

5-2020

Cybersecurity Methods for Grid-Connected Power Electronics

Stephen Joe Moquin
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 Architecture Commons](#)

Citation

Moquin, S. J. (2020). Cybersecurity Methods for Grid-Connected Power Electronics. *Graduate Theses and Dissertations* Retrieved from <https://scholarworks.uark.edu/etd/3680>

This Thesis is brought to you for free and open access by ScholarWorks@UARK. It has been accepted for inclusion in Graduate Theses and Dissertations by an authorized administrator of ScholarWorks@UARK. For more information, please contact scholar@uark.edu, uarepos@uark.edu.

Cybersecurity Methods for Grid-Connected Power Electronics

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

by

Stephen Joe Moquin
Auburn University
Bachelor of Science in Philosophy, 2009
University of Arkansas
Bachelor of Science in Electrical Engineering, 2017

May 2020
University of Arkansas

This thesis is approved for recommendation to the Graduate Council.

H. Alan Mantoosh, Ph.D.
Thesis Director

Jia Di, Ph.D.
Committee Member

Roy McCann, Ph.D.
Committee Member

Yue Zhao, Ph.D.
Committee Member

Chris Farnell
Committee Member

Abstract

The present work shows a secure-by-design process, defense-in-depth method, and security techniques for a secure distributed energy resource. The distributed energy resource is a cybersecure, solar inverter and battery energy storage system prototype, collectively called the Cybersecure Power Router. Consideration is given to the use of the Smart Green Power Node for a foundation of the present work. Metrics for controller security are investigated to evaluate firmware security techniques. The prototype's ability to mitigate, respond to, and recover from firmware integrity degradation is examined. The prototype shows many working security techniques within the context of a grid-connected, distributed energy resource. Further work is expected in the Cybersecure Power Router project. Consideration is also provided for the migration of the present research and the Smart Green Power Node to realize a pre-production prototype.

©2020 by Stephen Joe Moquin
All Rights Reserved

Acknowledgements

I would like to thank my advisor, Dr. Alan Mantooth, for taking a chance on me, affording many lessons on leadership, and the opportunity to participate in the environment he has so diligently cultivated over the years. It is an honor to work in the Power Mixed-Signal Computer Aided Design research group. Dr. Jia Di, Dr. Roy McCann, and Dr. Yue Zhao are owed much thanks for their advising and contributions to the success of my academic career. I owe a great deal to Chris Farnell, and have benefitted greatly from the years of mentoring and friendship he has provided me. The present work would not be possible without the edifice of work Chris Farnell has constructed, and the foundations laid by Dr. Mantooth and the fellow research faculty of the University of Arkansas. I am thankful for my teammates SangYun Kim and Nicholas Blair. Dr. Shannon Davis, Beth Benham, Karin Alvarado, Jamie Stafford, Cindy Pickney, Sharon Brasko, Connie Howard, and Tracey Long are owed thanks for their care and administration. Eric den Boer, Jeff Knox, and Mike Steger are owed thanks for their practical wisdom in life and in research. I am thankful for the support of my past and present fellow graduate students: Haider Mhiesan, Hamdi Albunashee, Haoyan Liu, Yuzhi Zhang, Janviere Umuhoza, Shuang Zhao, Yusi Liu, Sayan Seal, Andrea Wallace, and Audrey Dearien. Finally, I'd like to thank the Department of Energy and industrial partners who contributed to the Secure Evolvable Energy Delivery Systems research center for the funding of both the work and researchers of the Cybersecure Power Router.

Table of Contents

Chapter 1 - Introduction.....	1
Chapter 2 - Technical Background	5
Chapter 3 - Cybersecure Power Router	13
3.1 System Description	13
3.2 Power Electronics	16
3.3 Digital Signal Processor Board.....	19
3.4 Complex Programmable Logic Device Board.....	22
3.5 Signal Splitter.....	26
3.6 Hardware Authentication Module.....	27
3.7 BeagleBone Black.....	29
3.8 Test Bed	30
3.9 Power Flow	30
3.10 Data Flow.....	31
3.11 Control Multiplexing	33
3.12 Firmware and Boot Management.....	36
3.13 Hardware Authentication	37
3.14 Submodule Encrypted Communication	40
3.15 Hardware Protections.....	41
3.16 Display	42
Chapter 4 - Results.....	44
Chapter 5 - Future Work.....	50
5.1 Multi-Mission Controls.....	50
5.2 SGPN and CSPR Integration and Completion.....	51
Chapter 6 - Conclusion	55
References.....	56
Appendix.....	61
Appendix A: Hardware and Software Design Details	61
Appendix B: EEPROM_WRITE_PASSWORD	67
Appendix C: CSPR_V7.lpf.....	69
Appendix D: hardware_protections.vhd	75
Appendix E: CSPR_MODULES.vhdl	76
Appendix F: top.vhdl	131

List of Figures

Fig. 1. Design Inventory for Distributed Energy Resource	2
Fig. 2: Cost/Performance regions (left) for various security solutions (right).....	8
Fig. 3. Cybersecure Power Router prototype.....	13
Fig. 4. Block diagram (left) and figure (right) of Cybersecure Power Router prototype	14
Fig. 5. Minimal configuration of Cybersecure Power Router prototype	15
Fig. 6. Asynchronous buck converter schematic of Power Electronics Evaluation Board of the UCB project	16
Fig. 7. Asynchronous boost converter schematic of Power Electronics Evaluation Board of the UCB project	17
Fig. 8. 3-Phase inverter filter and current sensing schematic of Power Electronics Evaluation Board of the UCB project	17
Fig. 9. 3-Phase inverter/rectifier switching stage schematic of Power Electronics Evaluation Board of UCB project	18
Fig. 10. Power Electronics Evaluation Unified Controller Board, with Signal Splitter and Hardware Authentication Module, used in Cybersecure Power Router prototype.....	19
Fig. 11. USB to UART schematic of Digital Signal Processor board of the UCB project.....	20
Fig. 12. DIMM pinout of Digital Signal Processor board of the UCB project.....	21
Fig. 13. Majority of analog and digital I/O used by Digital Signal Processor board of UCB project	22
Fig. 14. IDCs in Complex Programmable Logic Device Unified Controller Board	23
Fig. 15. PCB layout of Complex Programmable Logic Device board of UCB project.....	25
Fig. 16. Fabricated Complex Programmable Logic Device board of UCB project used in Cybersecure Power Router prototype	26
Fig. 17. Complete schematic of Signal Splitter	27
Fig. 18. Complete schematic of Hardware Authentication Module	28
Fig. 19. Hardware Authentication Module used in Cybersecure Power Router prototype	28

Fig. 20. BeagleBone Black	29
Fig. 21. Block diagram of testbed used in Cybersecure Power Router project	30
Fig. 22. Simplified block diagram of grid-connected power flow capabilities of UCB hardware and Cybersecure Power Router.....	31
Fig. 23. Block diagram of clock generation within Complex Programmable Logic Device of the Cybersecure Power Router	32
Fig. 24. Block diagram of data bus controller within Complex Programmable Logic Device of the Cybersecure Power Router.....	33
Fig. 25 .Block diagram of serial interface within Complex Programmable Logic Device of the Cybersecure Power Router	33
Fig. 26. Firmware code snippet to generate heartbeat from Digital Signal Processors	34
Fig. 27. Heartbeats of Controllers 1 and 2 while running identical firmware	34
Fig. 28. Block diagram of control multiplexing using Digital Signal Processor signals and Hardware Authentication Module within the Complex Programmable Logic Device of the Cybersecure Power Router	35
Fig. 29. Block diagram of hot patching process	36
Fig. 30. Diagram of CSPR components involved in hardware authentication	38
Fig. 31. Excerpt from EEPROM datasheet, with erroneous information noted	39
Fig. 32: Hardware Authentication Module communication beside oscilloscope capture	39
Fig. 33. Block diagram of encrypted serial communication within Complex Programmable Logic Device used in the Cybersecure Power Router.....	40
Fig. 34. Shoot-Through hardware protection.....	42
Fig. 35. LED display diagram for Cybersecure Power Router	43
Fig. 36. Time between execution cycles of controller firmware vs. switching frequency	44
Fig. 37. Detail of Figure 36.....	45
Fig. 38. Inverter output voltage at 114, 118, and 128 kHz switching frequencies.	46

Fig. 39. Inverter output during controller transition	48
Fig. 40: Radar chart of missions for controls.....	51
Fig. 41: CSPR and SGPN migration.....	52
Fig. 42. Digital Signal Processor Unified Controller Board schematic	61
Fig. 43. Power Electronics Evaluation Unified Controller Board schematic	62
Fig. 44. Complex Programmable Logic Device Unified Controller Board schematic.....	63
Fig. 45. BeagleBone Black schematic	64
Fig. 46. Hardware Authentication Module schematic	64
Fig. 47. Analog Splitter schematic.....	65
Fig. 48. Crontab configuration on BeagleBone Black to run CPLD UCB LED control script on startup.....	65
Fig. 49. Content of LED control Python script running on the BeagleBone Black.....	66

List of Tables

TABLE I: Security-by-Design cycles	5
TABLE II: Defense-in-Depth Dependencies	6
TABLE III: Cyber-Physical Attack Matrix	15
TABLE IV: MachXO2 Family Features.....	24
TABLE V: Available MCLK frequencies	24

Chapter 1 - Introduction

The reliability and safety of the electrical grid face challenges. These challenges include aging infrastructure, tight regulatory environments, and the integration of new technologies. Power electronics are some of these new technologies, and provide a wide range of assets and liabilities to the electrical grid, its operation, and its evolution. Reactive power compensation, phase load balancing, battery energy storage systems, solar power, and flexible ac transmission are all potential assets. These devices may pose serious threats to both the electrical grid and interdependent critical infrastructure [1]. As these grid-connected power electronics permeate more of the electrical grid, the need for their security becomes greater [2].

A series of questions can begin this investigation of security for grid-connected power electronics, and establish appropriate security measures [3]. The first question is "what benefit does the device provide?" In the present case, the distributed energy device manages energy at the edge of the electrical grid, and supports grid resilience.

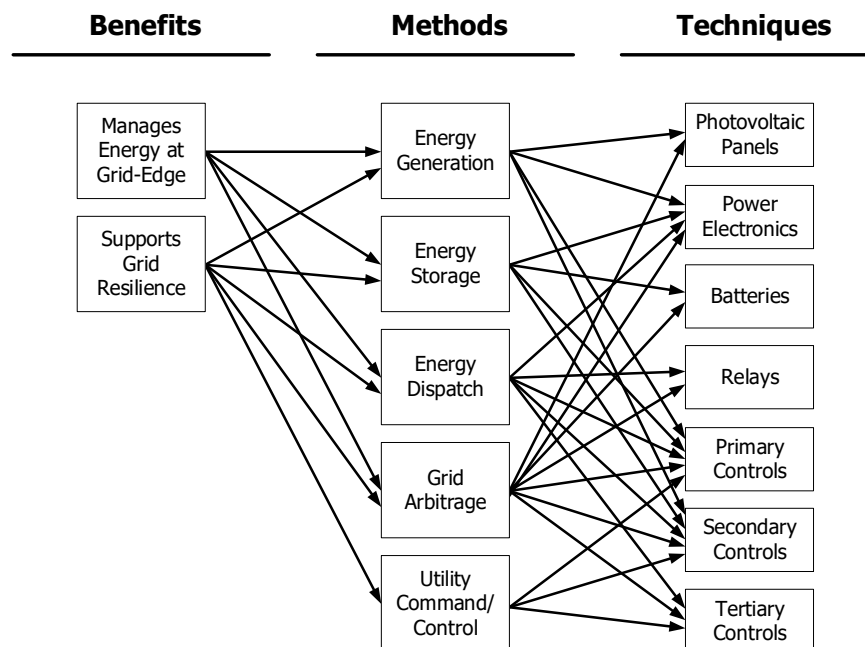


Fig. 1. Design Inventory for Distributed Energy Resource

The next question is "how does the device provide those benefits?" A distinction in method and technique is useful, here. The methods to realize energy management at the grid-edge are energy generation, storage, dispatch, and arbitrage (see "Methods" in Figure 1). These methods identify a particular process or type of system. The techniques to realize energy management at the grid-edge include the use of photovoltaic panels, dc converters, inverters, batteries, and hierarchical controls. These techniques identify the practical elements of a method, that is, of a process or system.

After arriving at a set of techniques, the next question is "what can happen to prevent this technique from working?" A boost converter or more complicated topology can be used to provide maximum power point tracking, current control, and hardware protection for a string of photovoltaic panels. In short, answering the question is difficult [4]. The hardware can fail, the switches can cause excessive electromagnetic interference, and noisy environments can corrupt transmitted data. Poor design can also cause various failures. The list of possible points and modes of failure is extensive for a complex system like a converter. Putting aside the incomplete answer, the next question is "what can be done to protect this technique?" In the case of a dc converter, more robust hardware can be used, filters can reduce electromagnetic interference, and error detection in communication can limit the use of corrupted data. This is not an exhaustive list of points and modes of failures or security measures. To better organize and address these questions, product lifecycle management and design dependency can be used (and will be discussed in greater depth in the next section). Product lifecycle management can be used to consider security of a device from specification all the way to end-of-life, and illustrates the security-by-design process. Design dependency can be used to build layers of security to protect an asset or the system as a whole, and illustrates the defense-in-depth method. A cyber-physical threat matrix is built on the

design dependencies listed and the implementation of security techniques to address the potential threats.

The Cybersecure Power Router (CSPR) uses the security-by-design process and defense-in-depth method to realize a cybersecure distributed energy resource. The CSPR operates as a solar inverter and battery energy storage system. It has hierarchical controls and network communication, allowing a utility operator to control the device, its operation, and its power flow. Finally, it employs a number of security features for a wide range of functionalities. These security features include AES-128 encryption for network communication, hardware-assisted monitoring for improved firmware integrity during runtime, hardware protection during nominal operation, and more.

The Smart Green Power Node (SGPN) is a device developed at the University of Arkansas to manage energy resources at the grid edge, specifically in residential applications. The SGPN predicts and optimizes power flow of a solar inverter and battery energy storage system. Hierarchical controls and network communication are also included within the design of the SGPN. The system optimizes power flow of the energy resources through powerful predictive algorithms and weather data collection. The system is rated for 2 kW operation.

The Unified Controller Board (UCB) project is a set of hardware, firmware, software, and instructional material also developed at the University of Arkansas. The devices within the UCB project include a DSP docking station; a complex programmable logic device (CPLD) PCB; buck and boost converter and inverter PCB; and several expansion boards. The Unified Controller Boards are designed around flexible controls and modular hardware.

A real-world system of sophisticated, grid-tied power electronics is needed to show the practical demands and limits of security. The use of the Smart Green Power Node and the Unified

Controller Board devices as a prototype for the Cybersecure Power Router was used for such a real-world system. The SGPN is a distributed energy resource with many sophisticated assets. The 2 kW power rating of the system provides an appreciable power flow for grid-connected applications. The UCB devices are modular, allowing for rapid configurability of hardware and controls. The UCB software and firmware is extensible, allowing for the integration of security features and changes in control hierarchy. The combination of these two systems provides the necessary power flows, complexity, and direct results necessary for this cyber-security investigation.

The present work shows the security-by-design process and defense-in-depth method for a grid-connected, distributed energy resource prototype. The security features chosen are developed and tested within a grid-connected power electronics context. Security features to protect firmware integrity at runtime are specifically investigated. The ability for the CSPR prototype to quantify firmware integrity degradation and respond to firmware integrity failure is shown. This ability is provided by the CSPR prototype monitoring and maintaining liveness of controllers through control multiplexing. Future research into greater flexibility and resiliency of controls is discussed. Finally, the necessary work to migrate research from the Smart Green Power Node and the Cybersecure Power Router into a pre-production prototype is presented.

Chapter 2 - Technical Background

Security-by-design ensures greater security of a device by considering both the processes behind the development and life of a device, and the device itself. Product lifecycle management [5] serves as the framework for security-by design. This process stands over and above the IEEE standard for system, software, and hardware verification and validation [6].

TABLE I: Security-by-Design cycles

Lifecycle Stage	Security Feature(s)
Hardware	
Specification	IEEE Standards (e.g., 1547)
Simulation	Accurate Modeling, Thermal Co-Simulation
Design	IEEE Standards (e.g., 3001, 3003), Thermal Co-Design
Verification	Electrical Rule Checking, Design Review, Hardware-In-the-Loop
Firmware	
Development	Restricted Access, Version Control, Standard Protocols, Standard Libraries
Distribution	Restricted Access, Message Digest, Server Authentication
Installation	Message Digest, Error Detection and Correction
Run-Time	Side Channel Analysis, Challenge-Response Authentication
Manufacture	
Fabrication	Trusted Supplier, ISO 9001 Certification, Hardware Authentication
Quality Control	Burn-In Testing, Fuzz Testing, Standard Metrics
Design Iteration	Restricted Source Code and Design Files
Operation	
Installation	Certified Installers, Standard Connections, Lockout-Tagout
Use	Key Management, Challenge-Response Authentication, Behavior Analysis
Aging	Hardware Health Diagnostics
Attack	User Authentication, Command Whitelisting, Asset Segmentation
Failure	Fails Safe, Hardware Protection, Resilient Communication
Recovery	Startup Sequence, Sanity Check, Firmware Integrity Check
Maintenance	
Update	Patching, Maintained Uptime
Replacement	Modular Design
Upgrade	Modular Design, Reconfigurable Architecture, Flexible Controls
End of Life	
Removal	Lockout-Tagout
Documentation	Failure Modes, Effects, and Diagnostic Analysis (FMEDA)
Reiteration	Restricted Access to Source Code, Specification and Quality Control
Disposal	Certified eWaste Recycling and Disposal

Any device, a distributed energy resource in this case, has a lifecycle. It is specified, designed, fabricated, tested, installed, operated, uninstalled, and disposed of during that lifecycle. Each step in the lifecycle of a device serves some purpose. For instance, hardware specification creates the exhaustive list of design requirements for a device. The result of the hardware specification step is a list. How could security be applied to this step? The use of standards (in this case, IEEE 1547 for the design of utility electric power systems and distributed energy resources) provides greater assurance that the list created in the hardware specification step is exhaustive. Stated another way, the IEEE standards secures the intended result of hardware specification. The product lifecycle security approach also requires a designer to consider the full lifecycle of a device, not just the useful life. In the context of the electric grid, people install and remove distributed energy resources. By considering the lifecycle of a device, the safe installation and removal of a device is considered and included in the hardware and firmware design stages. For instance, a Lockout/Tagout technique can be designed for a solar inverter to keep both people and hardware safe during installation and removal [7].

Defense-in-depth provides layered security for assets of a device [8]. Returning to the "Design Inventory" from the introductory section, each technique in the design inventory process has dependencies. These dependencies arise from the techniques chosen to realize a device. An inverter depends on various switches, gate drivers, feedback signal chains, capacitors, other hardware components, and firmware to operate. Examples of these dependencies are listed in Table II, along with possible methods of security.

TABLE II: Defense-in-Depth Dependencies

Design Category	Security Feature
Component Health	Hardware Authentication, Hardware Protections, Safety Factor
Feedback Signal Chain	Galvanic Isolation, Buffered I/O
Temperature Control	Thermal Management, Current Limitation
Current Control	Controller Current Limiting, Body Diodes, Fuses

Voltage Control	Controller Voltage Limiting
Firmware Execution	Hardware-Assisted Monitoring, Heartbeat, Hot Swapping
Network Communication	AES-128, MD5, Whitelisting

These dependencies are not spread across the lifecycle of a device, as illustrated in the security-by-design process. Rather, these dependencies are logical constituents of the design of a device, and are typically part of a complex, cyber-physical set of interdependencies [9]. For instance, how might one protect the current flow into the batteries from potentially damaging commands? Secure network communication [10] protects the battery energy storage system from noise and remote adversaries. If the secure network communication is defeated, current controls prevent harmful behavior of the device [11]. If the current controls are defeated, various design features (like galvanic isolation, buffered I/O, fuses, and over-design) allow the device to withstand or limit the harmful behavior [12]. In this case, the energy storage assets of the BESS are protected by layers of security.

Security features have a range of costs and performance gains [13]. Any change to the design of a system, including those to increase security, comes at a cost. This cost may include the price of more sophisticated integrated circuits, hardware to support increased power consumption, time to develop new firmware, or expertise to identify and execute security strategies. Improvements to security may come at a low cost. An existing system may extensively benefit from simple firmware management [14]. Such management, including revision, could greatly increase system security without incurring costs from additional hardware and hardware development. Beyond firmware management, an example of a firmware security feature is a checksum for network communication [15]. The inclusion of this security feature is lightweight: incurring a small increase in firmware size, computational load, and communication overhead. Checksums can prevent electromagnetic noise from corrupting communication, and weakly protect against a malicious actor tampering

with communicated data. A checksum is an instance of a common security feature implemented in firmware, but is far from the full benefits of improved firmware management and security.

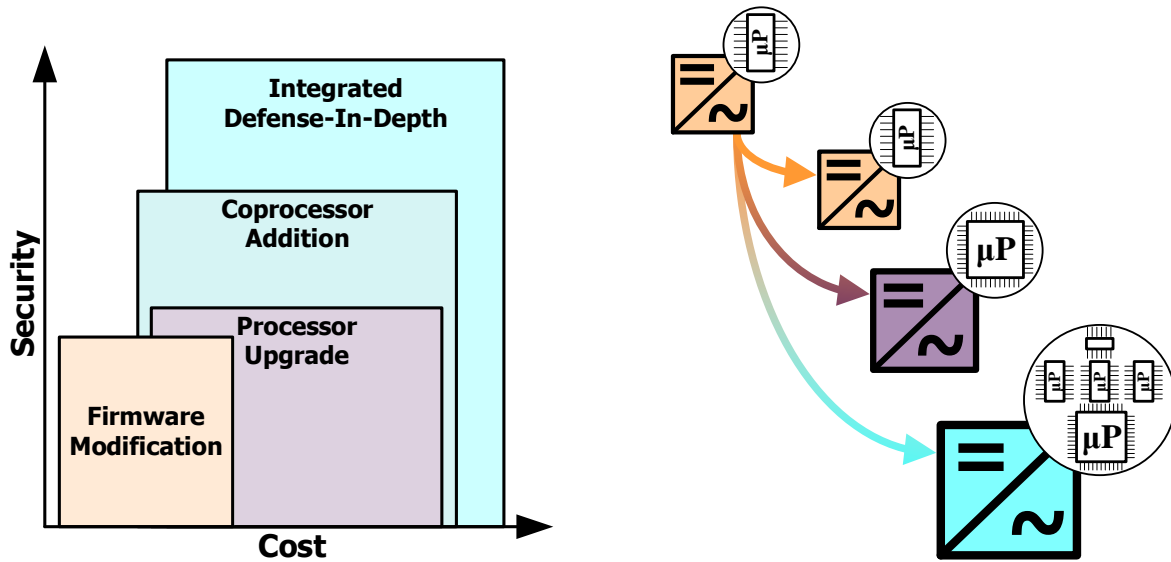


Fig. 2: Cost/Performance regions (left) for various security solutions (right)

Not all development leads to increased security. Consider the use of encrypted communication for a controller using a digital signal processor. The controller is responsible for the power flow through a solar inverter. Processing resources are required to encrypt and decrypt communication with the controller. The increased demands on the DSP during communication may cause overrun conditions [16] and degrade power processing. The system has more confidential communication, but at the cost of lower integrity of power processing. The general effect may be a less secure system, despite the addition of a security feature and the costs of development.

The cost/performance analysis for many security features used in the Cybersecure Power Router are readily available. The Advanced Encryption Standard 128-bit key (AES-128) is well defined [17],[18] and researched in various applications [19], [20], [21]. AES-128 is used in communication between hardware modules, and between outside controllers and the CSPR. The MD5 message-digest algorithm is also well researched [22], [23], [24]. The MD5 algorithm is used

to protect the integrity of firmware as it passes from an outside controller, to the Cybersecure Power Router, and into any on-board controller. Hardware protections against overcurrent [25] and shoot-through conditions [26] are also well understood. Overcurrent and shoot-through conditions are used in both controller firmware and within the hardware logic of the Digital Signal Processors. These techniques are robust and well researched. Security techniques are still needed for other essential functions of any grid-connected power electronics.

Firmware security at runtime for power electronics is less researched and widely implemented [27]. Grid-connected power electronics typically use microprocessors, microcontrollers, or processors to run firmware. Microprocessors and microcontrollers often use watchdog timers to reset the device in the case of firmware execution faults. If the controls of the power electronics run on the reset device, the power electronics will stop. The use of a watchdog timer may, therefore, be inappropriate for grid-connected power electronics, where downtime is to be minimized or eliminated [28]. Processors can provide more sophisticated techniques than microprocessors and microcontrollers to prevent, detect, identify, and recover from firmware execution faults, namely through using an embedded operating system [29], [30]. The choice of processors in grid-connected power electronics may also be inappropriate, given their sophistication and cost. The purpose of the Cybersecure Power Router is to develop and show security design techniques, including the security of grid-connected power electronics at runtime. Is there an option between a simple watchdog timer and a sophisticated embedded operating system? If so, how can the cost and performance of that runtime security be evaluated?

The two considered threats to CSPR firmware integrity during runtime are task overrun and firmware patching. Other threats are relevant [31], but fall outside the present scope of grid-connected power electronics. A task overrun condition occurs if a controller is not able to finish

the various tasks before another set of tasks are started [32]. The result is a degradation in the power flow of the power electronics, as shown later. Unlike task overrun, firmware patching is more likely to halt power flow altogether. The patching process requires the rebooting of the DSP running a controller, halting the controller during the process. The power flow through the electronics is, therefore, also halted as the DSP reboots.

Task overrun can be described as a controller's loss of liveness. Formally, liveness can be expressed as

$$\forall \alpha: \alpha \in S^*: (\exists \beta: \beta \in S^\omega: \alpha\beta \models P), \quad (1)$$

where α and β are a sequence of states, S^* is a set of finite sequence states, S^ω is a set of infinite sequence states, and P is a property (executability, in this case) [33]. Formally, this definition reads as there exists a state within an infinite set of states that satisfies a given property (executable), such that it does so given any arbitrarily sized sequence of compossible states. Or, simply, liveness means a task will be executed, even if there are many more tasks for a controller to complete. The tasks to be executed are part of interrupt service routines on the DSP, and in the present work are initiated every switching period. Each interrupt request (IRQ) initiating an interrupt service routine (ISR) is assigned a priority. The DSP resources handling an ISR are a critical section to other ISRs, especially those at the same priority. A critical section is a set of resources accessed or used by multiple processes [34]. For the present architecture, an IRQ can cause the interruption of an ISR in progress. This is also true if the ISR in progress and raised IRQ have the same priority. Here is an example from the Cybersecure Power Router. Assume the switching frequency is set at 30 kHz. Every 33.33 μ s, a number of interrupt requests will be raised within the DSP. These IRQs signal the DSP to read various voltages and currents, perform mathematical operations, look up values, and set the pulse width modulation of several switches. Let's assume the DSP requires 80 μ s to

handle all of these tasks. While processing the last round of ISRs, new ISRs are created. The previous ISRs are interrupted and started again by the new IRQs. The tasks never complete, given their interruption and restart during processing. Theoretically, the result is the controller losing liveness.

A simple consideration of timing can maintain liveness. The controller has a maximum duration of time to complete its tasks:

$$T_{d,max} = \frac{1}{f_s}, \quad (2)$$

where f_s is the switching frequency. In the example above, the 30 kHz switching frequency provides a maximum duration of 33.33 μ s for the controller to complete its tasks. This is the theoretical maximum amount of time the controller can take to process the tasks of one switching period before being interrupted by the next switching period. The time the controller requires to complete the switching period tasks is not dependent on the switching frequency, however. The time required depends more on the firmware, speed of the DSP, and competing interrupts (such as those from communication). This required time can be measured. This duration of time can vary, even if the firmware and processor remain the same. A mean time for the completion of tasks can be empirically found, and used to quantify available processing resources. The expected available resources can be articulated as

$$\text{Available Resources (\%)} = \left(1 - (T_{d,mean} * f_s)\right) * 100\%, \quad (3)$$

where $T_{d,mean}$ is the mean of the duration of time the controller requires to complete switching period tasks, and f_s is the switching frequency. As this percentage approaches 0%, tasks are more likely to be interrupted, and liveness of the controller is more likely to be compromised.

The Cybersecure Power Router uses a signal sensitive to controller liveness and hardware-assisted monitoring to protect controller liveness. The next section details the design of the

Cybersecure Power Router, especially the security features protecting controller liveness. Later sections provide the results and examination of the operation and performance of those security designs.

Chapter 3 - Cybersecure Power Router

3.1 System Description

The Cybersecure Power Router is a set of switch-mode power electronics, controllers, and processors. The majority of controller devices used are from the Unified Controller Board project, developed by Chris Farnell at the University of Arkansas. These devices include the Power Electronics Evaluation Unified Controller Board (PE Eval UCB), Complex Programmable Logic Device Unified Controller Board (CPLD UCB), and the Digital Signal Processor Unified Controller Board (DSP UCB).

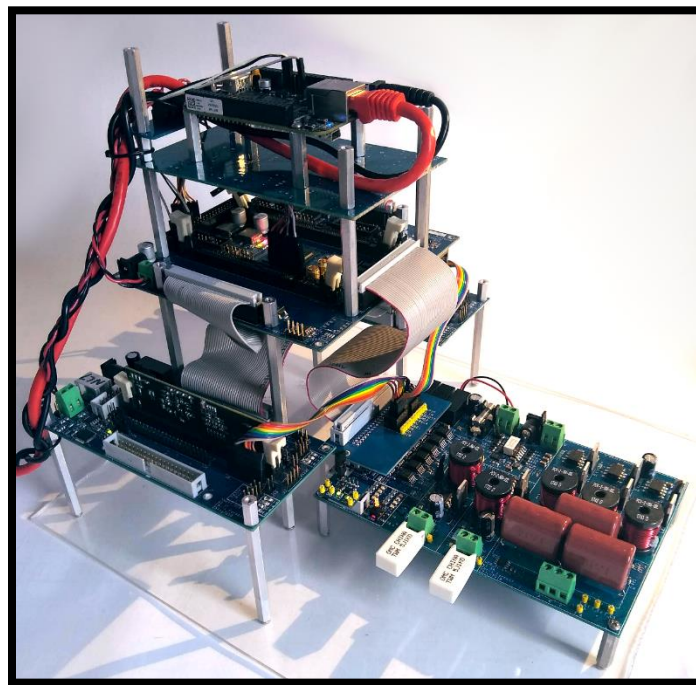


Fig. 3. Cybersecure Power Router prototype

The Digital Signal Processor Unified Controller Board uses many design features of the Texas Instruments' TMDSDOCK28335 Digital Signal Processor docking station. The power electronics of the PE Eval UCB are a buck converter, boost converter, and an inverter/rectifier. A testbed provides dc and ac power flow to provide safe and reliable conditions for testing. Two Digital

Signal Processors are used in the prototype. Each DSP is capable of controlling the power electronics. A Complex Programmable Logic Device UCB provides many security features, and multiplexes the control signals of the DSPs. The Hardware Authentication Module is used to authenticate the power electronics and enable power flow. The Signal Splitter routes analog signals to both DSPs from the power electronics. Finally, a BeagleBone Black provides high-level control and Ethernet communication to the CPLD UCB.

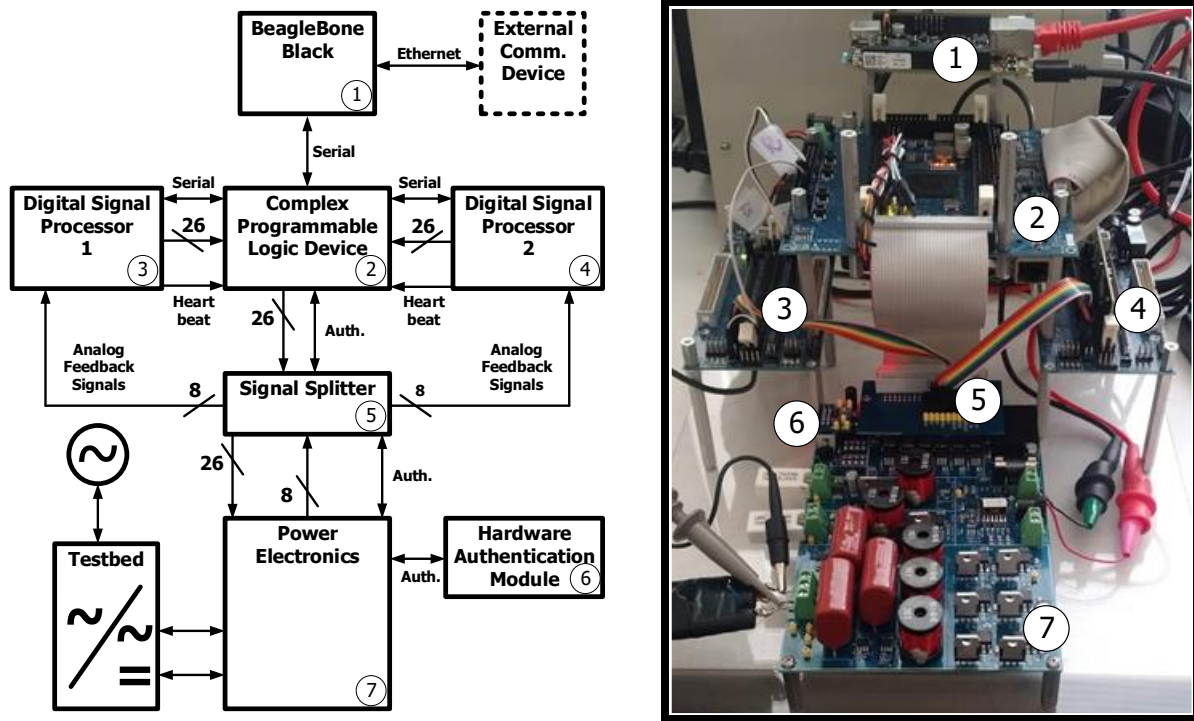


Fig. 4. Block diagram (left) and figure (right) of Cybersecure Power Router prototype

The CSPR prototype can be reconfigured. The modular design allows flexibility in both hardware and control. A more simplified configuration of the prototype could use one or two DSPs on the CPLD UCB, as shown below.

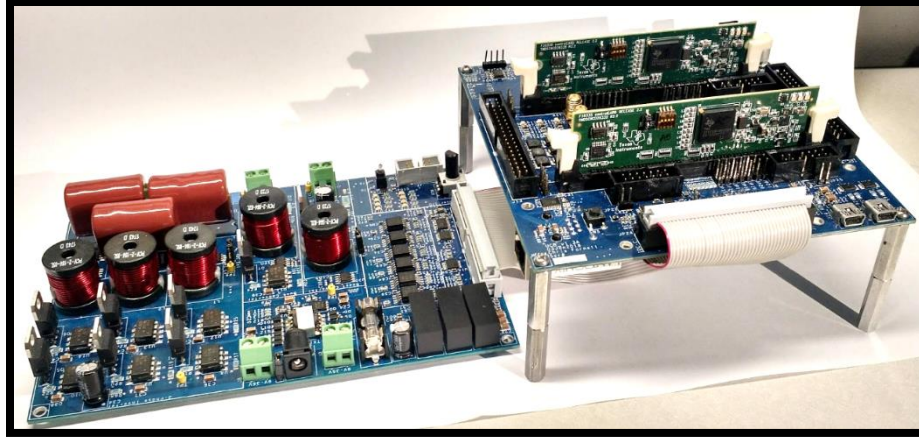


Fig. 5. Minimal configuration of Cybersecure Power Router prototype

The following table lists the security threats and mitigations chosen for the Cybersecure Power Router.

TABLE III: Cyber-Physical Attack Matrix

Asset	Threat	Mitigation
Communication		
Confidentiality	System Surveillance [35]	AES-128 Encryption [36]
Integrity	Corrupted Firmware [37]	Error Detection
Availability	Unauthorized User Access [38]	HW-Asst. Monitor, Key Mgmt.
Firmware		
Distribution	Tampered Firmware [39]	Encryption, Error-Detection
Installation	Reduced Integrity	MD5 Hash Check
Loading	System Downtime [40]	Control Multiplexing
Runtime	Operation Outside Parameters [41]	Heartbeat, HW-Asst. Monitor
Hardware		
Authenticity	Counterfeit Hardware [42]	Hardware Authentication
Power Processing		
Quality	Harmonic Distortion [43]	Robust Hardware/Controller Design
Availability	System Downtime [44]	Control Multiplexing
Response	Non-Recoverable State [45]	Robust Controller Design

Chris Farnell's contributions to the Cybersecure Power Router project

The asset inventory, threats, and mitigations are not exhaustive. The above table shows the current work done and where the security is implemented.

3.2 Power Electronics

The power electronics of the Power Electronics Evaluation Unified Controller Board consist of switch-mode power supplies, signal chains for controls and feedback, isolated power supplies, voltage and current sensors, filters, ports, ancillary circuits, and human-machine interfaces. Control signals routed from the Complex Programmable Logic Device enter the Power Electronics Evaluation Unified Controller Board through a 40 pin insulation-displacement connector. The control signals then pass through 120 Ω resistors to trigger HCPL-3120-300E optocouplers. The optocouplers then drive the STGP15H60DF insulated-gate bipolar transistors that act as switches. An isolated, flyback regulator using the LT3748EMS integrated circuit energizes the optocouplers to drive switching. The switch-mode power supplies of the board include an asynchronous buck converter, an asynchronous boost converter, and a three-phase inverter/rectifier. The buck converter provides current sensing before and after the 560 μ H, 100 μ F LC filter; and output voltage sensing. The buck converter is rated for an input voltage of 9 to 50 Vdc, and an output voltage of 0 to 50 Vdc. The voltage ratings can be increased if higher voltage rated capacitors are used on the input and output of the buck converter.

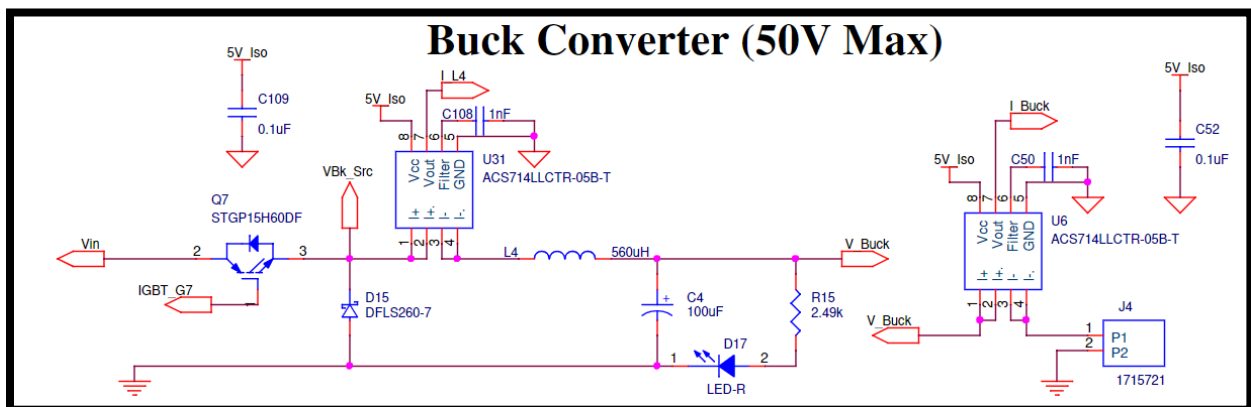


Fig. 6. Asynchronous buck converter schematic of Power Electronics Evaluation Board of the UCB project

The asynchronous boost converter provides current sensing at the input and output; and output voltage sensing. The boost converter is rated for an input voltage of 9 to 50 Vdc, and an output voltage of 9 to 50 Vdc. The voltage ratings can be increased if higher voltage rated capacitors are used on the input and output of the boost converter.

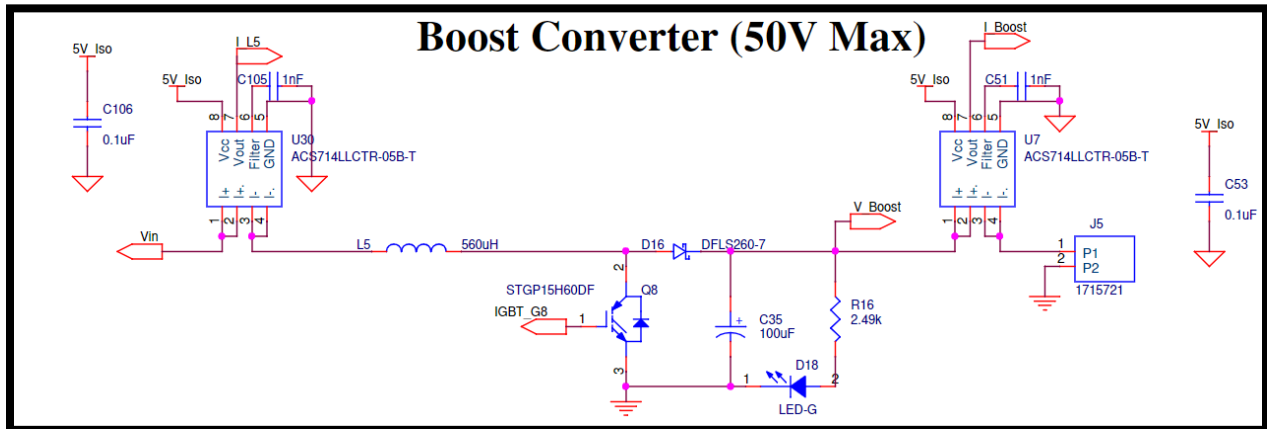


Fig. 7. Asynchronous boost converter schematic of Power Electronics Evaluation Board of the UCB project

The three-phase inverter/rectifier can operate bi-directionally. When acting as an inverter, it can output a 1 Hz to +1000 Hz sinusoid from 0 to 50 Vac. The inverter is rated for 0 to 50 Vdc input. Current sensing is available on all three phases after the inductive filtering, and after capacitive filtering on phase A. Voltage sensing is available on the output of all three phases.

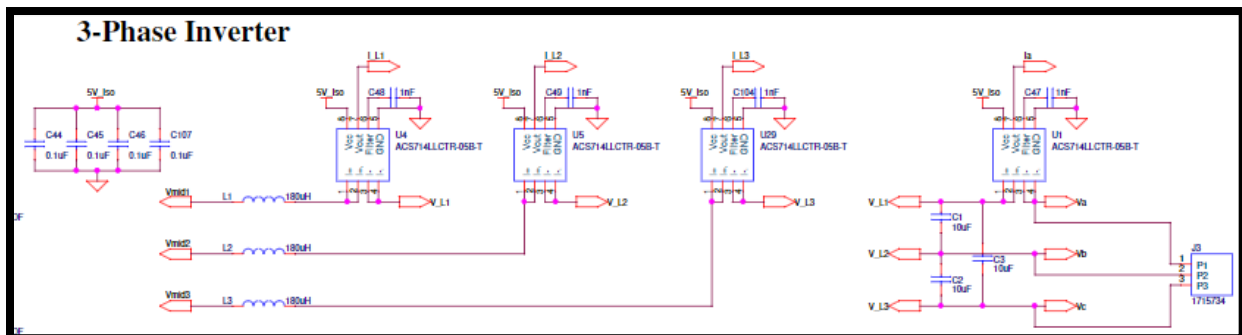


Fig. 8. 3-Phase inverter filter and current sensing schematic of Power Electronics Evaluation Board of the UCB project

Three switching legs using STGP15H60DF insulated-gate bipolar transistors are the switching stage for the inverter/rectifier. As stated earlier, isolated power is provided to the gate drivers. The

low switches share the same isolated power rail. Jumper 1 (JP1) can be used to connect or disconnect the inverter/rectifier with the dc input used on the rest of the Power Electronics Evaluation board. Current and voltage sensing is provided between the dc bus and the capacitive filtering of the inverter/rectifier.

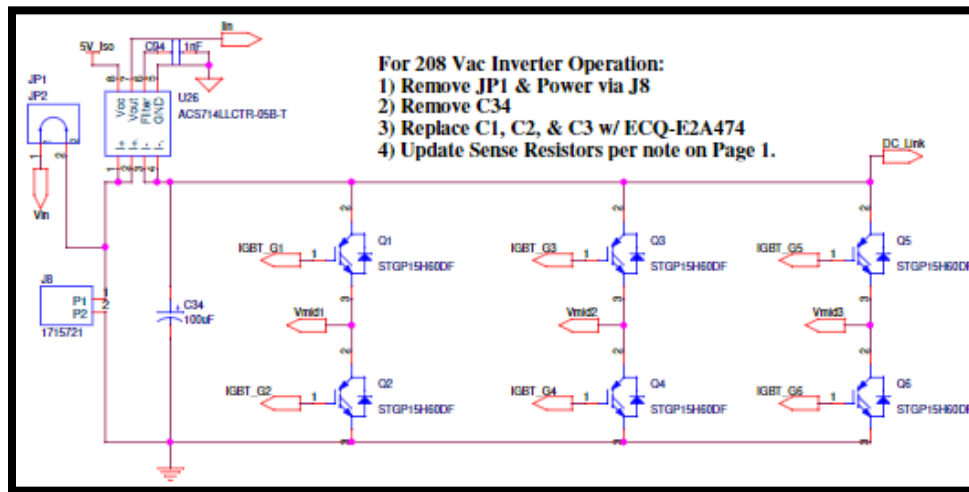


Fig. 9. 3-Phase inverter/rectifier switching stage schematic of Power Electronics Evaluation Board of UCB project

The PCB layout of the Power Electronics Evaluation board is shown below. Considerations were made for mixed analog/digital signals within the board. The isolated dc/dc power supplies and ACPL-C87B-000E optical isolation amplifiers provide isolation between analog signals in the mixed environment and the feedback signals supplied to the DSPs and CPLD.

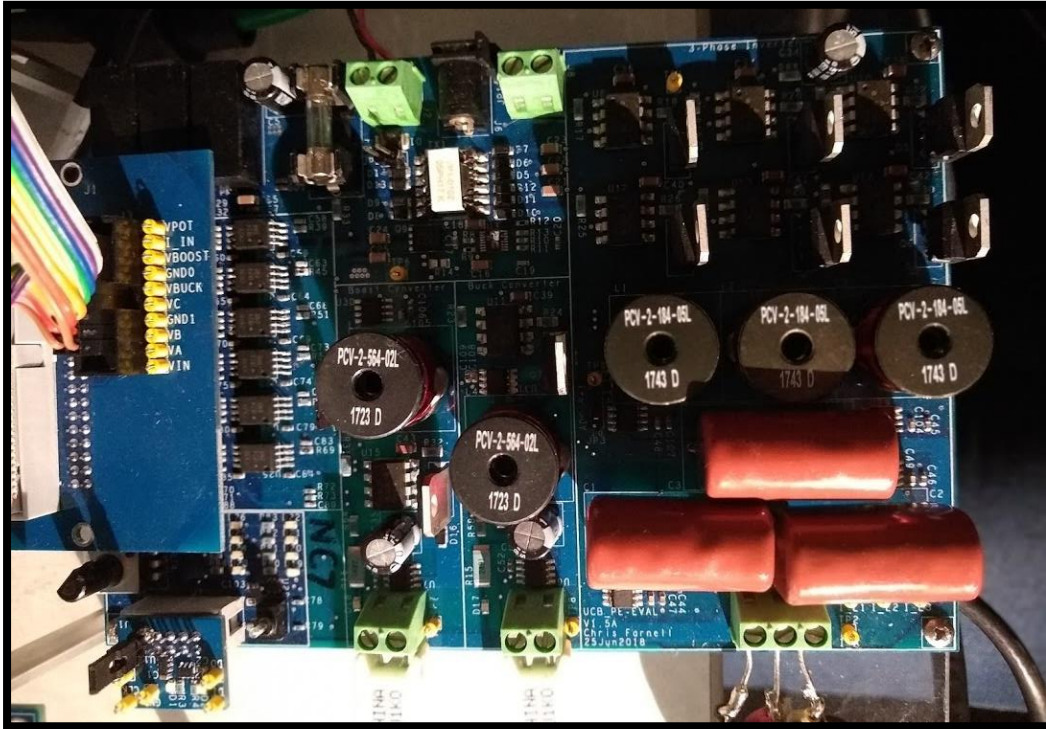


Fig. 10. Power Electronics Evaluation Unified Controller Board, with Signal Splitter and Hardware Authentication Module, used in Cybersecure Power Router prototype

The total system provides buffered, isolated control and feedback signal chains; and common types of switch-mode power conversion capable of various power flows, with switching frequencies exceeding 100 kHz.

3.3 Digital Signal Processor Board

Two Digital Signal Processor Unified Controller Boards are used in the Cybersecure Power Router prototype. The modular designs of the Unified Controller Board allow two DSPs to slot into the Complex Programmable Logic Device Unified Controller Board. However, having two, standalone DSP boards provide advantages to the testing of the prototype. One advantage is greater accessibility of tools and probes to the I/O of the boards. Another advantage is more direct access to the serial communication ports of the DSPs. A third advantage to using the standalone DSP

UCBs is the ability to apply power independent of the CPLD UCB. This allows DSPs to have controlled power disruptions while still allowing the rest of the CSPR prototype to continue operation. Serial communication with the DSPs in the DSP UCB is provided through a Future Technology Devices International (FTDI) Universal Serial Bus (USB) to Universal Asynchronous Receiver/Transmitter (UART) bridge. This USB to UART bridge is the FT2232D FTDI chip. The receive and transmit lines between the FTDI chip and the DSP are also accessible from the General Purpose Input/Output (GPIO)-28 and GPIO-29 pins. This configuration allows USB connectivity with a computer and with a UART device. For the Cybersecure Power Router prototype, the USB connectivity is used to control the DSP from Code Composer Studio and a LabVIEW script; and the UART connectivity is used to control the DSP from the CPLD UCB.

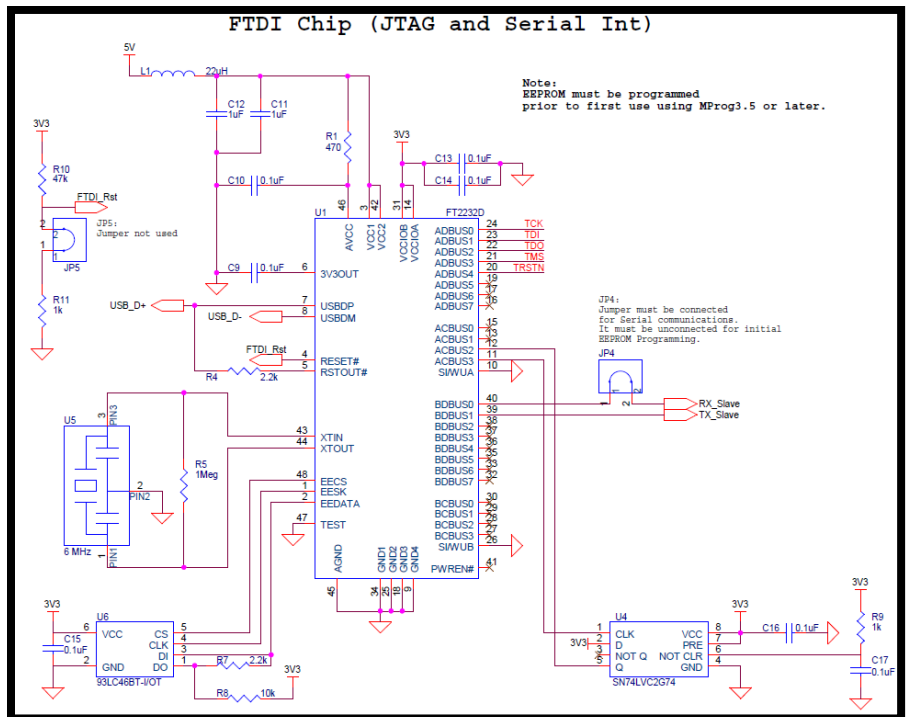


Fig. 11. USB to UART schematic of Digital Signal Processor board of the UCB project

The analog and digital I/O of the DSP within the DSP UCB is readily available to various tools and probes. A connection not pictured is the reset manually added to the TPS3828 \overline{RESET} pin on

U6 of the F28335 DSP controller card. This reset connection cycles power to the DSP during firmware patching.

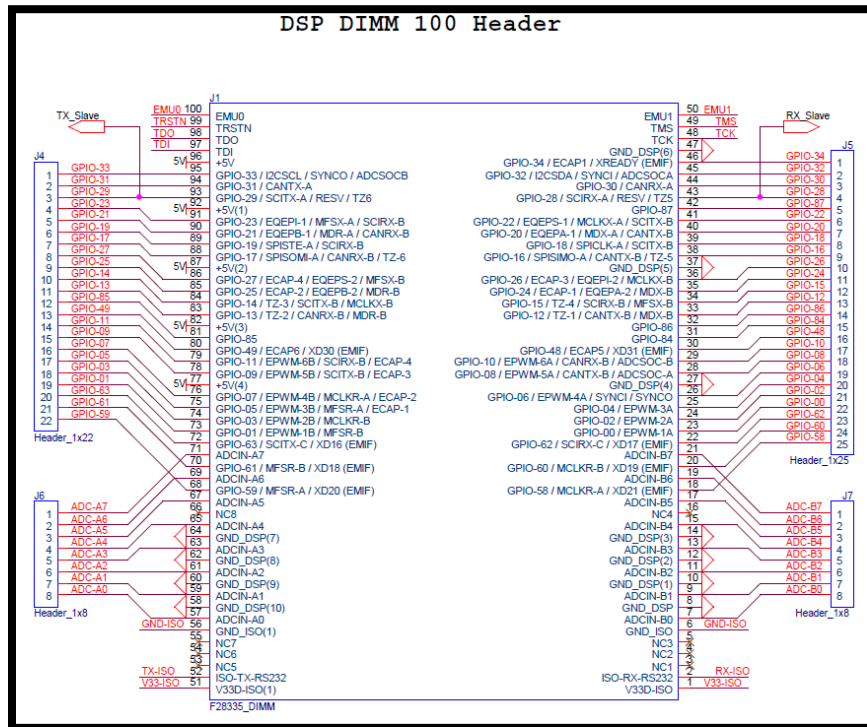


Fig. 12. DIMM pinout of Digital Signal Processor board of the UCB project

Two, 40-pin insulation-displacement connectors (IDCs) are used to bus analog and digital signals between the other Unified Controller Boards. The 28 digital signals use GPIO-00 through GPIO-27. The eight analog feedback signals use ADC-A0 through ADC-A7 for IDC A, and ADC-B0 through ADC-B7 for IDC B. Two 5 Vdc and Ground signals are provided within the 40-pin IDC.

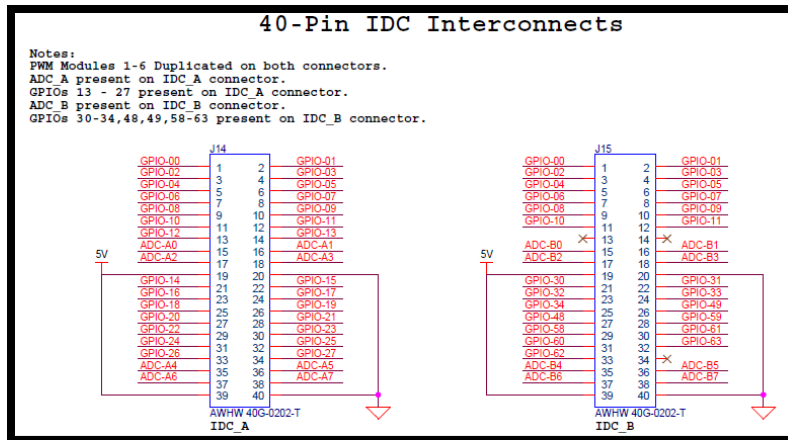


Fig. 13. Majority of analog and digital I/O used by Digital Signal Processor board of UCB project

The Digital Signal Processor Unified Controller Board differs from the Texas Instruments C2000 DAF docking station source design. Mechanical holes are included to allow the use of standoffs. An isolated power supply provides steady power and noise rejection. The isolated power also overcomes ground loops that may be very problematic in noisy environments with long communication cables (such as those used in this prototype).

3.4 Complex Programmable Logic Device Board

The Complex Programmable Logic Device Unified Controller Board is the core of the Cybersecure Power Router. It routes digital control and communication signals between other Unified Controller Boards. It also instantiates the VHDL modules that provide functionality essential to the CSPR project. Four IDCs are provided to allow up to four converters or controllers to interface with each other. Presently, the IDCs allow converters or controllers to interface with one another and the CPLD. Each IDC provides 28 digital, GPIO channels; eight analog channels (typically as part of a feedback signal chain); two 5 Vdc output pins; and two Ground pins. High

frequency filtering is provided between the 5 Vdc power of the CPLD UCB and the 5 Vdc outputs of the IDCs.

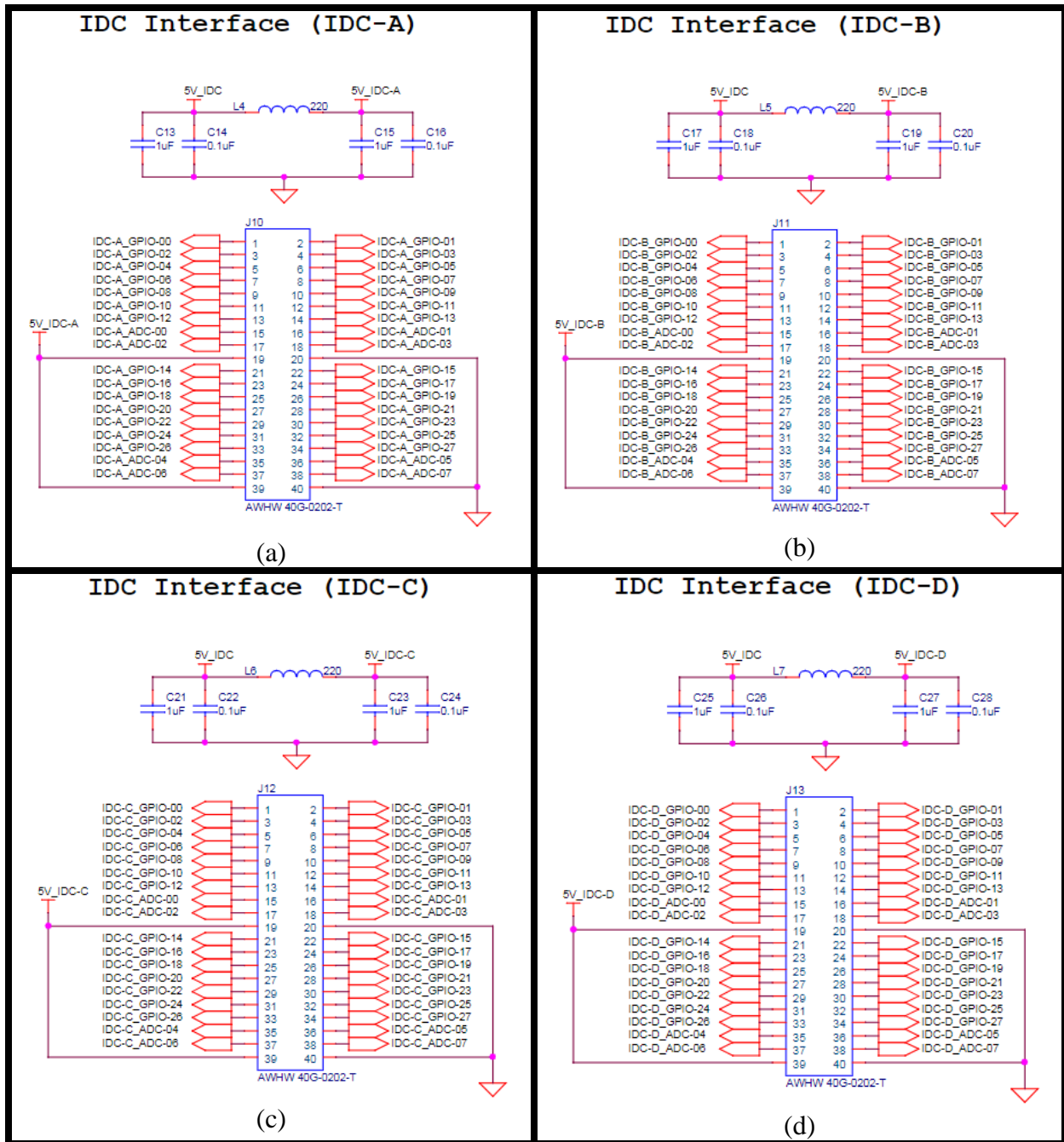


Fig. 14. IDCs in Complex Programmable Logic Device Unified Controller Board

The Complex Programmable Logic Device is Lattice's LCMXO2-7000HC. The IC is packaged as a TQFP with 144 pins. The XO2-7000 has the greatest resources of the MachXO2 family [46].

TABLE IV: MachXO2 Family Features

		XO2-256	XO2-640	XO2-640U ¹	XO2-1200	XO2-1200U ¹	XO2-2000	XO2-2000U ¹	XO2-4000	XO2-7000
LUTs		256	640	640	1280	1280	2112	2112	4320	6864
Distributed RAM (kbits)		2	5	5	10	10	16	16	34	54
EBR SRAM (kbits)		0	18	64	64	74	74	92	92	240
Number of EBR SRAM Blocks (9 kbits/block)		0	2	7	7	8	8	10	10	26
UFM (kbits)		0	24	64	64	80	80	96	96	256
Device Options:	HC ²	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	HE ³						Yes	Yes	Yes	Yes
	ZE ⁴	Yes	Yes		Yes		Yes		Yes	Yes
Number of PLLs		0	0	1	1	1	1	2	2	2
Hardened Functions:	I2C	2	2	2	2	2	2	2	2	2
	SPI	1	1	1	1	1	1	1	1	1
	Timer/Counter	1	1	1	1	1	1	1	1	1

The CPLDs of the MachXO2 family also provide a number of internal clock frequencies for use, in addition to the phase lock loops provided.

TABLE V: Available MCLK frequencies

MCLK (MHz, Nominal)	MCLK (MHz, Nominal)	MCLK (MHz, Nominal)
2.08 (default)	9.17	33.25
2.46	10.23	38
3.17	13.3	44.33
4.29	14.78	53.2
5.54	20.46	66.5
7	26.6	88.67
8.31	29.56	133

The extensive I/O of the MachXO2-7000HC is predominantly utilized by the GPIO of IDC A, B, C, and D; connections with the slots designed for DSP cards; serial peripheral interface with an analog to digital converter; Flash programmer/JTAG interface; button inputs; and an LED display. IDC A GPIO, dual in-line package (DIP) switches, push buttons, and four wire communication with the Lattice Flash Programmer connect to Bank 0 of the MachXO2-7000HC. Dual In-Line Memory Module (DIMM) slots connect DSPs cards to the MachXO2-7000HC. The connections between the DIMM-B slot and the MachXO2-7000HC are through Bank 1. The connections between the DIMM-C slot and the MachXO2-7000HC are through Bank 2. IDC C also connects with the MachXO2-7000HC through Bank 2. LED1 through LED8 interface with the MachXO2-

7000HC through Bank 1. The XPort connections interface with MachXO2-7000HC with the Lantronix Ethernet adapter. This interface is not currently developed. Serial communication interface (SCI) receive and transmit port are connected to Bank 3. The two, serial peripheral interface (SPI) analog to digital converters (ADCs) are also connected to Bank 3. The serial communication with the two FTDI chips (one for the DSPs, the other for the CPLD) interface at Bank 3. Bank 4 of the MachXO2-7000HC provides an external clock and reset signal. Finally, Bank 5 provides connections to the IDC D port.

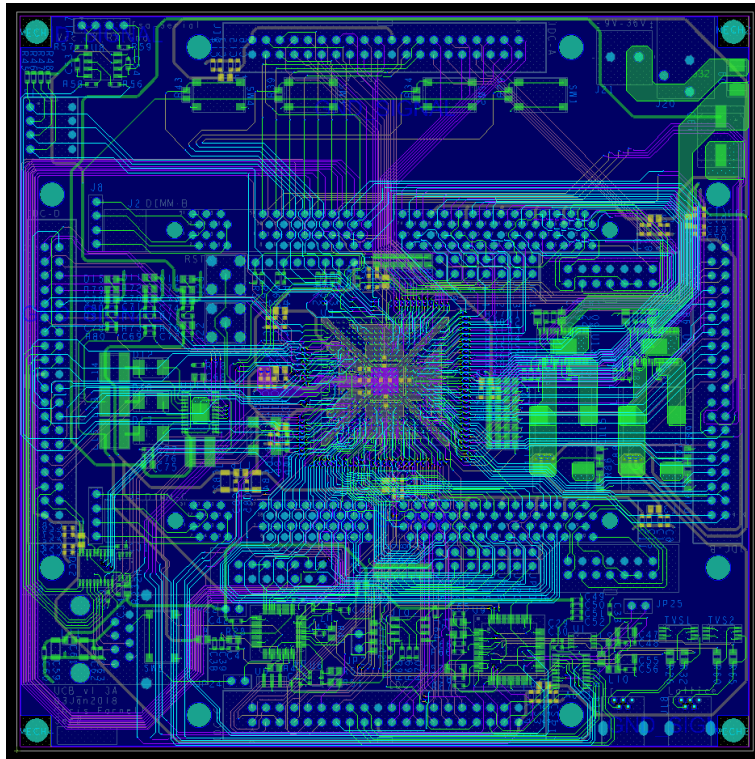


Fig. 15. PCB layout of Complex Programmable Logic Device board of UCB project

The components for the Complex Programmable Logic Device Unified Controller Board were reflowed using the Sikama 5/C reflow furnace in the Assembly Laboratory in the High Density Electronics Center at the University of Arkansas. A tin, silver, and copper alloy (SAC305) solder paste was applied using a stencil and squeegee. No solder paste was applied for the MachXO2-7000HC. A thin layer of no clean flux was applied to the pads of the TQFP footprint on the

Complex Programmable Logic Device Unified Controller Board PCB. After the application of solder paste, flux, and the population of surface mount components, the boards were flowed in the Sikama 5/C reflow furnace.

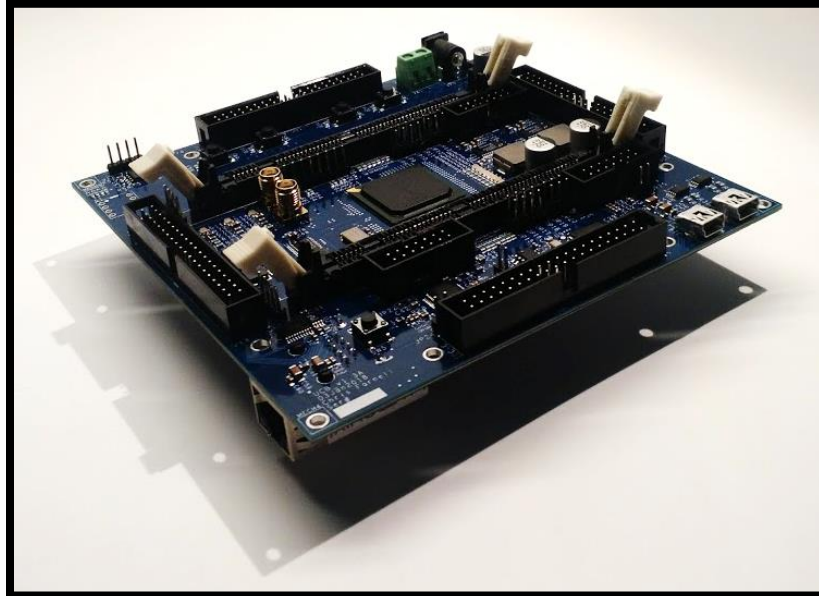


Fig. 16. Fabricated Complex Programmable Logic Device board of UCB project used in Cybersecure Power Router prototype

After inspection and debugging, the through-hole components were populated. Another round of visual, manual, and electrical inspection tested for shorts, unconnected legs, high integrity solder joints, and correct part orientation. Finally, a debug utility was flashed onto the CPLD and tested the I/O.

3.5 Signal Splitter

Two DSPs are used in the Cybersecure Power Router prototype. Two DSP Unified Controller Boards were used to interface the DSPs to the rest of the prototype. To route the analog feedback signals, eight Y connections needed to be created from the output of the Power Electronics Evaluation Unified Controller Board to the ADC input of the two DSP UCBs. A simple printed

circuit board was designed and fabricated to provide this split analog feedback signal. No buffering is provided on the Signal Splitter on account of existing buffered output from the PE Eval UCB.

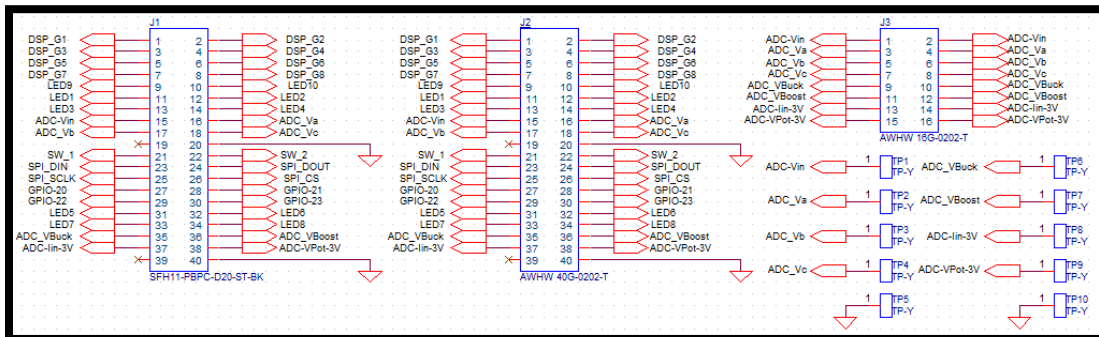


Fig. 17. Complete schematic of Signal Splitter

Test points are provided for each analog signal. Two ground test points are provided to reduce the size of the ground loop of a probe. Female/Female jumper wires connect the analog output of the Signal Splitter PCB to the ADC inputs of the two DSP UCBs.

3.6 Hardware Authentication Module

An embedded, 1 kilobyte password is used to authenticate the power electronics of the Cybersecure Power Router. This password is continually checked by the hardware-assisted monitor instantiated in the CPLD UCB. The password is stored in both the Microchip 93LC46BT-I/OT EEPROM and within the memory of the hardware-assisted monitor.

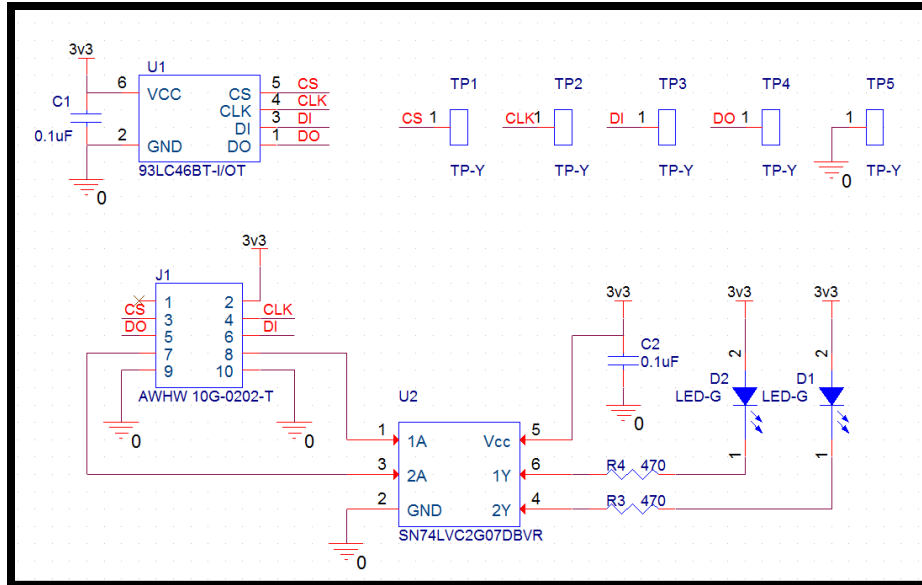


Fig. 18. Complete schematic of Hardware Authentication Module

The Hardware Authentication Module and the CPLD UCB communicate with a four-wire protocol, through the IDC D port. A Texas Instruments SN74LVC2G0 is included to drive two indicator LEDs. The pins for power, ground, the four-wire protocol, and the two indicator LEDs are routed through a 10-pin header.

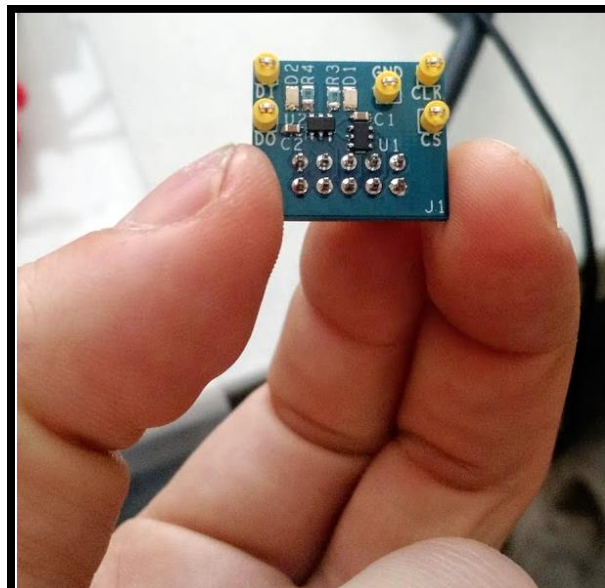


Fig. 19. Hardware Authentication Module used in Cybersecure Power Router prototype

This assembly plugs into the J7 header on the Power Electronics Evaluation Unified Controller Board. Test points are included for the four-wire protocol and a ground to minimize the ground loop during oscilloscope data collection.

3.7 BeagleBone Black

Texas Instruments' BeagleBone Black provides an embedded Linux environment for sophisticated, high-level operations.



Fig. 20. BeagleBone Black

The BeagleBone Black uses a 1 GHz ARM Cortex-A8 processor, and runs Debian Linux. The CPLD UCB and the BeagleBone Black communicate serially. The embedded operating system provides a platform for Python scripts and powerful utilities for automation, analysis, and debugging.

3.8 Test Bed

A testbed provides controlled ac and dc power flow to and from the Cybersecure Power Router.

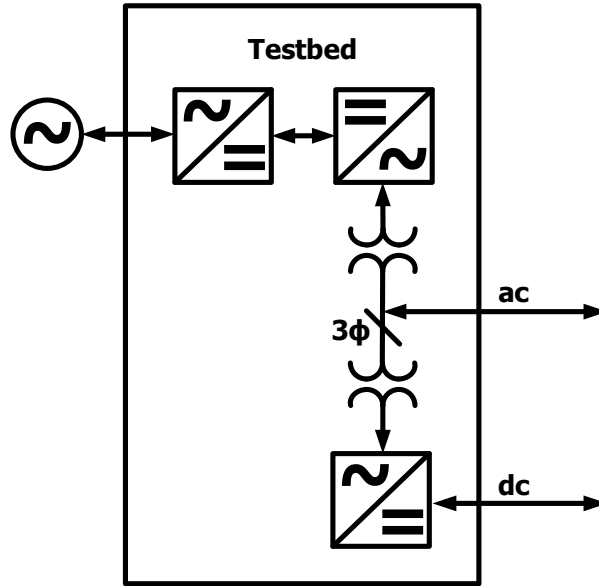


Fig. 21. Block diagram of testbed used in Cybersecure Power Router project

The testbed functions like a microgrid, allowing power assets to be added in a variety of ways while maintaining controlled power flow and a utility frequency (independent of the utility frequency of the electrical grid). The ac power is three phase, and galvanically isolated through low frequency transformers.

3.9 Power Flow

The Cybersecure Power Router shows a security-by-design process and defense-in-depth methods for a Distributed Energy Resource (DER). The security-by-design process outlines both the assets and their dependencies to be secured. To create this inventory of assets and dependencies, a specific device or system must be chosen. Presently, a modular, grid-tied

inverter/rectifier distributed energy resource is chosen. The power assets of this DER include on-site energy generation, energy storage, and bi-directional power flow with the grid.

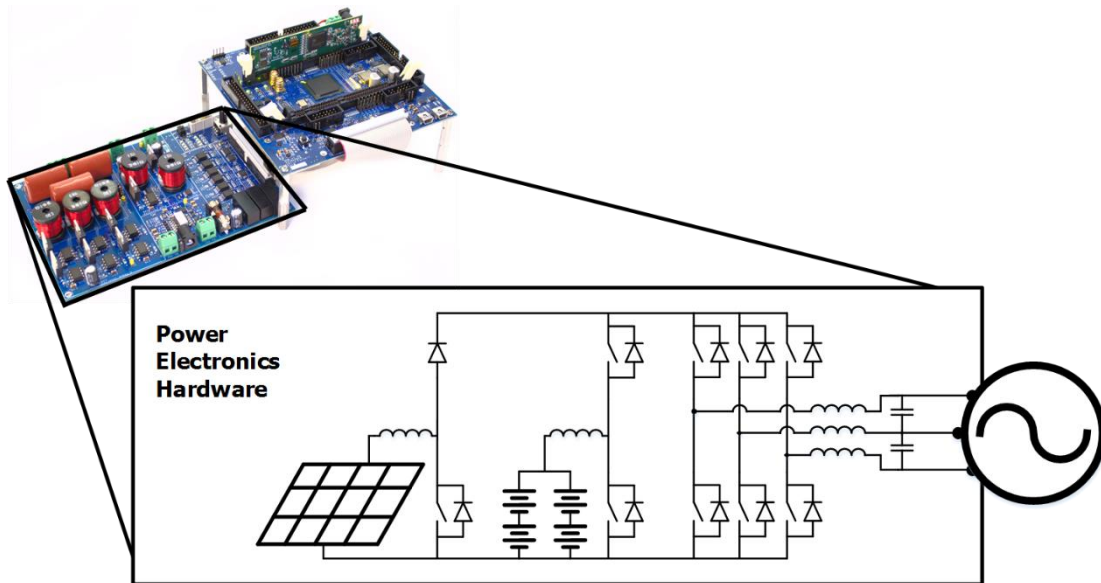


Fig. 22. Simplified block diagram of grid-connected power flow capabilities of UCB hardware and Cybersecure Power Router

Other resources can be integrated within the system. The diagram shows a three-phase inverter/rectifier working in tandem with a photovoltaic panel and battery energy storage system. The PE Eval UCB includes an asynchronous buck and boost converter that may be used to interface dc energy resources, in addition to the three-phase inverter/rectifier provided.

3.10 Data Flow

A hardware-assisted monitor and other utilities are instantiated within the complex programmable logic device on the CPLD Unified Controller Board. The monitor and utilities are developed in the Lattice Diamond integrated development environment. The monitor and utilities are developed using the Very high speed integrated circuit Hardware Description Language (VHDL). The complete source code is included in Appendix B. Not included in the appendix are

the Intellectual Property (IP) cores used within Lattice Diamond, such as the phase-locked loop (PLL) or the digital memory. An oscillator is instantiated within the CPLD to provide an internal clock at 53.2 MHz. This clock is supplied to an internal phase-locked loop. A 53.2 ($1 \cdot f_{clk}$), 24.93 ($1/2 \cdot f_{clk}$), and 1.5 ($1/32 \cdot f_{clk}$) MHz clock signal is derived from the original 53.2 MHz clock signal from the oscillator. The 53.2 MHz clock is used for the hardware-assisted monitor and other high speed applications. The 24.93 MHz clock is used for serial communication and the data bus. The 1.5 MHz clock is used for the four-wire communication with the Hardware Authentication Module.

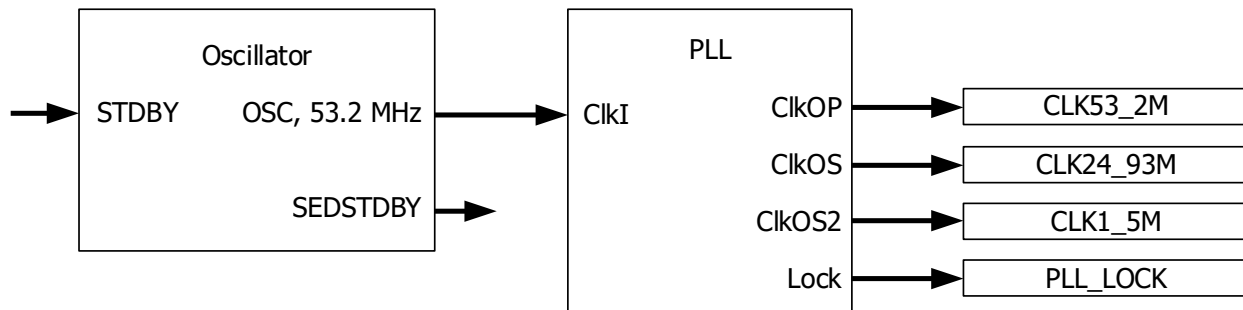


Fig. 23. Block diagram of clock generation within Complex Programmable Logic Device of the Cybersecure Power Router

A data bus is instantiated to allow data flow between various modules. The bus uses 16-bit addresses and 16-bit data. Prioritized access to the bus is given to modules. Currently, ten modules can be prioritized according to access privileges. In addition to controlling the data bus, the Bus Master contains Randomly Accessible Memory (RAM). This memory is used as registers for various controls and functions. A separate memory allocation is used for the booting partition of the digital signal processors.

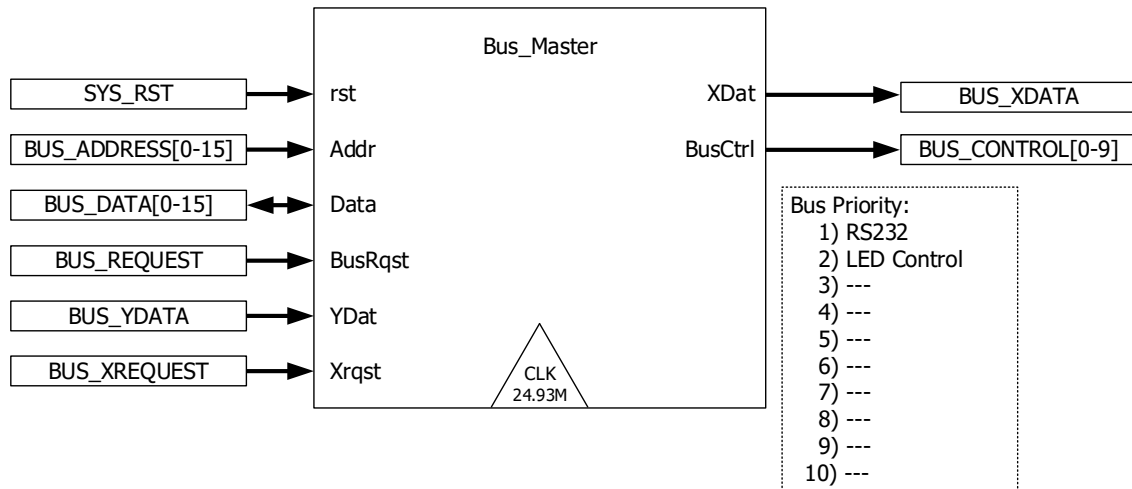


Fig. 24. Block diagram of data bus controller within Complex Programmable Logic Device of the Cybersecure Power Router

The serial communication with the modules instantiated in the CPLD uses a 9600 baud rate, and connects through Bank 3 of the MachXO2-7000HC CPLD. The current serial communication is designed for fixed packet lengths.

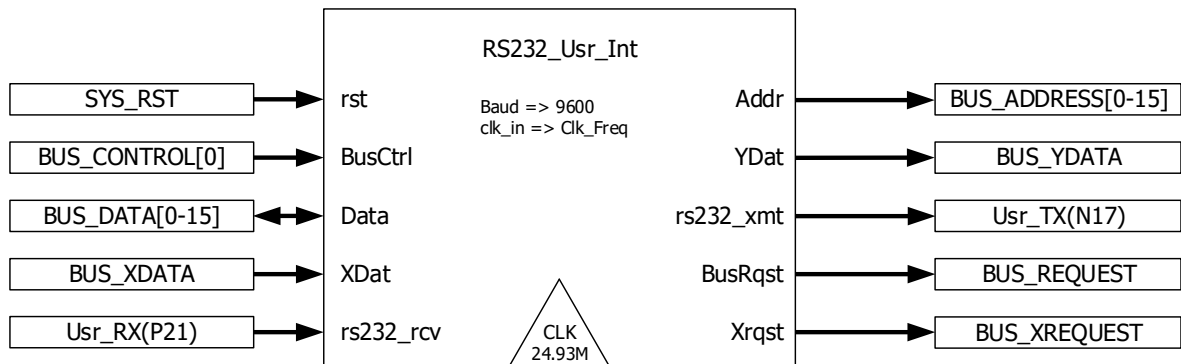


Fig. 25. Block diagram of serial interface within Complex Programmable Logic Device of the Cybersecure Power Router

3.11 Control Multiplexing

A defense-in-depth approach to controller security is explored in the Cybersecure Power Router. Control multiplexing strengthens the availability and integrity of the hardware controller, and the entire system by extension. The concept of multiplexing is common in telecommunication

[47], computer networks [48], and various signal conditioning and sampling [49] contexts. The concept is extended to an entire bus of control signals for the Cybersecure Power Router. The controllers running on the DSPs toggle a bit on GPIO-24 every time a switching cycle is completed. The firmware snippet is provided below.

```

783
784     if (HeartBeat_High){
785         GpioDataRegs.GPASET.bit.GPIO24 = 1;
786         HeartBeat_High = false;
787     }else{
788         GpioDataRegs.GPACLEAR.bit.GPIO24 = 1;
789         HeartBeat_High = true;
790     }
791

```

Fig. 26. Firmware code snippet to generate heartbeat from Digital Signal Processors

The toggled GPIO-24 pin creates a clock signal visible to external devices.

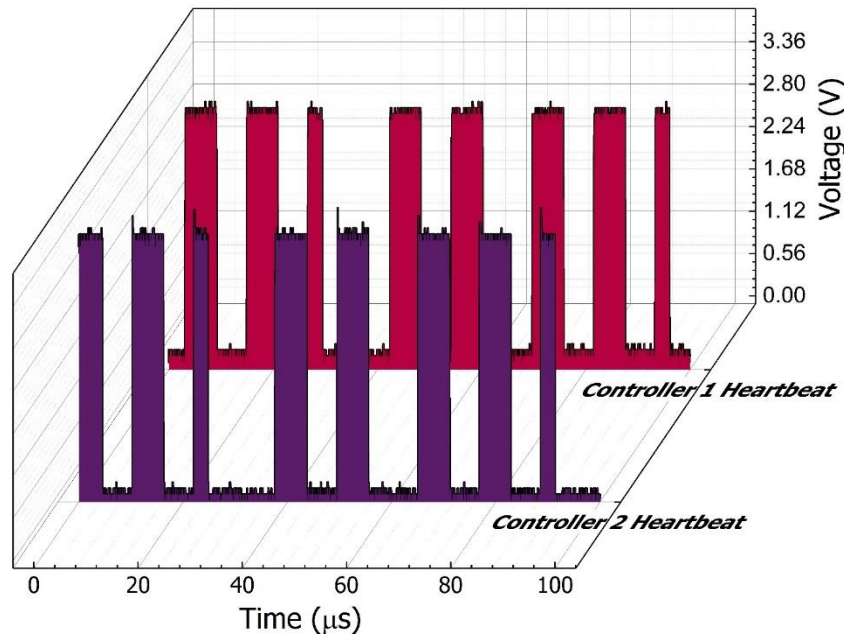


Fig. 27. Heartbeats of Controllers 1 and 2 while running identical firmware

This clock signal is the heartbeat of the controller, and is used by the Hardware Assisted Monitor to assess the liveness of the controller. Presently, if the heartbeat of a controller beats more often than 75 μs, it is considered to maintain liveness. If the heartbeat takes longer than the given 75 μs to toggle, the Hardware Assisted Monitor considers the controller to have lost liveness. The period

of the heartbeat is a function of the controller's clock rate and execution cycles of the firmware. A slower processor or a longer execution cycle would require a longer period between heartbeats.

The security features instantiated in the CPLD communicates with the Hardware Authentication Module on the UCB PE Eval board to authenticate the power electronic hardware. The password stored in the EEPROM of the UCB Hardware Authentication Module is checked against the password stored in the memory of the CPLD Hardware Authentication Module.

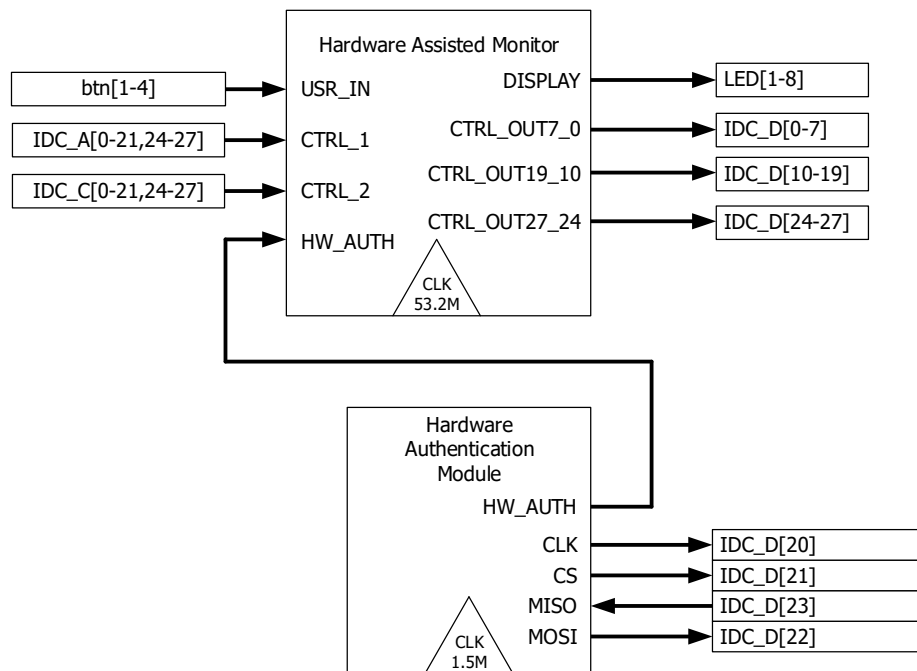


Fig. 28. Block diagram of control multiplexing using Digital Signal Processor signals and Hardware Authentication Module within the Complex Programmable Logic Device of the Cybersecure Power Router

If the two passwords match, then the "hardware is authorized" (HW_AUTH) signal goes HIGH (TRUE). If there is a mismatch between the symmetric keys, then the HW_AUTH signal goes LOW (FALSE). This mismatch will occur when the key stored in the power electronics (i.e., the Hardware Authentication Module) differs from the key stored in the controller (i.e., the Hardware Assisted Monitor).

The Hardware Assisted Monitor uses the liveness of the controllers and the authentication of the power electronics to decide the routing of control signals. When the hardware is authenticated,

control signals from Controller 1 or Controller 2 are routed to the power electronics. The Hardware Assisted Monitor assigns priority to Controllers 1 over Controller 2 if both controllers have liveness. If only one controller has liveness, that controller's control signals are routed to the power electronics. If no controller has liveness, the hardware is held in a lockout state. User inputs from buttons 1 through 4 can override this logic to manually set the routing of control signals.

3.12 Firmware and Boot Management

The control multiplexing behavior of the Hardware Assisted Monitor prevents downtime during firmware updates. The firmware is loaded into memory instantiated in the CPLD UCB allocated for boot loading. The firmware is loaded through encrypted serial communication and the internal data bus.

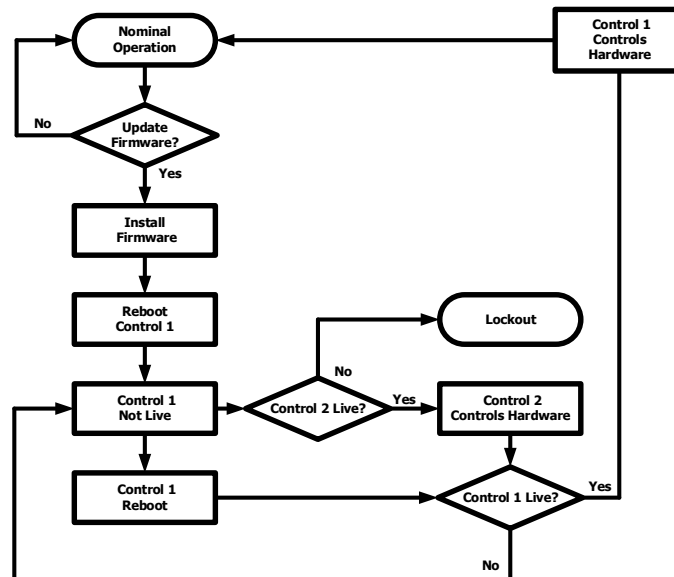


Fig. 29. Block diagram of hot patching process

When a command is given, the designated DSP is power cycled by the CPLD. The power cycled DSP boots from the allocated memory hosting the new firmware. The DSP boots using this new firmware. While the DSP is booting, hardware control is passed to the second DSP. The hardware

continues operating while the first DSP boots with the new firmware. When the first DSP resumes operation and provides a heartbeat with a period less than 75 μ s, it is either passed control of the hardware, or remains on standby. The above block diagram illustrates this process for Controller 1 being updated. A similar process is used for updating the firmware of Controller 2. As of the time of this writing, this uptime during update process is being developed.

Hot patching refers to modifying currently used data in system memory. Hot patching, strictly speaking, refers to a process that only applies to software. The present process is similar, but works at the firmware and hardware level. Here, the process modifies currently used data flow (like control signals) in a running system. The result of both techniques is the same: a running system while patches, updates, and other fixes are applied.

3.13 Hardware Authentication

Authentication of the Power Electronics Evaluation Unified Controller Board requires the Hardware Authentication Module PCB, the PE Eval UCB, and the CPLD UCB. The process begins when power is applied to the PE Eval UCB. The on-board power is used to energize the Hardware Authentication Module PCB. The IDC D port is used to connect the PE Eval UCB and the CPLD UCB. In the CSPR prototype, the IDC D ribbon cable plugs into the Signal Splitter board, which plugs into the PE Eval UCB.

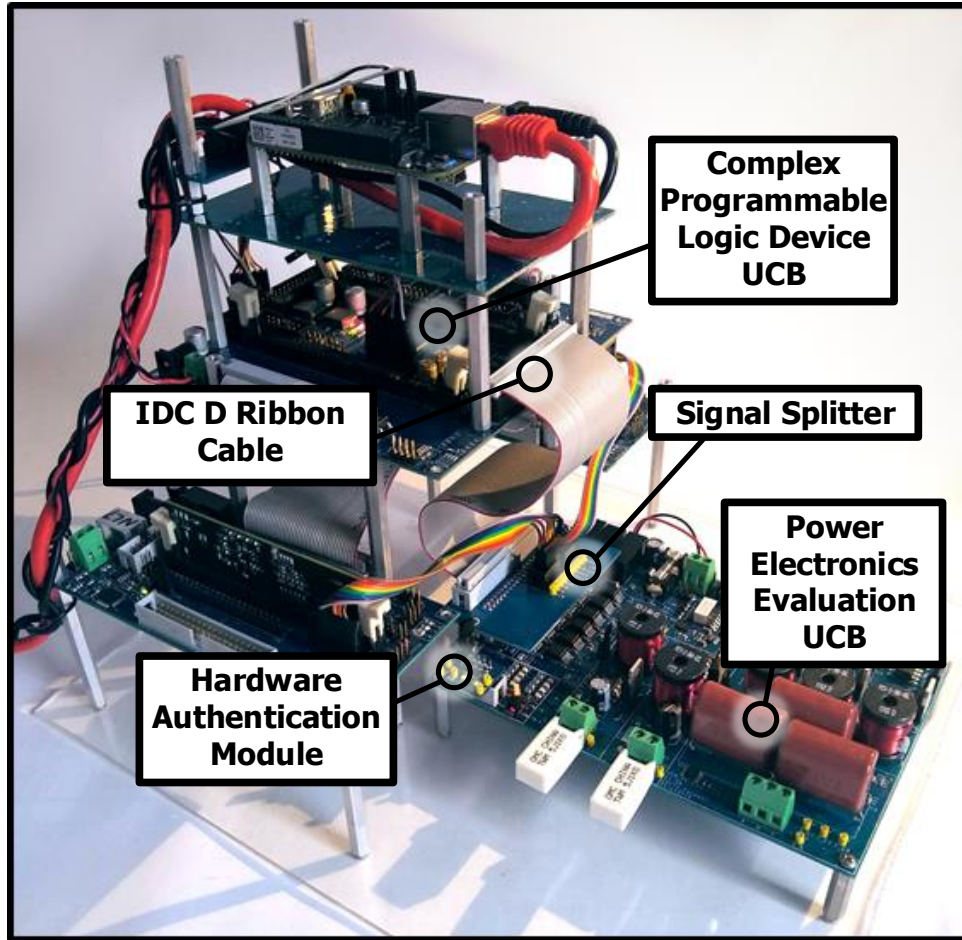


Fig. 30. Diagram of CSPR components involved in hardware authentication

The CPLD UCB interfaces with the Hardware Authentication Module PCB through four connections in the IDC D connection. These four connections provide the chip select (CS), clock (CLK), master in slave out (MISO), and master out slave in (MOSI) signals between the EEPROM of the Hardware Authentication Module on the PE Eval UCB and the VHDL Hardware Authentication Module instantiated in the CPLD UCB. The clock signal provided is 1.5 MHz. Only the read command for the EEPROM is used in the authentication process. To read a specific memory address from the 93LC46BT-I/OT EEPROM [50], the following steps are required.

Unaccounted clock cycle here.

TABLE 1-3: INSTRUCTION SET FOR X16 ORGANIZATION (93XX46B OR 93XX46C WITH ORG = 1)

Instruction	SB	Opcode	Address						Data In	Data Out	Req. CLK Cycles
ERASE	1	11	A5	A4	A3	A2	A1	A0	—	(RDY/BSY)	9
ERAL	1	00	1	0	X	X	X	X	—	(RDY/BSY)	9
EWDS	1	00	0	0	X	X	X	X	—	High-Z	9
EWEN	1	00	1	1	X	X	X	X	—	High-Z	9
READ	1	10	A5	A4	A3	A2	A1	A0	—	D15 - D0	25 26
WRITE	1	01	A5	A4	A3	A2	A1	A0	D15 - D0	(RDY/BSY)	25
WRAL	1	00	0	1	X	X	X	X	D15 - D0	(RDY/BSY)	25

Fig. 31. Excerpt from EEPROM datasheet, with erroneous information noted

First, the chip select must go high and remain high for the duration of the instruction. Sent data on the MOSI signal is read at the falling edge of the clock. The starting bit of "1" is given on the MOSI signal. The operational code for read, "10", is given over the next two clock cycles. A 6-bit address is then supplied. A hold cycle, not present in the datasheet, but present in the captured waveform below, is provided to transition between the final bit of the address (read on the falling edge of the clock) and the first data out (sent on the rising edge of the clock).

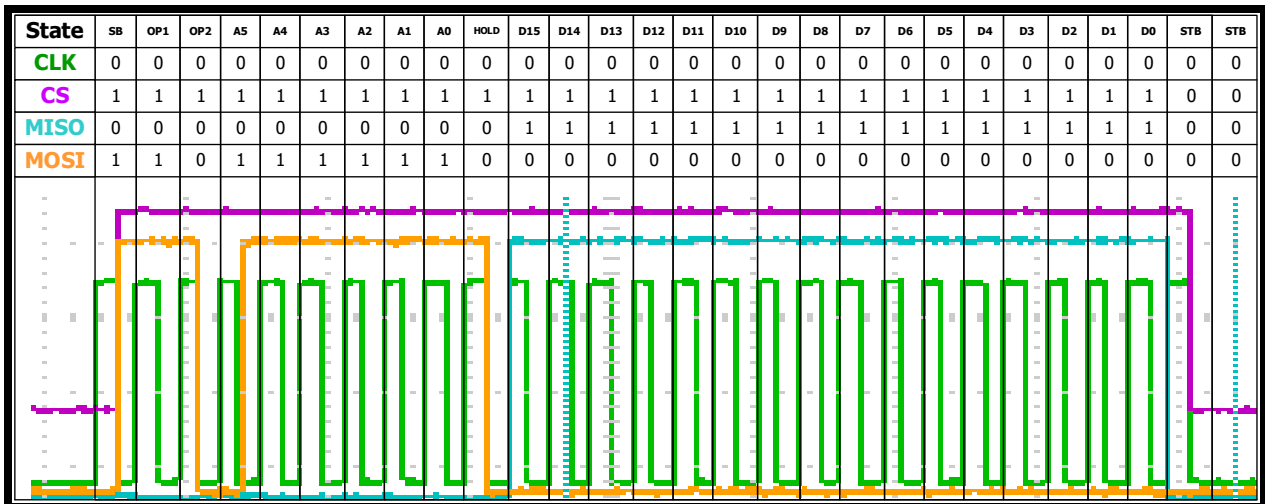


Fig. 32: Hardware Authentication Module communication beside oscilloscope capture

A 16-bit value is read from each address. After transmission, MOSI and CS are set low. The system is now in standby, and ready for another instruction cycle. The read values are parts of a 1 kilobit

password. There are 64 values in total. These values are flashed onto the EEPROM with an Arduino running the "EEPROM_Write_PASSWORD" program (source code provided in Appendix B). A matching password is included in the VHDL Hardware Authentication Module instantiated in the CPLD UCB. These values are polled by the Hardware Authentication Module in the CPLD UCB approximately five times a second. While the passwords match, a HW_AUTH ("hardware is authorized") signal is provided to the Hardware Assisted Monitor to allow control signals to pass to the hardware. The result is that the CSPR hardware will only run if the Hardware Authentication Module is operational.

3.14 Submodule Encrypted Communication

Communication between components of the CSPR prototype is designed to be encrypted. This encrypted communication is the serial communication between the two digital signal processors and the complex programmable logic device, and serial communication between the BeagleBone Black and the complex programmable logic device.

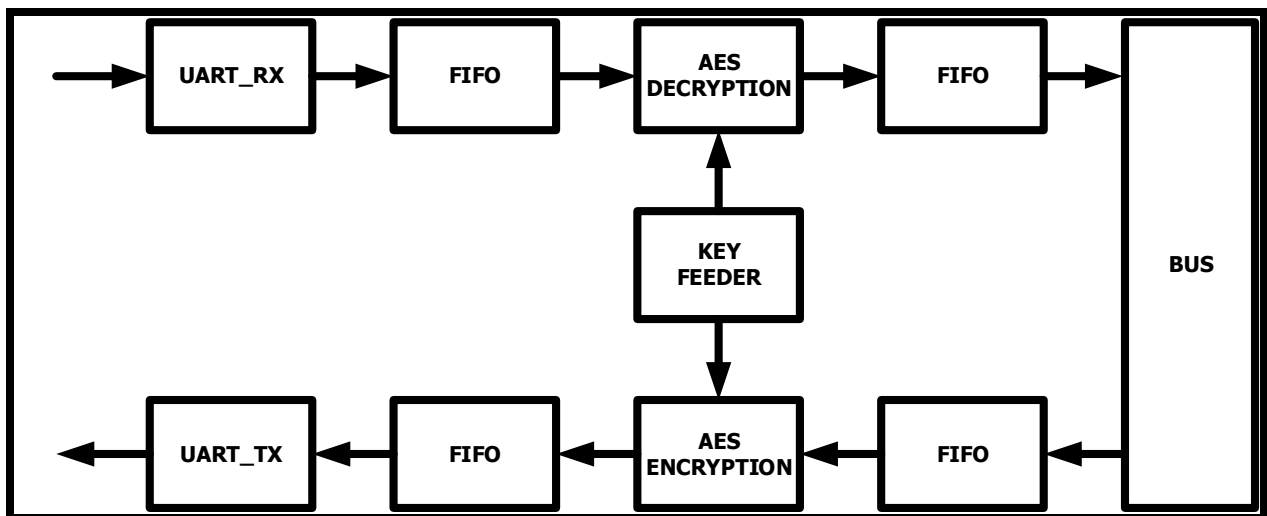


Fig. 33. Block diagram of encrypted serial communication within Complex Programmable Logic Device used in the Cybersecure Power Router

The Advanced Encryption Standard 128-bit (AES-128) is used for this encryption. A key feeder supplies the 128-bit key to the two AES-128 instantiations used to encrypt and decrypt serial communication. This allows a rolling key to be used in later development of the CSPR project. Additionally, different keys can be used for each communication port, and have separate keys for receiving and transmitting serial information. First In, First Out (FIFO) buffers are used to queue data until the appropriate block length is reached for encryption or decryption. Serial communication is made available to the data bus instantiated within the CPLD UCB. Presently, superficial, bidirectional, encrypted exchanges are available between the CSPR and the test bed. This sets the groundwork for an encrypted, MD5 secured communication pipeline for later development of the CSPR project.

3.15 Hardware Protections

Simple hardware protections are available for instantiation in the Cybersecure Power Router prototype. These hardware protections prevent shoot-through faults in the switching legs of the PE Eval UCB. These faults result when the control signals of both the high and low switch positions are set to HIGH. This creates a direct connection between the rails of the dc bus of the PE Eval UCB. The digital signal processors used in the CSPR prototype have shoot-through protections built into the PWM modules that generate the control signals of the switching legs. These built in protections can be reinforced by the Hardware Assisted Monitor of the CPLD UCB. The logic of the shoot-through protection is to allow only one switching position to be ON at a time, and both to be OFF in all other cases.

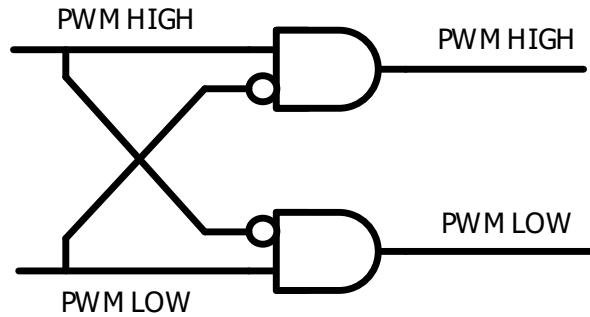


Fig. 34. Shoot-Through hardware protection

A delay or jitter of this protection on the order of 18.80 μs could be introduced by the sampling resolution of the CPLD UCB's Hardware Assisted Monitor. The resolution can be found with the following simple equation:

$$\frac{1}{f_{clock}} = T_{period} \quad (2)$$

Given the 52.3 MHz clock supplied to the Hardware Assisted Monitor, the resolution is limited to:

$$\frac{1}{53.2 \text{ MHz}} = 18.7969 \mu\text{s} \quad (3)$$

The resolution may be increased with a faster clock, up to the limitation of the hardware. CPLD cost, power demands, parasitic capacitance from layout, and EMI are all likely to limit the clock frequency at which the Hardware Assisted Monitor operates.

3.16 Display

A bank of LEDs are used to display various system information of the CPLD UCB.

LED(8)	LED(7)	LED(6)	LED(5)	LED(4)	LED(3)	LED(2)	LED(1)
Hardware IS NOT Authorized	Hardware IS Authorized	BBB Serial Comm. Operational	DSP Serial Comm. Operational	HARDWARE LOCKOUT	NOMINAL CONTROL	ONLY CONTROLLER 1	ONLY CONTROLLER 2

Fig. 35. LED display diagram for Cybersecure Power Router

LED(8) and LED(7) are dedicated to displaying the status of hardware authentication. The red LED(8) is ON and the green LED (7) is OFF if the hardware is not authenticated. The red LED(8) is OFF and the green LED (7) is ON if the hardware is authenticated. The yellow LED(6) is dedicated to displaying the serial connectivity between the BeagleBone Black and communication modules instantiated in the CPLD UCB. While serial connectivity is active, LED(6) will fade ON and OFF. LED(5) is dedicated to displaying the serial connectivity between the digital signal processors and the communication modules instantiated in the CPLD UCB. Finally, LED(4-1) are dedicated to display the state of the Hardware Assisted Monitor. LED(4) indicates a Lockout state. LED(3) indicates a nominal state of both Controller 1 and Controller 2 being live. LED(2) indicates that only Controller 1 is live. LED (1) indicates that only Controller 2 is live.

Chapter 4 - Results

The maximum duration of time between firmware execution cycles is sampled across switching frequencies. Specifically, GPIO-24 of Controller 1 is probed, and measured for the maximum pulse width. Ideally, this maximum pulse width corresponds to the duration of time between interrupt service routines triggered at every switching cycle. An ideal trend line of duration between switching periods is included in teal, below. An ideal trend line of available controller resources is included in purple. These samples are plotted in orange as switching frequency is swept from 1 kHz to 160 kHz.

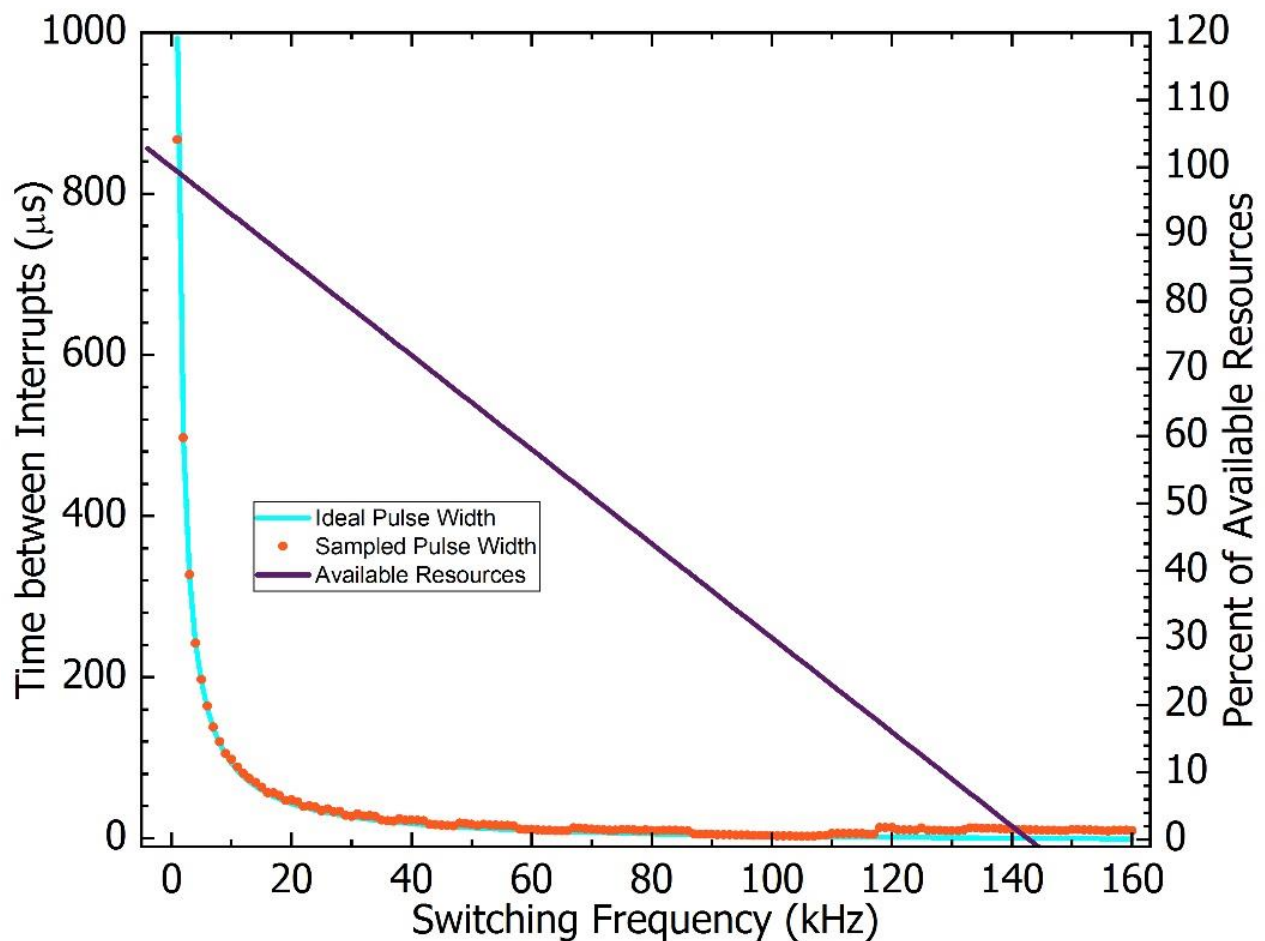


Fig. 36. Time between execution cycles of controller firmware vs. switching frequency

The Cybersecure Power Router can operate up to 2050 kHz, but the samples produce a monotonic trend beyond 120 kHz. The detail of Fig. 36 shows this transition in trends.

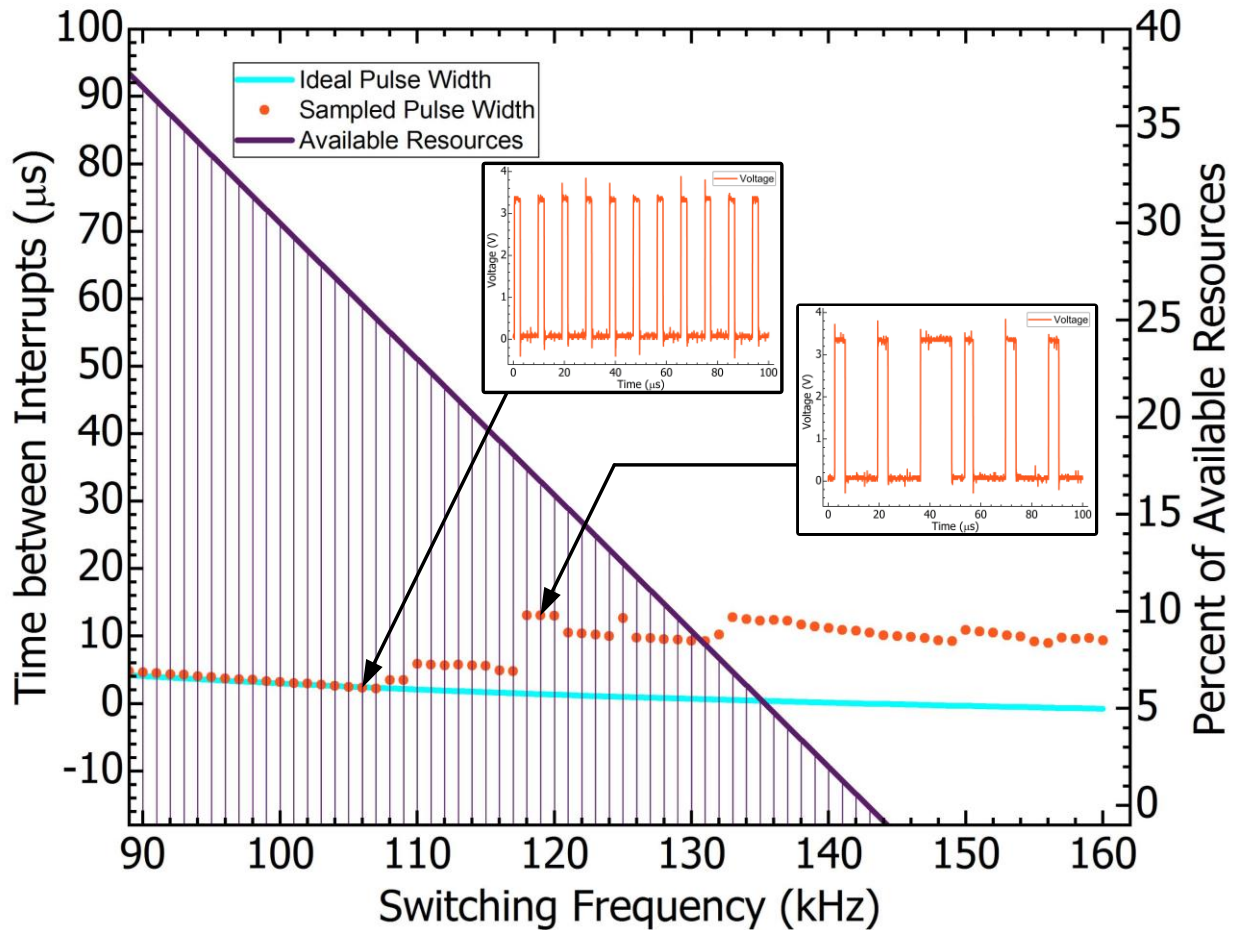


Fig. 37. Detail of Figure 36

The inverter is set to a 60 Hz output. The voltage waveform of the inverter output is captured as the switching frequency is set to 114, 118, and 128 kHz. Compare these switching frequencies to those in the figure above (Fig. 37) to see the threshold of an increasing switching frequency to degrading and decreasing the inverter output frequency. These trends are intrinsic to a controller's hardware. The operate at higher switching frequencies, all things being equal, a controller needs to do more processing in less time. This usually requires are more powerful controller.

The duration between switching period interrupts behaves ideally, until switching frequency is increased beyond 106 kHz. Increasing the switching frequency reduces the time a controller has to complete tasks required for controlling the power electronics. For the current controller hardware and firmware, a deterioration in power flow occurs when switching frequency is increased beyond 114 kHz. The output frequency of the inverter is set to 60 Hz. Yet, as switching frequency is increased, inverter output falls to 45.05 and finally 30.07 Hz.

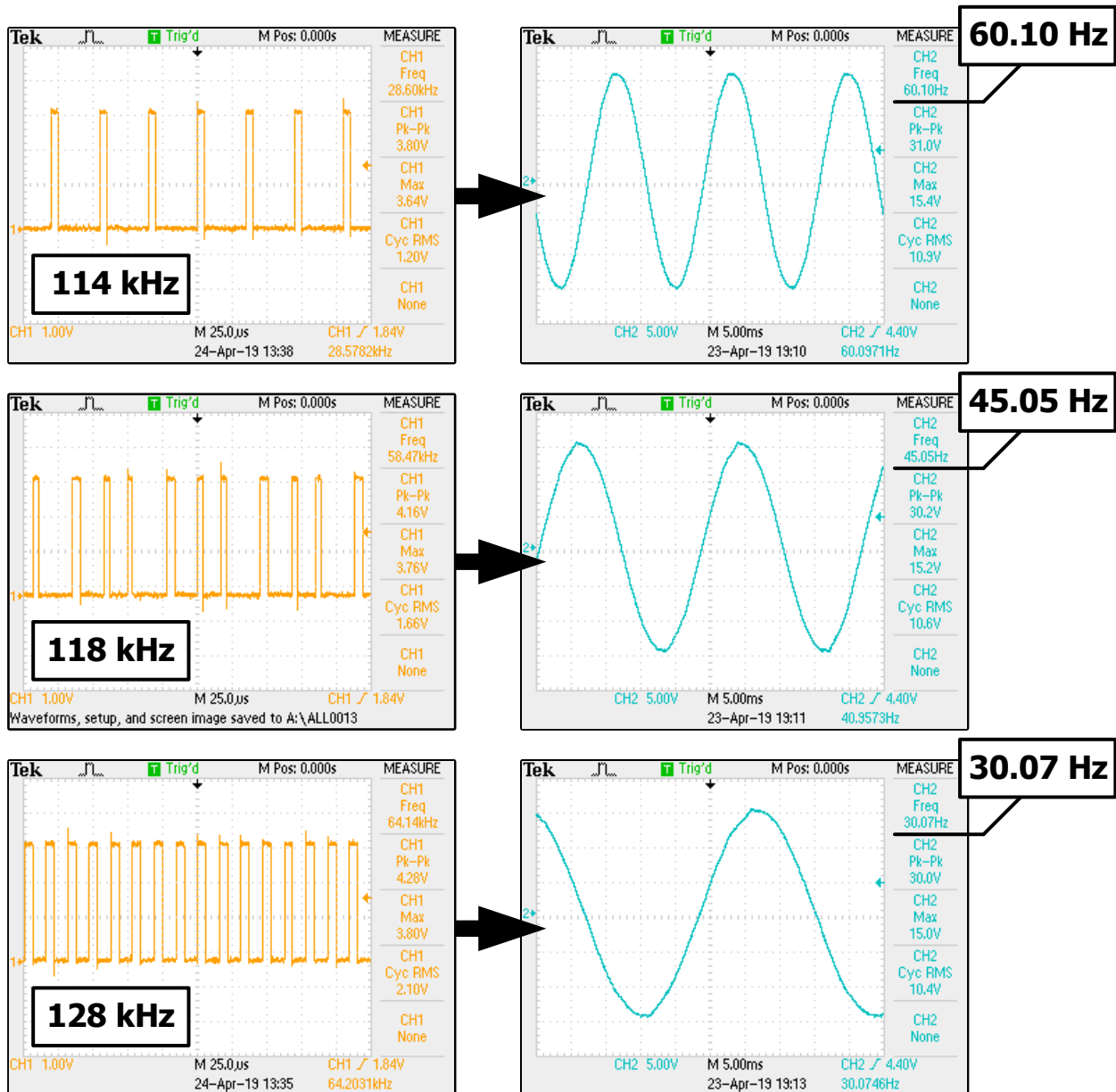


Fig. 38. Inverter output voltage at 114, 118, and 128 kHz switching frequencies.

The interruption of ISRs before completion accounts for this. At 128 kHz switching frequency, only 10% of processing resources are expected to be available for a particular interrupt. This amount of resource utilization provides little insulation between the resources used of one task from another. A possible result is a metastable condition resulting in the inverter output frequency one half of the set output frequency. While an ISR is running, it is interrupted by another ISR. When the new ISR completes, the previous ISR is able to finish. The results of the new ISR are overwritten by the previous ISR.

To safeguard against such overrun conditions, the sensitivity of the Hardware Assisted Monitor to loss of controller liveness can be adjusted. This can be done by adjusting the timing requirements of the controller heartbeat against the expected pulse width (as pictured in figures 36 and 37). The Hardware Assisted Monitor can reroute control away from the deteriorating controller, such as one causing 45.05 Hz or 30.07 Hz inverter output (as pictured in figure 38). If the Hardware Assisted Monitor and heartbeat features are thus employed, the power flow of the power electronics will be protected in case of firmware loss of liveness.

This security feature may result in disruptions and phase shifts if the controllers are not synchronized. Figure 39 shows the voltage waveform of the inverter output while control is rerouted.

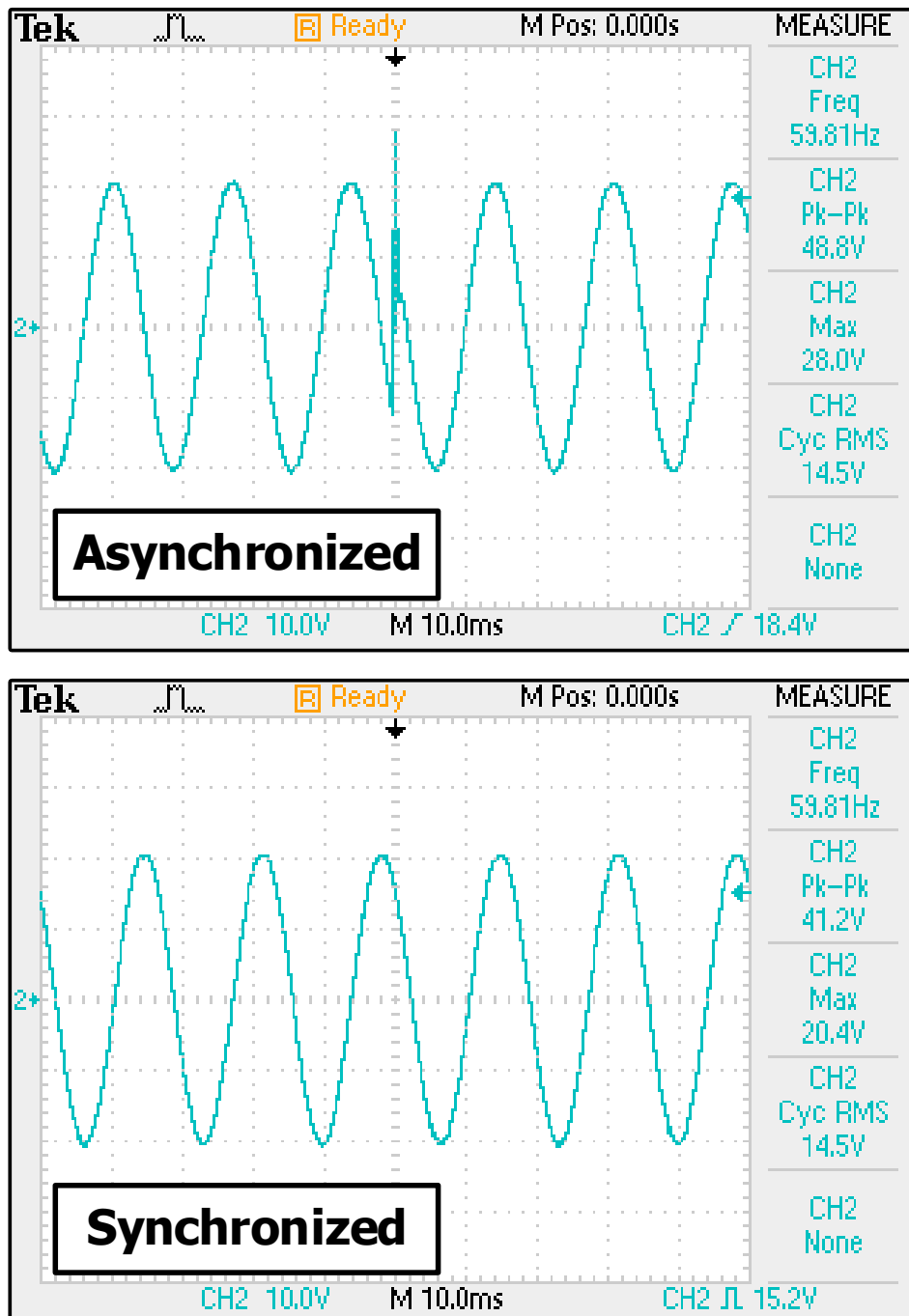


Fig. 39. Inverter output during controller transition

The inset labeled “Asynchronized” shows the possible result of routing control between controllers out of phase with each other. A sharp transient or instability may result as the hardware jumps from one point in the phase to another as control shifts between controllers. This problem is avoided if the controllers are kept in phase, as shown in the “Synchronized” inset. Here, both

controllers are synchronized to one another, or to the same signal. An example of this is grid-tied inverter controllers kept in phase with one another by locking onto the same grid frequency.

A Hardware Assisted Monitor, firmware modification, and a second controller are required for these security features. These features maintain the operation of the system, rather than cause downtime in case of fault or failure. These features use a co-processor to instantiate the Hardware Assisted Monitor and a second DSP to instantiate the second controller. Both of these design choices incur an economic and non-recurring engineering cost relatively high to the cost of the CSPR prototype. Using the Hardware Assisted Monitor as a second controller or a failsafe is also a possibility. The present work shows one example of a secure system. It also raises many possibilities for new cybersecure architectures that balance security, cost, and performance.

Chapter 5 - Future Work

5.1 Multi-Mission Controls

The Cybersecure Power Router allows great flexibility in the operation of controls. This flexibility can be extended to allow multi-mission controls. The digital signal processor controls used in the present prototype are redundant. This is not by way of necessity, but of convenience. A different controller could operate in each DSP. These controllers could be optimized for energy management, maximally secure operation, communication network facilitation, grid-reliability, or other objectives or missions. Hardware assisted monitoring within the CSPR could provide sufficient situational awareness to route hardware control to different controllers in different contexts. Consider the following as an example. Controller 1 is optimized for efficient use of energy resources. To provide more resources to power processing, communication is limited in both volume and sophistication. Controller 2 is balanced to manage energy resources and provide more secure communication. Controller 1 is used nominally, and accomplishes the primary mission of efficient use of energy resources. If a communication anomaly or attack is detected, Controller 2 is given system control. The transition between the two controllers is made smooth through the control multiplexing technique shown earlier. When the anomaly or attack is cleared, system control can be returned to Controller 1. Another possibility is booting a new controller onto the DSP used by Controller 1 while the system is operated by Controller 2. This would allow a new controller, say Controller 3, to be instantiated. Controller 3 could be optimized for a different mission, say, to perform more conservative power management or provide forensic data in case of hardware failure. Control could be switched from Controller 2 to Controller 3, and the process

could be repeated. In this way, two DSPs within the Cybersecure Power Router could be used to provide controllers with many different missions.

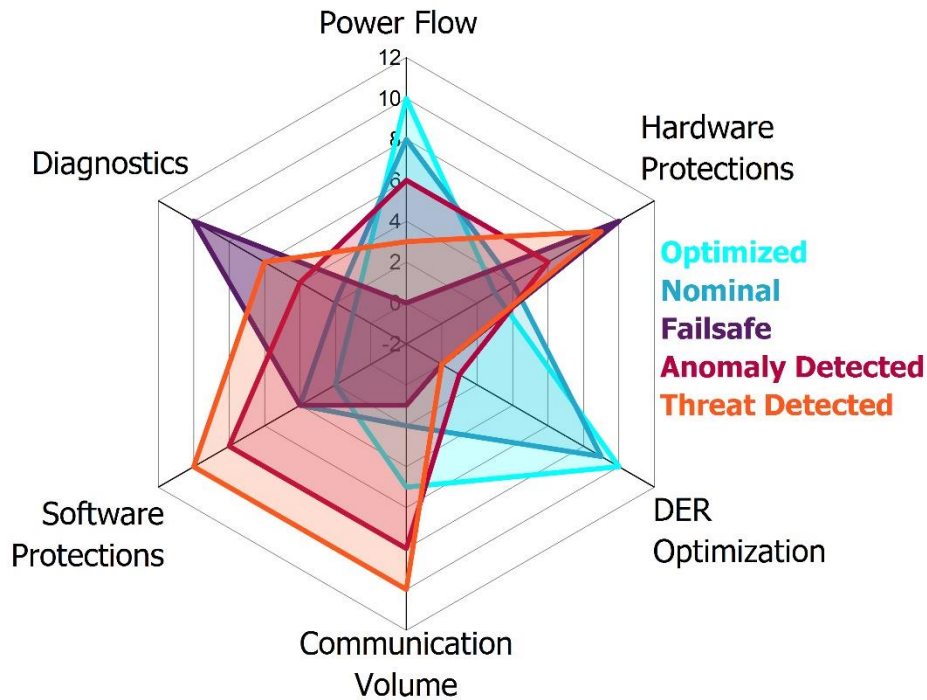


Fig. 40: Radar chart of missions for controls

To reduce cost, controllers could be instantiated in the complex programmable logic device or other hardware of the Cybersecure Power Router. This would remove the need for a second DSP, or possibly both DSPs.

5.2 SGPN and CSPR Integration and Completion

The integration of the Cybersecure Power Router and the Smart Green Power Node are required to realize a secure distributed energy resource pre-production prototype. The Smart Green Power Node contributes hardware designs rated for 2 kW operation, power flow optimization, energy generation prediction, grid arbitrage, and sophisticated controls.

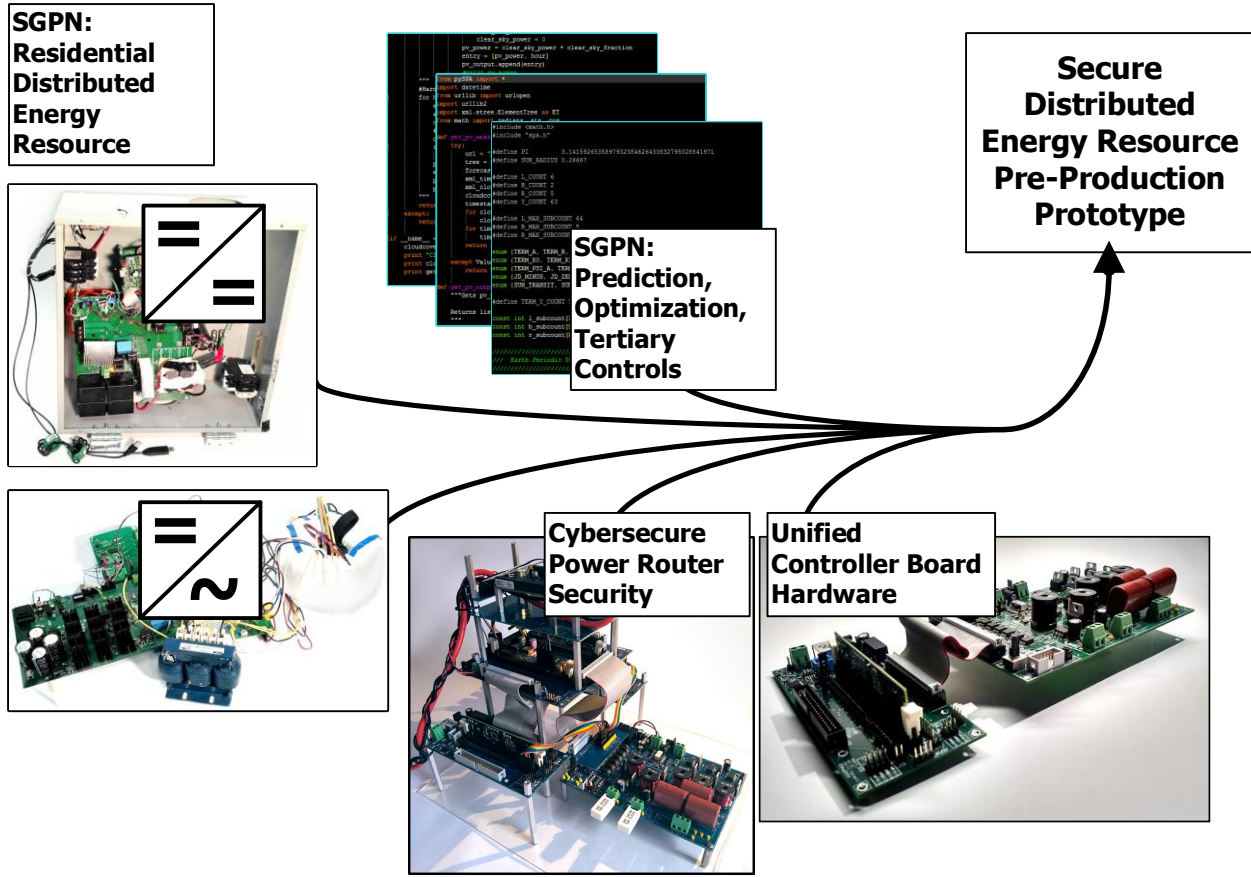


Fig. 41: CSPR and SGPN migration

The Cybersecure Power Router provides tested primary and secondary controls, protected system operation, and enhanced security. Both projects, however, have challenges to their forward development towards a pre-production prototype.

CSPR lacks the power ratings required in the prototype, and is not yet complete as a project. Five milestones remain in the CSPR project: external communication protocols, integrated user/server authentication, exhaustive testing, documentation, and project end. External communication protocols and User/Server authentication are developed, but have not been integrated with the CSPR prototype. Exhaustive testing of the cybersecure inverter is partially accomplished: the hardware has operated over 100 hours at various power levels. Superficial penetration tests on communication during operation were performed, and resulted in no

observable changes to system operation. Exhaustive penetration testing, specifically fuzz testing, during operation and hot-patching are suggested in further development. Finally, documentation and project end is also partially accomplished.

The SGPN lacks proper thermal management, optimized PCB layout, and coherent system-level design. Many switches are poorly cooled and some devices, such as snubber circuits, receive no forced cooling due to poor planning of thermal management. Electromagnetic interference from switching during typical operation destroyed gate drivers, disrupted serial communication, and deteriorated the integrity of feedback signals. The current SGPN has a poorly defined secondary and tertiary controller. The individual controllers are not able to coordinate power flow in a safe, autonomous way. The coordination and control of power flow through all these devices is manual. The prediction and optimization algorithms provide a simple schedule for charging and discharging the batteries throughout a day. These algorithms lack integration with the coordination and control of the system, and currently provide no improvement to system operation.

Several considerations are necessary to realize a secure DER pre-production prototype. First, an analysis of the design specifications are required to re-evaluate design parameters. The typical operating voltage of the photovoltaic panels, power rating of the dc/dc converters, topology of the dual half bridge, and capabilities of the human-machine interface may warrant re-evaluation. Additionally, the inclusion or exclusion of security features from the CSPR project require consideration. Secondly, the power electronics are suggested to be redesigned with thermal co-design. Forced air convection and an extruded heatsink common to all switching devices may provide a simple and effective means of cooling. The devices may be epoxied to aluminum nitride heat spreaders to provide high thermal conductivity and electrical insulation. Third, radiated and conducted EMI is suggested to be reduced through the reduction of PCB parasitics, and the

inclusion of snubbers and protection circuits during initial design. Finally, a well-defined, system-level control scheme is suggested to be designed early in development. A comprehensive, robust scheme is recommended for the integration of: power optimization, prediction, user preference, current protections, battery protections, various modes of operation, and individual controller operation. These schemes are suggested to be well defined before the design of individual converter controls.

Chapter 6 - Conclusion

A security-by-design process identified the key power and data assets and their dependencies for a distributed energy resource. The security-by-design process showed the dependencies between hardware, data, and power assets. Namely, firmware---as source code stored in memory and as a live instantiation as the controller---lies in the center of these dependencies. Defense-in-depth was shown by layered security using AES-128 encryption, error detection, hardware assisted monitoring, key management, MD5 hash checking, control multiplexing, heartbeat monitoring, and hardware authentication, and hardware protections. This defense-in-depth protects the integrity, confidentiality, and availability of hardware, data, and power at every layer of design. Communication security includes encryption and error checking of transmitted messages, firmware, and data shared between CSPR modules. Hardware security includes robust controls, shoot-through protection, hardware authentication, galvanic isolation, and hardware failsafe controls of connected resources. Securing the power and data flow through the Cybersecure Power Router primarily means securing the integrity and availability of the hardware controller. The Cybersecure Power Router determines the proper functioning of the controller by means of a hardware assisted monitor and a controller's heartbeat. The Cybersecure Power Router responds to a controller failure by multiplexing control signals, swapping control from a malfunctioning controller to a live one.

The Cybersecure Power Router illustrates the security-by-design process and defense-in-depth method in a single prototype. The Smart Green Power Node was evaluated for present and future use as the hardware of the Cybersecure Power Router prototype. The security features protecting liveness of controllers of the Cybersecure Power Router was researched in depth. Decreased integrity of power flow through the power electronics was shown to correlate with the loss of

liveness of the controllers. The use of a heartbeat from the controllers provides a signal sensitive to the liveness of the controller. The Hardware Assisted Monitor uses this heartbeat to provide control signals to hardware from controllers with liveness. The resulting system is resilient to firmware failure and loss of integrity at runtime. This resilience protects against controller overrun and system downtime, such as during firmware patching. Further research can investigate greater flexibility and resiliency in controls, and methods to reduce costs. A pre-production prototype can be realized from the migration of features from both the Cybersecure Power Router and the Smart Green Power Node.

References

- [1] O. Ivanchenko, V. Kharchenko, B. Moroz, L. Kabak, and S. Konovalenko, "Risk Assessment of Critical Energy Infrastructure Considering Physical and Cyber Assets: Methodology and Models," in *2018 IEEE 4th International Symposium on Wireless Systems within the International Conferences on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS-SWS)*, 2018, pp. 225–228.
- [2] C. Konstantinou, M. Maniatakos, F. Saqib, S. Hu, J. Plusquellic, and Y. Jin, "Cyber-physical systems: A security perspective," in *2015 20th IEEE European Test Symposium (ETS)*, 2015, pp. 1–8.
- [3] A. R. S. Farhan and G. M. M. Mostafa, "A Methodology for Enhancing Software Security During Development Processes," in *2018 21st Saudi Computer Society National Computer Conference (NCC)*, 2018, pp. 1–6.
- [4] S. Yang, A. Bryant, P. Mawby, D. Xiang, L. Ran, and P. Tavner, "An Industry-Based Survey of Reliability in Power Electronic Converters," *IEEE Trans. Ind. Appl.*, vol. 47, no. 3, pp. 1441–1451, May 2011.
- [5] J. Golovatchev, O. Budde, and Chin-Gi Hong, "Management of product complexity through integrated PLM in a multi-lifecycle environment," in *2009 IEEE International Technology Management Conference (ICE)*, 2009, pp. 1–9.
- [6] "IEEE Standard for System, Software, and Hardware Verification and Validation," *IEEE Std 1012-2016 Revis. IEEE Std 1012-2012 Inc. IEEE Std 1012-2016Cor1-2017*, pp. 1–260, Sep. 2017.

- [7] J. R. White and M. Doherty, “Hazards in the installation and maintenance of solar panels,” in *2017 IEEE IAS Electrical Safety Workshop (ESW)*, 2017, pp. 1–5.
- [8] Z. Li and M. Shahidehpour, “Defense-in-depth framework for microgrid secure operations against cyberattacks,” in *2017 IEEE Power Energy Society General Meeting*, 2017, pp. 1–5.
- [9] S. Sridhar, A. Hahn, and M. Govindarasu, “Cyber–Physical System Security for the Electric Power Grid,” *Proc. IEEE*, vol. 100, no. 1, pp. 210–224, Jan. 2012.
- [10] A. Mukherjee, S. A. A. Fakoorian, J. Huang, and A. L. Swindlehurst, “Principles of Physical Layer Security in Multiuser Wireless Networks: A Survey,” *IEEE Commun. Surv. Tutor.*, vol. 16, no. 3, pp. 1550–1573, Third 2014.
- [11] S. Teleke, M. E. Baran, S. Bhattacharya, and A. Q. Huang, “Rule-Based Control of Battery Energy Storage for Dispatching Intermittent Renewable Sources,” *IEEE Trans. Sustain. Energy*, vol. 1, no. 3, pp. 117–124, Oct. 2010.
- [12] E. W. Nahas, D. A. Mansour, H. A. A. el-Ghany, and M. M. Eissa, “Accurate Fault Analysis and Proposed Protection Scheme for Battery Energy Storage System Integrated with DC Microgrids,” in *2018 Twentieth International Middle East Power Systems Conference (MEPCON)*, 2018, pp. 911–917.
- [13] F. Xiao and J. D. McCalley, “Risk-Based Security and Economy Tradeoff Analysis for Real-Time Operation,” *IEEE Trans. Power Syst.*, vol. 22, no. 4, pp. 2287–2288, Nov. 2007.
- [14] A. Varghese and A. K. Bose, “Threat modelling of industrial controllers: A firmware security perspective,” in *2014 International Conference on Anti-Counterfeiting, Security and Identification (ASID)*, 2014, pp. 1–4.
- [15] N. R. Saxena and E. J. McCluskey, “Analysis of checksums, extended-precision checksums, and cyclic redundancy checks,” *IEEE Trans. Comput.*, vol. 39, no. 7, pp. 969–975, Jul. 1990.
- [16] A. Cervin, “ANALYSIS OF OVERRUN STRATEGIES IN PERIODIC CONTROL TASKS,” *IFAC Proc. Vol.*, vol. 38, no. 1, pp. 219–224, Jan. 2005.
- [17] National Institute of Standards and Technology, “FIPS 197, Advanced Encryption Standard (AES),” *Fed. Inf. Process. Stand. Publ. 197*, p. 51, Nov. 2001.
- [18] “ISO/IEC 18033-3:2010 - Information technology -- Security techniques -- Encryption algorithms -- Part 3: Block ciphers.” [Online]. Available: <https://www.iso.org/standard/54531.html>. [Accessed: 22-May-2019].

- [19] V. Saicheur and K. Piromsopa, "An implementation of AES-128 and AES-512 on Apple mobile processor," in *2017 14th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*, 2017, pp. 389–392.
- [20] M. P. Priyanka, E. L. Prasad, and A. R. Reddy, "FPGA implementation of image encryption and decryption using AES 128-bit core," in *2016 International Conference on Communication and Electronics Systems (ICCES)*, 2016, pp. 1–5.
- [21] R. Andriani, S. E. Wijayanti, and F. W. Wibowo, "Comparision Of AES 128, 192 And 256 Bit Algorithm For Encryption And Description File," in *2018 3rd International Conference on Information Technology, Information System and Electrical Engineering (ICITISEE)*, 2018, pp. 120–124.
- [22] X. Zheng and J. Jin, "Research for the application and safety of MD5 algorithm in password authentication," in *2012 9th International Conference on Fuzzy Systems and Knowledge Discovery*, 2012, pp. 2216–2219.
- [23] B. Preneel and P. C. van Oorschot, "On the security of iterated message authentication codes," *IEEE Trans. Inf. Theory*, vol. 45, no. 1, pp. 188–199, Jan. 1999.
- [24] K. Jarvinen, M. Tommiska, and J. Skytta, "Hardware Implementation Analysis of the MD5 Hash Algorithm," in *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*, 2005, pp. 298a–298a.
- [25] H. A. Abyaneh, M. Al-Dabbagh, H. K. Karegar, S. H. H. Sadeghi, and R. A. J. Khan, "A new optimal approach for coordination of overcurrent relays in interconnected power systems," *IEEE Trans. Power Deliv.*, vol. 18, no. 2, pp. 430–435, Apr. 2003.
- [26] S. Yin *et al.*, "Gate driver optimization to mitigate shoot-through in high-speed switching SiC half bridge module," in *2015 IEEE 11th International Conference on Power Electronics and Drive Systems*, 2015, pp. 484–491.
- [27] D. Arora, S. Ravi, A. Raghunathan, and N. K. Jha, "Secure embedded processing through hardware-assisted run-time monitoring," in *Design, Automation and Test in Europe*, 2005, pp. 178-183 Vol. 1.
- [28] H. Falaghi and M.- Haghifam, "Distributed Generation Impacts on Electric Distribution Systems Reliability: Sensitivity Analysis," in *EUROCON 2005 - The International Conference on "Computer as a Tool"*, 2005, vol. 2, pp. 1465–1468.
- [29] A. S. Tanenbaum, J. N. Herder, and H. Bos, "Can we make operating systems reliable and secure?," *Computer*, vol. 39, no. 5, pp. 44–51, May 2006.

- [30] J. N. Herder, H. Bos, B. Gras, P. Homburg, and A. S. Tanenbaum, "Fault isolation for device drivers," in *2009 IEEE/IFIP International Conference on Dependable Systems Networks*, 2009, pp. 33–42.
- [31] L. Li, M. Lu, and T. Gu, "Constructing runtime models of complex software-intensive systems for analysis of failure mechanism," in *2015 First International Conference on Reliability Systems Engineering (ICRSE)*, 2015, pp. 1–10.
- [32] D. Hristu-Varsakelis and W. S. Levine, *Handbook of Networked and Embedded Control Systems*. Springer Science & Business Media, 2007.
- [33] B. Alpern and F. B. Schneider, "Recognizing safety and liveness," *Distrib. Comput.*, vol. 2, no. 3, pp. 117–126, Sep. 1987.
- [34] M. Raynal, *Concurrent programming algorithms, principles, and foundations*. Heidelberg; New York: Springer-Verlag, 2013.
- [35] K. Stouffer, V. Pillitteri, S. Lightman, M. Abrams, and A. Hahn, "Guide to Industrial Control Systems (ICS) Security," National Institute of Standards and Technology, NIST SP 800-82r2, Jun. 2015.
- [36] J. Daemen and V. Rijmen, *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer Science & Business Media, 2013.
- [37] C. C. Erway, A. K p c , C. Papamanthou, and R. Tamassia, "Dynamic Provable Data Possession," *ACM Trans Inf Syst Secur*, vol. 17, no. 4, pp. 15:1–15:29, Apr. 2015.
- [38] H. Shukla, V. Singh, Y. Choi, J. Kwon, and C. Hahn, "Enhance OS security by restricting privileges of vulnerable application," in *2013 IEEE 2nd Global Conference on Consumer Electronics (GCCE)*, 2013, pp. 207–211.
- [39] Y. Li, J. M. McCune, and A. Perrig, "VIPER: Verifying the Integrity of PERipherals' Firmware," in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, New York, NY, USA, 2011, pp. 3–16.
- [40] A. Ramaswamy, S. Bratus, S. W. Smith, and M. E. Locasto, "Katana: A Hot Patching Framework for ELF Executables," in *2010 International Conference on Availability, Reliability and Security*, 2010, pp. 507–512.
- [41] Y. Huang, C. Kintala, N. Kolettis, and N. D. Fulton, "Software rejuvenation: analysis, module and applications," in *Twenty-Fifth International Symposium on Fault-Tolerant Computing. Digest of Papers*, 1995, pp. 381–390.
- [42] U. Guin, K. Huang, D. DiMase, J. M. Carulli, M. Tehranipoor, and Y. Makris, "Counterfeit Integrated Circuits: A Rising Threat in the Global Semiconductor Supply Chain," *Proc. IEEE*, vol. 102, no. 8, pp. 1207–1228, Aug. 2014.

- [43] G. Mazzanti, “Distortion limits in international standards vs. reliability of power components: Always on the safe side as to low-order voltage harmonics?,” in *2012 IEEE Power and Energy Society General Meeting*, 2012, pp. 1–8.
- [44] J. M. Fife, M. Scharf, S. G. Hummel, and R. W. Morris, “Field reliability analysis methods for photovoltaic inverters,” in *2010 35th IEEE Photovoltaic Specialists Conference*, 2010, pp. 002767–002772.
- [45] Y. Yuan, Q. Zhu, F. Sun, Q. Wang, and T. Başar, “Resilient control of cyber-physical systems against Denial-of-Service attacks,” in *2013 6th International Symposium on Resilient Control Systems (ISRCS)*, 2013, pp. 54–59.
- [46] Lattice Semiconductor, “MachXO2FamilyDataSheet-948089.pdf,” 2016. [Online]. Available: <https://www.mouser.com/datasheet/2/225/MachXO2FamilyDataSheet-948089.pdf>. [Accessed: 01-Jun-2019].
- [47] Z. Ying *et al.*, “Multiplexing efficiency of high order MIMO in mobile terminal for 5G communication at 15 GHz,” in *2016 International Symposium on Antennas and Propagation (ISAP)*, 2016, pp. 594–595.
- [48] J. Carpenter and R. Melhem, “Deterministic Multiplexing of NoC on Grid CMPs,” in *2013 IEEE 21st Annual Symposium on High-Performance Interconnects*, 2013, pp. 1–8.
- [49] E. Başar, Ü. Aygözü, E. Panayırıcı, and H. V. Poor, “Orthogonal Frequency Division Multiplexing With Index Modulation,” *IEEE Trans. Signal Process.*, vol. 61, no. 22, pp. 5536–5549, Nov. 2013.
- [50] Microchip, “1K Microwire Compatible Serial EEPROM,” 2008.

Appendix

Appendix A: Hardware and Software Design Details

The following figures are imbedded PDF objects. To fully view: right click on the figure, select Acrobat Document Object → Open. The PDF will open in a PDF reader.

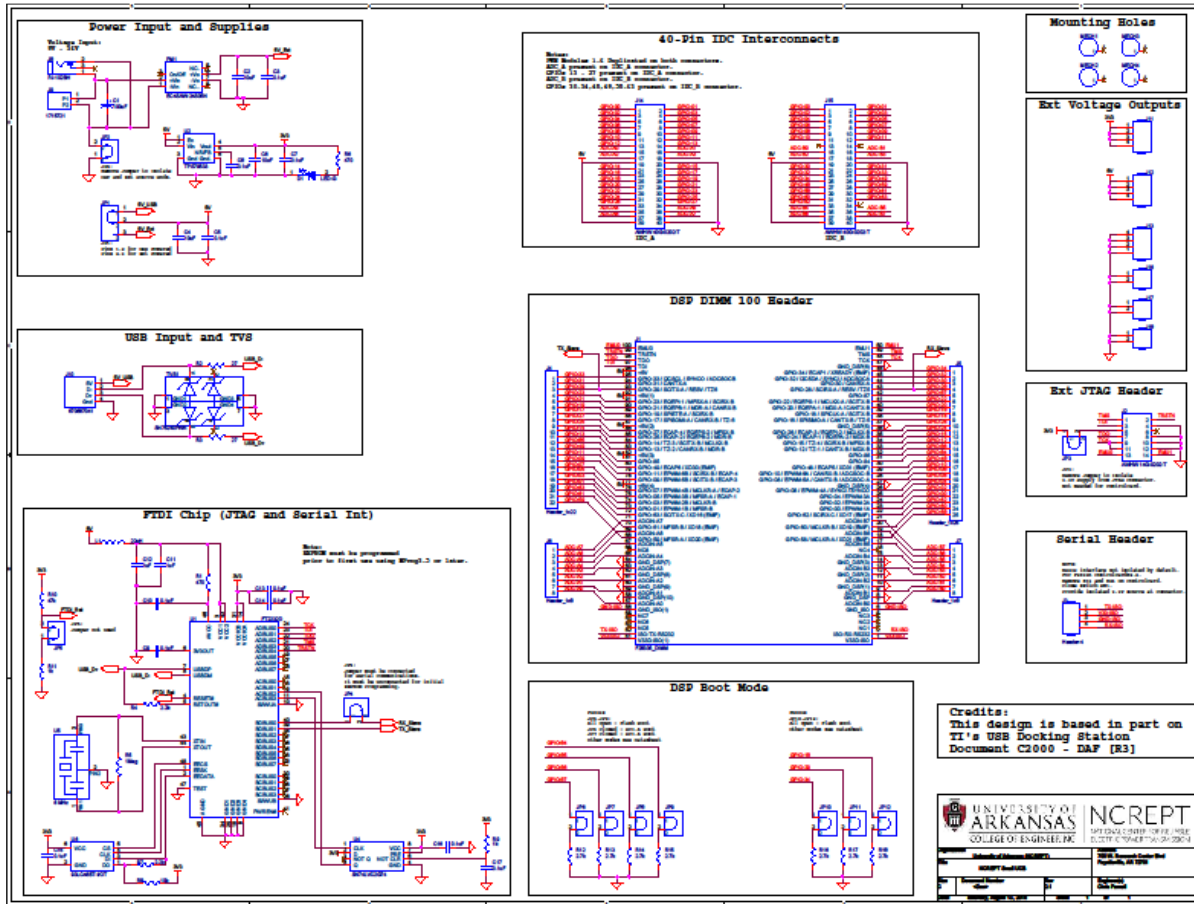


Fig. 42. Digital Signal Processor Unified Controller Board schematic

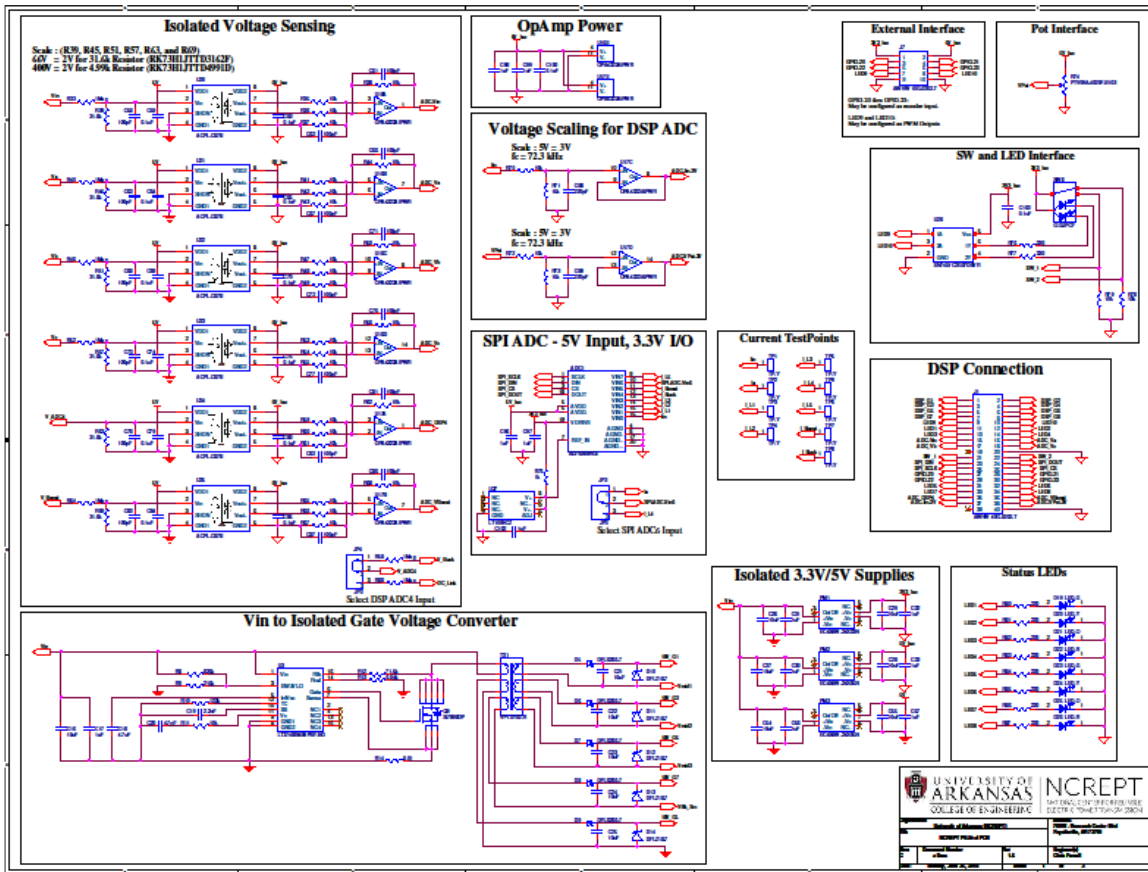


Fig. 43. Power Electronics Evaluation Unified Controller Board schematic

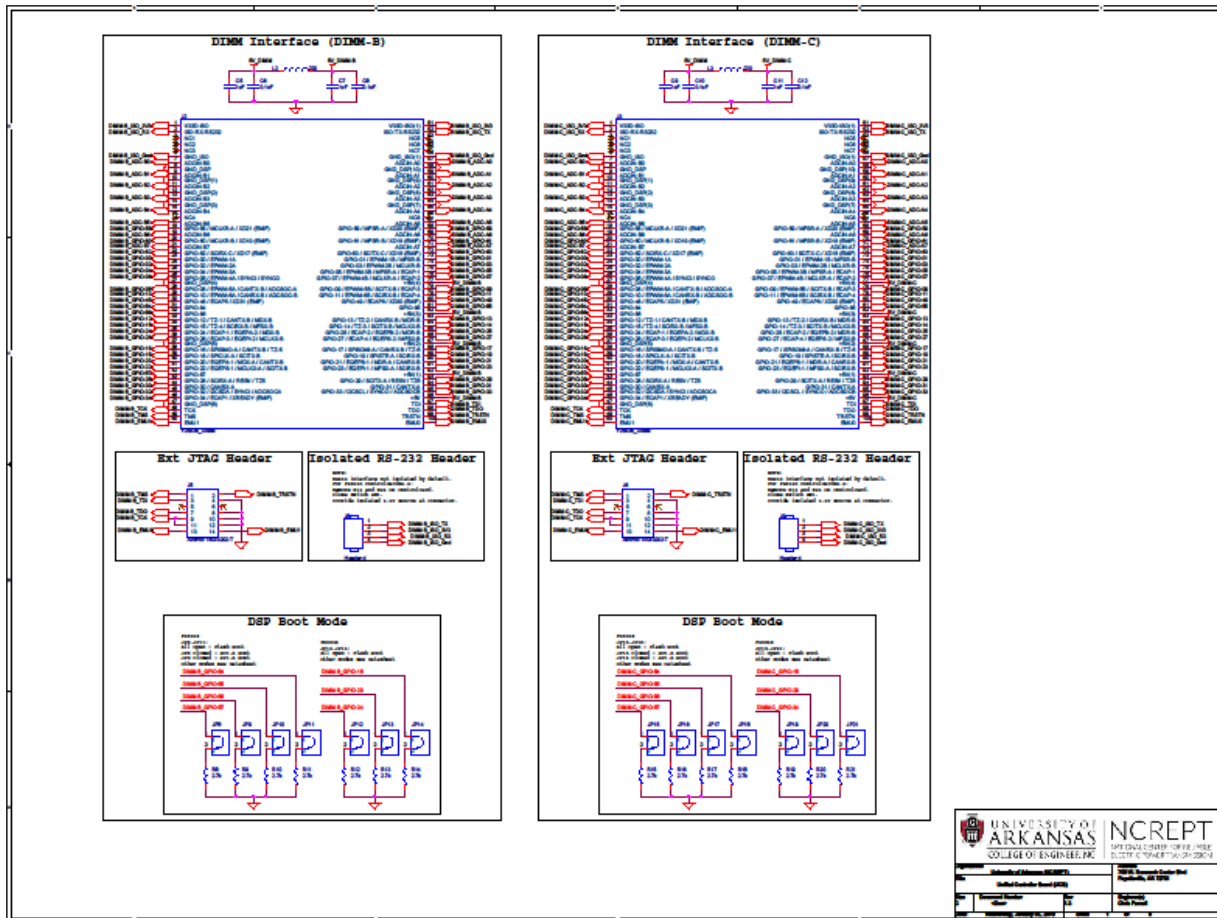


Fig. 44. Complex Programmable Logic Device Unified Controller Board schematic


REV	Description	DATE	BY
A4A	Initial production Release.	11/19/2012	GC
A5	On the initial production release the processors were to be found incorrect as supplied by TI. Parts while marked AM3359 were actually AM3352. This revision uses the correct parts.	1/2/2013	GC
ASA	1. Deleted R29-R44 from the LCD lines. 2. Added 47pF capacitors C158-C173 to LCD data lines to ground. 3. Changed schematic revision to ASA. 4. Changed a few footprints after PCB update for above changes. 5. Added access point for the battery function of the TP865217C. 6. Added Ferrite beads in series with LED power and 5V power rail of the USB host connector. Required to pass FCC/CE testing due to noise emissions on that pin. 7. Added power button to enable sleep, wakeup, power down and power up features on the system. 8. Added Modification to add 100K ohm resistor to ground to prevent crosstalk when serial cable is not plugged in.	2/6/2013	GC
ASB	1. Added 100K pulldown on J1 pin 4 to prevent crosstalk when serial cable is not connected into PCB layout. 2. Changed the LED resistors to 4.75K to lower the brightness.	5/21/2013	GC
ASC	1. Changed R56, R47, R45 to 0 ohms. 2. Changed R45 to 22 Ohms. Change was made due to production failures on some boards due to differences in impedances.	6/12/2013	GC
A6	1. Moved the enable for the VDD_3V3B regulator to VDD_3V3A rail. Change was made to reduce the delay between the ramp up of the 3.3V rails. 2. Added a AND gate to the SYS_RESETn circuitry. There is a small chance that on power up the RESETn signal on the processor may go high, causing the SYS_RESETn signal to go HI before it should. This change reinforces the reset with the POR2n resist signal. 3. Added optional zero ohm resistor to be GND_OSC0 to system ground.	7/25/2013	GC
ASA	1. Added optional zero ohm resistor to be GND_OSC1 to system ground. 2. Changed C106 to a 1uF capacitor. 3. Changed C24 to a 2.2uF capacitor. 4. Made R8 installed and R9 not installed.	12/13/2013	GC
B	1. Changed the processor to the AM3359B2C100.	1/20/2014	GC
C	1. Increased the eMMC from 2GB to 4GB.	3/21/2014	GC

PAGE NO.	SCHEMATIC PAGE
1	COVER PAGE
2	POWER MANAGEMENT
3	PROCESSOR 1 OF 3, JTAG HEADER
4	PROCESSOR 2 OF 3, UAB PORTS
5	PROCESSOR 3 OF 3
6	LED, CONFIGURATION AND BUTTON
7	DOR3 MEMORY
8	eMMC FLASH
9	10/100 ETHERNET
10	HDMI FRAMER
11	EXP CONN, uSD

NOTE: PCB Revision for this board is Rev B6

THIS SCHEMATIC IS "NOT SUPPORTED" AND DOES NOT CONSTITUTE a reference design. Only "community" support is allowed via resources at BeagleBoard.org/discuss.

THERE IS NO WARRANTY FOR THIS DESIGN, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THIS DESIGN "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE DESIGN IS WITH YOU. SHOULD THE DESIGN PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.



File	BeagleBone Black Cover Page	Rev	C
Size	Document Number		
B	450-5500-001		
Date	Friday, March 21, 2014	Sheet	1 of 11

Fig. 45. BeagleBone Black schematic

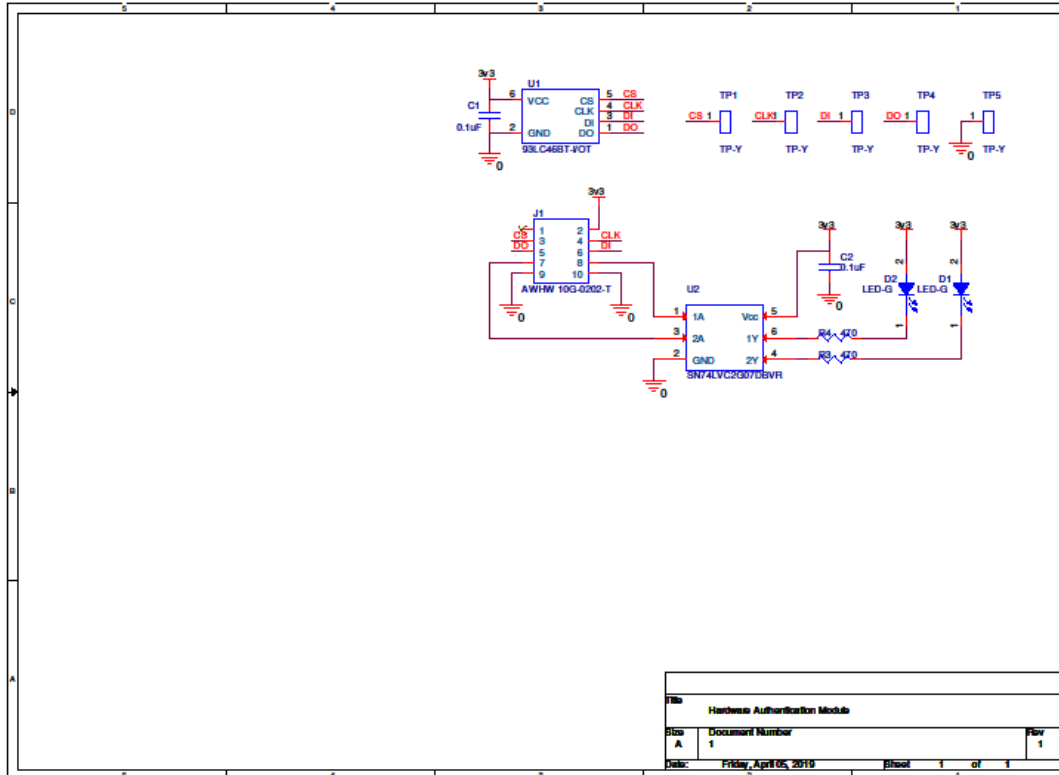


Fig. 46. Hardware Authentication Module schematic

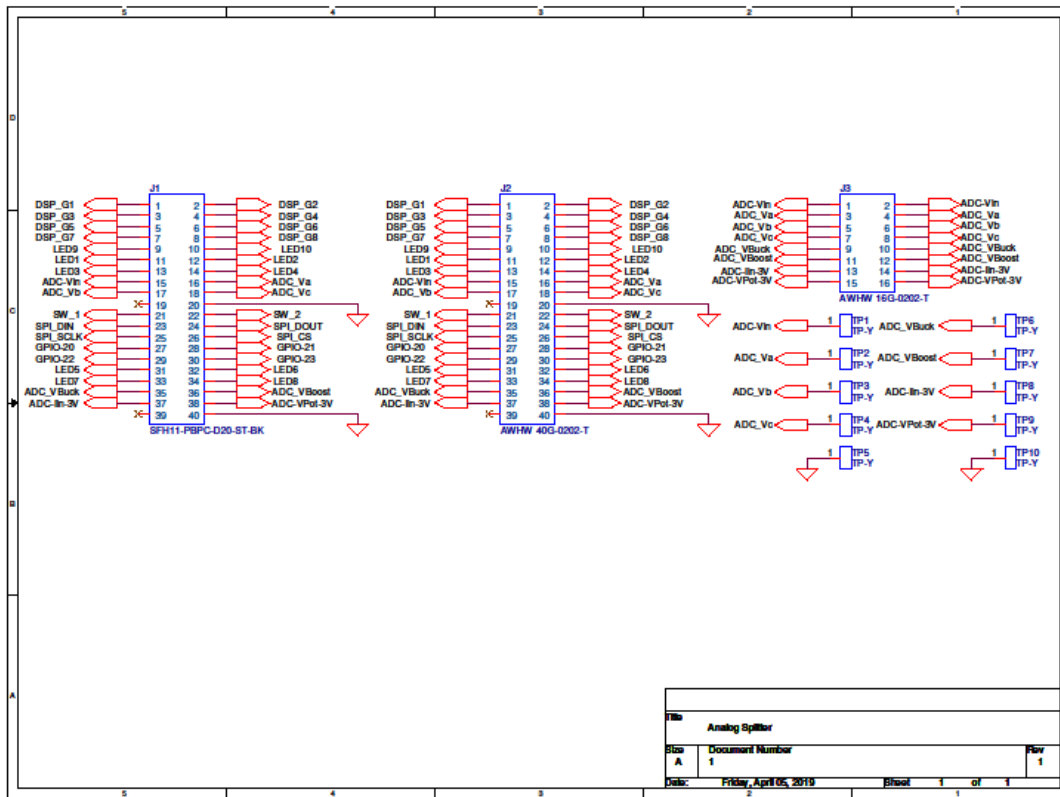


Fig. 47. Analog Splitter schematic

```

debian@beaglebone: /var/log
GNU nano 2.7.4 File: /tmp/crontab.aAsgIS/crontab Modified
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h dom mon dow   command
@reboot sleep 20 && /usr/bin/python3 /home/debian/CSPR/serial_LED_fade_v2.py > /var/log/fade.log 2>&1
^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos ^Y Prev Page
^X Exit ^R Read File ^\ Replace ^U Uncut Text ^T To Spell ^_ Go To Line ^V Next Page

```

Fig. 48. Crontab configuration on BeagleBone Black to run CPLD UCB LED control script on startup

```

1  #!/usr/bin/env python3
2  import serial
3  import time
4  import struct
5
6  #comm = input("Enter comm port (ex. \"COM16\")")
7  comm = "/dev/tty00"
8  ser = serial.Serial(comm,9600,timeout=0)
9  for a in range(0,3):
10     ser.write(b'\x71\x01\x00\x00\x01')#Enable LEDs
11     time.sleep(0.1)
12     ser.write(b'\x71\x01\x01\x00\x00')#Set Blink Frequency
13     time.sleep(0.1)
14     ser.write(b'\x71\x01\x02\x2F\xFF')#Set On Time
15     time.sleep(0.1)
16
17  prefix = b'\x71\x01\x03'
18
19  while True:
20     try:
21         for a in range(0,65534,256):
22             b = struct.pack('>H',a)
23             packet = b''.join([prefix,b])
24             ser.write(packet)
25
26         for a in range(65534,-1,-256):
27             b = struct.pack('>H',a)
28             packet = b''.join([prefix,b])
29             ser.write(packet)
30             ser.write(b'\x71\x01\x03\x00\x00')
31             time.sleep(0.5)
32     except serial.SerialException:
33         pass
34

```

Fig. 49. Content of LED control Python script running on the BeagleBone Black

Appendix B: EEPROM_WRITE_PASSWORD

```
/*
  MicrowireEEPROM Example Sketch
  Reads and writes a Microwire EEPROM.
  Written by Timo Schneider <timos@perlplexity.org> and Joe Moquin
*/
#include <MicrowireEEPROM.h>

// Microwire needs four wires (apart from VCC/GND) DO,DI,CS,CLK
// configure them here, note that DO and DI are the pins of the
// EEPROM, so DI is an output of the uC, while DO is an input
int CS=13; int CLK=12; int DI=7; int DO=2;
// EEPROMS have different sizes. Also the number of bits per page varies.
// We need to configure the page size in bits (PGS) and address bus width
// in bits (ADW). The speed at which the clock is run is configured in
// microseconds.
//int PGS=16; int ADW=8; int SPD=200;
int PGS=16; int ADW=6; int SPD=700;
unsigned int password[64] = {
  0xABBA,0xABED,0xBABE,0xBADE,0xBEAD,0xBEEF,0xCAFE,0xCEDE,
  0xDADA,0xDEAD,0xDEAF,0xDEED,0xFACE,0xFADE,0xFEED,0xFEE0,
  0xABBA,0xABED,0xBABE,0xBADE,0xBEAD,0xBEEF,0xCAFE,0xCEDE,
  0xDADA,0xDEAD,0xDEAF,0xDEED,0xFACE,0xFADE,0xFEED,0xFEE0,
  0xABBA,0xABED,0xBABE,0xBADE,0xBEAD,0xBEEF,0xCAFE,0xCEDE,
  0xDADA,0xDEAD,0xDEAF,0xDEED,0xFACE,0xFADE,0xFEED,0xFEE0,
  0xABBA,0xABED,0xBABE,0xBADE,0xBEAD,0xBEEF,0xCAFE,0xCEDE,
  0xDADA,0xDEAD,0xDEAF,0xDEED,0xFACE,0xFADE,0xFEED,0xFEE0
};

// initialize the library
MicrowireEEPROM ME(CS, CLK, DI, DO, PGS, ADW, SPD);

void setup() {
  Serial.begin(9600);
  set_memory_map();
}

void loop() {
  for (int addr=0; addr < 64; addr++) {
    unsigned int r = ME.read(addr);
    String addr_reading = "Address " + String(addr, HEX) + "(" + String(addr, DEC) + ") DO: "
+ String(r, HEX) + " ";
    Serial.println(addr_reading);
    delay(100);
  }
}
```

```
}  
  
void set_memory_map() {  
  ME.writeEnable();  
  delay(10);  
  for (int addr=0; addr < 64; addr++) {  
    ME.write(addr,password[addr]);  
    delay(10);  
  }  
  ME.writeDisable();  
  Serial.println("Write complete.");  
}
```

Appendix C: CSPR_V7.lpf

```
#LCMXO2-7000HC 4FG484C WITHIN THE UCB V1.3A
#DESIGNER: CHRIS FARNELL
#AUTHOR: JOE MOQUIN
#CONTACT: CFARNELL@UARK.EDU
#DATE: 3/12/2019
#
#CONTENTS:
# BLOCK
# GPIO:
# IDC-A
# IDC-B
# IDC-C
# IDC-D
# ADC:
# ADC1
# ADC2
# INTERFACE:
# BUTTONS
# LEDS
# DIP SWITCHES
# COMMUNICATION:
# SCI
# CLOCKS
#
#TO DO: XPORT, TI-RX/TX, LAT, EXT CLK/RST
BLOCK RESETPATHS ;
BLOCK ASYNCPATHS ;
BANK 0 VCCIO 3.3 V;
BANK 1 VCCIO 2.5 V;
BANK 2 VCCIO 3.3 V;
BANK 3 VCCIO 3.3 V;
BANK 4 VCCIO 3.3 V;
BANK 5 VCCIO 3.3 V;
#IDC A
LOCATE COMP "A0" SITE "A21" ;
LOCATE COMP "A1" SITE "C19" ;
LOCATE COMP "A2" SITE "A20" ;
LOCATE COMP "A3" SITE "D18" ;
LOCATE COMP "A4" SITE "B19" ;
LOCATE COMP "A5" SITE "C18" ;
LOCATE COMP "A6" SITE "F17" ;
LOCATE COMP "A7" SITE "A18" ;
#LOCATE COMP "A8" SITE "D17" ;
#LOCATE COMP "A9" SITE "E17" ;
```

LOCATE COMP "A10" SITE "A17" ;
LOCATE COMP "A11" SITE "C17" ;
LOCATE COMP "A12" SITE "F16" ;
LOCATE COMP "A13" SITE "E16" ;
LOCATE COMP "A14" SITE "D16" ;
LOCATE COMP "A15" SITE "B15" ;
LOCATE COMP "A16" SITE "C16" ;
LOCATE COMP "A17" SITE "E15" ;
LOCATE COMP "A18" SITE "B14" ;
LOCATE COMP "A19" SITE "F15" ;
#LOCATE COMP "A20" SITE "C15" ;
#LOCATE COMP "A21" SITE "B13" ;
#LOCATE COMP "A22" SITE "D15" ;
#LOCATE COMP "A23" SITE "G15" ;
LOCATE COMP "A24" SITE "A13" ;
LOCATE COMP "A25" SITE "E14" ;
LOCATE COMP "A26" SITE "D14" ;
LOCATE COMP "A27" SITE "B12" ;
#IDC B
LOCATE COMP "B0" SITE "AA22" ;
LOCATE COMP "B1" SITE "T19" ;
LOCATE COMP "B2" SITE "Y22" ;
LOCATE COMP "B3" SITE "W22" ;
LOCATE COMP "B4" SITE "W20" ;
LOCATE COMP "B5" SITE "V19" ;
LOCATE COMP "B6" SITE "V21" ;
LOCATE COMP "B7" SITE "V22" ;
LOCATE COMP "B8" SITE "U22" ;
LOCATE COMP "B9" SITE "U19" ;
LOCATE COMP "B10" SITE "T21" ;
LOCATE COMP "B11" SITE "R19" ;
LOCATE COMP "B12" SITE "U20" ;
LOCATE COMP "B13" SITE "T22" ;
LOCATE COMP "B14" SITE "R20" ;
LOCATE COMP "B15" SITE "R18" ;
LOCATE COMP "B16" SITE "R21" ;
LOCATE COMP "B17" SITE "P19" ;
LOCATE COMP "B18" SITE "T20" ;
LOCATE COMP "B19" SITE "R22" ;
LOCATE COMP "B20" SITE "P20" ;
LOCATE COMP "B21" SITE "P18" ;
#LOCATE COMP "B22" SITE "P21" ;
LOCATE COMP "U_{sr}_RX" SITE "P21" ;
#LOCATE COMP "B23" SITE "N17" ;
LOCATE COMP "U_{sr}_TX" SITE "N17" ;
LOCATE COMP "B24" SITE "N16" ;

LOCATE COMP "B25" SITE "N21" ;
LOCATE COMP "B26" SITE "N20" ;
LOCATE COMP "B27" SITE "M18" ;
#IDC-C
LOCATE COMP "C0" SITE "Y14" ;
LOCATE COMP "C1" SITE "AB15" ;
LOCATE COMP "C2" SITE "W12" ;
LOCATE COMP "C3" SITE "V12" ;
LOCATE COMP "C4" SITE "Y12" ;
LOCATE COMP "C5" SITE "V13" ;
LOCATE COMP "C6" SITE "AA15" ;
LOCATE COMP "C7" SITE "Y15" ;
#LOCATE COMP "C8" SITE "AB16" ;
#LOCATE COMP "C9" SITE "AA16" ;
LOCATE COMP "C10" SITE "T13" ;
LOCATE COMP "C11" SITE "U13" ;
LOCATE COMP "C12" SITE "Y16" ;
LOCATE COMP "C13" SITE "AB17" ;
LOCATE COMP "C14" SITE "W14" ;
LOCATE COMP "C15" SITE "V14" ;
LOCATE COMP "C16" SITE "Y17" ;
LOCATE COMP "C17" SITE "AB18" ;
LOCATE COMP "C18" SITE "W15" ;
LOCATE COMP "C19" SITE "V15" ;
#LOCATE COMP "C20" SITE "W16" ;
#LOCATE COMP "C21" SITE "W17" ;
#LOCATE COMP "C22" SITE "Y18" ;
#LOCATE COMP "C23" SITE "AA19" ;
LOCATE COMP "C24" SITE "AB20" ;
LOCATE COMP "C25" SITE "AB21" ;
LOCATE COMP "C26" SITE "V16" ;
LOCATE COMP "C27" SITE "U15" ;
#IDC D
LOCATE COMP "D0" SITE "C3" ;
LOCATE COMP "D1" SITE "C2" ;
LOCATE COMP "D2" SITE "F6" ;
LOCATE COMP "D3" SITE "F5" ;
LOCATE COMP "D4" SITE "E4" ;
LOCATE COMP "D5" SITE "D3" ;
LOCATE COMP "D6" SITE "G6" ;
LOCATE COMP "D7" SITE "H7" ;
LOCATE COMP "D8" SITE "B1" ;
LOCATE COMP "D9" SITE "C1" ;
LOCATE COMP "D10" SITE "H6" ;
LOCATE COMP "D11" SITE "G5" ;
LOCATE COMP "D12" SITE "E2" ;

```

LOCATE COMP "D13" SITE "D1" ;
LOCATE COMP "D14" SITE "F4" ;
LOCATE COMP "D15" SITE "G4" ;
LOCATE COMP "D16" SITE "F1" ;
LOCATE COMP "D17" SITE "G3" ;
LOCATE COMP "D18" SITE "J5" ;
LOCATE COMP "D19" SITE "J4" ;
LOCATE COMP "D20" SITE "G2" ;
LOCATE COMP "D21" SITE "G1" ;
LOCATE COMP "D22" SITE "K6" ;
LOCATE COMP "D23" SITE "K7" ;
LOCATE COMP "D24" SITE "H3" ;
LOCATE COMP "D25" SITE "H2" ;
LOCATE COMP "D26" SITE "K5" ;
LOCATE COMP "D27" SITE "L3" ;
#BUTTONS SW[1:4] ACTIVE LOW
LOCATE COMP "BTN[1]" SITE "G13" ;
LOCATE COMP "BTN[2]" SITE "F13" ;
LOCATE COMP "BTN[3]" SITE "A12" ;
LOCATE COMP "BTN[4]" SITE "C13" ;
DEFINE PORT GROUP "BTN" "BTN[1]"
"BTN[2]"
"BTN[3]"
"BTN[4]" ;
IOBUF GROUP "BTN" IO_TYPE=LVC MOS33 ;
#LEDS ACTIVE LOW
LOCATE COMP "LED[1]" SITE "R17" ;
LOCATE COMP "LED[2]" SITE "T18" ;
LOCATE COMP "LED[3]" SITE "R16" ;
LOCATE COMP "LED[4]" SITE "T17" ;
LOCATE COMP "LED[5]" SITE "Y21" ;
LOCATE COMP "LED[6]" SITE "Y20" ;
LOCATE COMP "LED[7]" SITE "U18" ;
LOCATE COMP "LED[8]" SITE "U17" ;
DEFINE PORT GROUP "LED" "LED[1]"
"LED[2]"
"LED[3]"
"LED[4]"
"LED[5]"
"LED[6]"
"LED[7]"
"LED[8]" ;
IOBUF GROUP "LED" IO_TYPE=LVC MOS25 PULLMODE=DOWN DRIVE=8
SLEWRATE=SLOW ;
#SCI
#LOCATE COMP "SCI_TX" SITE "W1" ;

```

```
#LOCATE COMP "SCI_RX" SITE "V2" ;
#CLOCKS
#LOCATE COMP "XTAL_CLK" SITE "V3" ;
#FREQUENCY NET "CLK" 53.200000 MHZ ;
IOBUF PORT "A0" IO_TYPE=LVCMOS33 ;
IOBUF PORT "A1" IO_TYPE=LVCMOS33 ;
IOBUF PORT "A2" IO_TYPE=LVCMOS33 ;
IOBUF PORT "A3" IO_TYPE=LVCMOS33 ;
IOBUF PORT "A4" IO_TYPE=LVCMOS33 ;
IOBUF PORT "A5" IO_TYPE=LVCMOS33 ;
IOBUF PORT "A6" IO_TYPE=LVCMOS33 ;
IOBUF PORT "A7" IO_TYPE=LVCMOS33 ;
IOBUF PORT "A10" IO_TYPE=LVCMOS33 ;
IOBUF PORT "A11" IO_TYPE=LVCMOS33 ;
IOBUF PORT "A12" IO_TYPE=LVCMOS33 ;
IOBUF PORT "A13" IO_TYPE=LVCMOS33 ;
IOBUF PORT "A14" IO_TYPE=LVCMOS33 ;
IOBUF PORT "A15" IO_TYPE=LVCMOS33 ;
IOBUF PORT "A16" IO_TYPE=LVCMOS33 ;
IOBUF PORT "A17" IO_TYPE=LVCMOS33 ;
IOBUF PORT "A18" IO_TYPE=LVCMOS33 ;
IOBUF PORT "A19" IO_TYPE=LVCMOS33 ;
IOBUF PORT "A24" IO_TYPE=LVCMOS33 ;
IOBUF PORT "A25" IO_TYPE=LVCMOS33 ;
IOBUF PORT "A26" IO_TYPE=LVCMOS33 ;
IOBUF PORT "A27" IO_TYPE=LVCMOS33 ;
IOBUF PORT "C0" IO_TYPE=LVCMOS33 ;
IOBUF PORT "C1" IO_TYPE=LVCMOS33 ;
IOBUF PORT "C2" IO_TYPE=LVCMOS33 ;
IOBUF PORT "C3" IO_TYPE=LVCMOS33 ;
IOBUF PORT "C4" IO_TYPE=LVCMOS33 ;
IOBUF PORT "C5" IO_TYPE=LVCMOS33 ;
IOBUF PORT "C6" IO_TYPE=LVCMOS33 ;
IOBUF PORT "C7" IO_TYPE=LVCMOS33 ;
IOBUF PORT "C10" IO_TYPE=LVCMOS33 ;
IOBUF PORT "C11" IO_TYPE=LVCMOS33 ;
IOBUF PORT "C12" IO_TYPE=LVCMOS33 ;
IOBUF PORT "C13" IO_TYPE=LVCMOS33 ;
IOBUF PORT "C14" IO_TYPE=LVCMOS33 ;
IOBUF PORT "C15" IO_TYPE=LVCMOS33 ;
IOBUF PORT "C16" IO_TYPE=LVCMOS33 ;
IOBUF PORT "C17" IO_TYPE=LVCMOS33 ;
IOBUF PORT "C18" IO_TYPE=LVCMOS33 ;
IOBUF PORT "C19" IO_TYPE=LVCMOS33 ;
IOBUF PORT "C24" IO_TYPE=LVCMOS33 ;
IOBUF PORT "C25" IO_TYPE=LVCMOS33 ;
```

```
IOBUF PORT "C26" IO_TYPE=LVCMOS33 ;
IOBUF PORT "C27" IO_TYPE=LVCMOS33 ;
IOBUF PORT "D23" IO_TYPE=LVCMOS33 ;
IOBUF PORT "D0" IO_TYPE=LVCMOS33 ;
IOBUF PORT "D1" IO_TYPE=LVCMOS33 ;
IOBUF PORT "D2" IO_TYPE=LVCMOS33 ;
IOBUF PORT "D3" IO_TYPE=LVCMOS33 ;
IOBUF PORT "D4" IO_TYPE=LVCMOS33 ;
IOBUF PORT "D5" IO_TYPE=LVCMOS33 ;
IOBUF PORT "D6" IO_TYPE=LVCMOS33 ;
IOBUF PORT "D7" IO_TYPE=LVCMOS33 ;
IOBUF PORT "D8" IO_TYPE=LVCMOS33 ;
IOBUF PORT "D9" IO_TYPE=LVCMOS33 ;
IOBUF PORT "D10" IO_TYPE=LVCMOS33 ;
IOBUF PORT "D11" IO_TYPE=LVCMOS33 ;
IOBUF PORT "D12" IO_TYPE=LVCMOS33 ;
IOBUF PORT "D13" IO_TYPE=LVCMOS33 ;
IOBUF PORT "D14" IO_TYPE=LVCMOS33 ;
IOBUF PORT "D15" IO_TYPE=LVCMOS33 ;
IOBUF PORT "D16" IO_TYPE=LVCMOS33 ;
IOBUF PORT "D17" IO_TYPE=LVCMOS33 ;
IOBUF PORT "D18" IO_TYPE=LVCMOS33 ;
IOBUF PORT "D19" IO_TYPE=LVCMOS33 ;
IOBUF PORT "D20" IO_TYPE=LVCMOS33 ;
IOBUF PORT "D21" IO_TYPE=LVCMOS33 ;
IOBUF PORT "D22" IO_TYPE=LVCMOS33 ;
IOBUF PORT "D24" IO_TYPE=LVCMOS33 ;
IOBUF PORT "D25" IO_TYPE=LVCMOS33 ;
IOBUF PORT "D26" IO_TYPE=LVCMOS33 ;
IOBUF PORT "D27" IO_TYPE=LVCMOS33 ;
```

Appendix D: hardware_protections.vhd

```
LIBRARY ieee;  
USE ieee.std_logic_1164.ALL;  
  
ENTITY censor IS  
    PORT (  
        pwm_i : IN STD_LOGIC_VECTOR(1 DOWNTO 0);  
        pwm_o : OUT STD_LOGIC_VECTOR(1 DOWNTO 0)  
    );  
END censor;  
  
ARCHITECTURE behavior OF censor IS  
BEGIN  
    WITH pwm_i SELECT pwm_o <=  
        "01" WHEN "01",  
        "10" WHEN "10",  
        "00" WHEN OTHERS;  
  
END behavior;
```

Appendix E: CSPR_MODULES.vhdl

```
-----
-- Company: University of Arkansas (NCREPT)
-- Engineer: Chris Farnell, Joe Moquin
--
-- Create Date:          March 12, 2019
-- Design Name:         Generic Components
-- Module Name:        Various
-- Project Name:       Cybersecure Power Router
-- Target Devices:     LCMXO2-7000HC-4FG484C (MachXO2 Eval Board)
--
-----
--#####Generic
Components#####--
-----Bus Interface-----
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE ieee.numeric_std.ALL;

ENTITY Bus_Int IS
    GENERIC (
        CONSTANT DATA_WIDTH : INTEGER := 16;
        CONSTANT Address_WIDTH : INTEGER := 16
    );
    PORT (
        clk : IN std_logic;
        rst : IN std_logic;
        DataIn : IN std_logic_vector(DATA_WIDTH - 1 DOWNT0 0);
        DataOut : OUT std_logic_vector(DATA_WIDTH - 1 DOWNT0 0);
        AddrIn : IN std_logic_vector(Address_WIDTH - 1 DOWNT0 0);
        WE : IN std_logic;
        RE : IN std_logic;
        Busy : OUT std_logic;
        Data : INOUT std_logic_vector(DATA_WIDTH - 1 DOWNT0 0);
        Addr : OUT std_logic_vector(Address_WIDTH - 1 DOWNT0 0);
        Xrqst : OUT std_logic;
        XDat : IN std_logic;
        YDat : OUT std_logic;
        BusRqst : OUT std_logic;
        BusCtrl : IN std_logic
    );
END;
ARCHITECTURE behavior OF Bus_Int IS
    TYPE state_type IS (S0, S1, S2, S3, S4, S5, S6, S7, S8, S9, S10);
```

```

    SIGNAL CS, NS : state_type;
    SIGNAL AddrIn_reg_o : std_logic_vector(DATA_WIDTH - 1 DOWNT0 0) :=
(Others => '0');
    SIGNAL DataIn_reg_o : std_logic_vector(DATA_WIDTH - 1 DOWNT0 0) :=
(Others => '0');
    SIGNAL LD_AddrIn, LD_DataIn, LD_Data : std_logic := '0';
BEGIN
    ---Registers
    Reg_Proc : PROCESS
    BEGIN
        WAIT UNTIL clk'event AND clk = '1';
        IF rst = '0' THEN
            AddrIn_reg_o <= (Others => '0');
            DataIn_reg_o <= (Others => '0');
            DataOut <= (Others => '0');
        ELSE
            IF (LD_AddrIn = '1') THEN
                AddrIn_reg_o <= AddrIn;
            END IF; --Register for reading input address
            IF (LD_DataIn = '1') THEN
                DataIn_reg_o <= DataIn;
            END IF; --Register for reading input address
            IF (LD_Data = '1') THEN
                DataOut <= Data;
            END IF; --Register for reading input address
        END IF;
    END PROCESS;
    ---End Registers
    ---Next State Logic Bus Interface
    NS_Bus_Int : PROCESS (CS, WE, RE, XDat, BusCtrl, AddrIn_reg_o, DataIn_reg_o)
    BEGIN
        ---Default States to remove latches
        Busy <= '1';
        Data <= (Others => 'Z');
        Addr <= (Others => 'Z');
        XRqst <= 'Z';
        YDat <= 'Z';
        BusRqst <= '0';
        NS <= S0;
        LD_AddrIn <= '0';
        LD_DataIn <= '0';
        LD_Data <= '0';
        CASE CS IS
            WHEN S0 => -- Waits until a read or write request is initiated.
                IF (RE = '1') THEN
                    NS <= S1;

```

```

        ELSIF (WE = '1') THEN
            NS <= S3;
        ELSE
            NS <= S0;
        END IF;
        Busy <= '0';
        LD_AddrIn <= '1'; -- Loads the Input Address
        LD_DataIn <= '1'; -- Loads the Input Data
        --Begin Read Process
    WHEN S1 => -- Request Control of the Bus and wait.
        IF (BusCtrl = '1') THEN
            NS <= S2;
        ELSE
            NS <= S1;
        END IF;
        BusRqst <= '1';
    WHEN S2 => -- Bus Control granted. Request data.
        IF (Xdat = '0') THEN --Active High
            NS <= S2;
        ELSE
            NS <= S0;
        END IF;
        Addr <= AddrIn_reg_o;
        XRqst <= '1'; --Active High--Active Low because of pull-ups for
internal tristate

        LD_Data <= '1';
        --End Read Process

        --Begin Write Process
    WHEN S3 => -- Request Control of the Bus and wait.
        IF (BusCtrl = '1') THEN
            NS <= S4;
        ELSE
            NS <= S3;
        END IF;
        BusRqst <= '1';
    WHEN S4 => -- Bus Control granted. Write data.
        Addr <= AddrIn_reg_o;
        Data <= DataIn_reg_o;
        YDat <= '1'; --Active High--Active Low because of pull-ups for
internal tristate

        NS <= S0;
        --End Write Process

    WHEN OTHERS =>
        NS <= S0;

```



```

        END CASE;
    END PROCESS;
    ----End Next State Logic for Bus Interface
    ----State Sync
    sync_States : PROCESS
    BEGIN
        WAIT UNTIL clk'event AND clk = '1';
        IF rst = '0' THEN
            CS <= S0;
        ELSE
            CS <= NS;
        END IF;
    END PROCESS;
    ----End State Sync
END behavior;
-----End Bus Interface-----
-----Generic FIFO-----
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE ieee.numeric_std.ALL;
ENTITY STD_FIFO IS
    GENERIC (
        DATA_WIDTH : INTEGER := 8; -- Width of FIFO
        FIFO_DEPTH : INTEGER := 512; --Depth of FIFO
        FIFO_ADDR_LEN : INTEGER := 9 -- Required number of bits to represent
        FIFO_Depth
    );
    PORT (
        CLK : IN STD_LOGIC; -- Clock input
        RST : IN STD_LOGIC; -- Active low reset
        WriteEn : IN STD_LOGIC; -- Write enable signal
        DataIn : IN STD_LOGIC_VECTOR (DATA_WIDTH - 1 DOWNT0); -- Data
        input bus
        ReadEn : IN STD_LOGIC; -- Read enable signal
        DataOut : OUT STD_LOGIC_VECTOR (DATA_WIDTH - 1 DOWNT0); --
        Data output bus
        Empty : OUT STD_LOGIC; -- FIFO empty flag
        Full : OUT STD_LOGIC -- FIFO full flag
    );
END STD_FIFO;
ARCHITECTURE Behavioral OF STD_FIFO IS
    TYPE FIFO_Memory IS ARRAY (0 TO FIFO_DEPTH - 1) OF STD_LOGIC_VECTOR
    (DATA_WIDTH - 1 DOWNT0);
    SIGNAL Memory : FIFO_Memory;
    SIGNAL Head : STD_LOGIC_VECTOR (FIFO_ADDR_LEN - 1 DOWNT0);

```

```

SIGNAL Tail : STD_LOGIC_VECTOR (FIFO_ADDR_LEN - 1 DOWNT0 0);
SIGNAL Looped : BOOLEAN;
BEGIN
  -- Memory Pointer Process
  fifo_proc : PROCESS (CLK)
  BEGIN
    IF rising_edge(CLK) THEN
      IF RST = '0' THEN
        Head <= (OTHERS => '0');
        Tail <= (OTHERS => '0');
        Looped <= false;
        Full <= '0';
        Empty <= '1';
      ELSE
        IF (ReadEn = '1') THEN
          IF ((Looped = true) OR (Head /= Tail)) THEN
            -- Update data output
            DataOut <= Memory(CONV_INTEGER(Tail));
            -- Update Tail pointer as needed
            IF (Tail = FIFO_DEPTH - 1) THEN
              Tail <= (OTHERS => '0');
              Looped <= false;
            ELSE
              Tail <= Tail + 1;
            END IF;
          END IF;
        END IF;
        IF (WriteEn = '1') THEN
          IF ((Looped = false) OR (Head /= Tail)) THEN
            -- Write Data to Memory
            Memory(CONV_INTEGER(Head)) <= DataIn;
            -- Increment Head pointer as needed
            IF (Head = FIFO_DEPTH - 1) THEN
              Head <= (OTHERS => '0');
              Looped <= true;
            ELSE
              Head <= Head + 1;
            END IF;
          END IF;
        END IF;
      END IF;
      -- Update Empty and Full flags
      IF (Head = Tail) THEN
        IF Looped THEN
          Full <= '1';
        ELSE
          Empty <= '1';
        END IF;
      END IF;
    END IF;
  END PROCESS;

```

```

                END IF;
            ELSE
                Empty <= '0';
                Full <= '0';
            END IF;
        END IF;
    END IF;
END PROCESS;
END Behavioral;
-----End Generic FIFO-----
-----16-Bit PWM with Phase shift-----
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE ieee.numeric_std.ALL;
ENTITY PWM_16b IS
    GENERIC (
        Freq_in : INTEGER := 25000000; --Clk (25 MHz)
        Max_PWM : INTEGER := 65535; --PWM Resolution (2^16-1)
        Freq_Sw : INTEGER := 6104); --PWM Switching Frequency      (Should be
derived from Main Clock) (25e6/2^12)
    PORT (
        clk : IN STD_LOGIC;
        rst : IN STD_LOGIC;
        DC : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
        Phase : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
        En : IN STD_LOGIC;
        PWM_Out : OUT STD_LOGIC);
END PWM_16b;
ARCHITECTURE Behavioral OF PWM_16b IS
    --Constants
    CONSTANT Max_Period : INTEGER := (Freq_in/Freq_Sw) - 1;
    CONSTANT PWM_Step_Inv : INTEGER := Max_PWM/Max_Period; --Clk cycle step
size for Duty cycle
    CONSTANT PWM_Max : INTEGER := Max_PWM;
    CONSTANT PWM_Min : INTEGER := PWM_Step_Inv;
    --Signals
    SIGNAL PWM_Count, DC_Read, Phase_Read : STD_LOGIC_VECTOR(15 DOWNTO
0) := (OTHERS => '0');
BEGIN
    DC_Update : PROCESS
    BEGIN
        WAIT UNTIL clk'event AND clk = '1';
        IF rst = '0' THEN
            DC_Read <= (OTHERS => '0');

```

```

        Phase_Read <= (OTHERS => '0');
    ELSE
        -- For 1.526 kHz
        -- DC_Read(15 downto 14)<=(others => '0');
        -- DC_Read(13 downto 0)<= DC(15 downto 2);
        --shift 2 places for divide by 4 (PWM_Step_Inv)
        -- Phase_Read(15 downto 14)<=(others => '0');
        -- Phase_Read(13 downto 0)<= Phase(15 downto 2);
        --shift 2 places for divide by 4 (PWM_Step_Inv)
        -- For 3.052 kHz
        -- DC_Read(15 downto 13)<=(others => '0');
        -- DC_Read(12 downto 0)<= DC(15 downto 3);
        --shift 3 places for divide by 8
        -- Phase_Read(15 downto 13)<=(others => '0');
        -- Phase_Read(12 downto 0)<= Phase(15 downto 3);
        --shift 3 places for divide by 8
        -- For 6.104 kHz
        DC_Read(15 DOWNTO 12) <= (OTHERS => '0');
        DC_Read(11 DOWNTO 0) <= DC(15 DOWNTO 4); --shift 4 places for
divide by 16 (PWM_Step_Inv)
        Phase_Read(15 DOWNTO 12) <= (OTHERS => '0');
        Phase_Read(11 DOWNTO 0) <= Phase(15 DOWNTO 4); --shift 4 places
for divide by 16 (PWM_Step_Inv)
    END IF;
END PROCESS;
Count_Update : PROCESS
BEGIN
    WAIT UNTIL clk'event AND clk = '1';
    IF rst = '0' THEN
        PWM_Count <= (OTHERS => '0');
    ELSIF (PWM_Count <= (Max_Period + Phase_Read)) THEN
        PWM_Count <= PWM_Count + 1;
    ELSE
        PWM_Count <= Phase_Read;
    END IF;
END PROCESS;
PWM_Update : PROCESS
BEGIN
    WAIT UNTIL clk'event AND clk = '1';
    IF rst = '0' THEN
        PWM_Out <= '0';
    ELSIF en = '0' THEN
        PWM_Out <= '0';
    ELSIF ((PWM_Count <= (DC_Read + Phase_Read)) AND ((PWM_Count) >
(Phase_Read))) THEN
        PWM_Out <= '1';

```

```

        ELSE
            PWM_Out <= '0';
        END IF;
    END PROCESS;
END Behavioral;
-----End 16-Bit PWM with Phase shift-----
-----16-Bit Shift Register(Parallel-to-Serial)-----
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE ieee.numeric_std.ALL;
ENTITY Sreg_PS_16 IS
    PORT (
        ld_D, sh_D, rst, clk : IN std_logic;
        Data_In : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
        Data_Out : OUT std_logic);
END;
ARCHITECTURE BEHAVIOR OF Sreg_PS_16 IS
    SIGNAL temp : STD_LOGIC_VECTOR(15 DOWNTO 0);
BEGIN
    --Data_Out <= temp(15);
    Counter_behav : PROCESS
    BEGIN
        WAIT UNTIL clk'event AND clk = '1';
        IF rst = '0' THEN
            temp <= (OTHERS => '0');
            Data_Out <= '0';
        ELSIF ld_D = '1' THEN
            temp <= Data_In;
            Data_Out <= temp(15);
        ELSIF sh_D = '1' THEN
            temp <= temp(14 DOWNTO 0) & '0';
            Data_Out <= temp(15);
        ELSE
            Data_Out <= temp(15);
        END IF;
    END PROCESS;
END BEHAVIOR;
-----End of 16-Bit Shift Register(Parallel-to-Serial)-----
-----16-Bit Shift Register(Serial-to-Parallel)-----
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE ieee.numeric_std.ALL;
ENTITY Sreg_SP_16 IS
    PORT (

```

```

        ld_D, rst, clk : IN std_logic;
        Data_In : IN std_logic;
        Data_Out : OUT STD_LOGIC_VECTOR(15 DOWNT0 0));
END;
ARCHITECTURE BEHAVIOR OF Sreg_SP_16 IS
    SIGNAL temp : STD_LOGIC_VECTOR(15 DOWNT0 0);
BEGIN
    Counter_behav : PROCESS
    BEGIN
        WAIT UNTIL clk'event AND clk = '1';
        IF rst = '0' THEN
            temp <= (OTHERS => '0');
            Data_Out <= (OTHERS => '0');
        ELSIF ld_D = '1' THEN
            temp <= temp(14 DOWNT0 0) & Data_In;
            --temp(0) <= Data_In;
            Data_Out <= temp;
        ELSE
            Data_Out <= temp;
        END IF;
    END PROCESS;
END BEHAVIOR;
-----End of 16-Bit Shift Register(Parallel-to-Serial)-----
----- Standard Counter-----
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE ieee.numeric_std.ALL;
ENTITY Std_Counter IS
    GENERIC (
        Width : INTEGER := 8 --width of counter
    );
    PORT (
        INC, rst, clk : IN std_logic;
        Count : OUT STD_LOGIC_VECTOR(Width - 1 DOWNT0 0));
END;
ARCHITECTURE BEHAVIOR OF Std_Counter IS
    SIGNAL temp : STD_LOGIC_VECTOR(Width - 1 DOWNT0 0);
BEGIN
    Counter_behav : PROCESS
    BEGIN
        WAIT UNTIL clk'event AND clk = '1';
        IF rst = '0' THEN
            temp <= (OTHERS => '0');
        ELSIF INC = '1' THEN
            temp <= temp + 1;

```

```

        ELSE
            NULL;
        END IF;
    END PROCESS;
    Count <= temp;
END BEHAVIOR;
--#####END Generic
Components#####--
--
--##### Serial
Components#####--
-- Company: University of Arkansas (NCREPT)
-- Engineer: Chris Farnell
--
-- Create Date:          3Dec2018
-- Design Name:          Bus_Interface_Common
-- Module Name:          Bus_Interface_Common
-- Project Name:         Bus Interface Example
-- Target Devices:       LCMXO2-7000HE-4TG144C (MachXO2 Eval Board)
-- Tool versions:        Lattice Diamond_x64 Build 3.10.2.115.1
-- Description:
-- This Package was created to allow for Memory Mapping as well as the declaration of various
needed constants.
---- Register and Memory Map Information:
-- This section describes the Memory Map used in this project.
-- This design contains a SPRAM Module which is 16 bits wide and 1024 entries deep.
-- Register addresses are from X"0000" to X"03FF".
-- All registers are 16-bits wide.
-- The SPRAM Module is located in the Bus_Master portion of the code.
-- This RAM Module may be accessed externally using either Serial Port interface.
-- Reserved for future use.
-- X"0200" - X"03FF"
-- LED Configuration Registers-
-- Range is X"0100" - X"010A"
-- Register Map is found as constants in Bus_Interface_Common and shared with all submodules
of this program.
-- Revisions:--
--
-- Revision 0.01 -
-- File Created; Basic\Classical Operation Implemented
--
--
-- Additional Comments:
--
--
-----

```

```

LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE ieee.numeric_std.ALL;
ENTITY Bus_Master IS
    PORT (
        clk : IN STD_LOGIC;
        rst : IN STD_LOGIC;
        Data : INOUT STD_LOGIC_VECTOR (15 DOWNT0 0);
        Addr : IN STD_LOGIC_VECTOR (15 DOWNT0 0);
        Xrqst : IN STD_LOGIC;
        XDat : OUT STD_LOGIC;
        YDat : IN STD_LOGIC;
        BusRqst : IN STD_LOGIC_VECTOR (9 DOWNT0 0);
        BusCtrl : OUT STD_LOGIC_VECTOR (9 DOWNT0 0));
END Bus_Master;
ARCHITECTURE Behavioral OF Bus_Master IS
    TYPE state_type IS (S0, S1, S2, S3, S4, S5, S6, S7, S8, S9, S10, S11);
    SIGNAL CS, NS : state_type;
    --Signals for Mem1
    SIGNAL Mem1_wea : STD_LOGIC := '0';
    SIGNAL Mem1_rst : STD_LOGIC := '0';
    SIGNAL Mem1_addra : STD_LOGIC_VECTOR(15 DOWNT0 0) := (OTHERS => '0');
    SIGNAL Mem1_dina, Mem1_douta : STD_LOGIC_VECTOR(15 DOWNT0 0) :=
(OOTHERS => '0');
    SIGNAL clk_en : STD_LOGIC := '1';
    --Signals for Registers
    SIGNAL LD_Addr, LD_Data, LD_BusCtrl : Std_Logic := '0';
    SIGNAL BusCtrl_Temp : STD_LOGIC_VECTOR(9 DOWNT0 0) := (OTHERS => '0');
    --declare SPRAM
    COMPONENT SPRAM
        PORT (
            Clock : IN std_logic;
            ClockEn : IN std_logic;
            Reset : IN std_logic;
            WE : IN std_logic;
            Address : IN std_logic_vector(9 DOWNT0 0);
            Data : IN std_logic_vector(15 DOWNT0 0);
            Q : OUT std_logic_vector(15 DOWNT0 0)
        );
    END COMPONENT;
BEGIN
    --Instantiate SPRAM_16bx1024
    Mem1 : SPRAM PORT MAP(
        Clock => clk,
        ClockEn => clk_en,

```



```

Reset => Mem1_rst,
WE => Mem1_wea,
Address => Mem1_addra(9 DOWNT0 0),
Data => Mem1_dina,
Q => Mem1_douta
);
----Registers
Reg_Proc : PROCESS
BEGIN
    WAIT UNTIL clk'event AND clk = '1';
    IF rst = '0' THEN
        Mem1_addra <= (OTHERS => '0');
        Mem1_dina <= (OTHERS => '0');
        BusCtrl <= (OTHERS => '0');
    ELSE
        IF (LD_Addr = '1') THEN
            Mem1_addra <= Addr;
        END IF; --Register for reading input address
        IF (LD_Data = '1') THEN
            Mem1_dina <= Data;
        END IF; --Register for writing input data
        IF (LD_BusCtrl = '1') THEN
            BusCtrl <= BusCtrl_Temp;
        END IF;
    END IF;
END PROCESS;
----End Registers
----Next State Logic Bus Control
NS_Bus_Ctrl : PROCESS (CS, BusRqst, XRqst, YDat, Mem1_douta)
BEGIN
    ----Default States to remove latches
    Data <= (OTHERS => 'Z');
    XDat <= '0';
    BusCtrl_Temp <= (OTHERS => '0');
    LD_BusCtrl <= '0';
    NS <= S0;
    Mem1_wea <= '0';
    LD_Addr <= '0';
    LD_Data <= '0';
    clk_en <= '1';
    CASE CS IS
        WHEN S0 => -- Waits until a request is made.
            IF (BusRqst > 0) THEN
                NS <= S1;
            ELSE
                NS <= S0;
            END IF;
    END CASE;
END PROCESS;

```

```

        END IF;
    WHEN S1 => -- Grant Control of the Bus (Priority Encoder)
        IF (BusRqst(0) = '1') THEN
            BusCtrl_Temp(0) <= '1';
        ELSIF (BusRqst(1) = '1') THEN
            BusCtrl_Temp(1) <= '1';
        ELSIF (BusRqst(2) = '1') THEN
            BusCtrl_Temp(2) <= '1';
        ELSIF (BusRqst(3) = '1') THEN
            BusCtrl_Temp(3) <= '1';
        ELSIF (BusRqst(4) = '1') THEN
            BusCtrl_Temp(4) <= '1';
        ELSIF (BusRqst(5) = '1') THEN
            BusCtrl_Temp(5) <= '1';
        ELSIF (BusRqst(6) = '1') THEN
            BusCtrl_Temp(6) <= '1';
        ELSIF (BusRqst(7) = '1') THEN
            BusCtrl_Temp(7) <= '1';
        ELSIF (BusRqst(8) = '1') THEN
            BusCtrl_Temp(8) <= '1';
        ELSIF (BusRqst(9) = '1') THEN
            BusCtrl_Temp(9) <= '1';
        END IF;
        LD_BusCtrl <= '1';
        NS <= S2;
    WHEN S2 => -- Bus Control granted. Wait until Read or Write Request.
        IF (XRqst = '1') THEN --Active High--Active Low because of
pull-ups for internal tristate
            NS <= S3;
        ELSIF (YDat = '1') THEN --Active High--Active Low because of
pull-ups for internal tristate
            NS <= S5;
        ELSE
            NS <= S2;
        END IF;
        LD_Addr <= '1';
        LD_Data <= '1';
    WHEN S3 => --(Read Operation) Send Data
        NS <= S4;
    WHEN S4 => --(Read Operation) Send Data
        data <= Mem1_douta;
        Xdat <= '1'; --Active High
        NS <= S6;
    WHEN S5 => --(Write Operation) Receive Data
        Mem1_wea <= '1';
        NS <= S6;

```

```

                WHEN S6 =>
                    LD_BusCtrl <= '1';
                    NS <= S0;
                WHEN OTHERS =>
                    NS <= S0;
            END CASE;
        END PROCESS;
    ----End Next State Logic for Bus Interface
    ----State Sync
    sync_States : PROCESS
    BEGIN
        WAIT UNTIL clk'event AND clk = '1';
        IF rst = '0' THEN
            Mem1_rst <= '1'; --reset Memory
            CS <= S0;
        ELSE
            Mem1_rst <= '0';
            CS <= NS;
        END IF;
    END PROCESS;
    ----End State Sync
END Behavioral;
--#####RS232 USR
INT#####
#####
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_unsigned.ALL;
USE IEEE.numeric_std.ALL;
USE IEEE.std_logic_arith.ALL;
ENTITY RS232_Usr_Int IS
    GENERIC (
        Baud : INTEGER := 9600; --9,600 bps
        clk_in : INTEGER := 24930000); --24.93MHz
    PORT (
        clk : IN STD_LOGIC;
        rst : IN STD_LOGIC;
        rs232_rcv : IN STD_LOGIC;
        rs232_xmt : OUT STD_LOGIC;
        Data : INOUT std_logic_vector(15 DOWNTO 0);
        Addr : OUT std_logic_vector(15 DOWNTO 0);
        Xrqst : OUT std_logic;
        XDat : IN std_logic;
        YDat : OUT std_logic;
        BusRqst : OUT std_logic;
        BusCtrl : IN std_logic

```

```

);
END RS232_Usr_Int;
ARCHITECTURE Behavioral OF RS232_Usr_Int IS
    TYPE state_type IS (S0, S1, S2, S3, S4, S5, S6, S7, S8, S9, S10, S11, S12, S13, S14,
S15, S16, S17, S18, S19, S20);
    SIGNAL CS_RS232_R, NS_RS232_R, CS_RS232_W, NS_RS232_W, CS_FIFO_Bus,
NS_FIFO_Bus : state_type;
    SIGNAL rx_done, tx_done : STD_LOGIC := '0';
    SIGNAL temp_rcv : STD_LOGIC_VECTOR(7 DOWNT0 0) := (OTHERS => '0');
    SIGNAL i, j : STD_LOGIC_VECTOR (15 DOWNT0 0) := (OTHERS => '0');
    SIGNAL uartclk : STD_LOGIC := '0';
    SIGNAL u : INTEGER;
    SIGNAL rs232_rcv_s, rs232_rcv_t : STD_LOGIC := '1';
    SIGNAL txbuff : STD_LOGIC_VECTOR(9 DOWNT0 0) := (OTHERS => '1'); --buff
used to transmit 1 bytes with start and stop bits
--Declare Signals for FIFO Serial Read
    SIGNAL STD_FIFO_R_WriteEn, STD_FIFO_R_ReadEn : STD_LOGIC := '0';
    SIGNAL STD_FIFO_R_DataIn, STD_FIFO_R_DataOut : STD_LOGIC_VECTOR(7
DOWNT0 0) := (OTHERS => '0');
    SIGNAL STD_FIFO_R_Empty, STD_FIFO_R_Full : STD_LOGIC := '0';
--Declare Signals for FIFO Serial Write
    SIGNAL STD_FIFO_W_WriteEn, STD_FIFO_W_ReadEn : STD_LOGIC := '0';
    SIGNAL STD_FIFO_W_DataIn, STD_FIFO_W_DataOut : STD_LOGIC_VECTOR(7
DOWNT0 0) := (OTHERS => '0');
    SIGNAL STD_FIFO_W_Empty, STD_FIFO_W_Full : STD_LOGIC := '0';
--Declare Signals for Bus Interface
    SIGNAL Bus_Int1_WE, Bus_Int1_RE, Bus_Int1_Busy : STD_LOGIC := '0';
    SIGNAL Bus_Int1_DataIn, Bus_Int1_DataOut, Bus_Int1_AddrIn :
STD_LOGIC_VECTOR(15 DOWNT0 0) := (OTHERS => '0');
--Declare Signals for Registers
    SIGNAL LD_busy, LD_busy2, LD_rx, LD_tx, LD_temp_data, LD_temp2 :
STD_LOGIC := '0';
    SIGNAL LD_Temp_Addr_High, LD_Temp_Addr_Low, LD_Temp_Data_High :
STD_LOGIC := '0';
    SIGNAL LD_Temp_Data_Low, ld_temp_cmd : STD_LOGIC := '0';
    SIGNAL busy, busy_reg_o, busy2, busy2_reg_o, rx, rx_reg_o, tx, tx_reg_o :
STD_LOGIC := '0';
    SIGNAL temp_data_reg_o, temp_data : STD_LOGIC_VECTOR(15 DOWNT0 0) :=
(OTHERS => '0');
    SIGNAL temp2_reg_o, temp2 : STD_LOGIC_VECTOR(7 DOWNT0 0) := (OTHERS
=> '0');
    SIGNAL Temp_Addr_High_reg_o, Temp_Addr_High : STD_LOGIC_VECTOR(7
DOWNT0 0) := (OTHERS => '0');
    SIGNAL Temp_Addr_Low_reg_o, Temp_Addr_Low : STD_LOGIC_VECTOR(7
DOWNT0 0) := (OTHERS => '0');

```

```

    SIGNAL Temp_Data_High_reg_o, Temp_Data_High : STD_LOGIC_VECTOR(7
DOWNT0 0) := (OTHERS => '0');
    SIGNAL Temp_Data_Low_reg_o, Temp_Data_Low : STD_LOGIC_VECTOR(7
DOWNT0 0) := (OTHERS => '0');
    SIGNAL Temp_Cmd_reg_o, Temp_Cmd : STD_LOGIC_VECTOR(7 DOWNT0 0) :=
(OTHERS => '0');
    ---User defined variables
    -- CM is the Clock Divder 25MHz/CM=115,200 Baud
    CONSTANT CM : INTEGER := clk_in/Baud;
    -- CN is the read offset for serial input
    CONSTANT CN : INTEGER := CM/2;
    ---End User defined variables
    --declare STD_FIFO
    COMPONENT STD_FIFO
        GENERIC (
            DATA_WIDTH : INTEGER; -- Width of FIFO
            FIFO_DEPTH : INTEGER; --      Depth of FIFO
            FIFO_ADDR_LEN : INTEGER -- Required number of bits to represent
FIFO_Depth
        );
        PORT (
            CLK : IN STD_LOGIC;
            RST : IN STD_LOGIC;
            WriteEn : IN STD_LOGIC;
            DataIn : IN STD_LOGIC_VECTOR (7 DOWNT0 0);
            ReadEn : IN STD_LOGIC;
            DataOut : OUT STD_LOGIC_VECTOR (7 DOWNT0 0);
            Empty : OUT STD_LOGIC;
            Full : OUT STD_LOGIC
        );
    END COMPONENT;
    --declare Bus Interface
    COMPONENT Bus_Int
        PORT (
            clk : IN std_logic;
            rst : IN std_logic;
            DataIn : IN std_logic_vector(15 DOWNT0 0);
            DataOut : OUT std_logic_vector(15 DOWNT0 0);
            AddrIn : IN std_logic_vector(15 DOWNT0 0);
            WE : IN std_logic;
            RE : IN std_logic;
            Busy : OUT std_logic;
            Data : INOUT std_logic_vector(15 DOWNT0 0);
            Addr : OUT std_logic_vector(15 DOWNT0 0);
            Xrqst : OUT std_logic;
            XDat : IN std_logic;

```

```

        YDat : OUT std_logic;
        BusRqst : OUT std_logic;
        BusCtrl : IN std_logic
    );
END COMPONENT;
BEGIN
--Instantiate STD_FIFO for Reading Serial Data
STD_FIFO_R : STD_FIFO
GENERIC MAP
(
    DATA_WIDTH => 8, -- Width of FIFO
    FIFO_DEPTH => 512, -- Depth of FIFO
    FIFO_ADDR_LEN => 9 -- Required number of bits to represent FIFO_Depth
)
PORT MAP
(
    CLK => clk,
    RST => rst,
    WriteEn => STD_FIFO_R_WriteEn,
    DataIn => STD_FIFO_R_DataIn,
    ReadEn => STD_FIFO_R_ReadEn,
    DataOut => STD_FIFO_R_DataOut,
    Empty => STD_FIFO_R_Empty,
    Full => STD_FIFO_R_Full
);
--Instantiate STD_FIFO for Writing Serial Data
STD_FIFO_W : STD_FIFO
GENERIC MAP
(
    DATA_WIDTH => 8, -- Width of FIFO
    FIFO_DEPTH => 512, -- Depth of FIFO
    FIFO_ADDR_LEN => 9 -- Required number of bits to represent FIFO_Depth
)
PORT MAP(
    CLK => clk,
    RST => rst,
    WriteEn => STD_FIFO_W_WriteEn,
    DataIn => STD_FIFO_W_DataIn,
    ReadEn => STD_FIFO_W_ReadEn,
    DataOut => STD_FIFO_W_DataOut,
    Empty => STD_FIFO_W_Empty,
    Full => STD_FIFO_W_Full
);
--Instantiate Bus Interface
Bus_Int1 : Bus_Int PORT MAP(
    clk => clk,

```

```

rst => rst,
DataIn => Bus_Int1_DataIn,
DataOut => Bus_Int1_DataOut,
AddrIn => Bus_Int1_AddrIn,
WE => Bus_Int1_WE,
RE => Bus_Int1_RE,
Busy => Bus_Int1_Busy,
Data => Data,
Addr => Addr,
Xrqst => Xrqst,
XDat => XDat,
YDat => YDat,
BusRqst => BusRqst,
BusCtrl => BusCtrl
);
----Registers
Reg_Proc : PROCESS
BEGIN
    WAIT UNTIL clk'event AND clk = '1';
    IF rst = '0' THEN
        busy_reg_o <= '0';
        busy2_reg_o <= '0';
        rx_reg_o <= '0';
        tx_reg_o <= '0';
        temp_data_reg_o <= (OTHERS => '0');
        temp2_reg_o <= (OTHERS => '0');
        Temp_Addr_High_reg_o <= (OTHERS => '0');
        Temp_Addr_Low_reg_o <= (OTHERS => '0');
        Temp_Data_High_reg_o <= (OTHERS => '0');
        Temp_Data_Low_reg_o <= (OTHERS => '0');
        Temp_Cmd_reg_o <= (OTHERS => '0');
    ELSE
        IF (LD_busy = '1') THEN
            busy_reg_o <= busy;
        END IF;
        IF (LD_busy2 = '1') THEN
            busy2_reg_o <= busy2;
        END IF;
        IF (LD_rx = '1') THEN
            rx_reg_o <= rx;
        END IF;
        IF (LD_tx = '1') THEN
            tx_reg_o <= tx;
        END IF;
        IF (LD_temp_data = '1') THEN
            temp_data_reg_o <= temp_data;
        END IF;
    END IF;
END PROCESS;

```

```

        END IF;
        IF (LD_temp2 = '1') THEN
            temp2_reg_o <= temp2;
        END IF;
        IF (LD_Temp_Addr_High = '1') THEN
            Temp_Addr_High_reg_o <= Temp_Addr_High;
        END IF;
        IF (LD_Temp_Addr_Low = '1') THEN
            Temp_Addr_Low_reg_o <= Temp_Addr_Low;
        END IF;
        IF (LD_Temp_Data_High = '1') THEN
            Temp_Data_High_reg_o <= Temp_Data_High;
        END IF;
        IF (LD_Temp_Data_Low = '1') THEN
            Temp_Data_Low_reg_o <= Temp_Data_Low;
        END IF;
        IF (LD_Temp_Cmd = '1') THEN
            Temp_Cmd_reg_o <= Temp_Cmd;
        END IF;
    END IF;
END PROCESS;
----End Registers
----Next State Logic for Serial Interface Read
NSL_RS232_R : PROCESS (CS_RS232_R, rs232_rcv_s, rx_done, STD_FIFO_R_Full,
temp_rcv)
BEGIN
    ----Default States to remove latches
    busy <= '0';
    rx <= '0';
    NS_RS232_R <= S0;
    LD_busy <= '0';
    LD_rx <= '0';
    --Signals for FIFO
    STD_FIFO_R_WriteEn <= '0';
    STD_FIFO_R_DataIn <= (OTHERS => '0');
    CASE CS_RS232_R IS
        WHEN S0 => -- Waits until data is detected on rs232_rcv_s.
            IF (rs232_rcv_s = '1') THEN
                NS_RS232_R <= S0;
            ELSE
                NS_RS232_R <= S1;
            END IF;
            busy <= '0'; -- the busy signal stops the baud generator
            rx <= '0'; -- signals to stop reading data
            LD_rx <= '1';
            LD_busy <= '1';
    
```



```

        WHEN S1 => -- Starts the baud rate generator and reading
            NS_RS232_R <= S2;
            busy <= '1'; -- the busy signal starts the baud generator
            rx <= '1'; -- signals to start reading data
            LD_rx <= '1';
            LD_busy <= '1';
        WHEN S2 => -- Waits until all data is read
            IF (rx_done = '0') THEN
                NS_RS232_R <= S2;
            ELSE
                NS_RS232_R <= S3;
            END IF;
        WHEN S3 =>
            IF (STD_FIFO_R_Full = '0') THEN
                STD_FIFO_R_DataIn <= temp_rcv;
                STD_FIFO_R_WriteEn <= '1';
            END IF;
            NS_RS232_R <= S0;
        WHEN OTHERS =>
            NS_RS232_R <= S0;
    END CASE;
END PROCESS;
----End Next State Logic for Serial Interface Read
----Next State Logic for Serial Interface Write
NSL_RS232_W : PROCESS (CS_RS232_W, tx_done, STD_FIFO_W_Empty,
STD_FIFO_W_DataOut)
BEGIN
    ----Default States to remove latches
    tx <= '0';
    NS_RS232_W <= S0;
    temp2 <= (OTHERS => '0');
    LD_tx <= '0';
    LD_temp2 <= '0';
    Busy2 <= '0';
    LD_Busy2 <= '0';
    --Signals for FIFO
    STD_FIFO_W_ReadEn <= '0';
    CASE CS_RS232_W IS
        WHEN S0 =>
            IF (STD_FIFO_W_Empty = '1') THEN
                NS_RS232_W <= S0;
            ELSE
                NS_RS232_W <= S1;
                STD_FIFO_W_ReadEn <= '1';
            END IF;
            busy2 <= '0'; -- the busy signal stops the baud generator

```

```

        tx <= '0'; -- signals to stop sending data
        LD_tx <= '1';
        LD_busy2 <= '1';
    WHEN S1 =>
        temp2 <= STD_FIFO_W_DataOut;
        LD_temp2 <= '1';
        NS_RS232_W <= S2;
    WHEN S2 =>
        busy2 <= '1'; -- the busy signal starts the baud generator
        tx <= '1'; -- signals to start sending data
        LD_tx <= '1';
        LD_busy2 <= '1';
        NS_RS232_W <= S3;
    WHEN S3 =>
        IF (tx_done = '0') THEN
            NS_RS232_W <= S3;
        ELSE
            NS_RS232_W <= S0;
        END IF;
    WHEN OTHERS =>
        NS_RS232_W <= S0;
    END CASE;
END PROCESS;
---End Next State Logic for Serial Interface Write
---Next State Logic for FIFO to Bus
NSL_FIFO_Bus : PROCESS (CS_FIFO_Bus, STD_FIFO_R_Empty, Temp_Cmd_reg_o,
Bus_Int1_Busy, STD_FIFO_R_DataOut, Temp_Addr_High_reg_o, Temp_Addr_Low_reg_o,
Temp_Data_High_reg_o, Temp_Data_Low_reg_o, Bus_Int1_DataOut, temp_data_reg_o)
BEGIN
    ---Default States to remove latches
    NS_FIFO_Bus <= S0;
    Temp_Cmd <= (OTHERS => '0');
    LD_Temp_Cmd <= '0';
    Temp_Addr_High <= (OTHERS => '0');
    LD_Temp_Addr_High <= '0';
    Temp_Addr_Low <= (OTHERS => '0');
    LD_Temp_Addr_Low <= '0';
    Bus_Int1_AddrIn <= (OTHERS => '0');
    Bus_Int1_RE <= '0';
    Bus_Int1_DataIn <= (OTHERS => '0');
    Bus_Int1_WE <= '0';
    Temp_Data <= (OTHERS => '0');
    LD_Temp_Data <= '0';
    Temp_Data_High <= (OTHERS => '0');
    LD_Temp_Data_High <= '0';
    Temp_Data_High <= (OTHERS => '0');

```

```

Temp_Data_Low <= (OTHERS => '0');
LD_Temp_Data_Low <= '0';
--Signals for FIFO
STD_FIFO_R_ReadEn <= '0';
STD_FIFO_W_DataIn <= (OTHERS => '0');
STD_FIFO_W_WriteEn <= '0';
CASE CS_FIFO_Bus IS
    WHEN S0 =>
        IF (STD_FIFO_R_Empty = '1') THEN --Check to see if
commands are in queue
            NS_FIFO_Bus <= S0;
        ELSE
            NS_FIFO_Bus <= S1;
            STD_FIFO_R_ReadEn <= '1'; --Assert Read Signal for
FIFO
        END IF;
    WHEN S1 => --Read Command from FIFO
        Temp_Cmd <= STD_FIFO_R_DataOut;
        LD_Temp_Cmd <= '1';
        NS_FIFO_Bus <= S2;
    WHEN S2 =>
        IF (STD_FIFO_R_Empty = '1') THEN --Check to see if
commands are in queue
            NS_FIFO_Bus <= S2;
        ELSE
            NS_FIFO_Bus <= S3;
            STD_FIFO_R_ReadEn <= '1'; --Assert Read Signal for
FIFO
        END IF;
    WHEN S3 => --Read Address_High from FIFO
        Temp_Addr_High <= STD_FIFO_R_DataOut;
        LD_Temp_Addr_High <= '1';
        NS_FIFO_Bus <= S4;
    WHEN S4 =>
        IF (STD_FIFO_R_Empty = '1') THEN --Check to see if
commands are in queue
            NS_FIFO_Bus <= S4;
        ELSE
            NS_FIFO_Bus <= S5;
            STD_FIFO_R_ReadEn <= '1';
        END IF;
    WHEN S5 => --Read Address_Low from FIFO
        Temp_Addr_Low <= STD_FIFO_R_DataOut;
        LD_Temp_Addr_Low <= '1';
        NS_FIFO_Bus <= S6;
    WHEN S6 =>

```

```

        IF (Temp_Cmd_reg_o = X"70") THEN --Check Cmd (Read)
            NS_FIFO_Bus <= S7;
        ELSIF (Temp_Cmd_reg_o = X"71") THEN --Check Cmd (Write)
            NS_FIFO_Bus <= S15;
        ELSE --Check Cmd (Invalid Data)
            NS_FIFO_Bus <= S0;
        END IF;
        --Read from Bus and Write to RS232 FIFO
    WHEN S7 =>
        Bus_Int1_AddrIn(15 DOWNT0 8) <= Temp_Addr_High_reg_o; -
-Send Address to Bus Interface for Read
        Bus_Int1_AddrIn(7 DOWNT0 0) <= Temp_Addr_Low_reg_o; --
Send Address to Bus Interface for Read
        Bus_Int1_RE <= '1'; --Read Flag to Bus Interface
        NS_FIFO_Bus <= S8;
    WHEN S8 => --Wait until data is ready
        IF (Bus_Int1_Busy = '1') THEN
            NS_FIFO_Bus <= S8;
        ELSE
            NS_FIFO_Bus <= S9;
        END IF;
        Temp_Data <= Bus_Int1_DataOut;
        LD_Temp_Data <= '1';
    WHEN S9 => --Form First byte of Packet(Start Deliminators)
        STD_FIFO_W_DataIn <= X"7E";
        STD_FIFO_W_WriteEn <= '1';
        NS_FIFO_Bus <= S10;
    WHEN S10 => --Form Second byte of Packet(Address_High)
        STD_FIFO_W_DataIn <= Temp_Addr_High_reg_o;
        STD_FIFO_W_WriteEn <= '1';
        NS_FIFO_Bus <= S11;
    WHEN S11 => --Form Third byte of Packet(Address_Low)
        STD_FIFO_W_DataIn <= Temp_Addr_Low_reg_o;
        STD_FIFO_W_WriteEn <= '1';
        NS_FIFO_Bus <= S12;
    WHEN S12 => --Form Fourth byte of Packet(Data_High)
        STD_FIFO_W_DataIn <= Temp_Data_reg_o(15 DOWNT0 8);
        STD_FIFO_W_WriteEn <= '1';
        NS_FIFO_Bus <= S13;
    WHEN S13 => --Form Fifth byte of Packet(Data_Low)
        STD_FIFO_W_DataIn <= Temp_Data_reg_o(7 DOWNT0 0);
        STD_FIFO_W_WriteEn <= '1';
        NS_FIFO_Bus <= S0;
        --End Read from Bus and Write to RS232 FIFO
        --Write to Bus from RS232 FIFO
    WHEN S15 =>

```

```

IF (STD_FIFO_R_Empty = '1') THEN --Check to see if
commands are in queue
    NS_FIFO_Bus <= S15;
ELSE
    NS_FIFO_Bus <= S16;
    STD_FIFO_R_ReadEn <= '1';
END IF;
WHEN S16 => --Read Data_High from FIFO
    Temp_Data_High <= STD_FIFO_R_DataOut;
    LD_Temp_Data_High <= '1';
    NS_FIFO_Bus <= S17;
WHEN S17 =>
IF (STD_FIFO_R_Empty = '1') THEN --Check to see if
commands are in queue
    NS_FIFO_Bus <= S17;
ELSE
    NS_FIFO_Bus <= S18;
    STD_FIFO_R_ReadEn <= '1';
END IF;
WHEN S18 => --Read Data_Low from FIFO
    Temp_Data_Low <= STD_FIFO_R_DataOut;
    LD_Temp_Data_Low <= '1';
    NS_FIFO_Bus <= S19;
WHEN S19 =>
    Bus_Int1_AddrIn(15 DOWNT0 8) <= Temp_Addr_High_reg_o; -
-Send Address to Bus Interface for Write
    Bus_Int1_AddrIn(7 DOWNT0 0) <= Temp_Addr_Low_reg_o; --
Send Address to Bus Interface for Write
    Bus_Int1_DataIn(15 DOWNT0 8) <= Temp_Data_High_reg_o; --
Send Data to Bus Interface for Write
    Bus_Int1_DataIn(7 DOWNT0 0) <= Temp_Data_Low_reg_o; --
Send Data to Bus Interface for Write
    Bus_Int1_WE <= '1'; --Write Flag to Bus Interface
    NS_FIFO_Bus <= S20;
WHEN S20 => --Wait until data is ready
    IF (Bus_Int1_Busy = '1') THEN
        NS_FIFO_Bus <= S20;
    ELSE
        NS_FIFO_Bus <= S0;
    END IF;
    --End Write to Bus from RS232 FIFO
WHEN OTHERS =>
    NS_FIFO_Bus <= S0;
END CASE;
END PROCESS;
----End Next State Logic for FIFO to Bus

```

```

----UART Clock Divider
UART_Clk : PROCESS
BEGIN
    WAIT UNTIL clk'event AND clk = '1';
    --Synchronize async signal
    rs232_rcv_t <= rs232_rcv; --Synchro1 rs232_rcv
    rs232_rcv_s <= rs232_rcv_t; --Synchro2 rs232_rcv
    IF (rst = '0' OR (busy_reg_o = '0' AND busy2_reg_o = '0')) THEN
        uartclk <= '0';
        i <= CONV_STD_LOGIC_VECTOR(CN, 16);
    ELSIF (i = CM) THEN
        uartclk <= '1';
        i <= X"0000";
    ELSE
        i <= i + 1;
        uartclk <= '0';
    END IF;
END PROCESS;
---- End UART Clock Divider
----UART_Read
UART_Read : PROCESS
BEGIN
    WAIT UNTIL clk'event AND clk = '1';
    IF rst = '0' OR rx_reg_o = '0' THEN
        temp_rcv <= x"00";
        j <= x"0000";
        rx_done <= '0';
    ELSIF rx_reg_o = '1' THEN
        IF uartclk = '1' THEN
            IF j < X"09" THEN
                temp_rcv(7) <= rs232_rcv_s;
                temp_rcv(6 DOWNT0 0) <= temp_rcv(7 DOWNT0 1);
                j <= j + 1;
                rx_done <= '0';
            ELSE
                j <= X"0000";
                rx_done <= '1';
            END IF;
        ELSE
            rx_done <= '0';
        END IF;
    END IF;
END PROCESS;
----End UART_Read
----UART_Xmit
UART_Xmit : PROCESS

```

```

BEGIN
    WAIT UNTIL clk'event AND clk = '1';
    IF (rst = '0' OR tx_reg_o = '0') THEN
        rs232_xmt <= '1';
        tx_done <= '0';
        u <= 0;
        --structure the 10-bit frame to be sent
        txbuff(9) <= '1'; --stopbit 2
        txbuff(8 DOWNT0 1) <= temp2_reg_o;
        txbuff(0) <= '0'; --startbit 2
    ELSE
        IF uartclk = '1' THEN
            IF (u < 10) THEN
                rs232_xmt <= txbuff(0);
                txbuff(8 DOWNT0 0) <= txbuff(9 DOWNT0 1);
                tx_done <= '0';
                u <= u + 1;
            ELSE
                u <= 0;
                tx_done <= '1';
            END IF;
        END IF;
    END IF;
END PROCESS;
----End UART_Xmit
----State Sync
sync_States : PROCESS
BEGIN
    WAIT UNTIL clk'event AND clk = '1';
    IF rst = '0' THEN
        CS_RS232_R <= S0;
        CS_RS232_W <= S0;
        CS_FIFO_Bus <= S0;
    ELSE
        CS_RS232_R <= NS_RS232_R;
        CS_RS232_W <= NS_RS232_W;
        CS_FIFO_Bus <= NS_FIFO_Bus;
    END IF;
END PROCESS;
----End State Sync
END Behavioral;
--#####LED
Controller#####
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;

```

```

USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE ieee.numeric_std.ALL;
ENTITY LED_Ctrl IS
    GENERIC (
        Addr_LED_En : STD_LOGIC_VECTOR(15 DOWNT0 0) := X"0100"; --Enable
LED Outputs (LSB)
        Addr_LED_Freq : STD_LOGIC_VECTOR(15 DOWNT0 0) := X"0101"; --LED
Blink Frequency
        Addr_LED_PW : STD_LOGIC_VECTOR(15 DOWNT0 0) := X"0102"; --LED
Pulse Width (On-Time)
        Addr_LED1_DC : STD_LOGIC_VECTOR(15 DOWNT0 0) := X"0103"; --
LED1 PWM Duty Cycle
        Addr_LED2_DC : STD_LOGIC_VECTOR(15 DOWNT0 0) := X"0104"; --
LED2 PWM Duty Cycle
        Addr_LED3_DC : STD_LOGIC_VECTOR(15 DOWNT0 0) := X"0105"; --
LED3 PWM Duty Cycle
        Addr_LED4_DC : STD_LOGIC_VECTOR(15 DOWNT0 0) := X"0106"; --
LED4 PWM Duty Cycle
        Addr_LED5_DC : STD_LOGIC_VECTOR(15 DOWNT0 0) := X"0107"; --
LED5 PWM Duty Cycle
        Addr_LED6_DC : STD_LOGIC_VECTOR(15 DOWNT0 0) := X"0108"; --
LED6 PWM Duty Cycle
        Addr_LED7_DC : STD_LOGIC_VECTOR(15 DOWNT0 0) := X"0109"; --
LED7 PWM Duty Cycle
        Addr_LED8_DC : STD_LOGIC_VECTOR(15 DOWNT0 0) := X"010A" --
LED8 PWM Duty Cycle
    );
    PORT (
        clk : IN STD_LOGIC;
        rst : IN STD_LOGIC;
        Data : INOUT std_logic_vector(15 DOWNT0 0);
        Addr : OUT std_logic_vector(15 DOWNT0 0);
        Xrqst : OUT std_logic;
        XDat : IN std_logic;
        YDat : OUT std_logic;
        BusRqst : OUT std_logic;
        BusCtrl : IN std_logic;
        LED1_Out : OUT STD_LOGIC
    );
END LED_Ctrl;
ARCHITECTURE Behavioral OF LED_Ctrl IS
    TYPE state_type IS (S0, S1, S2, S3, S4, S5, S6, S7, S8, S9, S10, S11, S12, S13, S14,
S15, S16, S17);
    SIGNAL CS_Bus, NS_Bus, CS_Blink, NS_Blink : state_type;
    --declare Std_Counter Component
    COMPONENT Std_Counter IS

```



```

    GENERIC (
        Width : INTEGER --width of counter
    );
    PORT (
        INC, rst, clk : IN std_logic;
        Count : OUT STD_LOGIC_VECTOR(Width - 1 DOWNT0 0));
END COMPONENT;
--Declare PWM
COMPONENT PWM_16b IS
    GENERIC (
        Freq_in : INTEGER; --Clk
        Max_PWM : INTEGER; --PWM Resolution (2^16-1)
        Freq_Sw : INTEGER --Switching Freq
    );
    PORT (
        clk : IN std_logic;
        rst : IN std_logic;
        DC : IN std_logic_vector(15 DOWNT0 0);
        Phase : IN std_logic_vector(15 DOWNT0 0);
        En : IN std_logic;
        PWM_Out : OUT std_logic
    );
END COMPONENT;
--declare Bus Interface
COMPONENT Bus_Int
    PORT (
        clk : IN std_logic;
        rst : IN std_logic;
        DataIn : IN std_logic_vector(15 DOWNT0 0);
        DataOut : OUT std_logic_vector(15 DOWNT0 0);
        AddrIn : IN std_logic_vector(15 DOWNT0 0);
        WE : IN std_logic;
        RE : IN std_logic;
        Busy : OUT std_logic;
        Data : INOUT std_logic_vector(15 DOWNT0 0);
        Addr : OUT std_logic_vector(15 DOWNT0 0);
        Xrqst : OUT std_logic;
        XDat : IN std_logic;
        YDat : OUT std_logic;
        BusRqst : OUT std_logic;
        BusCtrl : IN std_logic
    );
END COMPONENT;
----Signals

```

```

    SIGNAL PWM_En, PWM_Freq, PWM_PW, PWM1_DC, PWM2_DC, PWM3_DC,
    PWM4_DC : STD_LOGIC_VECTOR(15 DOWNT0 0) := X"0000"; --Set initial Duty Cycles to
0
    SIGNAL PWM1_En, PWM2_En, PWM3_En, PWM4_En : STD_LOGIC := '0';
    CONSTANT PWM1_Phase : STD_LOGIC_VECTOR(15 DOWNT0 0) := X"0000"; --
Set Phase shift for PWM1 to 0
    CONSTANT PWM2_Phase : STD_LOGIC_VECTOR(15 DOWNT0 0) := X"0000"; --
Set Phase shift for PWM2 to 0
    CONSTANT PWM3_Phase : STD_LOGIC_VECTOR(15 DOWNT0 0) := X"0000"; --
Set Phase shift for PWM3 to 0
    CONSTANT PWM4_Phase : STD_LOGIC_VECTOR(15 DOWNT0 0) := X"0000"; --
Set Phase shift for PWM4 to 0
    --Max PWM Values
    CONSTANT PWM_Max : std_logic_vector(15 DOWNT0 0) := X"FFFF";
    CONSTANT PWM_Min : std_logic_vector(15 DOWNT0 0) := X"0000";
    --Declare Signals for Bus Interface
    SIGNAL Bus_Int1_WE, Bus_Int1_RE, Bus_Int1_Busy : STD_LOGIC := '0';
    SIGNAL Bus_Int1_DataIn, Bus_Int1_DataOut, Bus_Int1_AddrIn :
STD_LOGIC_VECTOR(15 DOWNT0 0) := (OTHERS => '0');
    SIGNAL Bus_Cnt_rst, Bus_Cnt_INC : STD_LOGIC := '0';
    SIGNAL Bus_Cnt_Out : STD_LOGIC_VECTOR(15 DOWNT0 0) := (OTHERS => '0');
    SIGNAL Delay_Cnt_rst, Delay_Cnt_INC : STD_LOGIC := '0';
    SIGNAL Delay_Cnt_Out : STD_LOGIC_VECTOR(7 DOWNT0 0) := (OTHERS =>
'0');
    --Signals for Registers
    SIGNAL LD_PWM_En, LD_PWM_Freq, LD_PWM_PW, LD_PWM1_DC,
LD_PWM2_DC, LD_PWM3_DC, LD_PWM4_DC : STD_LOGIC := '0';
    --Signals for Clock Divider
    SIGNAL clk_temp : STD_LOGIC_VECTOR(15 DOWNT0 0) := (OTHERS => '0');
    SIGNAL clk_Blink : STD_LOGIC := '0';
    --Signals for blink
    SIGNAL Freq_Cnt_rst, Freq_Cnt_Inc : STD_LOGIC := '0';
    SIGNAL Freq_Cnt_out : STD_LOGIC_VECTOR(15 DOWNT0 0) := (OTHERS => '0');
BEGIN
    --instantiate Bus_Cnt
    Bus_Cnt : Std_Counter
    GENERIC MAP
    (
        Width => 16
    )
    PORT MAP
    (
        clk => clk,
        rst => Bus_Cnt_rst,
        INC => Bus_Cnt_INC,
        Count => Bus_Cnt_Out

```

```

);
--instantiate Delay_Cnt
Delay_Cnt : Std_Counter
GENERIC MAP
(
    Width => 8
)
PORT MAP(
    clk => clk,
    rst => Delay_Cnt_rst,
    INC => Delay_Cnt_INC,
    Count => Delay_Cnt_Out
);
-- Instantiate PWM1
PWM1 : PWM_16b
GENERIC MAP
(
    Freq_in => 24930000,
    Max_PWM => 65535,
    Freq_Sw => 6104
)
PORT MAP(
    clk => clk,
    rst => rst,
    DC => PWM1_DC,
    Phase => PWM1_Phase,
    En => PWM1_En,
    PWM_Out => LED1_Out
);

--Instantiate Bus Interface
Bus_Int1 : Bus_Int PORT MAP(
    clk => clk,
    rst => rst,
    DataIn => Bus_Int1_DataIn,
    DataOut => Bus_Int1_DataOut,
    AddrIn => Bus_Int1_AddrIn,
    WE => Bus_Int1_WE,
    RE => Bus_Int1_RE,
    Busy => Bus_Int1_Busy,
    Data => Data,
    Addr => Addr,
    Xrqst => Xrqst,
    XDat => XDat,
    YDat => YDat,
    BusRqst => BusRqst,

```

```

        BusCtrl => BusCtrl
    );
    --instantiate Freq_Cnt
    Freq_Cnt : Std_Counter
    GENERIC MAP
    (
        Width => 16
    )
    PORT MAP(
        clk => clk_Blink,
        rst => Freq_Cnt_rst,
        INC => Freq_Cnt_INC,
        Count => Freq_Cnt_Out
    );
    ----Registers
    Reg_Proc : PROCESS
    BEGIN
        WAIT UNTIL clk'event AND clk = '1';
        IF rst = '0' THEN
            PWM1_DC <= (OTHERS => '0');
            PWM2_DC <= (OTHERS => '0');
            PWM3_DC <= (OTHERS => '0');
            PWM4_DC <= (OTHERS => '0');
            PWM_Freq <= (OTHERS => '0');
            PWM_PW <= (OTHERS => '0');
            PWM_En <= (OTHERS => '0');
        ELSE
            IF (LD_PWM1_DC = '1') THEN
                PWM1_DC <= Bus_Int1_DataOut;
            END IF;
            IF (LD_PWM2_DC = '1') THEN
                PWM2_DC <= Bus_Int1_DataOut;
            END IF;
            IF (LD_PWM3_DC = '1') THEN
                PWM3_DC <= Bus_Int1_DataOut;
            END IF;
            IF (LD_PWM4_DC = '1') THEN
                PWM4_DC <= Bus_Int1_DataOut;
            END IF;
            IF (LD_PWM_Freq = '1') THEN
                PWM_Freq <= Bus_Int1_DataOut;
            END IF;
            IF (LD_PWM_PW = '1') THEN
                PWM_PW <= Bus_Int1_DataOut;
            END IF;
            IF (LD_PWM_En = '1') THEN

```

```

        PWM_En <= Bus_Int1_DataOut;
    END IF;
END IF;
END PROCESS;
----End Registers
----Next State Logic for Bus Interface
NSL_Bus : PROCESS (CS_Bus, Bus_Cnt_Out, Bus_Int1_Busy, Delay_Cnt_Out)
BEGIN
    ----Default States to remove latches
    NS_Bus <= S0;
    Bus_Int1_AddrIn <= (OTHERS => '0');
    Bus_Int1_RE <= '0';
    Bus_Int1_DataIn <= (OTHERS => '0');
    Bus_Int1_WE <= '0';
    Bus_Cnt_rst <= '1';
    Bus_Cnt_INC <= '0';
    LD_PWM1_DC <= '0';
    LD_PWM2_DC <= '0';
    LD_PWM3_DC <= '0';
    LD_PWM4_DC <= '0';
    LD_PWM_Freq <= '0';
    LD_PWM_PW <= '0';
    LD_PWM_En <= '0';
    Delay_Cnt_INC <= '0';
    Delay_Cnt_rst <= '1';

    CASE CS_Bus IS
        WHEN S0 =>
            Bus_Cnt_rst <= '0'; -- Reset Bus Counter
            Delay_Cnt_rst <= '0'; -- Reset Delay Counter
            NS_Bus <= S1;
        WHEN S1 => --Initial Delay count for sync
            IF (Delay_Cnt_Out < 40) THEN
                NS_Bus <= S1;
            ELSE
                NS_Bus <= S2;
            END IF;
            Delay_Cnt_INC <= '1';
        WHEN S2 => --Wait (2^12-34) Clk Cycles for 1x per fs
            IF (Bus_Cnt_Out < 4062) THEN
                NS_Bus <= S2;
            ELSE
                NS_Bus <= S3;
            END IF;
            Bus_Cnt_INC <= '1';
            --Read Command Data from Bus

```

```

    WHEN S3 =>
        IF (Bus_Int1_Busy = '1') THEN
            NS_Bus <= S3;
        ELSE
            NS_Bus <= S4;
        END IF;
        Bus_Cnt_rst <= '0'; -- Reset Bus Counter
    WHEN S4 =>
        Bus_Int1_AddrIn <= Addr_LED_En; --Read Data from LED_En

Register

        Bus_Int1_RE <= '1';
        NS_Bus <= S5;
    WHEN S5 =>
        IF (Bus_Int1_Busy = '1') THEN
            NS_Bus <= S5;
        ELSE
            LD_PWM_En <= '1';
            NS_Bus <= S6;
        END IF;
    WHEN S6 =>
        Bus_Int1_AddrIn <= Addr_LED_Freq; --Read Data from LED

Freq Register

        Bus_Int1_RE <= '1';
        NS_Bus <= S7;
    WHEN S7 =>
        IF (Bus_Int1_Busy = '1') THEN
            NS_Bus <= S7;
        ELSE
            LD_PWM_Freq <= '1';
            NS_Bus <= S8;
        END IF;
    WHEN S8 =>
        Bus_Int1_AddrIn <= Addr_LED_PW; --Read Data from LED

PulseWidth Register

        Bus_Int1_RE <= '1';
        NS_Bus <= S9;
    WHEN S9 =>
        IF (Bus_Int1_Busy = '1') THEN
            NS_Bus <= S9;
        ELSE
            LD_PWM_PW <= '1';
            NS_Bus <= S10;
        END IF;
    WHEN S10 =>
        Bus_Int1_AddrIn <= Addr_LED1_DC; --Read Data from

LED1_DC Register

```

```

        Bus_Int1_RE <= '1';
        NS_Bus <= S11;
    WHEN S11 =>
        IF (Bus_Int1_Busy = '1') THEN
            NS_Bus <= S11;
        ELSE
            LD_PWM1_DC <= '1';
            NS_Bus <= S12;
        END IF;
    WHEN S12 =>
        Bus_Int1_AddrIn <= Addr_LED2_DC; --Read Data from
LED2_DC Register
        Bus_Int1_RE <= '1';
        NS_Bus <= S13;
    WHEN S13 =>
        IF (Bus_Int1_Busy = '1') THEN
            NS_Bus <= S13;
        ELSE
            LD_PWM2_DC <= '1';
            NS_Bus <= S14;
        END IF;
    WHEN S14 =>
        Bus_Int1_AddrIn <= Addr_LED3_DC; --Read Data from
LED3_DC Register
        Bus_Int1_RE <= '1';
        NS_Bus <= S15;
    WHEN S15 =>
        IF (Bus_Int1_Busy = '1') THEN
            NS_Bus <= S15;
        ELSE
            LD_PWM3_DC <= '1';
            NS_Bus <= S16;
        END IF;
    WHEN S16 =>
        Bus_Int1_AddrIn <= Addr_LED4_DC; --Read Data from
LED4_DC Register
        Bus_Int1_RE <= '1';
        NS_Bus <= S17;
    WHEN S17 =>
        IF (Bus_Int1_Busy = '1') THEN
            NS_Bus <= S17;
        ELSE
            LD_PWM4_DC <= '1';
            NS_Bus <= S2;
        END IF;
    WHEN OTHERS =>

```

```

        NS_Bus <= S0;
    END CASE;
END PROCESS;
----End Next State Logic for Bus Interface
----Next State Logic for Blink Update
NSL_Blink : PROCESS (CS_Blink, Freq_Cnt_Out, PWM_Freq, PWM_PW)
BEGIN
    ----Default States to remove latches
    NS_Blink <= S0;
    Freq_Cnt_INC <= '0';
    Freq_Cnt_rst <= '1';
    PWM1_En <= '0';
    PWM2_En <= '0';
    PWM3_En <= '0';
    PWM4_En <= '0';
    CASE CS_Blink IS
        WHEN S0 =>
            Freq_Cnt_rst <= '0'; -- Reset Period Counter
            NS_Blink <= S1;
        WHEN S1 => -- Counter for Pulse Width
            IF (Freq_Cnt_Out < PWM_PW) THEN
                NS_Blink <= S1;
            ELSE
                NS_Blink <= S2;
            END IF;
            Freq_Cnt_INC <= '1';
            PWM1_En <= '1';
            PWM2_En <= '1';
            PWM3_En <= '1';
            PWM4_En <= '1';
        WHEN S2 => --Counter for Period
            IF (Freq_Cnt_Out < PWM_Freq) THEN
                NS_Blink <= S2;
            ELSE
                NS_Blink <= S0;
            END IF;
            Freq_Cnt_INC <= '1';
        WHEN OTHERS =>
            NS_Blink <= S0;
    END CASE;
END PROCESS;
----End Next State Logic for Blink Update
----State Sync
sync_States : PROCESS
BEGIN
    WAIT UNTIL clk'event AND clk = '1';

```



```

        IF rst = '0' THEN
            CS_Bus <= S0;
        ELSE
            CS_Bus <= NS_Bus;
        END IF;
    END PROCESS;
----End State Sync
----State Sync for Blink
sync_Blink : PROCESS
BEGIN
    WAIT UNTIL clk_Blink'event AND clk_Blink = '1';
    IF rst = '0' THEN
        CS_Blink <= S0;
    ELSE
        CS_Blink <= NS_Blink;
    END IF;
END PROCESS;
----End State Sync
-- Clock Divider for LED_Blink
Clk_Div_Blink : PROCESS
BEGIN
    WAIT UNTIL clk'event AND clk = '1';
    clk_temp <= clk_temp + 1;
    clk_Blink <= clk_temp(9);
END PROCESS;
END Behavioral;
-----#Bus Interface
Top#-----#
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE ieee.numeric_std.ALL;
LIBRARY lattice;
USE lattice.components.ALL;
LIBRARY machxo2;
USE machxo2.ALL;

ENTITY Bus_Interface_Top IS
    PORT (
        --RESETh : in STD_LOGIC;-- Global Reset
        -- RS232 Communication
        Usr_RX : IN STD_LOGIC; -- Serial In for User Control
        Usr_TX : OUT STD_LOGIC; -- Serial Out for User Control
        -- Board LEDs
        LED_1 : OUT STD_LOGIC -- Board LED
    );

```

```

END Bus_Interface_Top;
ARCHITECTURE Behavioral OF Bus_Interface_Top IS
  -- Declare Internal Oscillator
  COMPONENT OSCH
    GENERIC (NOM_FREQ : STRING := "8.31");
    PORT (
      STDBY : IN std_logic;
      OSC : OUT std_logic;
      SEDSTDBY : OUT std_logic
    );
  END COMPONENT;
  -- Declare PLL
  COMPONENT PLL_Clk
    PORT (
      ClkI : IN std_logic;
      ClkOP : OUT std_logic;
      Lock : OUT std_logic
    );
  END COMPONENT;
  -- Declare Bus_Master
  COMPONENT Bus_Master
    PORT (
      clk : IN std_logic;
      rst : IN std_logic;
      Data : INOUT std_logic_vector(15 DOWNTO 0);
      Addr : IN std_logic_vector(15 DOWNTO 0);
      Xrqst : IN std_logic;
      XDat : OUT std_logic;
      YDat : IN std_logic;
      BusRqst : IN std_logic_vector(9 DOWNTO 0);
      BusCtrl : OUT std_logic_vector(9 DOWNTO 0)
    );
  END COMPONENT;
  -- Declare RS232_Usr_Int
  COMPONENT RS232_Usr_Int
    GENERIC (
      Baud : INTEGER; -- Baud Rate
      clk_in : INTEGER -- Input Clk
    );
    PORT (
      clk : IN std_logic;
      rst : IN std_logic;
      rs232_rcv : IN std_logic;
      rs232_xmt : OUT std_logic;
      Data : INOUT std_logic_vector(15 DOWNTO 0);
      Addr : OUT std_logic_vector(15 DOWNTO 0);

```

```

        Xrqst : OUT std_logic;
        XDat : IN std_logic;
        YDat : OUT std_logic;
        BusRqst : OUT std_logic;
        BusCtrl : IN std_logic
    );
END COMPONENT;
-- Declare LED_Ctrl
COMPONENT LED_Ctrl IS
    PORT (
        clk : IN STD_LOGIC;
        rst : IN STD_LOGIC;
        Data : INOUT std_logic_vector(15 DOWNT0 0);
        Addr : OUT std_logic_vector(15 DOWNT0 0);
        Xrqst : OUT std_logic;
        XDat : IN std_logic;
        YDat : OUT std_logic;
        BusRqst : OUT std_logic;
        BusCtrl : IN std_logic;
        LED1_Out : OUT STD_LOGIC
    );
END COMPONENT;
-- Declare Std_Counter Component
COMPONENT Std_Counter IS
    GENERIC (
        Width : INTEGER -- width of counter
    );
    PORT (
        INC, rst, clk : IN std_logic;
        Count : OUT STD_LOGIC_VECTOR(Width - 1 DOWNT0 0));
END COMPONENT;
----Signals
-- Declare Signals for Bus Interface
SIGNAL Bus_Int1_WE, Bus_Int1_RE, Bus_Int1_Busy : STD_LOGIC := '0';
SIGNAL Bus_Int1_DataIn, Bus_Int1_DataOut, Bus_Int1_AddrIn :
STD_LOGIC_VECTOR(15 DOWNT0 0) := (OTHERS => '0');
-- Inputs
SIGNAL Addr : STD_LOGIC_VECTOR(15 DOWNT0 0) := (OTHERS => '0');
SIGNAL Xrqst : STD_LOGIC := '0';
SIGNAL YDat : STD_LOGIC := '0';
SIGNAL BusRqst : STD_LOGIC_VECTOR(9 DOWNT0 0) := (OTHERS => '0');
SIGNAL Data : STD_LOGIC_VECTOR(15 DOWNT0 0) := (OTHERS => '0');
SIGNAL XDat : STD_LOGIC := '0';
SIGNAL BusCtrl : STD_LOGIC_VECTOR(9 DOWNT0 0) := (OTHERS => '0');
-- Internal Clock
SIGNAL OSC_Stdbby, OSC_Out, OSC_SEDSTDBY, clk : STD_LOGIC := '0';

```

```

-- Reset
SIGNAL PLL_Lock, System_rst : STD_LOGIC := '0';
SIGNAL Reset_Cnt_INC, Reset_Cnt_rst : STD_LOGIC := '0';
SIGNAL Reset_Cnt_out : STD_LOGIC_VECTOR(7 DOWNT0 0) := (OTHERS => '0');
-- For inverting LED Outputs
SIGNAL LED_1n : STD_LOGIC := '0';
BEGIN
-- Instantiate Internal Oscillator
Int_OSC : OSCH PORT MAP(
    STDBY => OSC_Stdbby,
    OSC => OSC_Out,
    SEDSTDBY => OSC_SEDSTDBY
);
-- Instantiate PLL
PLL_1 : PLL_Clk PORT MAP(
    ClkI => OSC_Out,
    ClkOP => clk,
    Lock => Pll_Lock
);
-- Instantiate Bus_Master
BM : Bus_Master PORT MAP(
    clk => clk,
    rst => System_rst,
    Data => Data,
    Addr => Addr,
    Xrqst => Xrqst,
    XDat => XDat,
    YDat => YDat,
    BusRqst => BusRqst,
    BusCtrl => BusCtrl
);
-- Instantiate RS232_Usr_Int
RS232_Usr : RS232_Usr_Int
    GENERIC MAP
    (
        Baud => 9600, -- Baud Rate
        Clk_In => 24930000 -- Input Clk
    )
    PORT MAP(
        clk => clk,
        rst => System_rst,
        rs232_rcv => Usr_RX,
        rs232_xmt => Usr_TX,
        Data => Data,
        Addr => Addr,
        Xrqst => Xrqst,

```

```

        XDat => XDat,
        YDat => YDat,
        BusRqst => BusRqst(1),
        BusCtrl => BusCtrl(1)
    );
-- Instantiate LED_Ctrl
LED_Ctrl1 : LED_Ctrl PORT MAP(
    clk => clk,
    rst => System_rst,
    Data => Data,
    Addr => Addr,
    Xrqst => Xrqst,
    XDat => XDat,
    YDat => YDat,
    BusRqst => BusRqst(0),
    BusCtrl => BusCtrl(0),
    LED1_Out => LED_1n
);
-- Instantiate Reset_Cnt_8
Reset_Cnt : Std_Counter
GENERIC MAP
(
    Width => 8
)
PORT MAP(
    clk => OSC_Out,
    rst => Reset_Cnt_rst,
    INC => Reset_Cnt_INC,
    Count => Reset_Cnt_Out
);
-- Oscillator
OSC_Stdbby <= '0';
-- Tie unused ports to '0'
BusRqst(9 DOWNT0 2) <= (OTHERS => '0');
-- Reset Block1
Reset_Blkl : PROCESS
BEGIN
    WAIT UNTIL OSC_Out'event AND OSC_Out = '1';
    IF (PLL_Lock = '0') THEN
        Reset_Cnt_rst <= '0';
    ELSE
        Reset_Cnt_rst <= '1';
    END IF;
END PROCESS;
-- Reset Block
Reset_Blkl : PROCESS

```

```

BEGIN
    WAIT UNTIL OSC_Out'event AND OSC_Out = '1';
    IF (Reset_Cnt_out < X"7F") THEN
        System_rst <= '0';
        Reset_Cnt_Inc <= '1';
    ELSE
        System_rst <= '1';
        Reset_Cnt_Inc <= '0';
    END IF;
END PROCESS;
-- LED Invert due to Active Low Configuration on Dev Board
LED_Invert : PROCESS
BEGIN
    LED_1 <= NOT(LED_1n);
END PROCESS;
END Behavioral;
--#####Hardware Authentication
Module#####
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;
USE IEEE.std_logic_unsigned.ALL;
USE IEEE.numeric_std.ALL;
LIBRARY machxo2;
USE machxo2.ALL;
LIBRARY lattice;
USE lattice.components.ALL;
ENTITY HW_AUTH_MODULE IS
    PORT (
        MISO : IN STD_LOGIC;
        CLK_IN : IN STD_LOGIC;
        CS : OUT STD_LOGIC;
        CLK_OUT : OUT STD_LOGIC;
        MOSI : OUT STD_LOGIC;
        HW_GOOD : OUT STD_LOGIC
    );
END HW_AUTH_MODULE;

ARCHITECTURE Behavior OF HW_AUTH_MODULE IS
    TYPE MACHINE IS (START, SB, OPCODE_H, OPCODE_L, A5, A4, A3, A2, A1, A0,
W, D15, D14, D13, D12, D11, D10, D9, D8, D7, D6, D5, D4, D3, D2, D1, D0,
AUTHENTICATE, STDBY);
    SIGNAL STATE : MACHINE := STDBY;
    SIGNAL SB_VALUE : STD_LOGIC := '1';
    SIGNAL OP_READ : STD_LOGIC_VECTOR(1 DOWNT0 0) := "10";
    SIGNAL EEPROM_addr : STD_LOGIC_VECTOR(5 DOWNT0 0) := "000000";

```

```

SIGNAL DATA_IN : STD_LOGIC_VECTOR(15 DOWNTO 0) := (OTHERS => '0');
SIGNAL DELAY : INTEGER RANGE 0 TO 200_000 := 200_000; --1_500_000
TYPE T_ARRAY IS ARRAY(0 TO 63) OF STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL KEYS : T_ARRAY;
SIGNAL KEY_INDEX : INTEGER RANGE 0 TO 63 := 0;
BEGIN

    HW_AUTH_INTERRUPT : PROCESS (CLK_IN)
    BEGIN
        KEYS(0) <= X"ABBA";
        KEYS(1) <= X"ABED";
        KEYS(2) <= X"BABE";
        KEYS(3) <= X"BADE";
        KEYS(4) <= X"BEAD";
        KEYS(5) <= X"BEEF";
        KEYS(6) <= X"CAFE";
        KEYS(7) <= X"CEDE";
        KEYS(8) <= X"DADA";
        KEYS(9) <= X"DEAD";
        KEYS(10) <= X"DEAF";
        KEYS(11) <= X"DEED";
        KEYS(12) <= X"FACE";
        KEYS(13) <= X"FADE";
        KEYS(14) <= X"FEED";
        KEYS(15) <= X"FEE0";
        KEYS(16) <= X"ABBA";
        KEYS(17) <= X"ABED";
        KEYS(18) <= X"BABE";
        KEYS(19) <= X"BADE";
        KEYS(20) <= X"BEAD";
        KEYS(21) <= X"BEEF";
        KEYS(22) <= X"CAFE";
        KEYS(23) <= X"CEDE";
        KEYS(24) <= X"DADA";
        KEYS(25) <= X"DEAD";
        KEYS(26) <= X"DEAF";
        KEYS(27) <= X"DEED";
        KEYS(28) <= X"FACE";
        KEYS(29) <= X"FADE";
        KEYS(30) <= X"FEED";
        KEYS(31) <= X"FEE0";
        KEYS(32) <= X"ABBA";
        KEYS(33) <= X"ABED";
        KEYS(34) <= X"BABE";
        KEYS(35) <= X"BADE";
        KEYS(36) <= X"BEAD";

```

```

KEYS(37) <= X"BEEF";
KEYS(38) <= X"CAFE";
KEYS(39) <= X"CEDE";
KEYS(40) <= X"DADA";
KEYS(41) <= X"DEAD";
KEYS(42) <= X"DEAF";
KEYS(43) <= X"DEED";
KEYS(44) <= X"FACE";
KEYS(45) <= X"FADE";
KEYS(46) <= X"FEED";
KEYS(47) <= X"FEE0";
KEYS(48) <= X"ABBA";
KEYS(49) <= X"ABED";
KEYS(50) <= X"BABE";
KEYS(51) <= X"BADE";
KEYS(52) <= X"BEAD";
KEYS(53) <= X"BEEF";
KEYS(54) <= X"CAFE";
KEYS(55) <= X"CEDE";
KEYS(56) <= X"DADA";
KEYS(57) <= X"DEAD";
KEYS(58) <= X"DEAF";
KEYS(59) <= X"DEED";
KEYS(60) <= X"FACE";
KEYS(61) <= X"FADE";
KEYS(62) <= X"FEED";
KEYS(63) <= X"FEE0";

```

```

IF (CLK_IN' EVENT) THEN
  IF (CLK_IN = '1') THEN
    CLK_OUT <= '1';
  END IF;

```

```

  IF (CLK_IN = '0') THEN
    CLK_OUT <= '0';

```

```

    CASE STATE IS

```

```

      WHEN START =>
        CS <= '0';
        MOSI <= '0';
        STATE <= SB;

```

```

      WHEN SB =>
        CS <= '1';
        MOSI <= SB_VALUE;
        STATE <= OPCODE_H;

```



```

WHEN OPCODE_H =>
    CS <= '1';
    MOSI <= OP_READ(1);
    STATE <= OPCODE_L;

WHEN OPCODE_L =>
    CS <= '1';
    MOSI <= OP_READ(0);
    STATE <= A5;

WHEN A5 =>
    CS <= '1';
    MOSI <= EEPROM_addr(5);
    STATE <= A4;

WHEN A4 =>
    CS <= '1';
    MOSI <= EEPROM_addr(4);
    STATE <= A3;

WHEN A3 =>
    CS <= '1';
    MOSI <= EEPROM_addr(3);
    STATE <= A2;

WHEN A2 =>
    CS <= '1';
    MOSI <= EEPROM_addr(2);
    STATE <= A1;

WHEN A1 =>
    CS <= '1';
    MOSI <= EEPROM_addr(1);
    STATE <= A0;

WHEN A0 =>
    CS <= '1';
    MOSI <= EEPROM_addr(0);
    STATE <= W;

WHEN W =>
    CS <= '1';
    STATE <= D15;

WHEN D15 =>

```

```

        CS <= '1';
        MOSI <= '0';
        DATA_IN(15) <= MISO;
        STATE <= D14;

    WHEN D14 =>
        CS <= '1';
        MOSI <= '0';
        DATA_IN(14) <= MISO;
        STATE <= D13;

    WHEN D13 =>
        CS <= '1';
        MOSI <= '0';
        DATA_IN(13) <= MISO;
        STATE <= D12;

    WHEN D12 =>
        CS <= '1';
        MOSI <= '0';
        DATA_IN(12) <= MISO;
        STATE <= D11;

    WHEN D11 =>
        CS <= '1';
        MOSI <= '0';
        DATA_IN(11) <= MISO;
        STATE <= D10;

    WHEN D10 =>
        CS <= '1';
        MOSI <= '0';
        DATA_IN(10) <= MISO;
        STATE <= D9;

    WHEN D9 =>
        CS <= '1';
        MOSI <= '0';
        DATA_IN(9) <= MISO;
        STATE <= D8;

    WHEN D8 =>
        CS <= '1';
        MOSI <= '0';
        DATA_IN(8) <= MISO;
        STATE <= D7;

```

```

WHEN D7 =>
    CS <= '1';
    MOSI <= '0';
    DATA_IN(7) <= MISO;
    STATE <= D6;

WHEN D6 =>
    CS <= '1';
    MOSI <= '0';
    DATA_IN(6) <= MISO;
    STATE <= D5;

WHEN D5 =>
    CS <= '1';
    MOSI <= '0';
    DATA_IN(5) <= MISO;
    STATE <= D4;

WHEN D4 =>
    CS <= '1';
    MOSI <= '0';
    DATA_IN(4) <= MISO;
    STATE <= D3;

WHEN D3 =>
    CS <= '1';
    MOSI <= '0';
    DATA_IN(3) <= MISO;
    STATE <= D2;

WHEN D2 =>
    CS <= '1';
    MOSI <= '0';
    DATA_IN(2) <= MISO;
    STATE <= D1;

WHEN D1 =>
    CS <= '1';
    MOSI <= '0';
    DATA_IN(1) <= MISO;
    STATE <= D0;

WHEN D0 =>
    CS <= '1';
    MOSI <= '0';

```

```

        DATA_IN(0) <= MISO;
        STATE <= AUTHENTICATE;

    WHEN AUTHENTICATE =>
        CS <= '0';
        MOSI <= '0';
        IF (DATA_IN = KEYS(KEY_INDEX)) THEN
            HW_GOOD <= '1';
        ELSE
            HW_GOOD <= '0';
        END IF;
        STATE <= STDBY;

    WHEN STDBY =>
        CS <= '0';
        MOSI <= '0';
        IF (DELAY > 0) THEN
            DELAY <= DELAY - 1;
            STATE <= STDBY;
        ELSIF (DELAY = 0) THEN
            EEPROM_addr <= EEPROM_addr +

"000001";

            KEY_INDEX <= KEY_INDEX + 1;
            DELAY <= 200_000;
            STATE <= START;
        ELSE
            DELAY <= 200_000;
            STATE <= STDBY;
        END IF;

    WHEN OTHERS =>
        CS <= '0';
        MOSI <= '0';
        STATE <= STDBY;

    END CASE;
END IF;
END PROCESS;
END Behavior;
--#####Hardware Assisted
Supervisor#####
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
LIBRARY lattice;
USE lattice.components.ALL;
ENTITY HARDWARE_ASSISTED_SUPERVISOR IS

```

```

GENERIC (
    TIMEOUT : INTEGER := 2_000
);
PORT (
    CLK_IN : IN STD_LOGIC;
    --Input Controller 1
    I1_0 : IN STD_LOGIC;
    I1_1 : IN STD_LOGIC;
    I1_2 : IN STD_LOGIC;
    I1_3 : IN STD_LOGIC;
    I1_4 : IN STD_LOGIC;
    I1_5 : IN STD_LOGIC;
    I1_6 : IN STD_LOGIC;
    I1_7 : IN STD_LOGIC;
    I1_8 : IN STD_LOGIC;
    I1_9 : IN STD_LOGIC;
    I1_10 : IN STD_LOGIC;
    I1_11 : IN STD_LOGIC;
    I1_12 : IN STD_LOGIC;
    I1_13 : IN STD_LOGIC;
    I1_14 : IN STD_LOGIC;
    I1_15 : IN STD_LOGIC;
    I1_16 : IN STD_LOGIC;
    I1_17 : IN STD_LOGIC;
    I1_18 : IN STD_LOGIC;
    I1_19 : IN STD_LOGIC;
    I1_20 : IN STD_LOGIC;
    I1_21 : IN STD_LOGIC;
    I1_22 : IN STD_LOGIC;
    I1_23 : IN STD_LOGIC;
    I1_24 : IN STD_LOGIC;
    I1_25 : IN STD_LOGIC;
    I1_26 : IN STD_LOGIC;
    I1_27 : IN STD_LOGIC;
    --Input Controller 2
    I2_0 : IN STD_LOGIC;
    I2_1 : IN STD_LOGIC;
    I2_2 : IN STD_LOGIC;
    I2_3 : IN STD_LOGIC;
    I2_4 : IN STD_LOGIC;
    I2_5 : IN STD_LOGIC;
    I2_6 : IN STD_LOGIC;
    I2_7 : IN STD_LOGIC;
    I2_8 : IN STD_LOGIC;
    I2_9 : IN STD_LOGIC;
    I2_10 : IN STD_LOGIC;

```

```
I2_11 : IN STD_LOGIC;
I2_12 : IN STD_LOGIC;
I2_13 : IN STD_LOGIC;
I2_14 : IN STD_LOGIC;
I2_15 : IN STD_LOGIC;
I2_16 : IN STD_LOGIC;
I2_17 : IN STD_LOGIC;
I2_18 : IN STD_LOGIC;
I2_19 : IN STD_LOGIC;
I2_20 : IN STD_LOGIC;
I2_21 : IN STD_LOGIC;
I2_22 : IN STD_LOGIC;
I2_23 : IN STD_LOGIC;
I2_24 : IN STD_LOGIC;
I2_25 : IN STD_LOGIC;
I2_26 : IN STD_LOGIC;
I2_27 : IN STD_LOGIC;
--Output Controller
O_0 : OUT STD_LOGIC;
O_1 : OUT STD_LOGIC;
O_2 : OUT STD_LOGIC;
O_3 : OUT STD_LOGIC;
O_4 : OUT STD_LOGIC;
O_5 : OUT STD_LOGIC;
O_6 : OUT STD_LOGIC;
O_7 : OUT STD_LOGIC;
O_8 : OUT STD_LOGIC;
O_9 : OUT STD_LOGIC;
O_10 : OUT STD_LOGIC;
O_11 : OUT STD_LOGIC;
O_12 : OUT STD_LOGIC;
O_13 : OUT STD_LOGIC;
O_14 : OUT STD_LOGIC;
O_15 : OUT STD_LOGIC;
O_16 : OUT STD_LOGIC;
O_17 : OUT STD_LOGIC;
O_18 : OUT STD_LOGIC;
O_19 : OUT STD_LOGIC;
O_20 : OUT STD_LOGIC;
O_21 : OUT STD_LOGIC;
O_22 : OUT STD_LOGIC;
O_23 : OUT STD_LOGIC;
O_24 : OUT STD_LOGIC;
O_25 : OUT STD_LOGIC;
O_26 : OUT STD_LOGIC;
O_27 : OUT STD_LOGIC;
```

```

    LOCK_STATE : OUT STD_LOGIC;
    C1_STATE : OUT STD_LOGIC;
    C2_STATE : OUT STD_LOGIC;
    NOM_STATE : OUT STD_LOGIC;

    USR_IN : IN STD_LOGIC_VECTOR(4 DOWNT0 1);
    HW_AUTH : IN STD_LOGIC
);
END HARDWARE_ASSISTED_SUPERVISOR;

ARCHITECTURE BEHAVIOR OF HARDWARE_ASSISTED_SUPERVISOR IS
    TYPE MACHINE IS (LOCKOUT, CONTROL_1, CONTROL_2, NOMINAL);
    SIGNAL STATE : MACHINE := LOCKOUT;
    SIGNAL CTRL1_CNT : INTEGER RANGE 0 TO 4_000 := 0;
    SIGNAL CTRL2_CNT : INTEGER RANGE 0 TO 4_000 := 0;
    SIGNAL HRTBT1_LAST : STD_LOGIC := '0';
    SIGNAL HRTBT1 : STD_LOGIC;
    SIGNAL CTRL1_ISLIVE : BOOLEAN := FALSE;
    SIGNAL HRTBT2_LAST : STD_LOGIC := '0';
    SIGNAL HRTBT2 : STD_LOGIC;
    SIGNAL CTRL2_ISLIVE : BOOLEAN := FALSE;

BEGIN
    CTRL_MUX : PROCESS (CLK_IN, HW_AUTH, USR_IN)
    BEGIN
        HRTBT1 <= I1_24;
        HRTBT2 <= I2_24;

        --Update liveness timers
        IF (CLK_IN'EVENT AND CLK_IN = '1') THEN
            IF (HRTBT1 = NOT HRTBT1_LAST) THEN
                CTRL1_CNT <= 0;
                CTRL1_ISLIVE <= TRUE;
                HRTBT1_LAST <= HRTBT1;
            ELSE
                IF (CTRL1_CNT < TIMEOUT) THEN
                    CTRL1_CNT <= CTRL1_CNT + 1;
                    CTRL1_ISLIVE <= TRUE;
                ELSE
                    CTRL1_CNT <= TIMEOUT;
                    CTRL1_ISLIVE <= FALSE;
                END IF;
            END IF;
            IF (HRTBT2 = NOT HRTBT2_LAST) THEN
                CTRL2_CNT <= 0;
                CTRL2_ISLIVE <= TRUE;
            END IF;
        END IF;
    END PROCESS;

```

```

        HRTBT2_LAST <= HRTBT2;
    ELSE
        IF (CTRL2_CNT < TIMEOUT) THEN
            CTRL2_CNT <= CTRL2_CNT + 1;
            CTRL2_ISLIVE <= TRUE;
        ELSE
            CTRL2_CNT <= TIMEOUT;
            CTRL2_ISLIVE <= FALSE;
        END IF;
    END IF;
END IF;

```

```

--Set states from liveness
IF (CTRL1_ISLIVE AND CTRL2_ISLIVE) THEN
    STATE <= NOMINAL;
ELSE
    IF (CTRL1_ISLIVE) THEN
        STATE <= CONTROL_1;
    ELSIF (CTRL2_ISLIVE) THEN
        STATE <= CONTROL_2;
    ELSE
        STATE <= LOCKOUT;
    END IF;
END IF;

```

```

--Set state from hardware authentication module flag
IF (HW_AUTH = '0') THEN
    STATE <= LOCKOUT;
END IF;

```

```

--Set states from user input via push buttons
IF (USR_IN(1) = '0') THEN
    STATE <= LOCKOUT;
END IF;
IF (USR_IN(2) = '0') THEN
    STATE <= NOMINAL;
END IF;
IF (USR_IN(3) = '0') THEN
    STATE <= CONTROL_1;
END IF;
IF (USR_IN(4) = '0') THEN
    STATE <= CONTROL_2;
END IF;

```

```

--Route fabric according to set state
CASE STATE IS

```



```
WHEN LOCKOUT =>
    LOCK_STATE <= '1';
    C1_STATE <= '0';
    C2_STATE <= '0';
    NOM_STATE <= '0';
    O_0 <= '0';
    O_1 <= '0';
    O_2 <= '0';
    O_3 <= '0';
    O_4 <= '0';
    O_5 <= '0';
    O_6 <= '0';
    O_7 <= '0';
    O_8 <= '0';
    O_9 <= '0';
    O_10 <= '0';
    O_11 <= '0';
    O_12 <= '0';
    O_13 <= '0';
    O_14 <= '0';
    O_15 <= '0';
    O_16 <= '0';
    O_17 <= '0';
    O_18 <= '0';
    O_19 <= '0';
    O_20 <= '0';
    O_21 <= '0';
    O_22 <= '0';
    O_23 <= '0';
    O_24 <= '0';
    O_25 <= '0';
    O_26 <= '0';
    O_27 <= '0';
```

```
WHEN CONTROL_1 =>
    LOCK_STATE <= '0';
    C1_STATE <= '1';
    C2_STATE <= '0';
    NOM_STATE <= '0';
    O_0 <= I1_0;
    O_1 <= I1_1;
    O_2 <= I1_2;
    O_3 <= I1_3;
    O_4 <= I1_4;
    O_5 <= I1_5;
    O_6 <= I1_6;
```

```
O_7 <= I1_7;  
O_8 <= I1_8;  
O_9 <= I1_9;  
O_10 <= I1_10;  
O_11 <= I1_11;  
O_12 <= I1_12;  
O_13 <= I1_13;  
O_14 <= I1_14;  
O_15 <= I1_15;  
O_16 <= I1_16;  
O_17 <= I1_17;  
O_18 <= I1_18;  
O_19 <= I1_19;  
O_20 <= I1_20;  
O_21 <= I1_21;  
O_22 <= I1_22;  
O_23 <= I1_23;  
O_24 <= I1_24;  
O_25 <= I1_25;  
O_26 <= I1_26;  
O_27 <= I1_27;
```

```
WHEN CONTROL_2 =>  
  LOCK_STATE <= '0';  
  C1_STATE <= '0';  
  C2_STATE <= '1';  
  NOM_STATE <= '0';  
  O_0 <= I2_0;  
  O_1 <= I2_1;  
  O_2 <= I2_2;  
  O_3 <= I2_3;  
  O_4 <= I2_4;  
  O_5 <= I2_5;  
  O_6 <= I2_6;  
  O_7 <= I2_7;  
  O_8 <= I2_8;  
  O_9 <= I2_9;  
  O_10 <= I2_10;  
  O_11 <= I2_11;  
  O_12 <= I2_12;  
  O_13 <= I2_13;  
  O_14 <= I2_14;  
  O_15 <= I2_15;  
  O_16 <= I2_16;  
  O_17 <= I2_17;  
  O_18 <= I2_18;
```

```
O_19 <= I2_19;  
O_20 <= I2_20;  
O_21 <= I2_21;  
O_22 <= I2_22;  
O_23 <= I2_23;  
O_24 <= I2_24;  
O_25 <= I2_25;  
O_26 <= I2_26;  
O_27 <= I2_27;
```

```
WHEN NOMINAL =>  
  LOCK_STATE <= '0';  
  C1_STATE <= '0';  
  C2_STATE <= '0';  
  NOM_STATE <= '1';  
  O_0 <= I1_0;  
  O_1 <= I1_1;  
  O_2 <= I1_2;  
  O_3 <= I1_3;  
  O_4 <= I1_4;  
  O_5 <= I1_5;  
  O_6 <= I1_6;  
  O_7 <= I1_7;  
  O_8 <= I1_8;  
  O_9 <= I1_9;  
  O_10 <= I1_10;  
  O_11 <= I1_11;  
  O_12 <= I1_12;  
  O_13 <= I1_13;  
  O_14 <= I1_14;  
  O_15 <= I1_15;  
  O_16 <= I1_16;  
  O_17 <= I1_17;  
  O_18 <= I1_18;  
  O_19 <= I1_19;  
  O_20 <= I1_20;  
  O_21 <= I1_21;  
  O_22 <= I1_22;  
  O_23 <= I1_23;  
  O_24 <= I1_24;  
  O_25 <= I1_25;  
  O_26 <= I1_26;  
  O_27 <= I1_27;
```

```
WHEN OTHERS =>  
  LOCK_STATE <= '0';
```

```
C1_STATE <= '0';
C2_STATE <= '0';
NOM_STATE <= '0';
O_0 <= '0';
O_1 <= '0';
O_2 <= '0';
O_3 <= '0';
O_4 <= '0';
O_5 <= '0';
O_6 <= '0';
O_7 <= '0';
O_8 <= '0';
O_9 <= '0';
O_10 <= '0';
O_11 <= '0';
O_12 <= '0';
O_13 <= '0';
O_14 <= '0';
O_15 <= '0';
O_16 <= '0';
O_17 <= '0';
O_18 <= '0';
O_19 <= '0';
O_20 <= '0';
O_21 <= '0';
O_22 <= '0';
O_23 <= '0';
O_24 <= '0';
O_25 <= '0';
O_26 <= '0';
O_27 <= '0';
```

```
END CASE;
END PROCESS CTRL_MUX;
END BEHAVIOR;
```

Appendix F: top.vhdl

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
LIBRARY LATTICE;
USE LATTICE.COMPONENTS.ALL;
LIBRARY WORK;
USE WORK.CSPR_MODULES.ALL;
```

```
ENTITY CSPR IS
```

```
    PORT (
```

```
        --Question: Why aren't we just using STD_LOGIC_VECTORS for the IDC ports?
```

```
        --Answer: These IDC ports have both input and output pins. It is much more clear
```

```
(but verbose) to handle them individually.
```

```
        --IDC A
```

```
        A0 : IN STD_LOGIC;
```

```
        A1 : IN STD_LOGIC;
```

```
        A2 : IN STD_LOGIC;
```

```
        A3 : IN STD_LOGIC;
```

```
        A4 : IN STD_LOGIC;
```

```
        A5 : IN STD_LOGIC;
```

```
        A6 : IN STD_LOGIC;
```

```
        A7 : IN STD_LOGIC;
```

```
        --A8 : IN STD_LOGIC;
```

```
        --A9 : IN STD_LOGIC;
```

```
        A10 : IN STD_LOGIC;
```

```
        A11 : IN STD_LOGIC;
```

```
        A12 : IN STD_LOGIC;
```

```
        A13 : IN STD_LOGIC;
```

```
        A14 : IN STD_LOGIC;
```

```
        A15 : IN STD_LOGIC;
```

```
        A16 : IN STD_LOGIC;
```

```
        A17 : IN STD_LOGIC;
```

```
        A18 : IN STD_LOGIC;
```

```
        A19 : IN STD_LOGIC;
```

```
        --A20 : IN STD_LOGIC;
```

```
        --A21 : IN STD_LOGIC;
```

```
        --A22 : IN STD_LOGIC;
```

```
        --A23 : IN STD_LOGIC;
```

```
        A24 : IN STD_LOGIC;
```

```
        A25 : IN STD_LOGIC;
```

```
        A26 : IN STD_LOGIC;
```

```
        A27 : IN STD_LOGIC;
```

```
        --IDC B
```

```
        B0 : OUT STD_LOGIC := 'Z';
```

```
        B1 : OUT STD_LOGIC := 'Z';
```

```

B2 : OUT STD_LOGIC := 'Z';
B3 : OUT STD_LOGIC := 'Z';
B4 : OUT STD_LOGIC := 'Z';
B5 : OUT STD_LOGIC := 'Z';
B6 : OUT STD_LOGIC := 'Z';
B7 : OUT STD_LOGIC := 'Z';
B8 : OUT STD_LOGIC := 'Z';
B9 : OUT STD_LOGIC := 'Z';
B10 : OUT STD_LOGIC := 'Z';
B11 : OUT STD_LOGIC := 'Z';
B12 : OUT STD_LOGIC := 'Z';
B13 : OUT STD_LOGIC := 'Z';
B14 : OUT STD_LOGIC := 'Z';
B15 : OUT STD_LOGIC := 'Z';
B16 : OUT STD_LOGIC := 'Z';
B17 : OUT STD_LOGIC := 'Z';
B18 : OUT STD_LOGIC := 'Z';
B19 : OUT STD_LOGIC := 'Z';
B20 : OUT STD_LOGIC := 'Z';
B21 : OUT STD_LOGIC := 'Z';
--B22 : OUT STD_LOGIC := 'Z';
--B23 : OUT STD_LOGIC := 'Z';
B24 : OUT STD_LOGIC := 'Z';
B25 : OUT STD_LOGIC := 'Z';
B26 : OUT STD_LOGIC := 'Z';
B27 : OUT STD_LOGIC := 'Z';
--IDC C
C0 : IN STD_LOGIC;
C1 : IN STD_LOGIC;
C2 : IN STD_LOGIC;
C3 : IN STD_LOGIC;
C4 : IN STD_LOGIC;
C5 : IN STD_LOGIC;
C6 : IN STD_LOGIC;
C7 : IN STD_LOGIC;
--C8 : IN STD_LOGIC;
--C9 : IN STD_LOGIC;
C10 : IN STD_LOGIC;
C11 : IN STD_LOGIC;
C12 : IN STD_LOGIC;
C13 : IN STD_LOGIC;
C14 : IN STD_LOGIC;
C15 : IN STD_LOGIC;
C16 : IN STD_LOGIC;
C17 : IN STD_LOGIC;
C18 : IN STD_LOGIC;

```

```

C19 : IN STD_LOGIC;
--C20      :      IN STD_LOGIC;
--C21      :      IN STD_LOGIC;
--C22      :      IN STD_LOGIC;
--C23      :      IN STD_LOGIC;
C24 : IN STD_LOGIC;
C25 : IN STD_LOGIC;
C26 : IN STD_LOGIC;
C27 : IN STD_LOGIC;
--IDC D
D0 : OUT STD_LOGIC;
D1 : OUT STD_LOGIC;
D2 : OUT STD_LOGIC;
D3 : OUT STD_LOGIC;
D4 : OUT STD_LOGIC;
D5 : OUT STD_LOGIC;
D6 : OUT STD_LOGIC;
D7 : OUT STD_LOGIC;
D8 : OUT STD_LOGIC;
D9 : OUT STD_LOGIC;
D10 : OUT STD_LOGIC;
D11 : OUT STD_LOGIC;
D12 : OUT STD_LOGIC;
D13 : OUT STD_LOGIC;
D14 : OUT STD_LOGIC;
D15 : OUT STD_LOGIC;
D16 : OUT STD_LOGIC;
D17 : OUT STD_LOGIC;
D18 : OUT STD_LOGIC;
D19 : OUT STD_LOGIC;
D20 : OUT STD_LOGIC;
D21 : OUT STD_LOGIC;
D22 : OUT STD_LOGIC;
D23 : IN STD_LOGIC; --EEPROM Master In Slave Out
D24 : OUT STD_LOGIC;
D25 : OUT STD_LOGIC;
D26 : OUT STD_LOGIC;
D27 : OUT STD_LOGIC;

Usrc_RX : IN STD_LOGIC;
Usrc_TX : OUT STD_LOGIC;

BTN : IN STD_LOGIC_VECTOR(4 DOWNTO 1);
LED : OUT STD_LOGIC_VECTOR(8 DOWNTO 1)
);
END C_SPR;

```

ARCHITECTURE BEHAVIOR OF CSPR IS

```
COMPONENT OSCH
  GENERIC (NOM_FREQ : STRING := "53.2");
  PORT (
    STDBY : IN STD_LOGIC;
    OSC : OUT STD_LOGIC;
    SEDSTDBY : OUT STD_LOGIC
  );
END COMPONENT;
```

```
COMPONENT PLL
  PORT (
    CLKI : IN STD_LOGIC;
    CLKOP : OUT STD_LOGIC;
    CLKOS : OUT STD_LOGIC;
    CLKOS2 : OUT STD_LOGIC;
    CLKOS3 : OUT STD_LOGIC;
    LOCK : OUT STD_LOGIC
  );
END COMPONENT;
```

```
COMPONENT HARDWARE_ASSISTED_SUPERVISOR
  PORT (
    CLK_IN : IN STD_LOGIC;
    --Input Controller 1
    I1_0 : IN STD_LOGIC;
    I1_1 : IN STD_LOGIC;
    I1_2 : IN STD_LOGIC;
    I1_3 : IN STD_LOGIC;
    I1_4 : IN STD_LOGIC;
    I1_5 : IN STD_LOGIC;
    I1_6 : IN STD_LOGIC;
    I1_7 : IN STD_LOGIC;
    I1_8 : IN STD_LOGIC;
    I1_9 : IN STD_LOGIC;
    I1_10 : IN STD_LOGIC;
    I1_11 : IN STD_LOGIC;
    I1_12 : IN STD_LOGIC;
    I1_13 : IN STD_LOGIC;
    I1_14 : IN STD_LOGIC;
    I1_15 : IN STD_LOGIC;
    I1_16 : IN STD_LOGIC;
    I1_17 : IN STD_LOGIC;
    I1_18 : IN STD_LOGIC;
    I1_19 : IN STD_LOGIC;
```



```
I1_20 : IN STD_LOGIC;
I1_21 : IN STD_LOGIC;
I1_22 : IN STD_LOGIC;
I1_23 : IN STD_LOGIC;
I1_24 : IN STD_LOGIC;
I1_25 : IN STD_LOGIC;
I1_26 : IN STD_LOGIC;
I1_27 : IN STD_LOGIC;
--Input Controller 2
I2_0 : IN STD_LOGIC;
I2_1 : IN STD_LOGIC;
I2_2 : IN STD_LOGIC;
I2_3 : IN STD_LOGIC;
I2_4 : IN STD_LOGIC;
I2_5 : IN STD_LOGIC;
I2_6 : IN STD_LOGIC;
I2_7 : IN STD_LOGIC;
I2_8 : IN STD_LOGIC;
I2_9 : IN STD_LOGIC;
I2_10 : IN STD_LOGIC;
I2_11 : IN STD_LOGIC;
I2_12 : IN STD_LOGIC;
I2_13 : IN STD_LOGIC;
I2_14 : IN STD_LOGIC;
I2_15 : IN STD_LOGIC;
I2_16 : IN STD_LOGIC;
I2_17 : IN STD_LOGIC;
I2_18 : IN STD_LOGIC;
I2_19 : IN STD_LOGIC;
I2_20 : IN STD_LOGIC;
I2_21 : IN STD_LOGIC;
I2_22 : IN STD_LOGIC;
I2_23 : IN STD_LOGIC;
I2_24 : IN STD_LOGIC;
I2_25 : IN STD_LOGIC;
I2_26 : IN STD_LOGIC;
I2_27 : IN STD_LOGIC;
--Output Controller
O_0 : OUT STD_LOGIC;
O_1 : OUT STD_LOGIC;
O_2 : OUT STD_LOGIC;
O_3 : OUT STD_LOGIC;
O_4 : OUT STD_LOGIC;
O_5 : OUT STD_LOGIC;
O_6 : OUT STD_LOGIC;
O_7 : OUT STD_LOGIC;
```

```

O_8 : OUT STD_LOGIC;
O_9 : OUT STD_LOGIC;
O_10 : OUT STD_LOGIC;
O_11 : OUT STD_LOGIC;
O_12 : OUT STD_LOGIC;
O_13 : OUT STD_LOGIC;
O_14 : OUT STD_LOGIC;
O_15 : OUT STD_LOGIC;
O_16 : OUT STD_LOGIC;
O_17 : OUT STD_LOGIC;
O_18 : OUT STD_LOGIC;
O_19 : OUT STD_LOGIC;
O_20 : OUT STD_LOGIC;
O_21 : OUT STD_LOGIC;
O_22 : OUT STD_LOGIC;
O_23 : OUT STD_LOGIC;
O_24 : OUT STD_LOGIC;
O_25 : OUT STD_LOGIC;
O_26 : OUT STD_LOGIC;
O_27 : OUT STD_LOGIC;

LOCK_STATE : OUT STD_LOGIC;
C1_STATE : OUT STD_LOGIC;
C2_STATE : OUT STD_LOGIC;
NOM_STATE : OUT STD_LOGIC;

USR_IN : IN STD_LOGIC_VECTOR(4 DOWNT0 1);
HW_AUTH : IN STD_LOGIC
);
END COMPONENT;

COMPONENT HW_AUTH_MODULE
PORT (
MISO : IN STD_LOGIC;
CLK_IN : IN STD_LOGIC;
CS : OUT STD_LOGIC;
CLK_OUT : OUT STD_LOGIC;
MOSI : OUT STD_LOGIC;
HW_GOOD : OUT STD_LOGIC
);
END COMPONENT;

--COMPONENT Bus_Master
--PORT (
--clk : IN std_logic;
--rst : IN std_logic;

```

```

--Data  : INOUT std_logic_vector(15 DOWNT0 0);
--Addr  : IN std_logic_vector(15 DOWNT0 0);
--Xrqst : IN std_logic;
--XDat  : OUT std_logic;
--YDat  : IN std_logic;
--BusRqst : IN std_logic_vector(9 DOWNT0 0);
--BusCtrl : OUT std_logic_vector(9 DOWNT0 0)
--);
--END COMPONENT;

--COMPONENT RS232_Usr_Int
--GENERIC (
--Baud  : INTEGER; -- Baud Rate
--clk_in : INTEGER -- Input Clk
--);
--PORT (
--clk    : IN std_logic;
--rst    : IN std_logic;
--rs232_rcv : IN std_logic;
--rs232_xmt : OUT std_logic;
--Data    : INOUT std_logic_vector(15 DOWNT0 0);
--Addr    : OUT std_logic_vector(15 DOWNT0 0);
--Xrqst   : OUT std_logic;
--XDat    : IN std_logic;
--YDat    : OUT std_logic;
--BusRqst : OUT std_logic;
--BusCtrl : IN std_logic
--);
--END COMPONENT;

--COMPONENT LED_Ctrl
--PORT (
--clk    : IN STD_LOGIC;
--rst    : IN STD_LOGIC;
--Data   : INOUT std_logic_vector(15 DOWNT0 0);
--Addr   : OUT std_logic_vector(15 DOWNT0 0);
--Xrqst  : OUT std_logic;
--XDat   : IN std_logic;
--YDat   : OUT std_logic;
--BusRqst : OUT std_logic;
--BusCtrl : IN std_logic;
--LED1_Out : OUT STD_LOGIC
--);
--END COMPONENT;

--COMPONENT Std_Counter

```

```

--GENERIC (
--Width : INTEGER -- width of counter
--);
--PORT (
--INC, rst, clk : IN std_logic;
--Count      : OUT STD_LOGIC_VECTOR(Width - 1 DOWNT0 0));
--END COMPONENT;

SIGNAL OSC_Stdbby : STD_LOGIC := '0';
SIGNAL PLL_IN : STD_LOGIC := '0';
SIGNAL OSC_SEDSTDBY : STD_LOGIC := '0';

SIGNAL CLK53_2M : STD_LOGIC := '0';
SIGNAL CLK24_93M : STD_LOGIC := '0';
SIGNAL CLK8_31M : STD_LOGIC := '0';
SIGNAL CLK1_5M : STD_LOGIC := '0';
SIGNAL PLL_LOCK : STD_LOGIC := '0';

SIGNAL MISO0 : STD_LOGIC := '0';
SIGNAL CS0 : STD_LOGIC := '0';
SIGNAL CLK_OUT0 : STD_LOGIC := '0';
SIGNAL MOSI0 : STD_LOGIC := '0';
SIGNAL HW_AUTH_FLAG : STD_LOGIC := '0';

SIGNAL A : STD_LOGIC_VECTOR(27 DOWNT0 0) := (OTHERS => '0');
SIGNAL B : STD_LOGIC_VECTOR(27 DOWNT0 0) := (OTHERS => '0');
SIGNAL C : STD_LOGIC_VECTOR(27 DOWNT0 0) := (OTHERS => '0');
SIGNAL D : STD_LOGIC_VECTOR(21 DOWNT0 0) := (OTHERS => '0');

SIGNAL LOCK_S : STD_LOGIC := '0';
SIGNAL C1_S : STD_LOGIC := '0';
SIGNAL C2_S : STD_LOGIC := '0';
SIGNAL NOM_S : STD_LOGIC := '0';

SIGNAL LED_OUT : STD_LOGIC_VECTOR(8 DOWNT0 1);
--From Bus_Interface_Top
SIGNAL Bus_Int1_WE, Bus_Int1_RE, Bus_Int1_Busy : STD_LOGIC := '0';
SIGNAL Bus_Int1_DataIn, Bus_Int1_DataOut, Bus_Int1_AddrIn :
STD_LOGIC_VECTOR(15 DOWNT0 0) := (OTHERS => '0');
SIGNAL Addr : STD_LOGIC_VECTOR(15 DOWNT0 0) := (OTHERS => '0');
SIGNAL Xrqst : STD_LOGIC := '0';
SIGNAL YDat : STD_LOGIC := '0';
SIGNAL BusRqst : STD_LOGIC_VECTOR(9 DOWNT0 0) := (OTHERS => '0');
SIGNAL Data : STD_LOGIC_VECTOR(15 DOWNT0 0) := (OTHERS => '0');
SIGNAL XDat : STD_LOGIC := '0';
SIGNAL BusCtrl : STD_LOGIC_VECTOR(9 DOWNT0 0) := (OTHERS => '0');

```

```

SIGNAL System_rst : STD_LOGIC := '0';
SIGNAL Reset_Cnt_INC, Reset_Cnt_rst : STD_LOGIC := '0';
SIGNAL Reset_Cnt_out : STD_LOGIC_VECTOR(7 DOWNTO 0) := (OTHERS => '0');
SIGNAL LED_1n : STD_LOGIC := '0';

```

```

BEGIN

```

```

  Int_OSC : OSCH
  PORT MAP(
    STDBY => OSC_Stdby,
    OSC => PLL_IN,
    SEDSTDBY => OSC_SEDSTDBY
  );

```

```

  PLL0 : PLL
  PORT MAP(
    CLKI => PLL_IN,
    CLKOP => CLK53_2M,
    CLKOS => CLK24_93M,
    CLKOS2 => CLK8_31M,
    CLKOS3 => CLK1_5M,
    LOCK => PLL_LOCK
  );

```

```

  CTRL_MUX : HARDWARE_ASSISTED_SUPERVISOR
  PORT MAP(
    CLK_IN => CLK53_2M,
    I1_0 => A(0),
    I1_1 => A(1),
    I1_2 => A(2),
    I1_3 => A(3),
    I1_4 => A(4),
    I1_5 => A(5),
    I1_6 => A(6),
    I1_7 => A(7),
    I1_8 => '0', --A(8),
    I1_9 => '0', --A(9),
    I1_10 => A(10),
    I1_11 => A(11),
    I1_12 => A(12),
    I1_13 => A(13),
    I1_14 => A(14),
    I1_15 => A(15),
    I1_16 => A(16),
    I1_17 => A(17),
    I1_18 => A(18),

```

I1_19 => A(19),
I1_20 => '0', --A(20),
I1_21 => '0', --A(21),
I1_22 => '0', --A(22),
I1_23 => '0', --A(23),
I1_24 => A(24),
I1_25 => A(25),
I1_26 => A(26),
I1_27 => A(27),
I2_0 => C(0),
I2_1 => C(1),
I2_2 => C(2),
I2_3 => C(3),
I2_4 => C(4),
I2_5 => C(5),
I2_6 => C(6),
I2_7 => C(7),
I2_8 => '0', --C(8),
I2_9 => '0', --C(9),
I2_10 => C(10),
I2_11 => C(11),
I2_12 => C(12),
I2_13 => C(13),
I2_14 => C(14),
I2_15 => C(15),
I2_16 => C(16),
I2_17 => C(17),
I2_18 => C(18),
I2_19 => C(19),
I2_20 => '0', --C(20),
I2_21 => '0', --C(21),
I2_22 => '0', --C(22),
I2_23 => '0', --C(23),
I2_24 => C(24),
I2_25 => C(25),
I2_26 => C(26),
I2_27 => C(27),
O_0 => D(0),
O_1 => D(1),
O_2 => D(2),
O_3 => D(3),
O_4 => D(4),
O_5 => D(5),
O_6 => D(6),
O_7 => D(7),
O_8 => OPEN,

```

O_9 => OPEN,
O_10 => D(8),
O_11 => D(9),
O_12 => D(10),
O_13 => D(11),
O_14 => D(12),
O_15 => D(13),
O_16 => D(14),
O_17 => D(15),
O_18 => D(16),
O_19 => D(17),
O_20 => OPEN,
O_21 => OPEN,
O_22 => OPEN,
O_23 => OPEN,
O_24 => D(18),
O_25 => D(19),
O_26 => D(20),
O_27 => D(21),
LOCK_STATE => LOCK_S,
C1_STATE => C1_S,
C2_STATE => C2_S,
NOM_STATE => NOM_S,
USR_IN => BTN,
HW_AUTH => HW_AUTH_FLAG
);

AUTH_MODULE0 : HW_AUTH_MODULE
PORT MAP(
    MISO => MISO0,
    CLK_IN => CLK1_5M,
    CS => CS0,
    CLK_OUT => CLK_OUT0,
    MOSI => MOSI0,
    HW_GOOD => HW_AUTH_FLAG
);

--BM : Bus_Master
--PORT MAP(
--clk    => CLK24_93M,
--rst    => System_rst,
--Data   => Data,
--Addr   => Addr,
--Xrqst  => Xrqst,
--XDat   => XDat,
--YDat   => YDat,

```

```

--BusRqst => BusRqst,
--BusCtrl => BusCtrl
--);

--RS232_Usr : RS232_Usr_Int
--GENERIC MAP(
--Baud  => 9600, -- Baud Rate
--Clk_In => 24937500 --Clk_Freq
--)
--PORT MAP(
--clk    => CLK24_93M,
--rst    => System_rst,
--rs232_rcv => Usr_RX,
--rs232_xmt => Usr_TX,
--Data   => Data,
--Addr   => Addr,
--Xrqst  => Xrqst,
--XDat   => XDat,
--YDat   => YDat,
--BusRqst => BusRqst(1),
--BusCtrl => BusCtrl(1)
--);

--LED_Ctrl1 : LED_Ctrl
--PORT MAP(
--clk    => CLK24_93M,
--rst    => System_rst,
--Data   => Data,
--Addr   => Addr,
--Xrqst  => Xrqst,
--XDat   => XDat,
--YDat   => YDat,
--BusRqst => BusRqst(0),
--BusCtrl => BusCtrl(0),
--LED1_Out => LED_1n
--);

--Reset_Cnt : Std_Counter
--GENERIC MAP
--(
--Width => 8
--)
--PORT MAP(
--clk  => CLK8_31M,
--rst  => Reset_Cnt_rst,
--INC  => Reset_Cnt_INC,

```



```

--Count => Reset_Cnt_Out
--);

--OSC_Stdbby      <= '0';
--BusRqst(9 DOWNT0 2) <= (OTHERS => '0');

--IDC A, INPUT from CONTROLLER 1
A(0) <= A0;
A(1) <= A1;
A(2) <= A2;
A(3) <= A3;
A(4) <= A4;
A(5) <= A5;
A(6) <= A6;
A(7) <= A7;
A(8) <= '0';--A8;
A(9) <= '0';--A9;
A(10) <= A10;
A(11) <= A11;
A(12) <= A12;
A(13) <= A13;
A(14) <= A14;
A(15) <= A15;
A(16) <= A16;
A(17) <= A17;
A(18) <= A18;
A(19) <= A19;
A(20) <= '0';--A20;
A(21) <= '0';--A21;
A(22) <= '0';--A22;
A(23) <= '0';--A23;
A(24) <= A24;
A(25) <= A25;
A(26) <= A26;
A(27) <= A27;

--IDC C, INPUT from CONTROLLER 2
C(0) <= C0;
C(1) <= C1;
C(2) <= C2;
C(3) <= C3;
C(4) <= C4;
C(5) <= C5;
C(6) <= C6;
C(7) <= C7;
C(8) <= '0';--C8;

```

```
C(9) <= '0';--C9;
C(10) <= C10;
C(11) <= C11;
C(12) <= C12;
C(13) <= C13;
C(14) <= C14;
C(15) <= C15;
C(16) <= C16;
C(17) <= C17;
C(18) <= C18;
C(19) <= C19;
C(20) <= '0';--C20;
C(21) <= '0';--C21;
C(22) <= '0';--C22;
C(23) <= '0';--C23;
C(24) <= C24;
C(25) <= C25;
C(26) <= C26;
C(27) <= C27;
```

--IDC D, OUTPUT from CONTROLLER 1 or CONTROLLER 2 to POWER
ELECTRONICS

```
D0 <= D(0);
D1 <= D(1);
D2 <= D(2);
D3 <= D(3);
D4 <= D(4);
D5 <= D(5);
D6 <= D(6);
D7 <= D(7);
D8 <= '1';
D9 <= NOT HW_AUTH_FLAG;
D10 <= D(8);
D11 <= D(9);
D12 <= D(10);
D13 <= D(11);
D14 <= D(12);
D15 <= D(13);
D16 <= D(14);
D17 <= D(15);
D18 <= D(16);
D19 <= D(17);
D20 <= CLK_OUT0;
D21 <= CS0;
D22 <= MOSI0;
MISO0 <= D23;
```

```

D24 <= D(18);
D25 <= D(19);
D26 <= D(20);
D27 <= D(21);

LED(1) <= HW_AUTH_FLAG;
LED(2) <= NOT(HW_AUTH_FLAG);
LED(3) <= NOT(LED_1n);
LED(4) <= '1';
LED(5) <= NOT(LOCK_S);
LED(6) <= NOT(NOM_S);
LED(7) <= NOT(C1_S);
LED(8) <= NOT(C2_S);

--Reset_Blkl : PROCESS
--BEGIN
--WAIT UNTIL CLK8_31M'event AND CLK8_31M = '1';
--IF (PLL_Lock = '0') THEN
--Reset_Cnt_rst <= '0';
--ELSE
--Reset_Cnt_rst <= '1';
--END IF;
--END PROCESS;

--Reset_Blkr : PROCESS
--BEGIN
--WAIT UNTIL CLK8_31M'event AND CLK8_31M = '1';
--IF (Reset_Cnt_out < X"7F") THEN
--System_rst <= '0';
--Reset_Cnt_Inc <= '1';
--ELSE
--System_rst <= '1';
--Reset_Cnt_Inc <= '0';
--END IF;
--END PROCESS;

END BEHAVIOR;

```