been verified that the code will take a few clock cycles to move the data from the Modbus holding registers to local memory storage within the DSP. This is time that the system could be executing another task, which is considered a computational cost.

```

826          if (Reg[33] >= 1600)
C$L14:
3382c0: 761F0304    MOVW        DP, #0x304
3382c2: 1B210640    CMP         @0x21, #1600
3382c4: 680A        SB         C$L15, LO
828          ADC_valid3 = ADC_valid2;
3382c5: 761F0300    MOVW        DP, #0x300
3382c7: 9209        MOV         AL, @0x9
3382c8: 9602        MOV         @0x2, AL
829          ADC_valid2 = ADC_valid1;
3382c9: 9206        MOV         AL, @0x6
3382ca: 9609        MOV         @0x9, AL
830          ADC_valid1 = 1;
3382cb: 56BF0106    MOVB        @0x6, #0x01, UNC
831          }
3382cd: 6F05        SB         C$L16, UNC
834          ADC_valid1 = 0;
C$L15:
3382ce: 761F0300    MOVW        DP, #0x300
3382d0: 2B06        MOV         @0x6, #0
835          ADC_valid2 = 0;
3382d1: 2B09        MOV         @0x9, #0
840          if (ADC_valid3)
C$L16:
3382d2: 9202        MOV         AL, @0x2
3382d3: 6105        SB         C$L17, EQ

```

Figure 4.4. Control code Assembly breakdown.

Only a few move commands are considered when running the additional protection code. Presume that Code Composer Studio Assembly breakdown is correct, and that the move takes two clock cycles to compute. The operating frequency of the DSP is 150 MHz. Referenced in Eq. (4-1), this process only takes about 13 ns per move. With a minimal use of this, the operation will take no time at all.

$$\text{Clock Cycle Period} = (\text{Operating Frequency})^{-1} = \frac{1}{150\text{MHz}} = 6.667 \text{ ns} \quad (4-1)$$

Though the code execution does take resources away from the core modules, the impact on this system is irrelevant when considering the speed at which the processor is operating. Even if the control scheme were to increase in complexity, this additional operation in the code should not hinder the overall performance of the system.



### **4.2.3 Appending Hardware Security**

Though the CPLD is not within the hardware layer, it does have a significant role in the hardware protection application. Put simply, the goal of appending hardware security to the prototype is to protect against shoot-through fault currents. This is further explained with the hardware layer assessment in Chapter 3. There are a few ways to protect against these types of faults; gate drivers, with physically programmable dead band, exist to do just that. The hardware would have to go through a major revision to place these gate drivers, without them previously in the layout. The decision was made to sacrifice computational costs of the CPLD resources, rather than economical costs of purchasing more parts. This also required much less engineering work, which is another valuable component to the design process.

The CPLD operates at a much higher frequency than the DSP running the control code. This, combined with the fact that the CPLD is extremely underutilized for this purpose, allows for plenty of overhead in the design process. The CPLD could be used for managing communication as well as other data; however, the CPLD is simply acting as a routing fabric, connecting the digital signals across the architecture. The logical implementation of the hardware security is trivial, compared to the complexity and speed of the CPLD. Through VHDL programming, adding a logical inverter and a small arithmetic operation will go unnoticed in this environment. Unless the developer is using a dataflow approach to conserve any spare resources, this method would prove to be worth the computational costs because of how little they affect the system.


## **4.3 Economical Costs**


There are always trade-offs when considering the economic value of a product and its components. The important point to take away from this economical cost evaluation, is that the

chosen product will meet user needs and budget. For this design, monetary value can only be assigned to tangible security features. Programming implementations are difficult to evaluate, when considering other overhead costs such as engineering time and management; therefore, these considerations will not be directly addressed in this evaluation. This is also due to the controller hardware being required for base operation, so negligible monetary value is required to add security through code updates. The focus points will consider hardware application choices such as the TPM and Raspberry Pi platforms.

To a hobbyist or small engineering firm, the upfront costs associated with implementing additional hardware features to a product can be significant. Provided the design team is established, these upfront costs can be offset through economy of scale. The economy of scale is the idea that a purchase, made in large quantities and in advance, will result in a much lower individual price per product. Devices, such as the TPM and Raspberry Pi, can often benefit from this method. Although, large purchase orders are not an option and can even be a hinderance. Noted through Figure 4.5, sometimes purchasing items in bulk is not an option, depending on the retailer and stock options. Though, seen in upcharge some retailers will offer bulk options but will heavily upcharge a buyer to fulfill these orders due to low stock or other reasons. Economy of scale does not always benefit the consumer.

**VILROS** RASPBERRY PI ▾ ARDUINO ▾ MICROBOARDS ▾ NEWS LEARN ▾ B2B ▾ COVID-19


Search 








### Raspberry Pi 4 Model B - 2GB RAM


Model #: RP4B2GB In Stock








**\$35.00**

QTY  

**ADD TO CART**

 **Raspberry Pi**  
APPROVED RESELLER

Product description

**LIMIT 1 PER HOUSEHOLD**

Figure 4.5. Vilros stock limitation.

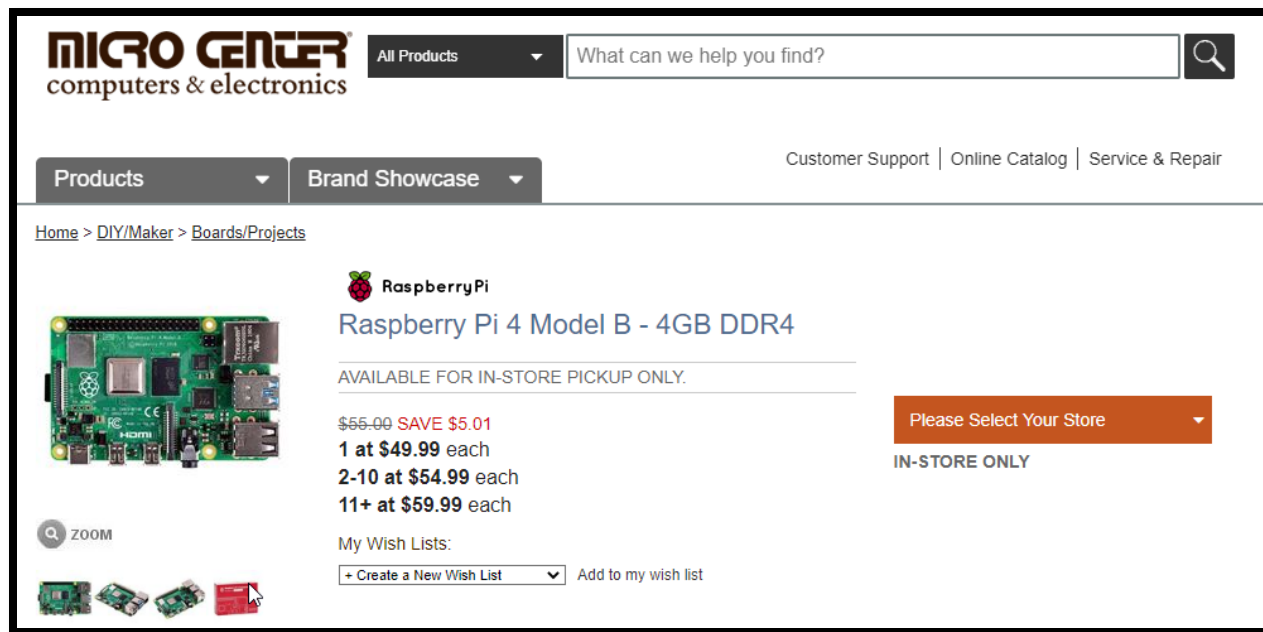


Figure 4.6. MicroCenter order upcharge.

Alternatively, the TPM can greatly benefit from the effects of economy of scale. Through a known online retailer, Digikey, the TPM used here is approximately \$3.70; seen in Figure 4.7. Within the same device listing, a table exists to display the economy of scale and how that can affect the order. On this same page, there is another listing for the same device, but with a minimum purchase quantity of 2500. The price here is discounted at less than half of the original purchase cost. This example is one of many that shows if a product is ordered in advance at a large agreed quantity, then the price can be discounted heavily. This is a great illustration on when the economic costs are offset enough to allow a purchase to be viable if the investment can be made.

**SLB9670VQ20FW785XUMA1**

[Datasheet](#)

Digi-Key Part Number: SLB9670VQ20FW785XUMA1CT-ND  
 Manufacturer: [Infineon Technologies](#)  
 Manufacturer Part Number: SLB9670VQ20FW785XUMA1  
 Description: TPM  
 Manufacturer Standard Lead Time: 12 Weeks

**Price & Procurement**

0  
[Check Lead Time](#)

**QUANTITY**

[Add to Cart](#)

All prices are in USD.

PRICE BREAK	UNIT PRICE	EXTENDED PRICE
1	3.67000	\$3.67
10	3.29800	\$32.98
25	3.11840	\$77.96
100	2.56390	\$256.39
250	2.30060	\$575.15
500	2.21744	\$1,108.72
1,000	1.84325	\$1,843.25
2,500	1.77395	\$4,434.88

**Documents & Media**

Datasheets: [SLB 9670 TPM2.0](#)  
 Online Catalog: [OPTIGA™ TPM Series](#)

**Product Attributes**

TYPE	DESCRIPTION	SELECT ALL
Categories	<a href="#">Integrated Circuits (ICs)</a>	<input type="checkbox"/>
	<a href="#">Embedded - Microcontrollers - Application Specific</a>	<input checked="" type="radio"/>
Manufacturer	<a href="#">Infineon Technologies</a>	<input type="checkbox"/>

Figure 4.7. TPM Economy of scale example.

#### 4.4 References

- [1] T. M. Mbelli, "Computational Secure ORAM (COMP SE-ORAM) with  $[\omega](\log n)$  Overhead: Amazon S3 Case Study – Random Access Location," *2019 IEEE Cloud Summit*, Washington, DC, USA, 2019, pp. 99-102, doi: 10.1109/CloudSummit47114.2019.00022.
- [2] Kun Li, Michael Maass, and Mike Ralph, Carnegie Mellon University School of Computer Science, "A Type-safe, TPM Backed TLS Infrastructure" [Online] Available: [https://www.cs.cmu.edu/~mmaass/tpm\\_tls/report.html](https://www.cs.cmu.edu/~mmaass/tpm_tls/report.html)

[3] Digikey, SLB9670VQ20FW785XUMA1 TPM Product Page, [Online] Available:

[https://www.digikey.com/product-detail/en/infineon-](https://www.digikey.com/product-detail/en/infineon-technologies/SLB9670VQ20FW785XUMA1/SLB9670VQ20FW785XUMA1CT-ND/9922617)

[technologies/SLB9670VQ20FW785XUMA1/SLB9670VQ20FW785XUMA1CT-](https://www.digikey.com/product-detail/en/infineon-technologies/SLB9670VQ20FW785XUMA1/SLB9670VQ20FW785XUMA1CT-ND/9922617)

[ND/9922617](https://www.digikey.com/product-detail/en/infineon-technologies/SLB9670VQ20FW785XUMA1/SLB9670VQ20FW785XUMA1CT-ND/9922617)

## CHAPTER 5

### CONCLUSIONS AND RECOMMENDATIONS

#### 5.1 Summary of Conclusions

This cyber-physical reference design is developed to assist power electronics engineers in selecting security components, to be included into the system, throughout the design process. The demand for this reference design is derived from numerous production power electronic devices deploying with insufficient security features. Even so, when these devices have security features considered, these features are normally an afterthought of the design and are not typically ideal solutions. This design proposes a three-layered approach to overcome this obstacle and teach the user how to address certain security concerns thoroughly. The three layers consider communications, controls, and hardware templates of a power electronics system. The system is evaluated for potential threats that can be identified through a threat model, which will show common weaknesses in each environment. Once these weaknesses are discovered, the risks are identified and characterized in order of criticality. Subsequently, serious risks can be focused on in order to mitigate most issues. Mitigation strategies are proposed, tested, and evaluated for a prototype design on an economical and computational cost metric. Costs of most elements are considered and justified for the implementation. The implementation is decided by the design engineer, and the reference design is the tool to help balance the trade-off choices between the monetary costs and value that it brings into the system. In conclusion, the reference design work detailed above will allow for a flexible and secure design methodology that can then add value to the end user design specifications. Future work is recommended following the completion of the work.

## 5.2 Recommendations and Future Work

Minor recommendations can be implemented with utilizing the TPM for other tasks outside of certificate safety. The TPM can be taken out of the communication layer and then be added to the controls layer. This could benefit any communication bus architecture, such as Wishbone, implemented directly on the CPLD communication bus. This would provide secure encryption, hardware hash generation, and many more security options that are not available in the current topology. Other areas for improvement would be to build a phone web application to better interface with the device. However, security may become a risk when involving more platforms. This reference design can then be expanded upon in other regions to apply to specific needs of each designer.

In addition to these recommendations, many other categories could be appended to the reference design to relate to the individual needs of the designer. This reference design is described as flexible because more situational attributes can be applied to the design process. For example, if there is a need to add higher priority communications that require more security than the rest; that can be added. A plethora of other options exist to work in the needs of the designer to the reference design. The decision comes down to the implementation of the engineer.



## APPENDICIES

### APPENDIX A: CODE

#### A-1 DSP C Code

```

/**
 * Engineer:      Nicholas Blair
 * Contact:       npb002@uark.edu
 * Company:       University of Arkansas (NCREPT)
 * Website:       https://ncrept.uark.edu/
 *
 * Create Date:   01Mar2020
 * Design Name:   Buck_Boost_Inverter_PI
 * Project Version: v0.2.3
 * Target Devices: TMS320F28335
 * Hardware:      UCB_Small_v2.1c; UCB_PE-EVAL_v1.5a
 * Tool Versions: Code Composer Studio v8.1.0.00011
 *                ControlSuite v3.4.9
 *
// Defines
#define DEBUG_EN      0           // Set to 1 for Debug Mode or 0 for Flash Mode
#define Freq_Clk      150e6      // Clock Frequency (150 MHz)
#define Freq_Sw       0x03C0     // Set Switching Frequency in Fixed Point
(30 kHz)
// #define Freq_Fund_AC 0x0780    // Set Fundamental Output Frequency in
Fixed Point (60 Hz)
#define Freq_Fund_AC  0x0140     // Set Fundamental Output Frequency in Fixed Point
(10 Hz)
// #define AC_Ref      0x0140    // Set Peak Output AC Magnitude Reference
in Fixed Point (10 V)
#define AC_Ref        0x0300     // Set Peak Output AC Magnitude Reference
in Fixed Point (24 V)
#define IQ_STEP       0.03125    // Step size for IQ5 fixed point format (2^-5)

// Includes
#include "ModbusSlave.h"
#include "DSP2833x_Device.h"
#include "DSP2833x_GlobalPrototypes.h"
#include "DSP28x_Project.h"
#include "math.h"
#include "C28x_FPU_FastRTS.h"
#include "DCL.h"

```

```

// ADC start parameters
#if (CPU_FRQ_150MHZ) // Default - 150 MHz SYSCLKOUT
    #define ADC_MODCLK 0x3 // HSPCLK = SYSCLKOUT/2*ADC_MODCLK2 = 150/(2*3)
    = 25.0 MHz
#endif
#if (CPU_FRQ_100MHZ)
    #define ADC_MODCLK 0x2 // HSPCLK = SYSCLKOUT/2*ADC_MODCLK2 = 100/(2*2)
    = 25.0 MHz
#endif
#define ADC_CKPS 0x1 // ADC module clock = HSPCLK/2*ADC_CKPS =
25.0MHz/(1*2) = 12.5MHz

///Prototype statements for functions
void Gpio_Init(void); // Initialize GPIO Outputs and Inputs
void ADC_Init(void); // Initialize ADC
void InitEPWM1(void); // Initialize EPWM1 Module
void InitEPWM2(void); // Initialize EPWM2 Module
void InitEPWM3(void); // Initialize EPWM3 Module
void InitEPWM4(void); // Initialize EPWM4 Module
void InitFlash(); // Initialize Flash Module
void MemCopy(); // Flash Memory Copy function

/// Prototype statements for Serial functions
//void scia_echoback_init(void); // Initialize Serial Module: SCIA DLB, 8-bit word,
// baud rate 0x000F, default, 1 STOP bit, no parity
//void scia_fifo_init(void); // Initialize FIFO for serial communication
//void scia_xmit(int a); // Send a single character via the serial port
//void scia_msg(char *msg); // Send a string of character via the serial port

///Interrupts
interrupt void adc_isr(void); // Interrupt routine for ADC

///External Functions
// For Ramfuncs
extern Uint16 RamfuncsLoadStart; // Variable for MemCopy for loading data into flash.
extern Uint16 RamfuncsLoadEnd; // Variable for MemCopy for loading data into flash.
extern Uint16 RamfuncsRunStart; // Variable for MemCopy for loading data into flash.
//For Dclfuncs
extern Uint16 DclfuncsLoadStart; // Variable for MemCopy for loading data into flash.
extern Uint16 DclfuncsLoadEnd; // Variable for MemCopy for loading data into flash.
extern Uint16 DclfuncsRunStart; // Variable for MemCopy for loading data into flash.

///For Debug/Flash Selection
#if DEBUG_EN > 0

```

```

//debug mode
#else
#pragma CODE_SECTION(adc_isr, "ramfuncs"); // flash mode
#endif

////Global variables
float32 DSP_ADC_Scale;           // Scaling factor for DSP ADC
float32 DSP_ADC_Scale2;         // Scaling factor for DSP ADC
float32 SPI_ADC_Scale;          // Scaling factor for SPI ADC
float32 Current_Scale;          // Scaling factor Current Sensors
float32 R_Div_Scale1;           // Scaling factor Voltage Divider
float32 R_Div_Scale2;           // Scaling factor Voltage Divider
float32 R_Div_Scale3;           // Scaling factor Voltage Divider

//Uint16 V_ADC[8];              // ADC Register Values

ModbusSlave mb;                 // Modbus Variables
long Temp_mb1, Temp_mb2, Temp_GPDAT, Temp_Read1, Temp_Read2 = 0; // Declare
Temp Variables as a Long and initialize to 0

//Registers
Uint16 Reg[160];                // Registers for command and data
int i;                          // Counter
float V_ADCmax = 3.0;           // Maximum ADC Voltage
Uint16 V_ADC[16];               // ADC Register Values
Uint16 V_ADC_Temp[16];          // ADC Register Values (Temp)

//// Constants for setting the correct parameters based on switching frequency////
//Prd_PWM = 2 x TBPRD x TTBCLK => TBPRD = [Freq_PWM/(2*TTBCLK)] -1
//Example1(30kHz): PWM_TBPRD = [(30e3^-1)/(2*60e6^-1)-1] = 999
//Example1(1kHz): PWM_TBPRD = [(1e3^-1)/(2*60e6^-1)-1] = 29999
float FreqClk;                  // Clock Freq (150 MHz)
float FreqPWM;                  // Switching Freq
int FreqFund_AC;                // Fundamental AC Freq

Uint16 PWM_TBPRD;               // PWM Period; \n Prd_PWM = 2 x TBPRD x
TTBCLK => TBPRD = [Freq_PWM/(2*TTBCLK)] -1 \n Example1(30kHz): PWM_TBPRD =
[(30e3^-1)/(2*60e6^-1)-1] = 999 \n
float Prd_Sw;                   // Switching cycles per fundamental period Ex:(30 kHz /
60Hz = 500)
float Prd_Sw_d4;                // Switching cycles per fundamental period divided by 4
Ex:((30 kHz / 60Hz)/4 = 125) \n This is used to sample at max and min AC values.

```

```

//Serial Comm
int Pkt_ID = 0; // Packet Variable for Serial Communications
(Packet ID)
int Pkt_Len = 0; // Packet Variable for Serial Communications (Packet
Length)
int Pkt_Reg = 0; // Packet Variable for Serial Communications (Number of
Registers)
int Pkt_Addr = 0; // Packet Variable for Serial Communications (Starting
Address)
int Pkt_CkSum = 0; // Packet Variable for Serial Communications (Checksum)
int Pkt_TX_Temp = 0; // Packet Variable for Serial Communications (Temp Value
of Data)
int Pkt_Data[512] = {0}; // Buffer for Serial Packet

int i; //!< Generic Counter variable

////ADC Values
//DSP ADC Values
float V_DCin; // Converted ADC Value for VDC Input
float V_PhA; // Converted ADC Value for Phase_A for 3-phase inverter
float V_PhB; // Converted ADC Value for Phase_B for 3-phase inverter
float V_PhC; // Converted ADC Value for Phase_C for 3-phase inverter
float V_Buck; // Converted ADC Value for Buck
float V_Boost; // Converted ADC Value for Boost
float I_DCin; // Converted ADC Value for IDC input [Scaled to 3V]
float V_Pot; // Converted ADC Value for Potentiometer [Scaled to 3V]
//SPI ADC Values
float I_DCin2; // Converted SPI ADC Value for VDC Current Input
float I_L1; // Converted SPI ADC Value for Phase_A for 3-phase inverter
float I_L2; // Converted SPI ADC Value for Phase_B for 3-phase inverter
float I_L3; // Converted SPI ADC Value for Phase_C for 3-phase inverter
float I_Buck; // Converted SPI ADC Value for Buck
float I_Boost; // Converted SPI ADC Value for Boost
float Ia; // Converted SPI ADC Value for Ia or I_L4 Current [Jumper JP3]
float I_L5; // Converted SPI ADC Value for Potentiometer
//ADC Scaling factors
float V_Scale1; // Scaling factor for ADC Values (V_ADC_Max/D_ADC_Max)
float V_Scale2; // Scaling factor for ADC Values
float V_Scale3; // Scaling factor for ADC Values for SPI (5V/4095)
float I_Scale1; // Scaling factor for Current Values (185mV/A)^-1
float I_Scale2; // Scaling factor for Current Values (185mV/A)^-1
float V_Scaled[16] = {0}; // Scaled ADC Values for Modbus
//ADC Validation Flags
char ADC_valid1 = 0; //Flag for determining if ADC voltage is in correct range
char ADC_valid2 = 0; //Flag for determining if ADC voltage is in correct range
char ADC_valid3 = 0; //Flag for determining if ADC voltage is in correct range

```

```

//3-Phase Inverter Variables
float AC_Mag;           // AC Magnitude reference (V)
float AC_Freq_Fund;    // AC Fundamental Frequency Ref (Hz)
float pi;               // Value of PI (3.1415926535898)
float pi_2;            // Value of 2 x PI
float phase_offset;    // Phase offset for 3-phase output voltages 120 degree offset (2/3)*pi
float Step_sin;        // Used in calculating the sin output for 3-phase inverter; Step_sin =
(Sin_Counter/Prd_Sw)*pi_2;
float Sine1;           // Duty to generate Duty Cycle for 3-Phase PWM
float Sine2;           // Duty to generate Duty Cycle for 3-Phase PWM
float Sine3;           // Duty to generate Duty Cycle for 3-Phase PWM
float Duty_AC;         // Duty Cycle for 3-Phase PWM; Based on AC Magnitude
reference and PWM Frequency.
Uint16 Sin_Counter;    // Used to calculate the value of sin output; Based on number of interrupts
Uint16 DeadTime;      // Deadtime; 30 clk cycles = (150e6^-1)*30 => 200ns

//SPI Variables
Uint16 spi_cmd;        // Used to store ADC Command to read all Channels Continuously;
"0xFFDF"
Uint16 rdata[8];       // Array for saving SPI ADC Data
Uint16 spi_temp;       // Temp received data from SPI
///PI Variables
PI pi1 = PI_DEFAULTS;  // PI Buck Control Variables
PI pi2 = PI_DEFAULTS;  // PI Boost Control Variables
float uk1;             // PI Buck Control Output Variable
float uk2;             // PI Boost Control Output Variable

float Buck_Cmd;
float Boost_Cmd;

////Main function
void main(void)
{
    spi_cmd = 0xFFDF;    //Set SPI command for ADC continuous read

    pi1.Umin = 0;
    pi2.Umin = 0;

    //      //Initialize Variables for Serial Communication
    //  Uint16 ReceivedChar;
    //  //char *msg;
    //  Pkt_ID =0;
    //  Pkt_Len=0;
    //  Pkt_Reg=0;
    //  Pkt_Addr=0;

```

```

// Pkt_CkSum = 0;

//Initialize Arrays to zero
for(i=0;i<512;i++)
{
    Pkt_Data[i] =0;
}

for(i=0;i<128;i++)
{
    Reg[i] =0;
}

for(i=0; i<8;i++)
{
    rdata[i] = 0;
}

//Setup the initial Switching frequency
//Prd_PWM = 2 x TBPRD x TTBCLK => TBPRD = [Freq_PWM/(2*TTBCLK)] -1
//Example1(30kHz): PWM_TBPRD = [(30e3^1)/(2*60e6^1)-1] = 999
//Example1(1kHz): PWM_TBPRD = [(1e3^1)/(2*60e6^1)-1] = 29999
FreqClk = Freq_Clk; //Clock Freq (150 MHz)
/*
-- Reg[0] => Switching Freq [0-1000kHz (in kHz)]
-- Reg[1] => Buck Reference [Vref Range= (0-2^12) (0-50V)]
-- Reg[2] => Boost Reference [Vref Range= (0-2^12) (0-50V)]
-- Reg[3] => AC Magnitude Reference [Vref Range= (0-2^12) (0-50V)]
-- Reg[4] => AC Fundamental Freq Reference [0-1000Hz]
-- Reg[5] => Kp Buck Parameter [0-100 in 0.01 increments]
-- Reg[6] => Ki Buck Parameter [0-100 in 0.01 increments]
-- Reg[7] => Kp Boost Parameter [0-100 in 0.01 increments]
-- Reg[8] => Ki Boost Parameter [0-100 in 0.01 increments]
*/
Reg[0] = Freq_Sw; //Set Switching Frequency (in kHz)
Reg[1] = 0; //Set Buck Voltage Reference
Reg[2] = 0; //Set Boost Voltage Reference
Reg[3] = AC_Ref; //Set AC Mag Reference
Reg[4] = Freq_Fund_AC; //Set AC Fund Reference
DeadTime = 30; //Deadtime; 30 clk
cycles = (150e6^1)*30 => 200ns

//V_Scale_DSP = (V_DSP_ADC_Max/D_DSP_ADC_Max); //Scaling factor for
DSP ADC
//V_Scale_SPI = (V_SPI_ADC_Max/D_SPI_ADC_Max); //Scaling factor for DSP ADC

```

```

//V_Scale2 = V_Scale1*32.65;           //Scaling factor for ADC
//V_Scale3 = (5.0/4095);               //Scaling factor for SPI ADC
//I_Scale1 = (5.405);                  //Scaling factor for Current Sensor
(185mV/A)^-1

DSP_ADC_Scale = (3.0/4095);           // Scaling factor for DSP ADC
//DSP_ADC_Scale2 = ((3.0/4095)-0.9);
SPI_ADC_Scale = (5.0/4095);           // Scaling factor for SPI ADC
Current_Scale = (5.4054);             // Scaling factor for Current Sensor (185mV/A)^-
1
R_Div_Scale1 = (32.6456);              // (31.6e3/(1e6+31.6e3))^(-1)
R_Div_Scale2 = (1.6667);              // (15e3/(10e3+15e3))^(-1)
R_Div_Scale3 = (1.1460);              // (13.7e3/(1e6+13.7e3))^(-1)

V_Scale1 = DSP_ADC_Scale * R_Div_Scale1;
//Voltage scaling factor for Vin,Va,Vb,Vc,V_Buck,V_Boost
V_Scale2 = SPI_ADC_Scale* R_Div_Scale1;
V_Scale3 = DSP_ADC_Scale2 * R_Div_Scale3;
I_Scale1 = DSP_ADC_Scale*Current_Scale*R_Div_Scale2;           //Current
scaling factor for Iin (DSP ADC)
I_Scale2 = SPI_ADC_Scale*Current_Scale;
//Current scaling factor for Iin,Ia,Ib,Ic,I_Buck,I_Boost

///Update Values (IQ5)
FreqPWM = (((Reg[0] >> 5) & 0x07FF) + ((Reg[0] & 0x001F)*IQ_STEP))*1e3;
//Switching Freq in Hz
Buck_Cmd = (((Reg[1] >> 5) & 0x07FF) + ((Reg[1] & 0x001F)*IQ_STEP));
Boost_Cmd = (((Reg[2] >> 5) & 0x07FF) + ((Reg[2] & 0x001F)*IQ_STEP));
AC_Mag = (((Reg[3] >> 5) & 0x07FF) + ((Reg[3] & 0x001F)*IQ_STEP));
//AC Magnitude
FreqFund_AC = (((Reg[4] >> 5) & 0x07FF) + ((Reg[4] & 0x001F)*IQ_STEP));
//Set AC Fundamental Freq
PWM_TBPRD = (1/FreqPWM)/(2*(1/FreqClk))-1;
//Set initial PWM Period

Prd_Sw = FreqPWM/FreqFund_AC;
//Switching cycles per fundamental period
Ex:(30 kHz / 60Hz = 500)
Prd_Sw_d4 = Prd_Sw/4;
//Switching cycles per 1/4 fundamental
period

pi=3.1415926535898;
pi_2=2*pi;
phase_offset=(2.0/3.0)*pi;

```

```

Buck_Cmd = 0.0;
Boost_Cmd = 0.0;

// Step 1. Initialize System Control:
// PLL, WatchDog, enable Peripheral Clocks
InitSysCtrl();

EALLOW;
SysCtrlRegs.HISPCP.all = ADC_MODCLK;      // HSPCLK =
SYSCLKOUT/ADC_MODCLK
EDIS;

// Step 2. Initialize GPIO:
// InitGpio();

// For this example, only init the pins for the SCI-A port.
// This function is found in the DSP2833x_Sci.c file.
//InitSciaGpio();

// Step 3. Clear all interrupts and initialize PIE vector table:
// Disable CPU interrupts
DINT;

// Initialize the PIE control registers to their default state.
// The default state is all PIE interrupts disabled and flags
// are cleared.
// This function is found in the DSP2803x_PieCtrl.c file.
InitPieCtrl();

// Disable CPU interrupts and clear all CPU interrupt flags:
IER = 0x0000;
IFR = 0x0000;

// Initialize the PIE vector table with pointers to the shell Interrupt
// Service Routines (ISR).
// This will populate the entire table, even if the interrupt
// is not used in this example. This is useful for debug purposes.
InitPieVectTable();

// Interrupts that are used in this example are re-mapped to
// ISR functions found within this file.
EALLOW; // This is needed to write to EALLOW protected registers
//PieVectTable.EPWM1_INT = &epwm1_isr;

```



```

//PieVectTable.EPWM2_INT = &epwm2_isr;
//PieVectTable.EPWM3_INT = &epwm3_isr;
PieVectTable.ADCINT = &adc_isr;
EDIS; // This is needed to disable write to EALLOW protected registers

EALLOW;
Gpio_Init(); //Initialize GPIO
InitSciaGpio();
EDIS;

// Debug or flash mode selection
#if DEBUG_EN > 0
    // do nothing, as default debug mode
#else
    MemCopy(&RamfuncsLoadStart, &RamfuncsLoadEnd, &RamfuncsRunStart);
    MemCopy(&DclfuncsLoadStart, &DclfuncsLoadEnd, &DclfuncsRunStart);
    InitFlash();
#endif

    ADC_Init(); // Initialize ADC

    InitEPwm1Gpio();
    InitEPwm2Gpio();
    InitEPwm3Gpio();
    InitEPwm4Gpio();

//      // Step 5. User specific code, enable interrupts:
//  scia_fifo_init();      // Initialize the SCI FIFO
//  scia_echoback_init(); // Initialize SCI for echoback

//LSPCLK = 50MHZ
    //SpiaRegs.SPIBRR.bit.SPI_BIT_RATE = 0x0031; //set the SPI baud rate
    //SpiaRegs.SPIBRR.all = 0x007F;
    //SpiaRegs.SPIBRR.all = 0x007F;
    //7Fh (R/W) = SPI Baud Rate = LSPCLK/128
    //13h (R/W) = SPI Baud Rate = LSPCLK/20

////SPI Configuration
EALLOW;

    SpiaRegs.SPICCR.bit.SPISWRESET = 0;           // SPI software reset

    //SpiaRegs.SPIBRR = 0x007F;                    // SPI Clock scaler
    SpiaRegs.SPIBRR = 0x0004;                      //SPI Clock scaler
    SpiaRegs.SPICCR.bit.SPICHR = 0xF;             //SPI character length 16 bit

```

```

SpiaRegs.SPICTL.bit.MASTER_SLAVE = 1;      //SPI (Master)
SpiaRegs.SPICTL.bit.TALK = 1;              //SPI Enable Comm
SpiaRegs.SPICCR.bit.SPILBK = 0;            //Disable loopback
SpiaRegs.SPIFFTX.bit.SPIFFENA = 1;         //Enable SPI FIFO
SpiaRegs.SPIFFTX.bit.SPIRST = 1;           //SPI FIFO can TX (or) RX
SpiaRegs.SPIFFTX.bit.TXFIFO = 1;           //Release TX FIFO from reset
SpiaRegs.SPIFFTX.bit.TXFFIENA = 0;         //disable TX FIFO interrupt
SpiaRegs.SPIFFRX.bit.RXFIFORESET = 0;      //Reset FIFO pointer to 0
SpiaRegs.SPIFFRX.bit.RXFFIENA = 0;         //disable RX FIFO interrupt
//SpiaRegs.SPIFFRX.bit.RXFFIL = 0x08;     //Generate RX FIFO interrupt after
receiving 8 words
SpiaRegs.SPIFFRX.bit.RXFIFORESET = 1;      //Enable RX FIFO operation

// Initialize SPI FIFO registers
SpiaRegs.SPIFFTX.all=0xE040;
SpiaRegs.SPIFFRX.all=0x204f;
SpiaRegs.SPIFFCT.all=0x0;

SpiaRegs.SPICCR.bit.CLKPOLARITY=0;
SpiaRegs.SPICTL.bit.CLK_PHASE=0;
SpiaRegs.SPICCR.bit.SPISWRESET = 1;        //Release SPI software reset
EDIS;

///End SPI Configuration

//Setup SPI ADC for Continuous Read
SpiaRegs.SPITXBUF = 0xFFFF; //Send Dummy Data to SPI
while(SpiaRegs.SPIFFRX.bit.RXFFST !=1);    //wait for response
rdata[0] = SpiaRegs.SPIRXBUF;
SpiaRegs.SPITXBUF = 0xFFFF; //Send Dummy Data to SPI
while(SpiaRegs.SPIFFRX.bit.RXFFST !=1);    //wait for response
rdata[1] = SpiaRegs.SPIRXBUF;
SpiaRegs.SPITXBUF = spi_cmd; //Send read all cmd to SPI
while(SpiaRegs.SPIFFRX.bit.RXFFST !=1);    //wait for response
rdata[2] = SpiaRegs.SPIRXBUF;

mb = construct_ModbusSlave();

//Main Loop
for(;;)
{
    mb.loopStates(&mb);

    //Copy Values from Modbus registers to global registers
    for(i=128;i<160;i++)

```

```

{
    Reg[i] = mb.holdingRegisters.Mod_Reg[i];
}

/*
* Reg[128] => GPADAT-1_Cmd (GPIO00 - GPIO15)
* Reg[129] => GPADAT-2_Cmd (GPIO16 - GPIO31)
* Reg[130] => GPBDAT-1_Cmd (GPIO32 - GPIO47)
* Reg[131] => GPBDAT-2_Cmd (GPIO48 - GPIO63)
* Reg[132] => GPADAT-1_Read (GPIO00 - GPIO15)
* Reg[133] => GPADAT-2_Read (GPIO16 - GPIO31)
* Reg[134] => GPBDAT-1_Read (GPIO32 - GPIO47)
* Reg[135] => GPBDAT-2_Read (GPIO48 - GPIO63)
* Reg[136] => ADC-A0 (IDC_A Connector)
* Reg[137] => ADC-A1 (IDC_A Connector)
* Reg[138] => ADC-A2 (IDC_A Connector)
* Reg[139] => ADC-A3 (IDC_A Connector)
* Reg[140] => ADC-A4 (IDC_A Connector)
* Reg[141] => ADC-A5 (IDC_A Connector)
* Reg[142] => ADC-A6 (IDC_A Connector)
* Reg[143] => ADC-A7 (IDC_A Connector)
* Reg[144] => ADC-B0 (IDC_B Connector)
* Reg[145] => ADC-B1 (IDC_B Connector)
* Reg[146] => ADC-B2 (IDC_B Connector)
* Reg[147] => ADC-B3 (IDC_B Connector)
* Reg[148] => ADC-B4 (IDC_B Connector)
* Reg[149] => ADC-B5 (IDC_B Connector)
* Reg[150] => ADC-B6 (IDC_B Connector)
* Reg[151] => ADC-B7 (IDC_B Connector)
*/

// Read or Write mode selection
#if R_W_FLAG > 0
    //Write Mode
    ///Read from Modbus and write to outputs.
    //GPADAT => GPIO00 - GPIO31
    Temp_mb1 = Reg[2];
    GpioDataRegs.GPADAT.all = (Reg[1] & 0x0000FFFF) | ((Temp_mb1 << 16) &
0xFFFF0000); //Merge upper and lower registers and write to the GPADAT Register.
    //GPBDAT => GPIO32 - GPIO63
    Temp_mb2 = Reg[4];
    GpioDataRegs.GPBDAT.all = (Reg[3] & 0x0000FFFF) | ((Temp_mb2 << 16) &
0xFFFF0000); //Merge upper and lower registers and write to the GPBDAT Register.

#else
    //Read Mode

```

```

#endif

////Read from GPIO and write to Modbus Registers
//GPADAT => GPIO00 - GPIO31
Reg[132] = (GpioDataRegs.GPADAT.all & 0xFFFF);
Reg[133] = (GpioDataRegs.GPADAT.all >> 16);
mb.holdingRegisters.Mod_Reg[0] = Reg[132]; // Mod_Reg[5] -> Mod_Reg[0]
mb.holdingRegisters.Mod_Reg[1] = Reg[133]; // Mod_Reg[6] -> Mod_Reg[1]

//GPBDAT => GPIO32 - GPIO63
Reg[134] = (GpioDataRegs.GPBDAT.all & 0xFFFF);
Reg[135] = (GpioDataRegs.GPBDAT.all >> 16);
mb.holdingRegisters.Mod_Reg[2] = Reg[134]; // Mod_Reg[7] -> Mod_Reg[2]
mb.holdingRegisters.Mod_Reg[3] = Reg[135]; // Mod_Reg[8] -> Mod_Reg[3]

//Reg[136] = 500;
//mb.holdingRegisters.Mod_Reg[1] = Reg[136];

//Set ADC Trigger (Cleared in ISR)
AdcRegs.ADCTRL2.bit.SOC_SEQ1 = 1;

for(i=0;i<16;i++)
{
    // Reconstructed ADC values to real world values
    //V_ADC[i] = V_ADC_Temp[i] * (V_ADCmax/4095); // V_ADC*(3.0/4095)
    // Write Data to Holding Registers
    mb.holdingRegisters.Mod_Reg[i+4] = V_Scaled[i]; // Mod_Reg[i+9] ->
Mod_Reg[i+4]
    // Store Holding Registers to Local Registers for Debugging
    Reg[i+136] = mb.holdingRegisters.Mod_Reg[i+4]; // Mod_Reg[i+9] ->
Mod_Reg[i+4]

}

if (Reg[33] >= 1600)
{
    ADC_valid3 = ADC_valid2;
    ADC_valid2 = ADC_valid1;
    ADC_valid1 = 1;
}
else
{
    ADC_valid1 = 0;
    ADC_valid2 = 0;
}

```

```

    //ADC_valid3 = 0;
}

if (ADC_valid3)
    Reg[3] = 0;
else
{
    for(i=0;i<9;i++)
    {
        // Store Holding Registers to Local Registers for Modbus Write Commands
        Reg[i] = mb.holdingRegisters.Mod_Reg[i+20];          // Mod_Reg[i+32] ->
Mod_Reg[i+20]
    }
}

//Temp_Read1 = GpioDataRegs.GPADAT.all;
//Temp_Read2 = GpioDataRegs.GPBDAT.all;

////SPI Communication
SpiaRegs.SPITXBUF = 0x0000;    //Send dummy cmd to SPI
while(SpiaRegs.SPIFFRX.bit.RXFFST <1); //wait for response
//rdata[spi_cnt] = SpiaRegs.SPIRXBUF;
spi_temp = SpiaRegs.SPIRXBUF;

    if((spi_temp & 0xF000) == 0x0000)                //Check if Address is 0; if
true then write lower 12-bits to rdata
        rdata[0] = (spi_temp & 0x0FFF);
    else if((spi_temp & 0xF000) == 0x1000)            //Check if Address is 1; if
true then write lower 12-bits to rdata
        rdata[1] = (spi_temp & 0x0FFF);
    else if((spi_temp & 0xF000) == 0x2000)            //Check if Address is 2; if
true then write lower 12-bits to rdata
        rdata[2] = (spi_temp & 0x0FFF);
    else if((spi_temp & 0xF000) == 0x3000)            //Check if Address is 3; if
true then write lower 12-bits to rdata
        rdata[3] = (spi_temp & 0x0FFF);
    else if((spi_temp & 0xF000) == 0x4000)            //Check if Address is 4; if
true then write lower 12-bits to rdata
        rdata[4] = (spi_temp & 0x0FFF);
    else if((spi_temp & 0xF000) == 0x5000)            //Check if Address is 5; if
true then write lower 12-bits to rdata
        rdata[5] = (spi_temp & 0x0FFF);
    else if((spi_temp & 0xF000) == 0x6000)            //Check if Address is 6; if
true then write lower 12-bits to rdata

```

```

        rdata[6] = (spi_temp & 0x0FFF);
        else if((spi_temp & 0xF000) == 0x7000)           //Check if Address is 7; if
true then write lower 12-bits to rdata
        rdata[7] = (spi_temp & 0x0FFF);

//      ///Serial Communication
//      if(SciaRegs.SCIFFRX.bit.RXFFST) //check to see if data is ready (Polling)
//      {
//          ReceivedChar = SciaRegs.SCIRXBUF.all; //Get Character
//          if(ReceivedChar== 0x7E) //Check if start delimiter
//          {
//              Pkt_CkSum = 0xFF; //Set Temp Check Sum to 0xFF
//              while(SciaRegs.SCIFFRX.bit.RXFFST !=1) { } // wait for XRDY
=1 for empty state (Blocking)
//              ReceivedChar = SciaRegs.SCIRXBUF.all; //Get Character
//              //Pkt_CkSum = Pkt_CkSum - ReceivedChar; // Subtract Received
data from Check Sum
//              Pkt_Len = ReceivedChar; //Set packet Length
//              for (i=0; i<Pkt_Len; i++)
//              {
//                  while(SciaRegs.SCIFFRX.bit.RXFFST !=1) { } // wait for
XRDY =1 for empty state (Blocking)
//                  ReceivedChar = SciaRegs.SCIRXBUF.all; //Get
Character
//                  Pkt_CkSum = Pkt_CkSum - ReceivedChar; // Subtract
Received data from Check Sum
//                  Pkt_Data[i]=ReceivedChar; //Copy received data to
Pkt_Data
//              }
//              //Read Check Sum Value
//              while(SciaRegs.SCIFFRX.bit.RXFFST !=1) { } // wait for XRDY
=1 for empty state (Blocking)
//              ReceivedChar = SciaRegs.SCIRXBUF.all; //Get Character
//              Pkt_CkSum = Pkt_CkSum & 0x00FF;
//              if(Pkt_CkSum == ReceivedChar) //Checksums match!
//              {
//                  Pkt_ID = Pkt_Data[0]; //Operation ID
//                  Pkt_Reg = Pkt_Data[1]; //Number of Registers
//                  Pkt_Addr = (Pkt_Data[2]*256)+Pkt_Data[3]; //Starting
Address
//                  if(Pkt_ID == 0x0A) //Set Registers
//                  {
//                      for(i=0;i<Pkt_Reg;i++)
//                      {

```

```
//      Reg[Pkt_Addr+i]=(Pkt_Data[4+i*2]*256)+Pkt_Data[5+i*2];  
//          }  
//      }  
//      else if(Pkt_ID == 0x0F)  
//      {  
//          Pkt_TX_Temp = 0; //Used to prevent data from  
updating during ChkSum Calculations  
//          Pkt_CkSum = 0xFF; //Reset Checksum  
//          scia_xmit(0x7E); //Send Start Delimiter  
//          scia_xmit(Pkt_Reg*2 + 4); //Send Packet Length  
((Register Count times 2) +4)  
//          scia_xmit(0x0A); //Send OP ID  
//          Pkt_CkSum = Pkt_CkSum - 0x0A; //Update  
Checksum  
//          scia_xmit(Pkt_Reg); //Send Number of registers  
//          Pkt_CkSum = Pkt_CkSum - Pkt_Reg; //Update  
Checksum  
//          //scia_xmit(Pkt_Addr); //Send Starting Address  
//          scia_xmit(Pkt_Addr>>8 & 0xFF); ///Send Starting Address (Upper Byte)  
//          Pkt_CkSum = Pkt_CkSum - (Pkt_Addr>>8 & 0xFF); //Update Checksum  
//          scia_xmit(Pkt_Addr & 0xFF); ///Send Starting Address (Lower Byte)  
//          Pkt_CkSum = Pkt_CkSum - (Pkt_Addr & 0xFF); //Update Checksum  
//          for(i=0;i<Pkt_Reg;i++)  
//              {  
//                  Pkt_TX_Temp = Reg[Pkt_Addr+i]; //Used  
to prevent data from updating during ChkSum Calculations  
//                      scia_xmit(Pkt_TX_Temp>>8 & 0xFF);  
//                          //Send upper byte  
//                              Pkt_CkSum = Pkt_CkSum -  
(Pkt_TX_Temp>>8 & 0xFF);  
//                                  scia_xmit(Pkt_TX_Temp & 0xFF); //Send  
lower byte  
//                                      Pkt_CkSum = Pkt_CkSum - (Pkt_TX_Temp  
& 0xFF);  
//                                          }  
//                                              scia_xmit(Pkt_CkSum & 0xFF);  
//                                                  }  
//                                                      }  
//                                                          }  
}   
////End Serial Communication  
  
////Update Values (IQ5)  
FreqPWM = (((Reg[0] >> 5) & 0x07FF) + ((Reg[0] & 0x001F)*IQ_STEP))*1e3;  
           //Switching Freq in Hz
```

```

    Buck_Cmd = (((Reg[1] >> 5) & 0x07FF) + ((Reg[1] & 0x001F)*IQ_STEP));
    Boost_Cmd = (((Reg[2] >> 5) & 0x07FF) + ((Reg[2] & 0x001F)*IQ_STEP));
    AC_Mag = (((Reg[3] >> 5) & 0x07FF) + ((Reg[3] & 0x001F)*IQ_STEP));
                //AC Magnitude
    FreqFund_AC = (((Reg[4] >> 5) & 0x07FF) + ((Reg[4] & 0x001F)*IQ_STEP));
                //Set AC Fundamental Freq
    PWM_TBPRD = (1/FreqPWM)/(2*(1/FreqClk))-1;
                //Set initial PWM Period

    Prd_Sw = FreqPWM/FreqFund_AC;
                //Switching cycles per fundamental
period Ex:(30 kHz / 60Hz = 500)
    Prd_Sw_d4 = Prd_Sw/4;
                //Switching cycles per 1/4
fundamental period
    //Buck PI
    pi1.Kp = (((Reg[5] >> 5) & 0x07FF) + ((Reg[5] & 0x001F)*IQ_STEP));
                //Set Kp for PI
    pi1.Ki = (((Reg[6] >> 5) & 0x07FF) + ((Reg[6] & 0x001F)*IQ_STEP));
                //Set Ki for PI
    pi1.Umax = PWM_TBPRD;
                //Set Max Duty Cycle

    //Boost PI
    pi2.Kp = (((Reg[7] >> 5) & 0x07FF) + ((Reg[7] & 0x001F)*IQ_STEP));
                //Set Kp for PI
    pi2.Ki = (((Reg[8] >> 5) & 0x07FF) + ((Reg[8] & 0x001F)*IQ_STEP));
                //Set Ki for PI
    pi2.Umax = PWM_TBPRD;
                //Set Max Duty Cycle

    //Update PWM Freq
    EPwm1Regs.TBPRD = PWM_TBPRD;
    EPwm2Regs.TBPRD = PWM_TBPRD;
    EPwm3Regs.TBPRD = PWM_TBPRD;
    EPwm4Regs.TBPRD = PWM_TBPRD;
    //End Update Values
}
}

void ADC_Init(void)
{
    // Configure ADC
    EALLOW;
    PieVectTable.ADCINT = &adc_isr;
    EDIS;

    InitAdc();

```



```

IER |= M_INT1;                                // Enable CPU Interrupt 1
//PieCtrlRegs.PIEIER1.bit.INTx1 = 1;          // Enable INT 1.1 in the PIE
PieCtrlRegs.PIEIER1.bit.INTx6 = 1;
EINT;                                           // Enable Global interrupt INTM
ERTM;                                           // Enable Global realtime interrupt
DBGM

EALLOW;
// Specific ADC setup for this example:
//Sample and Hold window
AdcRegs.ADCTRL1.bit.ACQ_PS = 0x1;              //Window length(1+#of
Cycles)
//ADC Clock Rate
AdcRegs.ADCTRL3.bit.ADCCLKPS = ADC_CKPS; //0 works faster but may
result in errors // 12.5 MHz 80ns
AdcRegs.ADCTRL1.bit.SEQ_CASC = 1;              // 1 Cascaded mode
AdcRegs.ADCTRL1.bit.SEQ_OVRD = 0;
//Specify # of Conversions
AdcRegs.ADCMAXCONV.bit.MAX_CONV1 = 0x8; // convert only used
channels

//AdcRegs.ADCTRL3.bit.SMODE_SEL = 0; //setup ADC for sequential
sampling

//Convert used channels
AdcRegs.ADCCHSELSEQ1.bit.CONV00 = 0x0; //ADCA0
AdcRegs.ADCCHSELSEQ1.bit.CONV01 = 0x1; //ADCA1
AdcRegs.ADCCHSELSEQ1.bit.CONV02 = 0x2; //ADCA2
AdcRegs.ADCCHSELSEQ1.bit.CONV03 = 0x3; //ADCA3
AdcRegs.ADCCHSELSEQ2.bit.CONV04 = 0x4; //ADCA4
AdcRegs.ADCCHSELSEQ2.bit.CONV05 = 0x5; //ADCA5
AdcRegs.ADCCHSELSEQ2.bit.CONV06 = 0x6; //ADCA6
AdcRegs.ADCCHSELSEQ2.bit.CONV07 = 0x7; //ADCA7

AdcRegs.ADCTRL1.bit.CONT_RUN = 0;              // Setup start/stop run
AdcRegs.ADCTRL2.bit.EPWM_SOCA_SEQ1 = 1; // Enable SOCA from ePWM
to start SEQ1
AdcRegs.ADCTRL2.bit.INT_ENA_SEQ1 = 1; // Enable SEQ1 interrupt (every
End Of Sequence)
AdcRegs.ADCTRL2.bit.INT_MOD_SEQ1 = 0; //
EDIS;

//ADC Triggering Setup
EPwm1Regs.ETSEL.bit.SOCAEN = 1;

```

```

    EPwm1Regs.ETSEL.bit.SOCASEL = 0x2;//4; // 2=Select SOC from CPMA on Period;
4=Select SOC from from CPMA on upcount
    EPwm1Regs.ETPS.bit.SOCAPRD = 1;    // Generate pulse every event
}

void Gpio_Init(void)
{
    // Step 2. Initialize GPIO:
    // Initialize GPIO
    EALLOW;

    GpioCtrlRegs.GPAMUX1.all = 0x0000; // GPIO functionality GPIO0-GPIO15
    GpioCtrlRegs.GPAMUX2.all = 0x0000; // GPIO functionality GPIO16-GPIO31
    GpioCtrlRegs.GPBMUX1.all = 0x0000; // GPIO functionality GPIO32-GPIO47

    GpioCtrlRegs.GPADIR.all = 0xFFFFFFFF; // GPIO0-GPIO31 are outputs
    GpioCtrlRegs.GPBDIR.all = 0xFFFFFFFF; // GPIO32-GPIO63 are outputs

    GpioCtrlRegs.GPAQSEL1.all = 0x0000; // GPIO0-GPIO15 Synch to SYSCLKOUT
    GpioCtrlRegs.GPAQSEL2.all = 0x0000; // GPIO16-GPIO31 Synch to SYSCLKOUT
    GpioCtrlRegs.GPBQSEL1.all = 0x0000; // GPIO32-GPIO47 Synch to SYSCLKOUT

    GpioCtrlRegs.GPAPUD.all = 0xFFFFFFFF; // Pullup's disabled GPIO0-GPIO31
    GpioCtrlRegs.GPBPUD.all = 0xFFFFFFFF; // Pullup's disabled GPIO32-GPIO63

    //Serial RX/TX Config
    GpioCtrlRegs.GPADIR.bit.GPIO28 = 0;    // Configure as input
    GpioCtrlRegs.GPADIR.bit.GPIO29 = 1;    // Configure as output
    GpioCtrlRegs.GPAPUD.bit.GPIO28 = 1;    // Disable internal pull-up
    GpioCtrlRegs.GPAPUD.bit.GPIO29 = 1;    // Disable internal pull-up

    //Switch Setup
    GpioCtrlRegs.GPADIR.bit.GPIO14 = 0;    // Configure as input (SW1)
    GpioCtrlRegs.GPADIR.bit.GPIO15 = 0;    // Configure as input (SW2)
    GpioCtrlRegs.GPAPUD.bit.GPIO14 = 1;    // Disable internal pull-up (SW1)
    GpioCtrlRegs.GPAPUD.bit.GPIO15 = 1;    // Disable internal pull-up (SW2)

    ///Configure LEDs
    //Control Card LEDs
    GpioCtrlRegs.GPADIR.bit.GPIO31 = 1;    // Configure as output (Red_Led1 On Board)
    GpioCtrlRegs.GPAPUD.bit.GPIO31 = 1;    // Disable internal pull-up
    GpioDataRegs.GPACLEAR.bit.GPIO31 = 1;    //initialize to zero
    GpioCtrlRegs.GPBDIR.bit.GPIO34 = 1;    // Configure as output (Red_Led2 On Board)
    GpioCtrlRegs.GPBPUD.bit.GPIO34 = 1;    // Disable internal pull-up

```

```

    GpioDataRegs.GPBCLEAR.bit.GPIO34 = 1;          //initialize to zero

    //Demo Board LEDs
    GpioCtrlRegs.GPADIR.bit.GPIO8 = 1;             // Configure as output (LED9 On Demo
Board)
    GpioCtrlRegs.GPAPUD.bit.GPIO8 = 1;             // Disable internal pull-up
    GpioDataRegs.GPACLEAR.bit.GPIO8 = 1;           //initialize to zero
    GpioCtrlRegs.GPADIR.bit.GPIO9 = 1;             // Configure as output (LED10 On Demo
Board)
    GpioCtrlRegs.GPAPUD.bit.GPIO9 = 1;             // Disable internal pull-up
    GpioDataRegs.GPACLEAR.bit.GPIO9 = 1;           //initialize to zero
    GpioCtrlRegs.GPADIR.bit.GPIO10 = 1;            // Configure as output (LED1 On Demo
Board)
    GpioCtrlRegs.GPAPUD.bit.GPIO10 = 1;            // Disable internal pull-up
    GpioDataRegs.GPACLEAR.bit.GPIO10 = 1;          //initialize to zero
    GpioCtrlRegs.GPADIR.bit.GPIO11 = 1;            // Configure as output (LED2 On Demo
Board)
    GpioCtrlRegs.GPAPUD.bit.GPIO11 = 1;            // Disable internal pull-up
    GpioDataRegs.GPACLEAR.bit.GPIO11 = 1;          //initialize to zero
    GpioCtrlRegs.GPADIR.bit.GPIO12 = 1;            // Configure as output (LED3 On Demo
Board)
    GpioCtrlRegs.GPAPUD.bit.GPIO12 = 1;            // Disable internal pull-up
    GpioDataRegs.GPACLEAR.bit.GPIO12 = 1;          //initialize to zero
    GpioCtrlRegs.GPADIR.bit.GPIO13 = 1;            // Configure as output (LED4 On Demo
Board)
    GpioCtrlRegs.GPAPUD.bit.GPIO13 = 1;            // Disable internal pull-up
    GpioDataRegs.GPACLEAR.bit.GPIO13 = 1;          //initialize to zero
    GpioCtrlRegs.GPADIR.bit.GPIO24 = 1;            // Configure as output (LED5 On Demo
Board)
    GpioCtrlRegs.GPAPUD.bit.GPIO24 = 1;            // Disable internal pull-up
    GpioDataRegs.GPACLEAR.bit.GPIO24 = 1;          //initialize to zero
    GpioCtrlRegs.GPADIR.bit.GPIO25 = 1;            // Configure as output (LED6 On Demo
Board)
    GpioCtrlRegs.GPAPUD.bit.GPIO25 = 1;            // Disable internal pull-up
    GpioDataRegs.GPACLEAR.bit.GPIO25 = 1;          //initialize to zero
    GpioCtrlRegs.GPADIR.bit.GPIO26 = 1;            // Configure as output (LED7 On Demo
Board)
    GpioCtrlRegs.GPAPUD.bit.GPIO26 = 1;            // Disable internal pull-up
    GpioDataRegs.GPACLEAR.bit.GPIO26 = 1;          //initialize to zero
    GpioCtrlRegs.GPADIR.bit.GPIO27 = 1;            // Configure as output (LED8 On Demo
Board)
    GpioCtrlRegs.GPAPUD.bit.GPIO27 = 1;            // Disable internal pull-up
    GpioDataRegs.GPACLEAR.bit.GPIO27 = 1;          //initialize to zero

    //Configure SPI

```

```
GpioCtrlRegs.GPAPUD.bit.GPIO16 = 0; // Enable pull-up on GPIO16 (SPISIMOA)
GpioCtrlRegs.GPAPUD.bit.GPIO17 = 0; // Enable pull-up on GPIO17 (SPISOMIA)
GpioCtrlRegs.GPAPUD.bit.GPIO18 = 0; // Enable pull-up on GPIO18 (SPICLKA)
GpioCtrlRegs.GPAPUD.bit.GPIO19 = 0; // Enable pull-up on GPIO19 (SPISTEIA)
```

```
//GpioCtrlRegs.GPAQSEL2.bit.GPIO16 = 3; // Asynch input GPIO16 (SPISIMOA)
GpioCtrlRegs.GPAQSEL2.bit.GPIO17 = 3; // Asynch input GPIO17 (SPISOMIA)
//GpioCtrlRegs.GPAQSEL2.bit.GPIO18 = 3; // Asynch input GPIO18 (SPICLKA)
//GpioCtrlRegs.GPAQSEL2.bit.GPIO19 = 3; // Asynch input GPIO19 (SPISTEIA)
```

```
GpioCtrlRegs.GPAMUX2.bit.GPIO16 = 1; // Configure GPIO16 as SPISIMOA
GpioCtrlRegs.GPAMUX2.bit.GPIO17 = 1; // Configure GPIO17 as SPISOMIA
GpioCtrlRegs.GPAMUX2.bit.GPIO18 = 1; // Configure GPIO18 as SPICLKA
GpioCtrlRegs.GPAMUX2.bit.GPIO19 = 1; // Configure GPIO19 as SPISTEIA
```

```
GpioCtrlRegs.GPADIR.bit.GPIO19 = 1; // Configure as output
GpioCtrlRegs.GPADIR.bit.GPIO16 = 1; // Configure as output
GpioCtrlRegs.GPADIR.bit.GPIO18 = 1; // Configure as output
GpioCtrlRegs.GPADIR.bit.GPIO17 = 0; // Configure as input
```

```
EDIS;
```

```
InitEPwm1Gpio();
InitEPwm2Gpio();
InitEPwm3Gpio();
InitEPwm4Gpio();
```

```
EALLOW;
SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 0;
EDIS;
```

```
InitEPWM1();
InitEPWM2();
InitEPWM3();
InitEPWM4();
```

```
EALLOW;
SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 1;
EDIS;
```

```
}
```

```
interrupt void adc_isr(void)
{
```

```
// Sample ADCs and converter variables to real world values
```

```
V_ADC[0] = AdcRegs.ADCRESULT0 >>4;
V_ADC[1] = AdcRegs.ADCRESULT1 >>4;
V_ADC[2] = AdcRegs.ADCRESULT2 >>4;
V_ADC[3] = AdcRegs.ADCRESULT3 >>4;
V_ADC[4] = AdcRegs.ADCRESULT4 >>4;
V_ADC[5] = AdcRegs.ADCRESULT5 >>4;
V_ADC[6] = AdcRegs.ADCRESULT6 >>4;
V_ADC[7] = AdcRegs.ADCRESULT7 >>4;
```

```
//Store ADC Values in Register Bank (DSP)
```

```
Reg[32]=V_ADC[0];
Reg[33]=V_ADC[1];
Reg[34]=V_ADC[2];
Reg[35]=V_ADC[3];
Reg[36]=V_ADC[4];
Reg[37]=V_ADC[5];
Reg[38]=V_ADC[6];
Reg[39]=V_ADC[7];
```

```
//Store ADC Values in Register Bank (AD7928)
```

```
Reg[40]=rdata[0] & 0x0FFF; //Remove address data
Reg[41]=rdata[1] & 0x0FFF; //Remove address data
Reg[42]=rdata[2] & 0x0FFF; //Remove address data
Reg[43]=rdata[3] & 0x0FFF; //Remove address data
Reg[44]=rdata[4] & 0x0FFF; //Remove address data
Reg[45]=rdata[5] & 0x0FFF; //Remove address data
Reg[46]=rdata[6] & 0x0FFF; //Remove address data
Reg[47]=rdata[7] & 0x0FFF; //Remove address data
```

```
/*
```

```
* Reg[32] => ADC-A_0 (DC Input Voltage) \n
* Reg[33] => ADC-A_1 (Phase A Voltage) \n
* Reg[34] => ADC-A_2 (Phase B Voltage) \n
* Reg[35] => ADC-A_3 (Phase C Voltage) \n
* Reg[36] => ADC-A_4 (Buck Voltage) \n
* Reg[37] => ADC-A_5 (Boost Voltage) \n
* Reg[38] => ADC-A_6 (DC Input Current [3V-Scaled]) \n
* Reg[39] => ADC-A_7 (Pot Voltage [3V-Scaled]) \n
* Reg[40] => ADC-SPI_0 (DC Input Current) \n
* Reg[41] => ADC-SPI_1 (L1 Current) [Phase A] \n
* Reg[42] => ADC-SPI_2 (L2 Current) [Phase B] \n
* Reg[43] => ADC-SPI_3 (L3 Current) [Phase C] \n
* Reg[44] => ADC-SPI_4 (Buck Current) \n
* Reg[45] => ADC-SPI_5 (Boost Current) \n
* Reg[46] => ADC-SPI_6 (Ia or I_L4 Current) [Jumper JP3] \n
```

```

* Reg[47] => ADC-SPI_7 (L5 Current) \n
*/

// Reconstruct ADC values to real world values
V_DCin = Reg[32] * V_Scale1; //
V_ADC*(3/4095) * (31.6e3/(1e6+31.6e3))^1
V_PhA = ((Reg[33]*DSP_ADC_Scale)-0.9)*R_Div_Scale3; //
[(V_ADC*(3/4095))-0.9] * (13.7e3/(1e6+13.7e3))^1
V_PhB = ((Reg[34]*DSP_ADC_Scale)-0.9)*R_Div_Scale3; //
[(V_ADC*(3/4095))-0.9] * (13.7e3/(1e6+13.7e3))^1
V_PhC = ((Reg[35]*DSP_ADC_Scale)-0.9)*R_Div_Scale3; //
[(V_ADC*(3/4095))-0.9] * (13.7e3/(1e6+13.7e3))^1
V_Buck = Reg[36] * V_Scale1; //
V_ADC*(3/4095) * (31.6e3/(1e6+31.6e3))^1
V_Boost = Reg[37] * V_Scale1; //
V_ADC*(3/4095) * (31.6e3/(1e6+31.6e3))^1
I_DCin = (Reg[38]*I_Scale1)-13.5; // [V_ADC*(5/4095) *
(185mV/A)^-1] -13.5
V_Pot = Reg[39] * DSP_ADC_Scale * R_Div_Scale2; //
V_ADC*(3/4095)
I_DCin2 = (Reg[40]*I_Scale2)-13.5; // [V_ADC*(5/4095) *
(185mV/A)^-1] -13.5
I_L1 = (Reg[41]*I_Scale2)-13.5 ; // [V_ADC*(5/4095) *
(185mV/A)^-1] -13.5
I_L2 = (Reg[42]*I_Scale2)-13.5 ; // [V_ADC*(5/4095) *
(185mV/A)^-1] -13.5
I_L3 = (Reg[43]*I_Scale2)-13.5 ; // [V_ADC*(5/4095) *
(185mV/A)^-1] -13.5
I_Buck = (Reg[44]*I_Scale2)-13.5 ; // [V_ADC*(5/4095) *
(185mV/A)^-1] -13.5
I_Boost = (Reg[45]*I_Scale2)-13.5 ; // [V_ADC*(5/4095) *
(185mV/A)^-1] -13.5
Ia = (Reg[46]*I_Scale2)-13.5 ; //
[V_ADC*(5/4095) * (185mV/A)^-1] -13.5
I_L5 = (Reg[47]*I_Scale2)-13.5 ; //
[V_ADC*(5/4095) * (185mV/A)^-1] -13.5

// Map scaled ADC values to itterable array
V_Scaled[0] = V_DCin;
V_Scaled[1] = V_PhA;
V_Scaled[2] = V_PhB;
V_Scaled[3] = V_PhC;
V_Scaled[4] = V_Buck;
V_Scaled[5] = V_Boost;

```

```

V_Scaled[6] = I_DCin;
V_Scaled[7] = V_Pot;
V_Scaled[8] = I_DCin2;
V_Scaled[9] = I_L1;
V_Scaled[10] = I_L2;
V_Scaled[11] = I_L3;
V_Scaled[12] = I_Buck;
V_Scaled[13] = I_Boost;
V_Scaled[14] = Ia;
V_Scaled[15] = I_L5;

// Check Potentiometer and update LEDs based on voltage
if(V_Pot<1.0)
{
    GpioDataRegs.GPASET.bit.GPIO31 = 1;           //turn off 1st Red LED
(Active Low)
    GpioDataRegs.GPBSET.bit.GPIO34 = 1;           //turn off 2nd Red LED
(Active Low)
    GpioDataRegs.GPACLEAR.bit.GPIO10 = 1; //turn off Board LED1
    GpioDataRegs.GPACLEAR.bit.GPIO11 = 1; //turn off Board LED2
    GpioDataRegs.GPACLEAR.bit.GPIO12 = 1; //turn off Board LED3
    GpioDataRegs.GPACLEAR.bit.GPIO13 = 1; //turn off Board LED4
    GpioDataRegs.GPACLEAR.bit.GPIO24 = 1; //turn off Board LED5
    GpioDataRegs.GPACLEAR.bit.GPIO25 = 1; //turn off Board LED6
    GpioDataRegs.GPACLEAR.bit.GPIO26 = 1; //turn off Board LED7
    GpioDataRegs.GPACLEAR.bit.GPIO27 = 1; //turn off Board LED8
    GpioDataRegs.GPASET.bit.GPIO8 = 1; //turn off Board LED9 (Sw8 Active
Low)
    GpioDataRegs.GPASET.bit.GPIO9 = 1; //turn off Board LED10 (Sw8 Active
Low)
}
else if(V_Pot>1.0 && V_Pot<2.0 )
{
    GpioDataRegs.GPACLEAR.bit.GPIO31 = 1;           //turn on 1st Red LED
    GpioDataRegs.GPBSET.bit.GPIO34 = 1;           //turn off 2nd Red LED
    GpioDataRegs.GPASET.bit.GPIO10 = 1; //turn on Board LED1
    GpioDataRegs.GPASET.bit.GPIO11 = 1; //turn on Board LED2
    GpioDataRegs.GPASET.bit.GPIO12 = 1; //turn on Board LED3
    GpioDataRegs.GPASET.bit.GPIO13 = 1; //turn on Board LED4
    GpioDataRegs.GPACLEAR.bit.GPIO24 = 1; //turn off Board LED5
    GpioDataRegs.GPACLEAR.bit.GPIO25 = 1; //turn off Board LED6
    GpioDataRegs.GPACLEAR.bit.GPIO26 = 1; //turn off Board LED7
    GpioDataRegs.GPACLEAR.bit.GPIO27 = 1; //turn off Board LED8
    GpioDataRegs.GPASET.bit.GPIO8 = 1; //turn off Board LED9 (Sw8 Active
Low)

```

```

        GpioDataRegs.GPACLEAR.bit.GPIO9 = 1; //turn on Board LED10 (Sw8 Active
Low)
    }
    else if(V_Pot>2.0 && V_Pot<2.5 )
    {
        GpioDataRegs.GPACLEAR.bit.GPIO31 = 1;      //turn on 1st Red LED
        GpioDataRegs.GPBCLEAR.bit.GPIO34 = 1;      //turn on 2nd Red LED
        GpioDataRegs.GPACLEAR.bit.GPIO10 = 1; //turn off Board LED1
        GpioDataRegs.GPACLEAR.bit.GPIO11 = 1; //turn off Board LED2
        GpioDataRegs.GPACLEAR.bit.GPIO12 = 1; //turn off Board LED3
        GpioDataRegs.GPACLEAR.bit.GPIO13 = 1; //turn off Board LED4
        GpioDataRegs.GPASET.bit.GPIO24 = 1; //turn on Board LED5
        GpioDataRegs.GPASET.bit.GPIO25 = 1; //turn on Board LED6
        GpioDataRegs.GPASET.bit.GPIO26 = 1; //turn on Board LED7
        GpioDataRegs.GPASET.bit.GPIO27 = 1; //turn on Board LED8
        GpioDataRegs.GPACLEAR.bit.GPIO8 = 1; //turn on Board LED9 (Sw8 Active
Low)
        GpioDataRegs.GPASET.bit.GPIO9 = 1; //turn off Board LED10 (Sw8 Active
Low)
    }
    else
    {
        GpioDataRegs.GPACLEAR.bit.GPIO31 = 1;      //turn on 1st Red LED
        GpioDataRegs.GPBCLEAR.bit.GPIO34 = 1;      //turn on 2nd Red LED
        GpioDataRegs.GPASET.bit.GPIO10 = 1; //turn on Board LED1
        GpioDataRegs.GPASET.bit.GPIO11 = 1; //turn on Board LED2
        GpioDataRegs.GPASET.bit.GPIO12 = 1; //turn on Board LED3
        GpioDataRegs.GPASET.bit.GPIO13 = 1; //turn on Board LED4
        GpioDataRegs.GPASET.bit.GPIO24 = 1; //turn on Board LED5
        GpioDataRegs.GPASET.bit.GPIO25 = 1; //turn on Board LED6
        GpioDataRegs.GPASET.bit.GPIO26 = 1; //turn on Board LED7
        GpioDataRegs.GPASET.bit.GPIO27 = 1; //turn on Board LED8
        GpioDataRegs.GPACLEAR.bit.GPIO8 = 1; //turn on Board LED9 (Sw8 Active
Low)
        GpioDataRegs.GPACLEAR.bit.GPIO9 = 1; //turn on Board LED10 (Sw8 Active
Low)
    }

    // Create an AC reference
    if(Sin_Counter >= Prd_Sw) //here we count to a number based on how many switching
cycles occur for every fundamental cycle
    {
        Sin_Counter = 0;
    }
    else
    {

```



```

        Sin_Counter++;
    }

    //Feedforward Command to maintain magnitude of Inverter output as DC Input is varied.
    Duty_AC = AC_Mag/V_DCin;

    if(Duty_AC > 0.9)
    {
        Duty_AC = 0.9;
    }
    else if(Duty_AC < 0.0)
    {
        Duty_AC = 0.0;
    }

    Step_sin = (Sin_Counter/Prd_Sw)*pi_2;
    Duty_AC = Duty_AC*PWM_TBPRD;

    // Check SW1 Status and base motor rotation on reading
    if(GpioDataRegs.GPADAT.bit.GPIO14 == 1)
    {
        Sine1 = ((sin(Step_sin+0)+1)/2)*Duty_AC;
        Sine2 = ((sin(Step_sin-phase_offset)+1)/2)*Duty_AC;
        Sine3 = ((sin(Step_sin+phase_offset)+1)/2)*Duty_AC;
    }
    else
    {
        Sine1 = ((sin(Step_sin+0)+1)/2)*Duty_AC;
        Sine3 = ((sin(Step_sin-phase_offset)+1)/2)*Duty_AC;
        Sine2 = ((sin(Step_sin+phase_offset)+1)/2)*Duty_AC;
    }

    uk1 = DCL_runPI(&pi1, Buck_Cmd, V_Buck);
    uk2 = DCL_runPI(&pi2, Boost_Cmd, V_Boost);

    ///Update Duty Cycle for Buck Converter
    //EPwm1Regs.CMPA.half.CMPA= (Uint16) (DC_Buck*PWM_TBPRD);

    EPwm4Regs.CMPA.half.CMPA= (Uint16) (uk1);

    ///Update Duty Cycle for Boost Converter
    //EPwm1Regs.CMPB = (Uint16) (DC_Boost*PWM_TBPRD);

    EPwm4Regs.CMPB = (Uint16) (uk2);

```

```

//Update Duty Cycle for 3-phase inverter
EPwm1Regs.CMPA.half.CMPA = (Uint16) Sine1;
EPwm2Regs.CMPA.half.CMPA = (Uint16) Sine2;
EPwm3Regs.CMPA.half.CMPA = (Uint16) Sine3;

// Reinitialize for next ADC sequence
AdcRegs.ADCTRL2.bit.RST_SEQ1 = 1;    // Reset SEQ1
AdcRegs.ADCST.bit.INT_SEQ1_CLR = 1;  // Clear INT SEQ1 bit
PieCtrlRegs.PIEACK.all = PIEACK_GROUP1; // Acknowledge interrupt to PIE

return;
}

void InitEPWM1()
{
    EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN;
    EPwm1Regs.TBPRD = PWM_TBPRD;        // Period = 2* 1000 TBCLK
counts => 30 kHz

    EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE;    // Enable phase loading
    EPwm1Regs.TBPHS.half.TBPHS = 0;
    EPwm1Regs.TBCTL.bit.PHSDIR = TB_DOWN;
    EPwm1Regs.TBCTL.bit.PRDLT = TB_SHADOW;
    EPwm1Regs.TBCTL.bit.SYNCOSEL = TB_CTR_ZERO; //Sync down-stream module
    EPwm1Regs.TBCTR = 0x0000;                // Clear counter
    EPwm1Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;    // Clock ratio to
SYSCLKOUT
    EPwm1Regs.TBCTL.bit.CLKDIV = TB_DIV1;

    EPwm1Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;    // Load registers every
ZERO
    EPwm1Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
    EPwm1Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;
    EPwm1Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;

    // Setup compare
    EPwm1Regs.CMPA.half.CMPA = 0;
    EPwm1Regs.CMPB = 0;

    // Set actions
    EPwm1Regs.AQCTLA.bit.CAU = AQ_CLEAR;
    EPwm1Regs.AQCTLA.bit.CAD = AQ_SET;
    EPwm1Regs.AQCTLB.bit.CBU = AQ_CLEAR;
    EPwm1Regs.AQCTLB.bit.CBD = AQ_SET;

```

```

// Setup Deadband
EPwm1Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE;
EPwm1Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;
EPwm1Regs.DBCTL.bit.IN_MODE = DBA_ALL;
EPwm1Regs.DBRED = 45; //30;
EPwm1Regs.DBFED = 45; //30;

//ADC Triggering Setup
EPwm1Regs.ETSEL.bit.SOCAEN = 1;
EPwm1Regs.ETSEL.bit.SOCASEL = 0x2; //2; //0x2;    // Select SOC from from CPMA
on Period
    EPwm1Regs.ETPS.bit.SOCAPRD = 1;    // Generate pulse every event
}

void InitEPWM2()
{
    EPwm2Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN;
    EPwm2Regs.TBPRD = PWM_TBPRD;    // Period = 2*1000
TBCLK counts => 30 kHz
    EPwm2Regs.TBCTL.bit.PHSEN = TB_DISABLE;    // Enable phase
loading
    EPwm2Regs.TBPHS.half.TBPHS = 0;
    EPwm2Regs.TBCTL.bit.PHSDIR = TB_DOWN;
    EPwm2Regs.TBCTL.bit.PRDL = TB_SHADOW;
    EPwm2Regs.TBCTL.bit.SYNCSEL = TB_SYNC_IN; // sync flow-through
    EPwm2Regs.TBCTR = 0x0000;    // Clear counter
    EPwm2Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;    // Clock ratio to SYSCLKOUT
    EPwm2Regs.TBCTL.bit.CLKDIV = TB_DIV1;

    EPwm2Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW; // Load registers every
ZERO
    EPwm2Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
    EPwm2Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;
    EPwm2Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;

    // Setup compare
    EPwm2Regs.CMPA.half.CMPA = 0; //fixed 50% duty cycle
    EPwm2Regs.CMPB = 0;

    // Set actions qualifiers
    EPwm2Regs.AQCTLA.bit.CAU = AQ_CLEAR;
    EPwm2Regs.AQCTLA.bit.CAD = AQ_SET;
    EPwm2Regs.AQCTLB.bit.CBU = AQ_CLEAR;
    EPwm2Regs.AQCTLB.bit.CBD = AQ_SET;

```

```

//EPwm2Regs.ETSEL.bit.SOCAEN = 0;    // Disable SOC on A group

// Setup Deadband
EPwm2Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE;
EPwm2Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;
EPwm2Regs.DBCTL.bit.IN_MODE = DBA_ALL;
EPwm2Regs.DBRED = DeadTime;
EPwm2Regs.DBFED = DeadTime;

}

void InitEPWM3()
{
    EPwm3Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN;
    EPwm3Regs.TBPRD = PWM_TBPRD;    // Period = 2*1000 TBCLK counts =>
30 kHz
    EPwm3Regs.TBCTL.bit.PHSEN = TB_DISABLE;
    EPwm3Regs.TBPHS.half.TBPHS = 0;
    EPwm3Regs.TBCTL.bit.PHSDIR = TB_DOWN;
    EPwm3Regs.TBCTL.bit.PRDL = TB_SHADOW;
    EPwm3Regs.TBCTL.bit.SYNCSEL = TB_SYNC_IN;    // sync flow-through
    EPwm3Regs.TBCTR = 0x0000;    // Clear counter
    EPwm3Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;    // Clock ratio to
SYSCLKOUT
    EPwm3Regs.TBCTL.bit.CLKDIV = TB_DIV1;

    // Setup shadow register load on ZERO
    EPwm3Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
    EPwm3Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
    EPwm3Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;
    EPwm3Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;

    // Set Compare values
    EPwm3Regs.CMPA.half.CMPA = 0;    // Set compare A value
    EPwm3Regs.CMPB = 0;

    // Set action qualifiers
    EPwm3Regs.AQCTLA.bit.CAU = AQ_CLEAR;
    EPwm3Regs.AQCTLA.bit.CAD = AQ_SET;
    EPwm3Regs.AQCTLB.bit.CBU = AQ_CLEAR;
    EPwm3Regs.AQCTLB.bit.CBD = AQ_SET;

    //EPwm3Regs.AQCTLA.bit.CAU = AQ_SET;
    //EPwm3Regs.AQCTLA.bit.CAD = AQ_CLEAR;
    //EPwm3Regs.AQCTLB.bit.CBU = AQ_SET;

```

```

//EPwm3Regs.AQCTLB.bit.CBD = AQ_CLEAR;

// Setup Deadband
EPwm3Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE;
EPwm3Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;
EPwm3Regs.DBCTL.bit.IN_MODE = DBA_ALL;
EPwm3Regs.DBRED = DeadTime;
EPwm3Regs.DBFED = DeadTime;

}

void InitEPWM4()
{
    EPwm4Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN;
    EPwm4Regs.TBPRD = PWM_TBPRD;          // Period = 2*1000 TBCLK counts =>
30 kHz
    EPwm4Regs.TBCTL.bit.PHSEN = TB_DISABLE;
    EPwm4Regs.TBPHS.half.TBPHS = 0;
    EPwm4Regs.TBCTL.bit.PHSDIR = TB_DOWN;
    EPwm4Regs.TBCTL.bit.PRDL = TB_SHADOW;
    EPwm4Regs.TBCTL.bit.SYNCSEL = TB_SYNC_IN; // sync flow-through
    EPwm4Regs.TBCTR = 0x0000;                // Clear counter
    EPwm4Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1; // Clock ratio to
SYSCLKOUT
    EPwm4Regs.TBCTL.bit.CLKDIV = TB_DIV1;

    // Setup shadow register load on ZERO
    EPwm4Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
    EPwm4Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
    EPwm4Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;
    EPwm4Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;

    // Set Compare values
    EPwm4Regs.CMPA.half.CMPA = 0; // Set compare A value
    EPwm4Regs.CMPB = 0;

    // Set action qualifiers
    EPwm4Regs.AQCTLA.bit.CAU = AQ_CLEAR;
    EPwm4Regs.AQCTLA.bit.CAD = AQ_SET;
    EPwm4Regs.AQCTLB.bit.CBU = AQ_CLEAR;
    EPwm4Regs.AQCTLB.bit.CBD = AQ_SET;

    //EPwm4Regs.AQCTLA.bit.CAU = AQ_SET;
    //EPwm4Regs.AQCTLA.bit.CAD = AQ_CLEAR;
    //EPwm4Regs.AQCTLB.bit.CBU = AQ_SET;

```

```

//EPwm4Regs.AQCTLB.bit.CBD = AQ_CLEAR;

// Setup Deadband
// EPwm4Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE;
// EPwm4Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;
// EPwm4Regs.DBCTL.bit.IN_MODE = DBA_ALL;
// EPwm4Regs.DBRED = DeadTime;
// EPwm4Regs.DBFED = DeadTime;

}

//// Test 1,SCIA DLB, 8-bit word, baud rate 0x000F, default, 1 STOP bit, no parity
//void scia_echoback_init()
//{
// // Note: Clocks were turned on to the SCIA peripheral
// // in the InitSysCtrl() function
//
// SciaRegs.SCICCR.all =0x0007; // 1 stop bit, No loopback
// // No parity,8 char bits,
// // async mode, idle-line protocol
// SciaRegs.SCICTL1.all =0x0003; // enable TX, RX, internal SCICLK,
// // Disable RX ERR, SLEEP, TXWAKE
// SciaRegs.SCICTL2.all =0x0003;
// SciaRegs.SCICTL2.bit.TXINTENA =1;
// SciaRegs.SCICTL2.bit.RXBKINTENA =1;
// #if (CPU_FRQ_150MHZ)
// SciaRegs.SCIHBAUD =0x0001; // 9600 baud @LSPCLK = 37.5MHz.
// SciaRegs.SCILBAUD =0x00E7;
// #endif
// #if (CPU_FRQ_100MHZ)
// SciaRegs.SCIHBAUD =0x0001; // 9600 baud @LSPCLK = 20MHz.
// SciaRegs.SCILBAUD =0x0044;
// #endif
// SciaRegs.SCICTL1.all =0x0023; // Relinquish SCI from Reset
//}
//
//// Transmit a character from the SCI
//void scia_xmit(int a)
//{
// while (SciaRegs.SCIFFTX.bit.TXFFST != 0) {}
// SciaRegs.SCITXBUF=a;
//}
//
//void scia_msg(char * msg)
//{

```

```
// int i;
// i = 0;
// while(msg[i] != '\0')
// {
//     scia_xmit(msg[i]);
//     i++;
// }
//}
//
//// Initialize the SCI FIFO
//void scia_fifo_init()
//{
//    SciaRegs.SCIFFTX.all=0xE040;
//    SciaRegs.SCIFFRX.all=0x204f;
//    SciaRegs.SCIFFCT.all=0x0;
//}
```

## A-2 CPLD VHDL Code

### A-2-1 Main Top Function

```

-----

-- Company: University of Arkansas (NCREPT)
-- Engineer: Nicholas Blair
--
-- Create Date:          12Mar2020
-- Design Name:          CPLD_CSPR_v1
-- Module Name:          CPLD_CSPR_v1 - Behavioral
-- Project Name:          CPLD_CSPR_v1
-- Target Devices:       LCMXO2-7000HC-4FG484C (UCB v1.3a)
-- Tool versions:        Lattice Diamond_x64 Build 3.10.2.115.1
-- Description:
-- This Project was developed to demonstrate a system-level design which incorporates shared
-- registers and communications.
-- Eight individual on-board LEDs will be controlled via a simple PWM interface.
-- Each PWM will have its duty cycle set via system registers which may be modified via the
-- serial interface.
--
---- PinOut:
-- Signal          CPLD_Pin   Description
-- LED_1           R17        LED1          (Output)
-- LED_2           T18        LED2          (Output)
-- LED_3           R16        LED3          (Output)
-- LED_4           T17        LED4          (Output)
-- LED_5           Y21        LED5          (Output)
-- LED_6           Y20        LED6          (Output)
-- LED_7           U18        LED7          (Output)
-- LED_8           U17        LED8          (Output)
-- PWM_Test_Out    K2         PWM Test     (Output)
-- Usr_TX          W1         Serial-TX   (Output)
-- Usr_RX          V2         Serial-RX   (Input)
--
--
--
--
--
--
--
--
-- Additional Comments:
--
--
-----

```



```

Library IEEE;
use IEEE.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;

```

```

library machxo2;
use machxo2.all;

```

```

entity CPLD_Modbus_Top is
    Port (

```

	---- UCB TI FTDI Serial Interface	
	TI_RX_Slave : in STD_LOGIC;	-- Serial In for User
Control		
	TI_TX_Slave : out STD_LOGIC;	-- Serial Out for User
Control		
	---- UCB XPORT Ethernet Interface	
	XPORT_TX : in STD_LOGIC;	-- Serial In for
XPORT Comms		
	XPORT_RX : out STD_LOGIC;	-- Serial Out
for XPORT Comms		
--	XPORT_RTS : in STD_LOGIC;	-- Request to
Send for XPORT Comms		
--	XPORT_DTR : in STD_LOGIC;	-- Data
Terminal Ready for XPORT Comms		
--	XPORT_CTS : out STD_LOGIC;	-- Clear to
Send for XPORT Comms		
	---- UCB DIP Switches	
	SW_1 : in STD_LOGIC;	-- Board DIP
Switch		
	SW_2 : in STD_LOGIC;	-- Board DIP
Switch		
	SW_3 : in STD_LOGIC;	-- Board DIP
Switch		
	SW_4 : in STD_LOGIC;	-- Board DIP
Switch		
	---- UCB Push Buttons	
	BTN_1 : in STD_LOGIC;	-- Board Push
Button		
	BTN_2 : in STD_LOGIC;	-- Board Push
Button		

Button	BTN_3 : in STD_LOGIC;	-- Board Push
Button	BTN_4 : in STD_LOGIC;	-- Board Push
	---- UCB Board LEDs	
	LED_1 : out STD_LOGIC;	-- Board LED
	LED_2 : out STD_LOGIC;	-- Board LED
	LED_3 : out STD_LOGIC;	-- Board LED
	LED_4 : out STD_LOGIC;	-- Board LED
	LED_5 : out STD_LOGIC;	-- Board LED
	LED_6 : out STD_LOGIC;	-- Board LED
	LED_7 : out STD_LOGIC;	-- Board LED
	LED_8 : out STD_LOGIC;	-- Board LED
	---- UCB SPI ADC1	
Clock	ADC1_SCLK : out STD_LOGIC;	-- Serial SPI
ADC Data In	ADC1_DIN : out STD_LOGIC;	-- Serial SPI
ADC Chip Select	ADC1_CS :out STD_LOGIC;	-- Serial SPI
ADC Data Out	ADC1_DOUT : in STD_LOGIC;	-- Serial SPI
	---- UCB SPI ADC2	
Clock	ADC2_SCLK : out STD_LOGIC;	-- Serial SPI
ADC Data In	ADC2_DIN : out STD_LOGIC;	-- Serial SPI
ADC Chip Select	ADC2_CS :out STD_LOGIC;	-- Serial SPI
ADC Data Out	ADC2_DOUT : in STD_LOGIC;	-- Serial SPI
	---- DIMM_B Serial Interface	
Serial RX (DIMM_B_GPIO_28)	DIMM_B_SCI_RX : out STD_LOGIC;	-- DSP B
Serial TX (DIMM_B_GPIO_29)	DIMM_B_SCI_TX : in STD_LOGIC;	-- DSP B
	---- DIMM_B GPIO Pins	
Connector GPIO	DIMM_B_GPIO_00 : in STD_LOGIC;	-- DIMM_B
Connector GPIO	DIMM_B_GPIO_01 : in STD_LOGIC;	-- DIMM_B
Connector GPIO	DIMM_B_GPIO_02 : in STD_LOGIC;	-- DIMM_B
Connector GPIO	DIMM_B_GPIO_03 : in STD_LOGIC;	-- DIMM_B

Connector GPIO	DIMM_B_GPIO_04 : in STD_LOGIC;	-- DIMM_B
Connector GPIO	DIMM_B_GPIO_05 : in STD_LOGIC;	-- DIMM_B
Connector GPIO	DIMM_B_GPIO_06 : in STD_LOGIC;	-- DIMM_B
Connector GPIO	DIMM_B_GPIO_07 : in STD_LOGIC;	-- DIMM_B
Connector GPIO	DIMM_B_GPIO_08 : in STD_LOGIC;	-- DIMM_B
Connector GPIO	DIMM_B_GPIO_09 : in STD_LOGIC;	-- DIMM_B
Connector GPIO	DIMM_B_GPIO_10 : in STD_LOGIC;	-- DIMM_B
Connector GPIO	DIMM_B_GPIO_11 : in STD_LOGIC;	-- DIMM_B
Connector GPIO	DIMM_B_GPIO_12 : in STD_LOGIC;	-- DIMM_B
Connector GPIO	DIMM_B_GPIO_13 : in STD_LOGIC;	-- DIMM_B
Connector GPIO	DIMM_B_GPIO_14 : in STD_LOGIC;	-- DIMM_B
Connector GPIO	DIMM_B_GPIO_15 : in STD_LOGIC;	-- DIMM_B
Connector GPIO	DIMM_B_GPIO_16 : in STD_LOGIC;	-- DIMM_B
Connector GPIO	DIMM_B_GPIO_17 : in STD_LOGIC;	-- DIMM_B
Connector GPIO	DIMM_B_GPIO_18 : in STD_LOGIC;	-- DIMM_B
Connector GPIO	DIMM_B_GPIO_19 : in STD_LOGIC;	-- DIMM_B
Connector GPIO	DIMM_B_GPIO_20 : in STD_LOGIC;	-- DIMM_B
Connector GPIO	DIMM_B_GPIO_21 : in STD_LOGIC;	-- DIMM_B
Connector GPIO	DIMM_B_GPIO_22 : in STD_LOGIC;	-- DIMM_B
Connector GPIO	DIMM_B_GPIO_23 : in STD_LOGIC;	-- DIMM_B
Connector GPIO	DIMM_B_GPIO_24 : in STD_LOGIC;	-- DIMM_B
Connector GPIO	DIMM_B_GPIO_25 : in STD_LOGIC;	-- DIMM_B
Connector GPIO	DIMM_B_GPIO_26 : in STD_LOGIC;	-- DIMM_B

Connector GPIO	DIMM_B_GPIO_27 : in STD_LOGIC;	-- DIMM_B
Connector GPIO	--DIMM_B_GPIO_28 : in STD_LOGIC;	-- DIMM_B
Connector GPIO	--DIMM_B_GPIO_29 : in STD_LOGIC;	-- DIMM_B
Connector GPIO	DIMM_B_GPIO_30 : in STD_LOGIC;	-- DIMM_B
Connector GPIO	DIMM_B_GPIO_31 : in STD_LOGIC;	-- DIMM_B
Connector GPIO	DIMM_B_GPIO_32 : in STD_LOGIC;	-- DIMM_B
Connector GPIO	DIMM_B_GPIO_33 : in STD_LOGIC;	-- DIMM_B
Connector GPIO	DIMM_B_GPIO_34 : in STD_LOGIC;	-- DIMM_B
Connector GPIO	DIMM_B_GPIO_48 : in STD_LOGIC;	-- DIMM_B
Connector GPIO	DIMM_B_GPIO_49 : in STD_LOGIC;	-- DIMM_B
Connector GPIO	DIMM_B_GPIO_58 : in STD_LOGIC;	-- DIMM_B
Connector GPIO	DIMM_B_GPIO_59 : in STD_LOGIC;	-- DIMM_B
Connector GPIO	DIMM_B_GPIO_60 : in STD_LOGIC;	-- DIMM_B
Connector GPIO	DIMM_B_GPIO_61 : in STD_LOGIC;	-- DIMM_B
Connector GPIO	DIMM_B_GPIO_62 : in STD_LOGIC;	-- DIMM_B
Connector GPIO	DIMM_B_GPIO_63 : in STD_LOGIC;	-- DIMM_B
---- IDC_A GPIO Pins		
Connector GPIO	IDC_A_GPIO_00 : out STD_LOGIC;	-- IDC_A
Connector GPIO	IDC_A_GPIO_01 : out STD_LOGIC;	-- IDC_A
Connector GPIO	IDC_A_GPIO_02 : out STD_LOGIC;	-- IDC_A
Connector GPIO	IDC_A_GPIO_03 : out STD_LOGIC;	-- IDC_A
Connector GPIO	IDC_A_GPIO_04 : out STD_LOGIC;	-- IDC_A
Connector GPIO	IDC_A_GPIO_05 : out STD_LOGIC;	-- IDC_A

Connector GPIO	IDC_A_GPIO_06 : out STD_LOGIC;	-- IDC_A
Connector GPIO	IDC_A_GPIO_07 : out STD_LOGIC;	-- IDC_A
Connector GPIO	IDC_A_GPIO_08 : out STD_LOGIC;	-- IDC_A
Connector GPIO	IDC_A_GPIO_09 : out STD_LOGIC;	-- IDC_A
Connector GPIO	IDC_A_GPIO_10 : out STD_LOGIC;	-- IDC_A
Connector GPIO	IDC_A_GPIO_11 : out STD_LOGIC;	-- IDC_A
Connector GPIO	IDC_A_GPIO_12 : out STD_LOGIC;	-- IDC_A
Connector GPIO	IDC_A_GPIO_13 : out STD_LOGIC;	-- IDC_A
Connector GPIO	IDC_A_GPIO_14 : out STD_LOGIC;	-- IDC_A
Connector GPIO	IDC_A_GPIO_15 : out STD_LOGIC;	-- IDC_A
Connector GPIO	IDC_A_GPIO_16 : out STD_LOGIC;	-- IDC_A
Connector GPIO	IDC_A_GPIO_17 : out STD_LOGIC;	-- IDC_A
Connector GPIO	IDC_A_GPIO_18 : out STD_LOGIC;	-- IDC_A
Connector GPIO	IDC_A_GPIO_19 : out STD_LOGIC;	-- IDC_A
Connector GPIO	IDC_A_GPIO_20 : out STD_LOGIC;	-- IDC_A
Connector GPIO	IDC_A_GPIO_21 : out STD_LOGIC;	-- IDC_A
Connector GPIO	IDC_A_GPIO_22 : out STD_LOGIC;	-- IDC_A
Connector GPIO	IDC_A_GPIO_23 : out STD_LOGIC;	-- IDC_A
Connector GPIO	IDC_A_GPIO_24 : out STD_LOGIC;	-- IDC_A
Connector GPIO	IDC_A_GPIO_25 : out STD_LOGIC;	-- IDC_A
Connector GPIO	IDC_A_GPIO_26 : out STD_LOGIC;	-- IDC_A
Connector GPIO	IDC_A_GPIO_27 : out STD_LOGIC;	-- IDC_A
Connector GPIO	---- IDC_B GPIO Pins	

Connector GPIO	IDC_B_GPIO_00 : out STD_LOGIC;	-- IDC_B
Connector GPIO	IDC_B_GPIO_01 : out STD_LOGIC;	-- IDC_B
Connector GPIO	IDC_B_GPIO_02 : out STD_LOGIC;	-- IDC_B
Connector GPIO	IDC_B_GPIO_03 : out STD_LOGIC;	-- IDC_B
Connector GPIO	IDC_B_GPIO_04 : out STD_LOGIC;	-- IDC_B
Connector GPIO	IDC_B_GPIO_05 : out STD_LOGIC;	-- IDC_B
Connector GPIO	IDC_B_GPIO_06 : out STD_LOGIC;	-- IDC_B
Connector GPIO	IDC_B_GPIO_07 : out STD_LOGIC;	-- IDC_B
Connector GPIO	IDC_B_GPIO_08 : out STD_LOGIC;	-- IDC_B
Connector GPIO	IDC_B_GPIO_09 : out STD_LOGIC;	-- IDC_B
Connector GPIO	IDC_B_GPIO_10 : out STD_LOGIC;	-- IDC_B
Connector GPIO	IDC_B_GPIO_11 : out STD_LOGIC;	-- IDC_B
Connector GPIO	IDC_B_GPIO_12 : out STD_LOGIC;	-- IDC_B
Connector GPIO	IDC_B_GPIO_13 : out STD_LOGIC;	-- IDC_B
Connector GPIO	IDC_B_GPIO_14 : out STD_LOGIC;	-- IDC_B
Connector GPIO	IDC_B_GPIO_15 : out STD_LOGIC;	-- IDC_B
Connector GPIO	IDC_B_GPIO_16 : out STD_LOGIC;	-- IDC_B
Connector GPIO	IDC_B_GPIO_17 : out STD_LOGIC;	-- IDC_B
Connector GPIO	IDC_B_GPIO_18 : out STD_LOGIC;	-- IDC_B
Connector GPIO	IDC_B_GPIO_19 : out STD_LOGIC;	-- IDC_B
Connector GPIO	IDC_B_GPIO_20 : out STD_LOGIC;	-- IDC_B
Connector GPIO	IDC_B_GPIO_21 : out STD_LOGIC;	-- IDC_B
Connector GPIO	IDC_B_GPIO_22 : out STD_LOGIC;	-- IDC_B

```

Connector GPIO      IDC_B_GPIO_23 : out STD_LOGIC;          -- IDC_B
Connector GPIO      IDC_B_GPIO_24 : out STD_LOGIC;          -- IDC_B
Connector GPIO      IDC_B_GPIO_25 : out STD_LOGIC;          -- IDC_B
Connector GPIO      IDC_B_GPIO_26 : out STD_LOGIC;          -- IDC_B
Connector GPIO      IDC_B_GPIO_27 : out STD_LOGIC;          -- IDC_B

);

end CPLD_Modbus_Top;

architecture Behavioral of CPLD_Modbus_Top is
begin

    -- Main Routing Process (Combinatorial)
    --process(SW_1, SW_2, SW_3, SW_4,
    BTN_1,BTN_2,BTN_3,BTN_4,SCI_RX,DIMM_C_SCI_TX,
    DIMM_C_GPIO_00,DIMM_C_GPIO_01,DIMM_C_GPIO_02,DIMM_C_GPIO_03,DIMM_C_
    GPIO_04,DIMM_C_GPIO_05,DIMM_C_GPIO_06,DIMM_C_GPIO_07,DIMM_C_GPIO_08,
    DIMM_C_GPIO_09,DIMM_C_GPIO_10,DIMM_C_GPIO_11,DIMM_C_GPIO_12,DIMM_C_
    GPIO_13,DIMM_C_GPIO_14,DIMM_C_GPIO_15,DIMM_C_GPIO_16,DIMM_C_GPIO_17,
    DIMM_C_GPIO_18,DIMM_C_GPIO_19,DIMM_C_GPIO_20,DIMM_C_GPIO_21,DIMM_C_
    GPIO_22,DIMM_C_GPIO_23,DIMM_C_GPIO_24,DIMM_C_GPIO_25,DIMM_C_GPIO_26,
    DIMM_C_GPIO_27)
    --process(all)
    process(DIMM_B_GPIO_00,DIMM_B_GPIO_01,DIMM_B_GPIO_02,DIMM_B_GPIO
    _03,DIMM_B_GPIO_04,DIMM_B_GPIO_05)
    begin
        if (SW_1 = '1') then
            -- DIMM_B Active
            -- For Routing Serial Communications
            DIMM_B_SCI_RX <= TI_RX_Slave;          -- Send Data
            Recieved from On-Board FTDI serial interface to DSP
            TI_TX_Slave <= DIMM_B_SCI_TX;          -- Send Data
            Recieved from DSP to On-Board FTDI serial interface
            if(SW_2 = '1') then
                --IDC_A is Primary and IDC_B is Secondary
                IDC_A_GPIO_00 <= DIMM_B_GPIO_00 AND
            NOT(DIMM_B_GPIO_01);
                IDC_A_GPIO_01 <= DIMM_B_GPIO_01 AND
            NOT(DIMM_B_GPIO_00);
                IDC_A_GPIO_02 <= DIMM_B_GPIO_02 AND
            NOT(DIMM_B_GPIO_03);

```

```

NOT(DIMM_B_GPIO_02);
NOT(DIMM_B_GPIO_05);
NOT(DIMM_B_GPIO_04);

IDC_A_GPIO_03 <= DIMM_B_GPIO_03 AND
IDC_A_GPIO_04 <= DIMM_B_GPIO_04 AND
IDC_A_GPIO_05 <= DIMM_B_GPIO_05 AND

IDC_A_GPIO_06 <= DIMM_B_GPIO_06;
IDC_A_GPIO_07 <= DIMM_B_GPIO_07;
IDC_A_GPIO_08 <= DIMM_B_GPIO_08;
IDC_A_GPIO_09 <= DIMM_B_GPIO_09;
IDC_A_GPIO_10 <= DIMM_B_GPIO_10;
IDC_A_GPIO_11 <= DIMM_B_GPIO_11;
IDC_A_GPIO_12 <= DIMM_B_GPIO_12;
IDC_A_GPIO_13 <= DIMM_B_GPIO_13;
IDC_A_GPIO_14 <= DIMM_B_GPIO_14;
IDC_A_GPIO_15 <= DIMM_B_GPIO_15;
IDC_A_GPIO_16 <= DIMM_B_GPIO_16;
IDC_A_GPIO_17 <= DIMM_B_GPIO_17;
IDC_A_GPIO_18 <= DIMM_B_GPIO_18;
IDC_A_GPIO_19 <= DIMM_B_GPIO_19;
IDC_A_GPIO_20 <= DIMM_B_GPIO_20;
IDC_A_GPIO_21 <= DIMM_B_GPIO_21;
IDC_A_GPIO_22 <= DIMM_B_GPIO_22;
IDC_A_GPIO_23 <= DIMM_B_GPIO_23;
IDC_A_GPIO_24 <= DIMM_B_GPIO_24;
IDC_A_GPIO_25 <= DIMM_B_GPIO_25;
IDC_A_GPIO_26 <= DIMM_B_GPIO_26;
IDC_A_GPIO_27 <= DIMM_B_GPIO_27;
--IDC_B_GPIO is set to Secondary
IDC_B_GPIO_00 <= DIMM_B_GPIO_30;
IDC_B_GPIO_01 <= DIMM_B_GPIO_31;
IDC_B_GPIO_02 <= DIMM_B_GPIO_32;
IDC_B_GPIO_03 <= DIMM_B_GPIO_33;
IDC_B_GPIO_04 <= DIMM_B_GPIO_34;
IDC_B_GPIO_05 <= DIMM_B_GPIO_48;
IDC_B_GPIO_06 <= DIMM_B_GPIO_49;
IDC_B_GPIO_07 <= DIMM_B_GPIO_58;
IDC_B_GPIO_08 <= DIMM_B_GPIO_59;
IDC_B_GPIO_09 <= DIMM_B_GPIO_60;
IDC_B_GPIO_10 <= DIMM_B_GPIO_61;
IDC_B_GPIO_11 <= DIMM_B_GPIO_62;
IDC_B_GPIO_12 <= DIMM_B_GPIO_63;
IDC_B_GPIO_13 <= '0';
IDC_B_GPIO_14 <= '0';
IDC_B_GPIO_15 <= '0';
IDC_B_GPIO_16 <= '0';

```



```

IDC_B_GPIO_17 <= '0';
IDC_B_GPIO_18 <= '0';
IDC_B_GPIO_19 <= '0';
IDC_B_GPIO_20 <= '0';
IDC_B_GPIO_21 <= '0';
IDC_B_GPIO_22 <= '0';
IDC_B_GPIO_23 <= '0';
IDC_B_GPIO_24 <= '0';
IDC_B_GPIO_25 <= '0';
IDC_B_GPIO_26 <= '0';
IDC_B_GPIO_27 <= '0';
else
--IDC_B is Primary and IDC_A is Secondary
IDC_B_GPIO_00 <= DIMM_B_GPIO_00;
IDC_B_GPIO_01 <= DIMM_B_GPIO_01;
IDC_B_GPIO_02 <= DIMM_B_GPIO_02;
IDC_B_GPIO_03 <= DIMM_B_GPIO_03;
IDC_B_GPIO_04 <= DIMM_B_GPIO_04;
IDC_B_GPIO_05 <= DIMM_B_GPIO_05;
IDC_B_GPIO_06 <= DIMM_B_GPIO_06;
IDC_B_GPIO_07 <= DIMM_B_GPIO_07;
IDC_B_GPIO_08 <= DIMM_B_GPIO_08;
IDC_B_GPIO_09 <= DIMM_B_GPIO_09;
IDC_B_GPIO_10 <= DIMM_B_GPIO_10;
IDC_B_GPIO_11 <= DIMM_B_GPIO_11;
IDC_B_GPIO_12 <= DIMM_B_GPIO_12;
IDC_B_GPIO_13 <= DIMM_B_GPIO_13;
IDC_B_GPIO_14 <= DIMM_B_GPIO_14;
IDC_B_GPIO_15 <= DIMM_B_GPIO_15;
IDC_B_GPIO_16 <= DIMM_B_GPIO_16;
IDC_B_GPIO_17 <= DIMM_B_GPIO_17;
IDC_B_GPIO_18 <= DIMM_B_GPIO_18;
IDC_B_GPIO_19 <= DIMM_B_GPIO_19;
IDC_B_GPIO_20 <= DIMM_B_GPIO_20;
IDC_B_GPIO_21 <= DIMM_B_GPIO_21;
IDC_B_GPIO_22 <= DIMM_B_GPIO_22;
IDC_B_GPIO_23 <= DIMM_B_GPIO_23;
IDC_B_GPIO_24 <= DIMM_B_GPIO_24;
IDC_B_GPIO_25 <= DIMM_B_GPIO_25;
IDC_B_GPIO_26 <= DIMM_B_GPIO_26;
IDC_B_GPIO_27 <= DIMM_B_GPIO_27;
--IDC_A_GPIO is set to Secondary
IDC_A_GPIO_00 <= DIMM_B_GPIO_30;
IDC_A_GPIO_01 <= DIMM_B_GPIO_31;
IDC_A_GPIO_02 <= DIMM_B_GPIO_32;
IDC_A_GPIO_03 <= DIMM_B_GPIO_33;

```

```

IDC_A_GPIO_04 <= DIMM_B_GPIO_34;
IDC_A_GPIO_05 <= DIMM_B_GPIO_48;
IDC_A_GPIO_06 <= DIMM_B_GPIO_49;
IDC_A_GPIO_07 <= DIMM_B_GPIO_58;
IDC_A_GPIO_08 <= DIMM_B_GPIO_59;
IDC_A_GPIO_09 <= DIMM_B_GPIO_60;
IDC_A_GPIO_10 <= DIMM_B_GPIO_61;
IDC_A_GPIO_11 <= DIMM_B_GPIO_62;
IDC_A_GPIO_12 <= DIMM_B_GPIO_63;
IDC_A_GPIO_13 <= '0';
IDC_A_GPIO_14 <= '0';
IDC_A_GPIO_15 <= '0';
IDC_A_GPIO_16 <= '0';
IDC_A_GPIO_17 <= '0';
IDC_A_GPIO_18 <= '0';
IDC_A_GPIO_19 <= '0';
IDC_A_GPIO_20 <= '0';
IDC_A_GPIO_21 <= '0';
IDC_A_GPIO_22 <= '0';
IDC_A_GPIO_23 <= '0';
IDC_A_GPIO_24 <= '0';
IDC_A_GPIO_25 <= '0';
IDC_A_GPIO_26 <= '0';
IDC_A_GPIO_27 <= '0';
end if;

else
-- DIMM_B Active
-- For Routing Serial Communications
XPort Server to DSP
DIMM_B_SCI_RX <= XPORT_TX;-- Send Data Recieved from
DSP to XPort Server
XPORT_RX <= DIMM_B_SCI_TX;-- Send Data Recieved from

if(SW_2 = '1') then
--IDC_A is Primary and IDC_B is Secondary
IDC_A_GPIO_00 <= DIMM_B_GPIO_00;
IDC_A_GPIO_01 <= DIMM_B_GPIO_01;
IDC_A_GPIO_02 <= DIMM_B_GPIO_02;
IDC_A_GPIO_03 <= DIMM_B_GPIO_03;
IDC_A_GPIO_04 <= DIMM_B_GPIO_04;
IDC_A_GPIO_05 <= DIMM_B_GPIO_05;
IDC_A_GPIO_06 <= DIMM_B_GPIO_06;
IDC_A_GPIO_07 <= DIMM_B_GPIO_07;
IDC_A_GPIO_08 <= DIMM_B_GPIO_08;
IDC_A_GPIO_09 <= DIMM_B_GPIO_09;
IDC_A_GPIO_10 <= DIMM_B_GPIO_10;

```

```

IDC_A_GPIO_11 <= DIMM_B_GPIO_11;
IDC_A_GPIO_12 <= DIMM_B_GPIO_12;
IDC_A_GPIO_13 <= DIMM_B_GPIO_13;
IDC_A_GPIO_14 <= DIMM_B_GPIO_14;
IDC_A_GPIO_15 <= DIMM_B_GPIO_15;
IDC_A_GPIO_16 <= DIMM_B_GPIO_16;
IDC_A_GPIO_17 <= DIMM_B_GPIO_17;
IDC_A_GPIO_18 <= DIMM_B_GPIO_18;
IDC_A_GPIO_19 <= DIMM_B_GPIO_19;
IDC_A_GPIO_20 <= DIMM_B_GPIO_20;
IDC_A_GPIO_21 <= DIMM_B_GPIO_21;
IDC_A_GPIO_22 <= DIMM_B_GPIO_22;
IDC_A_GPIO_23 <= DIMM_B_GPIO_23;
IDC_A_GPIO_24 <= DIMM_B_GPIO_24;
IDC_A_GPIO_25 <= DIMM_B_GPIO_25;
IDC_A_GPIO_26 <= DIMM_B_GPIO_26;
IDC_A_GPIO_27 <= DIMM_B_GPIO_27;
--IDC_B_GPIO is set to Secondary
IDC_B_GPIO_00 <= DIMM_B_GPIO_30;
IDC_B_GPIO_01 <= DIMM_B_GPIO_31;
IDC_B_GPIO_02 <= DIMM_B_GPIO_32;
IDC_B_GPIO_03 <= DIMM_B_GPIO_33;
IDC_B_GPIO_04 <= DIMM_B_GPIO_34;
IDC_B_GPIO_05 <= DIMM_B_GPIO_48;
IDC_B_GPIO_06 <= DIMM_B_GPIO_49;
IDC_B_GPIO_07 <= DIMM_B_GPIO_58;
IDC_B_GPIO_08 <= DIMM_B_GPIO_59;
IDC_B_GPIO_09 <= DIMM_B_GPIO_60;
IDC_B_GPIO_10 <= DIMM_B_GPIO_61;
IDC_B_GPIO_11 <= DIMM_B_GPIO_62;
IDC_B_GPIO_12 <= DIMM_B_GPIO_63;
IDC_B_GPIO_13 <= '0';
IDC_B_GPIO_14 <= '0';
IDC_B_GPIO_15 <= '0';
IDC_B_GPIO_16 <= '0';
IDC_B_GPIO_17 <= '0';
IDC_B_GPIO_18 <= '0';
IDC_B_GPIO_19 <= '0';
IDC_B_GPIO_20 <= '0';
IDC_B_GPIO_21 <= '0';
IDC_B_GPIO_22 <= '0';
IDC_B_GPIO_23 <= '0';
IDC_B_GPIO_24 <= '0';
IDC_B_GPIO_25 <= '0';
IDC_B_GPIO_26 <= '0';
IDC_B_GPIO_27 <= '0';

```

else

--IDC\_B is Primary and IDC\_A is Secondary

```
IDC_B_GPIO_00 <= DIMM_B_GPIO_00;
IDC_B_GPIO_01 <= DIMM_B_GPIO_01;
IDC_B_GPIO_02 <= DIMM_B_GPIO_02;
IDC_B_GPIO_03 <= DIMM_B_GPIO_03;
IDC_B_GPIO_04 <= DIMM_B_GPIO_04;
IDC_B_GPIO_05 <= DIMM_B_GPIO_05;
IDC_B_GPIO_06 <= DIMM_B_GPIO_06;
IDC_B_GPIO_07 <= DIMM_B_GPIO_07;
IDC_B_GPIO_08 <= DIMM_B_GPIO_08;
IDC_B_GPIO_09 <= DIMM_B_GPIO_09;
IDC_B_GPIO_10 <= DIMM_B_GPIO_10;
IDC_B_GPIO_11 <= DIMM_B_GPIO_11;
IDC_B_GPIO_12 <= DIMM_B_GPIO_12;
IDC_B_GPIO_13 <= DIMM_B_GPIO_13;
IDC_B_GPIO_14 <= DIMM_B_GPIO_14;
IDC_B_GPIO_15 <= DIMM_B_GPIO_15;
IDC_B_GPIO_16 <= DIMM_B_GPIO_16;
IDC_B_GPIO_17 <= DIMM_B_GPIO_17;
IDC_B_GPIO_18 <= DIMM_B_GPIO_18;
IDC_B_GPIO_19 <= DIMM_B_GPIO_19;
IDC_B_GPIO_20 <= DIMM_B_GPIO_20;
IDC_B_GPIO_21 <= DIMM_B_GPIO_21;
IDC_B_GPIO_22 <= DIMM_B_GPIO_22;
IDC_B_GPIO_23 <= DIMM_B_GPIO_23;
IDC_B_GPIO_24 <= DIMM_B_GPIO_24;
IDC_B_GPIO_25 <= DIMM_B_GPIO_25;
IDC_B_GPIO_26 <= DIMM_B_GPIO_26;
IDC_B_GPIO_27 <= DIMM_B_GPIO_27;
```

--IDC\_A\_GPIO is set to Secondary

```
IDC_A_GPIO_00 <= DIMM_B_GPIO_30;
IDC_A_GPIO_01 <= DIMM_B_GPIO_31;
IDC_A_GPIO_02 <= DIMM_B_GPIO_32;
IDC_A_GPIO_03 <= DIMM_B_GPIO_33;
IDC_A_GPIO_04 <= DIMM_B_GPIO_34;
IDC_A_GPIO_05 <= DIMM_B_GPIO_48;
IDC_A_GPIO_06 <= DIMM_B_GPIO_49;
IDC_A_GPIO_07 <= DIMM_B_GPIO_58;
IDC_A_GPIO_08 <= DIMM_B_GPIO_59;
IDC_A_GPIO_09 <= DIMM_B_GPIO_60;
IDC_A_GPIO_10 <= DIMM_B_GPIO_61;
IDC_A_GPIO_11 <= DIMM_B_GPIO_62;
IDC_A_GPIO_12 <= DIMM_B_GPIO_63;
IDC_A_GPIO_13 <= '0';
IDC_A_GPIO_14 <= '0';
```

```

        IDC_A_GPIO_15 <= '0';
        IDC_A_GPIO_16 <= '0';
        IDC_A_GPIO_17 <= '0';
        IDC_A_GPIO_18 <= '0';
        IDC_A_GPIO_19 <= '0';
        IDC_A_GPIO_20 <= '0';
        IDC_A_GPIO_21 <= '0';
        IDC_A_GPIO_22 <= '0';
        IDC_A_GPIO_23 <= '0';
        IDC_A_GPIO_24 <= '0';
        IDC_A_GPIO_25 <= '0';
        IDC_A_GPIO_26 <= '0';
        IDC_A_GPIO_27 <= '0';
    end if;

end if;

-- For Testing Onboard Buttons, Switches, and LEDs.
LED_1 <= BTN_1;
LED_2 <= BTN_2;
LED_3 <= BTN_3;
LED_4 <= BTN_4;
LED_5 <= SW_1;
LED_6 <= SW_2;
LED_7 <= SW_3;
LED_8 <= SW_4;

end process;

end Behavioral;

```

### A-2-2 Top Testbench Simulation

```

Library IEEE;
use IEEE.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;

library machxo2;
use machxo2.all;

entity CPLD_Testbench is

end CPLD_Testbench;

```

architecture Behavioral of CPLD\_Testbench is

```

    COMPONENT CPLD_Modbus_Top
    PORT (

        ---- DIMM_B GPIO Pins
        DIMM_B_GPIO_00 : in  STD_LOGIC;           -- DIMM_B
Connector GPIO
        DIMM_B_GPIO_01 : in  STD_LOGIC;           -- DIMM_B
Connector GPIO
        DIMM_B_GPIO_02 : in  STD_LOGIC;           -- DIMM_B
Connector GPIO
        DIMM_B_GPIO_03 : in  STD_LOGIC;           -- DIMM_B
Connector GPIO
        DIMM_B_GPIO_04 : in  STD_LOGIC;           -- DIMM_B
Connector GPIO
        DIMM_B_GPIO_05 : in  STD_LOGIC;           -- DIMM_B
Connector GPIO

        ---- IDC_A GPIO Pins
        IDC_A_GPIO_00 : out  STD_LOGIC;           -- IDC_A
Connector GPIO
        IDC_A_GPIO_01 : out  STD_LOGIC;           -- IDC_A
Connector GPIO
        IDC_A_GPIO_02 : out  STD_LOGIC;           -- IDC_A
Connector GPIO
        IDC_A_GPIO_03 : out  STD_LOGIC;           -- IDC_A
Connector GPIO
        IDC_A_GPIO_04 : out  STD_LOGIC;           -- IDC_A
Connector GPIO
        IDC_A_GPIO_05 : out  STD_LOGIC           -- IDC_A Connector
GPIO

    );
END COMPONENT;

SIGNAL s_CLK : STD_LOGIC := '0';
SIGNAL s_IDC_A_GPIO_00 : STD_LOGIC := '0';
SIGNAL s_IDC_A_GPIO_01 : STD_LOGIC := '0';
SIGNAL s_IDC_A_GPIO_02 : STD_LOGIC := '0';
SIGNAL s_IDC_A_GPIO_03 : STD_LOGIC := '0';
SIGNAL s_IDC_A_GPIO_04 : STD_LOGIC := '0';
SIGNAL s_IDC_A_GPIO_05 : STD_LOGIC := '0';
SIGNAL s_DIMM_B_GPIO_00 : STD_LOGIC := '0';
SIGNAL s_DIMM_B_GPIO_01 : STD_LOGIC := '0';

```

```

SIGNAL s_DIMM_B_GPIO_02 : STD_LOGIC := '0';
SIGNAL s_DIMM_B_GPIO_03 : STD_LOGIC := '0';
SIGNAL s_DIMM_B_GPIO_04 : STD_LOGIC := '0';
SIGNAL s_DIMM_B_GPIO_05 : STD_LOGIC := '0';

```

```

constant CLK : TIME := 10ns;

```

```

begin

```

```

    UUT: CPLD_Modbus_Top PORT MAP(

```

```

        --INPUTS

```

```

        IDC_A_GPIO_00 => s_IDC_A_GPIO_00,
        IDC_A_GPIO_01 => s_IDC_A_GPIO_01,
        IDC_A_GPIO_02 => s_IDC_A_GPIO_02,
        IDC_A_GPIO_03 => s_IDC_A_GPIO_03,
        IDC_A_GPIO_04 => s_IDC_A_GPIO_04,
        IDC_A_GPIO_05 => s_IDC_A_GPIO_05,

```

```

        --OUTPUTS

```

```

        DIMM_B_GPIO_00 => s_DIMM_B_GPIO_00,
        DIMM_B_GPIO_01 => s_DIMM_B_GPIO_01,
        DIMM_B_GPIO_02 => s_DIMM_B_GPIO_02,
        DIMM_B_GPIO_03 => s_DIMM_B_GPIO_03,
        DIMM_B_GPIO_04 => s_DIMM_B_GPIO_04,
        DIMM_B_GPIO_05 => s_DIMM_B_GPIO_05
    );

```

```

--    clk:PROCESS

```

```

--        begin

```

```

--            s_CLK_I<='1';
--            wait for CLK/2;
--            s_CLK_I<='0';
--            wait for CLK/2;
--        END PROCESS

```

```

PROCESS

```

```

begin

```

```

    wait for 100ns;
    s_DIMM_B_GPIO_00 <= '0';
    s_DIMM_B_GPIO_01 <= '0';
    s_DIMM_B_GPIO_02 <= '0';
    s_DIMM_B_GPIO_03 <= '0';
    s_DIMM_B_GPIO_04 <= '0';
    s_DIMM_B_GPIO_05 <= '0';

```

```

wait for 100ns;
s_DIMM_B_GPIO_00 <= '1';
s_DIMM_B_GPIO_01 <= '0';
s_DIMM_B_GPIO_02 <= '0';
s_DIMM_B_GPIO_03 <= '1';
s_DIMM_B_GPIO_04 <= '1';
s_DIMM_B_GPIO_05 <= '0';

```

```

wait for 100ns;
s_DIMM_B_GPIO_00 <= '1';
s_DIMM_B_GPIO_01 <= '0';
s_DIMM_B_GPIO_02 <= '0';
s_DIMM_B_GPIO_03 <= '1';
s_DIMM_B_GPIO_04 <= '0';
s_DIMM_B_GPIO_05 <= '1';

```

```

wait for 100ns;
s_DIMM_B_GPIO_00 <= '1';
s_DIMM_B_GPIO_01 <= '0';
s_DIMM_B_GPIO_02 <= '1';
s_DIMM_B_GPIO_03 <= '0';
s_DIMM_B_GPIO_04 <= '0';
s_DIMM_B_GPIO_05 <= '1';

```

```

        wait for 100ns;
s_DIMM_B_GPIO_00 <= '1';
s_DIMM_B_GPIO_01 <= '1';
s_DIMM_B_GPIO_02 <= '1';
s_DIMM_B_GPIO_03 <= '1';
s_DIMM_B_GPIO_04 <= '0';
s_DIMM_B_GPIO_05 <= '1';

```

```

END PROCESS;

```

```

end Behavioral;

```

### A-3 Web Server Python Code

```

# This file handles the communication for modbus
import minimalmodbus
# Port name, slave address (in decimal)
instrument = minimalmodbus.Instrument('/dev/ttyUSB0',1)

# Serial Port Name

```



```

instrument.serial.port = '/dev/ttyUSB0'

# Baud Rate
instrument.serial.baudrate = 115200

# Byte size
instrument.serial.bytesize = 8

# Stop Bits
instrument.serial.stopbits = 2

# Slave Address
instrument.address

# RTU or ASCII Mode
instrument.mode = minimalmodbus.MODE_RTU

# Clear Buffer Before Each Transaction
instrument.clear_buffers_before_each_transaction = True

# Timeout (in seconds)
instrument.serial.timeout = 2

# To Display Current Settings
print("Slave Address:", instrument.address)
print("Port Name:", instrument.serial.port)
print("Mode:", minimalmodbus.MODE_RTU)
print("Baud Rate:", instrument.serial.baudrate)
print("Byte Size:", instrument.serial.bytesize)
print("Parity:", instrument.serial.parity)
print("Stop Bits:", instrument.serial.stopbits, '\n')

def mod_read():
    print("In serial function")
    #s = 2
    #t = 8
    #instrument.write_register(int(s), int(t), 0)
    #i = 0

    # Registers 0-31 allocated as read registers
    V_DCin = instrument.read_register(4,0)
    V_PhA = instrument.read_register(5,0)
    V_PhB = instrument.read_register(6,0)
    V_PhC = instrument.read_register(7,0)
    V_Buck = instrument.read_register(8,0)

```

```

V_Boost = instrument.read_register(9,0)
I_DCin = instrument.read_register(10,0)
V_Pot = instrument.read_register(11,0)
I_DCin2 = instrument.read_register(12,0)
I_L1 = instrument.read_register(13,0)
I_L2 = instrument.read_register(14,0)
I_L3 = instrument.read_register(15,0)
I_Buck = instrument.read_register(16,0)
I_Boost = instrument.read_register(17,0)
Ia = instrument.read_register(18,0)
I_L5 = instrument.read_register(19,0)

arr = [V_DCin, V_PhA, V_PhB, V_PhC, V_Buck, V_Boost, I_DCin, V_Pot, I_DCin2, I_L1,
I_L2, I_L3, I_Buck, I_Boost, Ia, I_L5]

print(arr)
return arr

def mod_write(arr):

    # arr = [myfreq, mybuck, myboost, myvac, myvacfund, mybuckkp, mybuckki, myboostkp,
myboostki]
    print("In Modbus Write Function")

    a = int(format(30,"b")+ '00000',2)
    print(a)

    arr[0] = int(format(int(arr[0]),"b")+ '00000',2)
    arr[1] = int(format(int(arr[1]),"b")+ '00000',2)
    arr[2] = int(format(int(arr[2]),"b")+ '00000',2)
    arr[3] = int(format(int(arr[3]),"b")+ '00000',2)
    arr[4] = int(format(int(arr[4]),"b")+ '00000',2)
    arr[5] = int(format(int(arr[5]),"b")+ '00000',2)
    arr[6] = int(format(int(arr[6]),"b")+ '00000',2)
    arr[7] = int(format(int(arr[7]),"b")+ '00000',2)
    arr[8] = int(format(int(arr[8]),"b")+ '00000',2)

    # Registers 32-63 allocated as write registers
    instrument.write_register(20, int(arr[0]), 0)
    instrument.write_register(21, int(arr[1]), 0)
    instrument.write_register(22, int(arr[2]), 0)
    instrument.write_register(23, int(arr[3]), 0)
    instrument.write_register(24, int(arr[4]), 0)
    instrument.write_register(25, int(arr[5]), 0)
    instrument.write_register(26, int(arr[6]), 0)
    instrument.write_register(27, int(arr[7]), 0)

```

```
instrument.write_register(28, int(arr[8]), 0)
```

```
print(arr)
```

```
if __name__ == "__main__":  
    mod_func()
```