HARDWARE-ACCELERATED CAN SECURITY FOR ELECTRIC VEHICLE CHARGING STATIONS

by

Benjamin Joseph Macht

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

of

Master of Science

in

Electrical Engineering

MONTANA STATE UNIVERSITY
Bozeman, Montana

May 2026

©COPYRIGHT

by

Benjamin Joseph Macht

2026

All Rights Reserved

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Brock LaMeres, for the guidance, opportunities, and support that made this thesis possible. I am also grateful to the engineering team at Resilient Computing, Dr. Chris Major and Dr. Hezekiah Austin, for their patience and their unfailing generosity fielding my late-night questions. To my fellow research assistants, thank you for your support, collaboration, and the moments of humor that kept this work moving forward. Above all, I want to thank my wife, whose patience, encouragement, and steady support carried me through to the end of this work.

TABLE OF CONTENTS

1. INTRODUCTION 1

2. BACKGROUND AND MOTIVATION 3

Growth of EV Charging Infrastructure and the Emergence of a National-Scale Cyber-Physical System 3
Increasing Cyber-Physical Complexity in EV Charging Systems 5
Heterogeneous Communication Protocols 5
Shift Toward Fully Networked, Cloud-Managed EVSEs 5
Escalating Cyber-Physical Consequences 6
AI/ML-Based Security Trends 6
EVCS Vulnerabilities and System-Level Risks 7
EVCS as a Cyber-Physical System 7
Increasing Cyber-Physical Impact of Attacks 7
High-Consequence Risks in DC Fast Charging Systems 7
INL's Experimental Exploitation of DCFC Control Networks 8
Single-Network Internal Architecture 8
CAN Exposure Through CHAdeMO 8
Grid-Level Risks from EVCS Compromise 9
Public Data Enabling Load-Altering Attacks 9
MaDEVIoT Attacks and Urban Grid Instability 9
Protocol-Level Vulnerabilities and Their Implications 9
OCPP Vulnerabilities 9
ISO 15118 and PLC-Based Risks 10
EVCSMS Vulnerabilities 10
CAN as a Critical Cyber-Physical Vulnerability 10
CAN as an Exposed Interface 10
CAN's Structural Weaknesses 11
CAN Compromise as a Gateway to Grid-Level Risk 11
Motivation for Hardware-Enforced CAN Safety 11
Motivation for this thesis 12

3. RELATED WORK 13

Protocol-Level Security in EV Charging Systems 13
OCPP Security Weaknesses and Proposed Enhancements 13
ISO 15118 and PLC-Layer Vulnerabilities 13
EVCSMS and Backend Platform Vulnerabilities 14
Cyber-Physical Risks in DC Fast Charging Systems 14
Grid-Level Cyber-Physical Threats from EVCS Compromise 15
Load-Altering Attacks Using Public Data 15
MaDEVIoT Attacks and Urban Grid Instability 16

TABLE OF CONTENTS CONTINUED

EVs as Distributed Attack Vectors 16
CAN-Based Vulnerabilities in EV Charging Ecosystems 16
CAN Exposure Through Charging Standards 16
CAN as a Vehicle-Level Attack Surface 17
CAN-Based Attacks Relevant to EV Charging 17
Intrusion Detection and Prevention for CAN Networks 18
Software-Based CAN Authentication and IDS 18
Hardware-Based CAN Security 18
EVCS-Specific Cybersecurity Frameworks and Analyses 19
Early EVCS Security Frameworks 19
System-Level EVCS Security Models 19
False-Data Injection and Federated Learning Defenses 19
Summary and Research Gap 20

4. EXPLINATION OF CAN PROTOCOL 21

Physical Layer 21
Message Frame Formats 22
Standard Data Frame 22
Extended CAN Data Frame 24
Error Detection and Fault Confinement 25
Arbitration and Medium Access 25

5. OVERVIEW OF EV CHARGING STATION COMPONENTS AND EXPERIMENTAL SETUP 27

EV Charging Station Architecture 27
Safety-Critical Components 27
Authorization and User-Access Components 28
Billing and Metering Components 28
Network and Communication Components 29
Load-Management and Power-Control Components 29
Bidirectional Power Transfer (BPT) and DER Components 29
Fault-Detection and Diagnostic Components 30
Remote Monitoring and Control Components 30
Cybersecurity Components 30
Supervisory Control Unit 31
PNNL's Role and Provided Research Platforms 31
Emulator #1: EVCS Box for Signal-Path Discovery 33
Emulator #2: DCFC Power Electronics Box and Software Control Model 34
Power-Electronics Controller Simulation 36
Hardware Accelerator Platform (RadPC Hydra) 38
CAN Bus Integration and Test Environment 40

TABLE OF CONTENTS CONTINUED

6. SYSTEM ARCHITECTURE 43

System Overview 43
SPI Master Component 45
Component Interface 46
Internal Data Flow 46
Decision-Making Behavior 47
MCP2515 Control Component 48
Component Interface 48
Internal Data Flow 49
Decision-Making Behavior 50
Rule Engine Component 51
Component Interface 51
Internal Data Flow 52
Decision-Making Behavior 53
Reciprocal Lookup Table Component 54
Component Interface 55
Internal Data Flow 55
Decision-Making Behavior 56
Register Bank Component 57
Component Interface 57
Internal Data Flow 63
Decision-Making Behavior 64
CAN Limit Checking Top Component 64
Component Interface 65
Internal Data Flow 65
Decision-Making Behavior 66

7. SYSTEM VERIFICATION 67

Verification Objectives 67
Functional Correctness 67
Safety-Rule Enforcement 68
Normal-Operation Behavior 68
Timing Consistency 69
System-Integration Behavior 70
Verification Scope and Assumptions 71
Scope of Verification 71
Out-of-Scope Items 72
Assumptions 72
Verification Methodology 73
Test Categories 74
Test Procedure 75

TABLE OF CONTENTS CONTINUED

Evaluation Criteria 76
Test Cases and Expected Results 76
Normal-Operation Test Case 77
Limit-Violation Test Cases 77
Identifier-Validation Test Case 80
Frame-Format Validation Test Case 81
Timing-Consistency Test Cases 81
System-Integration Test Case 82
Results and Discussion 82
Functional Correctness 83
Safety-Rule Enforcement 85
Identifier and Format Validation 91
Timing Consistency 94
System-Integration Behavior 95
Summary of Findings 95

8. FUTURE WORK 97

Full EVCS Control-Loop Integration 97
Integration with the CyberShield Malware-Tolerant Architecture 99
Runtime-Configurable Rule Engine and Dynamic Safety Policies 101
Extending Rule Coverage and Parallelizing Evaluation 102

9. CONCLUSION 104

10. REFERENCES CITED 106

LIST OF TABLES

Table Page

1. Table 1: Summary of rule conditions, corresponding accelerator actions, and resulting fault-bit assignments. 54

2. Table 2: Configuration parameters used for validation testing. 83

LIST OF FIGURES

Figure Page

1. Figure 1: U.S. light-duty plug-in electric vehicle inventory predictions [1]. 3

2. Figure 2: NREL's predictions of public charging infrastructure requirements versus current availability [2]. 4

3. Figure 3: Standard data frame [32]. 23

4. Figure 4: Extended data frame [32]. 24

5. Figure 5:Original prototype setup using Emulator #1 to sniff I^2C traffic. A) RadPC running on a Nexys A7 board. B) Analog Discovery II used to monitor I2C bus for comparison to RadPC sniffing. C) EVCS emulator. D) Serial monitor used to monitor RadPC sniffing. 34

6. Figure 6: External testbench used to validate the C-based power-electronics controller and CAN-monitoring accelerator. The setup includes a Raspberry Pi (A) with an RS485 CAN HAT generating supervisory CAN traffic, a second RS485 CAN HAT (B) connected to the RadPC MCU for CAN reception, an Analog Discovery 3 (C) used to observe SPI communication between the MCP2515 and the accelerator, and a RadPC Hydra (D) executing the controller firmware. 37

7. Figure 7: Resilient Computing's RadPC-SBC-100 with the RS485 CAN Hat connected via a custom PCB. 39

8. Figure 8: Conceptual block diagram of the experimental setup used to test the CAN limit checking hardware accelerator. 40

9. Figure 9: Final experimental testbed used for system-integration verification. The setup includes the Raspberry Pi with RS485 CAN HAT (A) generating supervisory CAN frames, a RadPC Hydra with its own RS485 CAN HAT (B) receiving and processing frames through the accelerator, the RadPC programming and power interface (C), and a UART connection to a host PC (D) for serial-terminal monitoring. 42

10. Figure 10: Conceptual block diagram showing the components of the CAN limit checking hardware accelerator. 45

LIST OF FIGURES CONTINUED

11. Figure 11: Memory mapped voltage limit configuration register defining the minimum and maximum allowable voltage values. The accelerator compares incoming CAN frame voltage requests against these thresholds to classify frames as safe or unsafe. 58

12. Figure 12: Memory-mapped current-limit configuration register specifying the allowable current range for incoming CAN requests. These limits are used by the rule engine to enforce hardware-level current-safety constraints. 58

13. Figure 13: Memory-mapped lower power-limit register defining the minimum allowable power request. The accelerator computes power from voltage and current fields and flags frames whose calculated power falls below this threshold. 59

14. Figure 14: Memory-mapped upper power-limit register defining the maximum allowable power request. Frames whose computed power exceeds this limit are classified as unsafe and reported to the MCU. 59

15. Figure 15: Memory-mapped configuration control register used to configure accelerator operation. The register contains the 16-bit SPI clock-divider field, the extended-identifier enable bit, and the interrupt-clear bit, allowing the RadPC MCU to control timing behavior and acknowledge accelerator interrupts. 60

16. Figure 16: Memory-mapped status register containing frame-classification results, fault codes, and interrupt flags. The RadPC MCU reads this register to determine whether the most recent frame was safe and to identify any rule-violation conditions. 61

17. Figure 17: Memory-mapped frame-identifier register storing the 29-bit CAN ID of the most recently processed frame. The accelerator writes this field after rule-engine evaluation, allowing the MCU to inspect the identifier of safe or unsafe frames. 61

18. Figure 18: Memory-mapped data-length-code register storing the DLC value of the most recent CAN frame. This register allows the MCU to verify the frame length and correlate it with the received data bytes. 62

19. Figure 19: Memory-mapped data register containing bytes 0–3 of the most recently processed CAN frame. These values are written by the accelerator after SPI retrieval and used by the MCU for application-level processing. 62

LIST OF FIGURES CONTINUED

20. Figure 20: Memory-mapped data register containing bytes 4–7 of the most recently processed CAN frame. Together with Figure 19, this register provides the full eight-byte payload for MCU consumption. 63

21. Figure 21: Verification testing setup. 70

22. Figure 22: External logic-analyzer capture showing a complete safe CAN frame and its corresponding MCP2515 SPI transaction. The lower trace shows the CAN frame as observed directly on the bus, including the standard identifier, DLC value, and six data bytes. The upper trace shows the MCP2515's SPI transfer of the same frame to the accelerator, with the ID, DLC, and data fields visible in the decoded SPI stream. The alignment between the CAN-bus capture and the SPI transaction confirms correct frame retrieval and byte-ordering behavior at the hardware interface. 84

23. Figure 23: Internal ILA capture demonstrating correct register-loading behavior for the safe frame shown in Figure 22. The ILA trace captures the rising edge of the frame-ready interrupt (1), followed by the sequential latching of the DLC register (2), data-byte registers (3), and the 29-bit CAN identifier (4) into the accelerator's register bank. The status register (5) indicates that the frame-ready interrupt is set and that the frame is classified as safe. The corresponding UART output (bottom) shows the decoded frame as reported by the RadPC MCU, confirming agreement between internal FPGA state and external system-level behavior. 85

24. Figure 24: External logic-analyzer capture showing a high-voltage request CAN frame and its corresponding MCP2515 SPI transaction. The lower trace shows the CAN frame as observed on the bus, including the standard identifier, DLC value, and six data bytes representing a voltage request that exceeds the configured safety limit. The upper trace shows the MCP2515's SPI transfer of the same frame to the accelerator, with the ID, DLC, and data fields visible in the decoded SPI stream. The alignment between the CAN-bus capture and the SPI transaction confirms correct retrieval of the out-of-range request and proper byte ordering at the hardware interface. 86

LIST OF FIGURES CONTINUED

25. Figure 25: External logic-analyzer capture showing a low-voltage request CAN frame and the corresponding MCP2515 SPI transaction. The lower trace shows a CAN frame whose data bytes encode a voltage request below the configured minimum safety threshold. The upper trace shows the MCP2515's SPI transfer of the same frame to the accelerator, with the identifier, DLC, and data fields visible in the decoded SPI stream. The alignment between the CAN-bus capture and the SPI transaction confirms correct retrieval of the out-of-range voltage request and proper byte-ordering behavior at the hardware interface. 86

26. Figure 26: Internal ILA capture demonstrating correct register-loading and fault-classification behavior for the high-voltage request frame shown in Figure 24. The ILA trace captures the rising edge of the frame-ready interrupt (1), followed by the sequential latching of the DLC register (2), data-byte registers (3), and the 29-bit CAN identifier (4) into the accelerator's register bank. The status register indicates that the frame-ready interrupt is set, the frame is classified as unsafe, and the voltage-limit fault code is asserted (5). The corresponding UART output (bottom) shows the decoded unsafe frame and the reported error code, confirming agreement between internal FPGA state and external system-level behavior. 87

27. Figure 27: Internal ILA capture demonstrating correct register-loading and fault-classification behavior for the low-voltage request frame shown in Figure 25. The ILA trace captures the rising edge of the frame-ready interrupt (1), followed by the sequential latching of the DLC register (2), data-byte registers (3), and the 29-bit CAN identifier (4) into the accelerator's register bank. The status register indicates that the frame-ready interrupt is set, the frame is classified as unsafe, and the voltage-limit fault code is asserted (5). The corresponding UART output (bottom) shows the decoded unsafe frame and the reported error code, confirming agreement between internal FPGA state and external system-level behavior. 88

28. Figure 28: External logic-analyzer capture showing a high-current request CAN frame and the corresponding MCP2515 SPI transaction. The lower trace shows a CAN frame whose data bytes encode a current request exceeding the configured safety limit. The upper trace shows the MCP2515's SPI transfer of the same frame to the accelerator, with the identifier, DLC, and data fields visible in the decoded SPI stream. The alignment between the CAN-bus capture and the SPI transaction confirms correct retrieval of the out-of-range current request and proper byte ordering at the hardware interface. 89

LIST OF FIGURES CONTINUED

29. Figure 29: Internal ILA capture demonstrating correct register-loading and fault-classification behavior for the high-current request frame shown in Figure 28. The ILA trace captures the rising edge of the frame-ready interrupt (1), followed by the sequential latching of the DLC register (2), data-byte registers (3), and the 29-bit CAN identifier (4) into the accelerator's register bank. The status register (5) indicates that the frame-ready interrupt is set, the frame is classified as unsafe, and the current-limit fault code is asserted. The UART output (bottom) shows the decoded unsafe frame and the reported error code, confirming agreement between internal FPGA state and external system-level behavior. 89

30. Figure 30: External logic-analyzer capture showing a high-power request CAN frame and the corresponding MCP2515 SPI transaction. The lower trace shows a CAN frame whose encoded voltage and current values produce a power request exceeding the accelerator's configured power limit. The upper trace shows the MCP2515's SPI transfer of the same frame, with the identifier, DLC, and data fields visible in the decoded SPI stream. The capture verifies that the MCP2515 delivers the full frame correctly and that the accelerator receives all fields required to compute and evaluate the power-limit condition. 90

31. Figure 31 Internal ILA capture demonstrating correct register-loading and fault-classification behavior for the high-power request frame shown in Figure 30. The ILA trace captures the rising edge of the frame-ready interrupt (1), followed by the sequential latching of the DLC register (2), data-byte registers (3), and the 29-bit CAN identifier (4) into the accelerator's register bank. The accelerator computes the requested power from the received voltage and current fields and classifies the frame as unsafe when the calculated value exceeds the configured power threshold. The status register (5) asserts the power-limit fault code, and the UART output (bottom) reports the unsafe frame and associated error code, confirming agreement between internal FPGA state and external system-level behavior. 91

32. Figure 32: External logic-analyzer capture showing a non-whitelisted CAN identifier and the corresponding MCP2515 SPI transaction. The lower trace shows a CAN frame with an identifier not included in the accelerator's configured whitelist. The upper trace shows the MCP2515's SPI transfer of the same frame, with the ID, DLC, and data fields visible in the decoded SPI stream. The capture verifies that the MCP2515 delivers the frame correctly and that the accelerator receives the full identifier for whitelist evaluation. 92

LIST OF FIGURES CONTINUED

33. Figure 33: Internal ILA capture demonstrating correct rejection of the non-whitelisted CAN identifier shown in Figure 32. The ILA trace shows the frame-ready interrupt (1) followed by the sequential latching of the DLC and data fields, along with the non-whitelisted identifier (2). The status register (3) indicates that the frame-ready interrupt is set, the frame is classified as unsafe, and the ID-violation fault code is asserted. The UART output (bottom) shows the corresponding unsafe-frame report, confirming that the accelerator's whitelist logic and system-level reporting behave consistently. 92

34. Figure 34: External logic-analyzer capture showing a standard-frame CAN message received when an extended-frame (IDE-bit asserted) format is expected. The lower trace shows a CAN frame transmitted with the IDE bit not asserted, indicating a standard 11-bit identifier. Because the accelerator is configured to accept only extended 29-bit identifiers for this test, the frame constitutes a format-mismatch violation. The upper trace shows the MCP2515's SPI transfer of the same frame to the accelerator, with the standard-ID field visible in the decoded SPI stream. The capture confirms that the MCP2515 correctly forwards the standard frame and that the accelerator receives the full identifier and data fields for validation against the expected extended-frame format. 93

35. Figure 35: Internal ILA capture demonstrating correct detection and rejection of a standard-frame message when an extended-frame (IDE-bit asserted) format is expected. The ILA trace captures the rising edge of the frame-ready interrupt (1), followed by the attempted loading of the standard 11-bit identifier, DLC, and data fields into the accelerator's register bank (2). Because the IDE bit is not asserted, the accelerator classifies the frame as unsafe and asserts the extended-frame format-violation fault code. The status register (3) reflects the unsafe classification, and the UART output (bottom) shows the corresponding unsafe-frame report, confirming agreement between internal FPGA state and external system-level behavior. 94

LIST OF FIGURES CONTINUED

36. Figure 36: Internal ILA capture demonstrating consistent interrupt-to-data-ready latency across all verified frames. The trace shows the falling edge of the MCP2515 frame-ready interrupt (1) followed by the rising edge of the accelerator's frame-data-ready signal (2). The number of clock cycles between these two events (3) remains constant across all ILA captures, indicating that the SPI-to-register pipeline, rule-engine evaluation, and register-bank update sequence execute with fixed latency. This repeatable cycle count confirms that the accelerator provides deterministic timing behavior suitable for integration into real-time EVCS control loops. 95

37. Figure 37: System block diagram of *CyberShield* [34]. 100

NOMENCLATURE

Acronyms

AC	Alternating Current
ACK	Acknowledge
AFDC	Alternative Fuels Data Center
API	Application Programming Interface
BPT	Bidirectional Power Transfer
CAN	Controller Area Network
CCS	Combined Charging System
CHAdeMO	Trade name for a DC fast-charging standard (“Charge de Move”)
CP	Control Pilot
CRC	Cyclic Redundancy Check
CS	Chip Select
CSMS	Charging Station Management System
COTS	Commercial Off-The-Shelf
DC	Direct Current
DCFC	Direct Current Fast Charger
DLC	Data Length Code
DNS	Domain Name System
EV	Electric Vehicle
EVCS	Electric Vehicle Charging Station
EVCSMS	Electric Vehicle Charging Station Management System
EVSE	Electric Vehicle Supply Equipment
FSM	Finite State Machine
GB/T	Guobiao Standards (Chinese national standards)
HMI	Human–Machine Interface
HWA	Hardware Accelerator
I2C	Inter-Integrated Circuit
IDE	Identifier Extension (CAN frame bit)
ILA	Integrated Logic Analyzer
INL	Idaho National Laboratory
IRQ	Interrupt Request
ISO	International Organization for Standardization
L2	Level 2 AC charging
LUT	Lookup Table
MAC	Message Authentication Code
MaDEVIoT	Massively Distributed Electric Vehicle Internet-of-Things
MCU	Microcontroller Unit

NOMENCLATURE - CONTINUED

MCP2515	Microchip CAN Controller (part number)
MISO	Master In Slave Out
ML	Machine Learning
MOSI	Master Out Slave In
MSU	Montana State University
NACS	North American Charging Standard
NFC	Near-Field Communication
NREL	National Renewable Energy Laboratory
NRZ	Non-Return to Zero
NTP	Network Time Protocol
OBC	Onboard Charger
OCPP	Open Charge Point Protocol
OCPI	Open Charge Point Interface
OpenADR	Open Automated Demand Response
PE	Protective Earth
PF	Power Factor
PLC	Power Line Communication
PNNL	Pacific Northwest National Laboratory
PP	Proximity Pilot
PV	Photovoltaic
RFID	Radio-Frequency Identification
RTR	Remote Transmission Request
RX	Receive
SAE	Society of Automotive Engineers
SOF	Start of Frame
SPI	Serial Peripheral Interface
SRR	Substitute Remote Request
THD	Total Harmonic Distortion
TLS	Transport Layer Security
TX	Transmit
UART	Universal Asynchronous Receiver–Transmitter
V2B	Vehicle-to-Building
V2G	Vehicle-to-Grid
V2H	Vehicle-to-Home
VPN	Virtual Private Network
Wi-Fi	Wireless Fidelity

ABSTRACT

The rapid expansion of electric vehicle (EV) charging infrastructure has increased the need for secure and reliable embedded control systems, particularly as charging stations operate simultaneously at the grid edge and the network edge. While modern EV charging stations depend on the Controller Area Network (CAN) bus to coordinate safety-critical power-electronics functions, this interface lacks authentication and is vulnerable to spoofing, malformed messages, and unsafe command injection. To address these risks, this thesis presents the design and implementation of a secure, man-in-the-middle hardware accelerator CAN monitor intended for integration with the power-electronics controller in EV charging stations. The accelerator performs real-time validation of CAN identifiers and enforces hardware-accelerated limit checking for voltage, current, and power, ensuring that only authorized and physically safe commands reach the power stage. By offloading these checks into a dedicated hardware module, the system provides deterministic, low-latency protection that is resistant to firmware tampering and software-based attacks. Experimental evaluation demonstrates that the accelerator introduces negligible communication overhead while reliably detecting spoofed, malformed, or out-of-range CAN messages. The results show that embedding a dedicated security and safety co-processor within the EVCS architecture can significantly enhance resilience against CAN-borne cyberattacks and operational faults, contributing to safer and more robust charging infrastructure.

INTRODUCTION

Electric vehicle charging stations (EVCSs) are rapidly becoming critical components of the modern electric grid. As adoption of electric vehicles accelerates, charging infrastructure is expanding in scale, complexity, and connectivity, transforming what was once a simple power-delivery interface into a distributed cyber-physical system operating simultaneously at the grid edge and the network edge. This evolution introduces new operational dependencies and exposes charging stations to a growing range of cyber-physical threats. Ensuring the safety and reliability of these systems requires secure embedded controllers capable of withstanding both accidental faults and malicious manipulation.

Despite the increasing sophistication of EVCS communication protocols, many internal control functions continue to rely on the Controller Area Network (CAN) bus. Originally designed for automotive environments, CAN provides no authentication, message integrity, or sender identification. As a result, any device with physical or logical access to the bus can inject spoofed, malformed, or unsafe messages. In the context of EV charging, where CAN frames directly influence power-electronics behavior, this lack of protection creates a high-impact attack surface. Malicious or erroneous commands can request unsafe voltages, currents, or power levels, potentially damaging equipment, interrupting charging sessions, or destabilizing local grid conditions.

This thesis addresses these vulnerabilities by introducing a hardware-accelerated CAN-safety monitor designed to operate as a man-in-the-middle co-processor within the EVCS control stack. The accelerator intercepts every CAN frame destined for the power-electronics controller, validates its identifier and format, computes derived quantities such as power, and

enforces hardware-level limits on voltage, current, and power. By performing these checks in dedicated hardware rather than firmware, the system provides deterministic timing, eliminates software bypass paths, and reduces susceptibility to malware or firmware tampering.

The remainder of this thesis presents the motivation for hardware-enforced CAN safety, reviews related work in EVCS cybersecurity, describes the architecture of the accelerator, and evaluates its performance through logic-analyzer and internal logic-analyzer (ILA) captures. The results demonstrate that the accelerator reliably detects unsafe or malformed frames, maintains consistent timing behavior, and integrates cleanly with existing EVCS control components. Together, these contributions show that hardware-accelerated CAN validation is a practical and effective approach for improving the safety and resilience of modern electric vehicle charging systems.

BACKGROUND AND MOTIVATION

Growth of EV Charging Infrastructure and the Emergence of a National-Scale Cyber-Physical System

The United States is undergoing a rapid transition toward transportation electrification as part of national climate goals targeting economy-wide net-zero emissions by 2050. Analysts project that 48–61% of U.S. light-duty vehicle sales could be electric by 2030 [1] (Figure 1). Supporting this level of adoption requires a charging ecosystem far larger and more sophisticated than what exists today. The National Renewable Energy Laboratory's (NREL) 2030 National Charging Network report estimates that accommodating 30 - 42 million EVs will require approximately 28 million charging ports, including 182,000 public DC fast-charging ports and over one million public Level 2 ports [1].

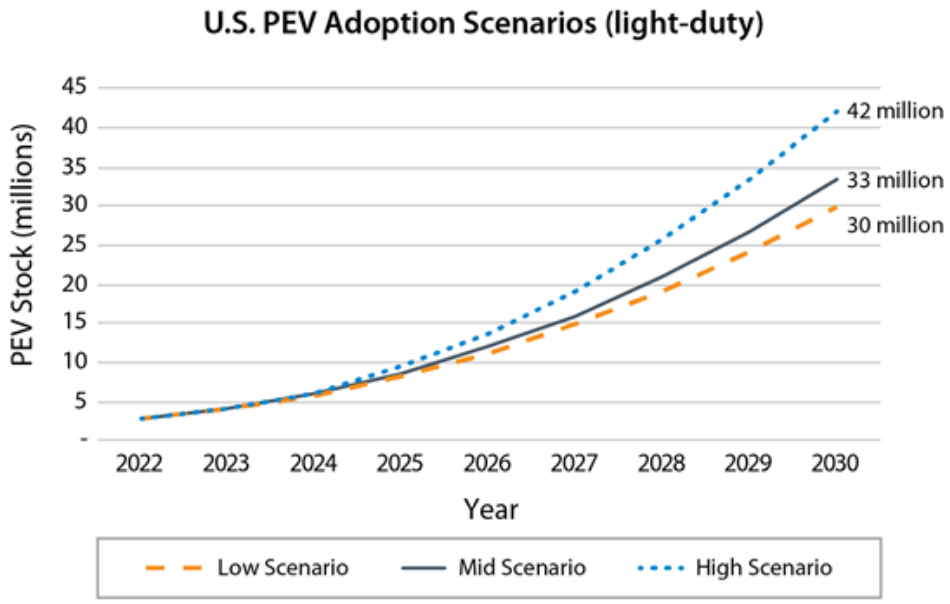

Figure 1: U.S. light-duty plug-in electric vehicle inventory predictions [1].

Real-time deployment data from the Alternative Fuels Data Center (AFDC) illustrates both the momentum and the gap. As of Q2 2024, the United States had 211,382 charging ports (Figure 2), a 6.3% increase in a single quarter [2]. Growth was strongest in the Northeast (13.2%), while California continues to host the largest share of public ports (26.3%). DC fast-charging ports grew by 7.4%, adding more than 3,000 new ports in one quarter [2]. Yet only 13.3% of required DC fast ports and 13.4% of required Level 2 ports have been deployed, and when Tesla-only ports are excluded, the gap widens to 4.9% and 12.4%, respectively [2].

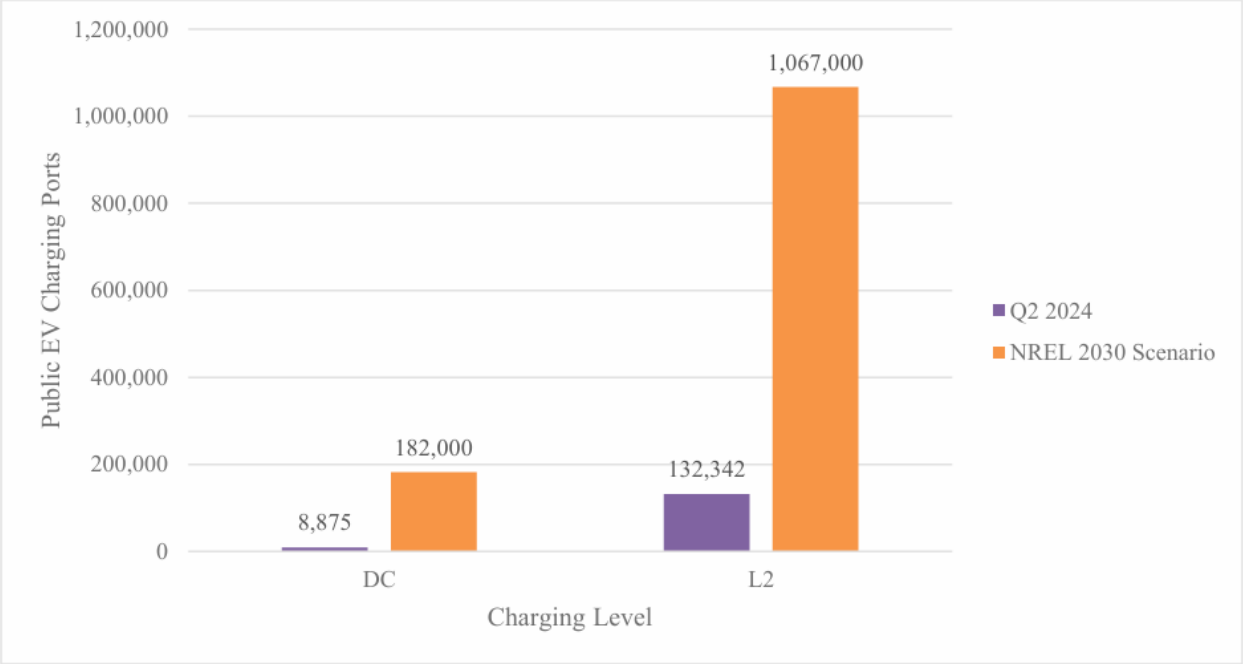

Figure 2: NREL's predictions of public charging infrastructure requirements versus current availability [2].

Together, these reports depict a charging ecosystem expanding rapidly but still far from meeting projected national needs. The combination of massive infrastructure growth, increasing network complexity, and accelerating deployment of high-power DC fast chargers is transforming EV charging into a national-scale cyber-physical system. As millions of new

chargers come online, networked, cloud-managed, and interoperable across vendors, the attack surface for cyber threats expands proportionally. Ensuring cybersecurity is therefore foundational to maintaining grid stability, protecting consumers, and supporting continued EV adoption.

Increasing Cyber-Physical Complexity in EV Charging Systems

As EV charging infrastructure scales, the cyber-physical risks associated with this ecosystem expand in both scope and severity. EV charging stations now sit at the intersection of transportation, energy, and digital communication networks, creating a dense, distributed system with millions of potential entry points for attackers.

Heterogeneous Communication Protocols

Modern EVCS deployments incorporate a diverse set of communication technologies, ranging from legacy CAN bus signaling to the Power Line Communication (PLC) channels used in ISO 15118, as well as cloud-connected protocols such as the Open Charge Point Protocol (OCPP) and Open Automated Demand Response (OpenADR).

This heterogeneity creates uneven security baselines. CAN remains fundamentally unprotected, making physical or local cyberattacks feasible at scale [3]. Meanwhile, cloud-connected chargers introduce remote attack surfaces that grow proportionally with network size, especially as more EVSEs rely on centralized management systems for authentication, billing, and configuration.

Shift Toward Fully Networked, Cloud-Managed EVSEs

Most public EVSEs now authenticate users, process payments, report status, and receive updates through cloud platforms. This connectivity enables efficient operation but also exposes

the system to credential theft, remote manipulation, and denial-of-service attacks. Weak internal protections, such as unencrypted storage, unsecured maintenance interfaces, and easily accessible enclosures, make physical access a powerful attack vector [3].

Escalating Cyber-Physical Consequences

Disrupting charging sessions, once a nuisance, now has broader implications as EV adoption rises. Coordinated manipulation of charging behavior can create load spikes, voltage instability, or frequency deviations on local distribution networks. One survey warns that "attacks using EVCSs to disrupt the power grid could prove to be very detrimental by just 2030 if the prevalence of EVs continues to increase at the current rate" [3].

AI/ML-Based Security Trends

With millions of distributed devices generating continuous operational data, traditional rule-based intrusion detection becomes insufficient. AI and ML-based approaches show promise, but the lack of high-quality EVCS cybersecurity datasets remains a bottleneck [3].

These trends collectively show that the rapid growth of EV charging infrastructure is creating a large, complex, and increasingly attractive cyber-physical attack surface. As the charging network becomes more interconnected and deeply integrated with the electric grid, cybersecurity becomes a prerequisite for transportation reliability, grid stability, and public safety.

EVCS Vulnerabilities and System-Level Risks

EVCS as a Cyber-Physical System

Hamdare et al. describe EV charging ecosystems as "a connected system paradigm at the core of the smart grid, consisting of a complex cyber-physical system compromising linked hardware parts, software elements, and communication protocols" [4]. This interconnectedness means that vulnerabilities in any component, from charger hardware to cloud management platforms, can propagate across the entire system.

The authors identify four primary points of entry that become increasingly vulnerable as deployment accelerates: EV connectors, user terminals, Internet connections, and maintenance interfaces. Each of these interfaces expands in number and accessibility as the charging network grows, amplifying the opportunities for attackers to exploit weaknesses in the EVCS ecosystem [4].

Increasing Cyber-Physical Impact of Attacks

As EVCS density grows, the consequences of compromise escalate. Disrupting a single charger may be a nuisance, but compromising many chargers simultaneously can destabilize charging availability, disrupt transportation patterns, or even affect local grid stability [4].

High-Consequence Risks in DC Fast Charging Systems

DC fast chargers (DCFCs) represent one of the most consequential cyber-physical attack surfaces in the EV ecosystem. Their high-power levels, tightly integrated control architectures, and frequent use in public, unattended environments make them attractive targets.

INL's Experimental Exploitation of DCFC Control Networks

Research from Idaho National Laboratory (INL) shows that once an attacker gains access, either physically or through a compromised EV, the internal communications network of a DC fast charger can be manipulated using off-the-shelf tools [5]. Such access allows an adversary to disrupt coordination among power-electronics modules, falsify HMI charging data, force abrupt load transitions, degrade input power quality (with power factor dropping below 0.8 and current THD exceeding 20%), and violate industry limits on harmonic distortion and transient behavior. INL demonstrated that even a single charger can be forced from 50 kW to standby in roughly 20 ms, and that coordinated manipulation of hundreds of DCFCs could produce grid-scale disturbances [5].

Single-Network Internal Architecture

Many DCFCs rely on a single internal communications network linking the HMI, power-electronics controllers, cooling systems, and protocol modules [5]. Compromising any one component, including through a malicious EV, can provide a pathway to manipulate charger behavior or propagate malware between vehicles and infrastructure.

CAN Exposure Through CHAdeMO

Most importantly for CAN-focused research, INL shows that CHAdeMO-based charging exposes a direct CAN-bus pathway between the EV and the charger, enabling firmware extraction, reflashing, and message manipulation directly over CAN [5]. This finding demonstrates that CAN remains a viable and under-secured attack surface in modern DC fast-charging systems.

Grid-Level Risks from EVCS Compromise

Public Data Enabling Load-Altering Attacks

Acharya et al. demonstrate that publicly available data, such as smartphone-app charging-station usage and utility-published grid topology, can be combined to infer where high-wattage EV loads are located and when they are active [6]. Because EVCSs are not continuously monitored by utilities, these demand-side assets form a large, distributed, and relatively unprotected attack surface.

MaDEVIoT Attacks and Urban Grid Instability

A related study introduces MaDEVIoT attacks, showing that coordinated cyberattacks on Internet-connected EVCSs can manipulate large amounts of charging demand simultaneously, creating abrupt load changes capable of destabilizing urban power systems [7]. Using Manhattan as a case study, the authors show that a sudden interruption of charging across the city could push grid frequency to 62.095 Hz, exceeding over-frequency relay thresholds and potentially causing cascading outages [7].

These findings demonstrate that EVCS cybersecurity is not merely a transportation issue but a grid-reliability concern.

Protocol-Level Vulnerabilities and Their Implications

OCPP Vulnerabilities

OCPP plays a central role in coordinating authorization, metering, billing, and power delivery across public charging infrastructure. Vulnerabilities in OCPP can directly affect both

charging availability and grid stability. As one study notes, “possible subversion or malicious end-points in the protocol can also lead to destabilization of power networks” [8].

ISO 15118 and PLC-Based Risks

ISO 15118 introduces convenience through Plug-and-Charge but also new risks. Researchers have demonstrated that unencrypted PLC traffic can be eavesdropped on, charging sessions can be hijacked, and wireless disruption attacks such as *Brokenwire* can abort CCS charging sessions at distances up to 47 meters [9].

EVCSMS Vulnerabilities

A comprehensive analysis of EV Charging Station Management Systems (EVCSMS) reveals widespread high-severity vulnerabilities, including SQL injection, cross-site scripting, server-side request forgery, and cross-site request forgery, that allow remote compromise of charging stations and their management platforms [10].

CAN as a Critical Cyber-Physical Vulnerability

Across the EVCS ecosystem, the Controller Area Network emerges as one of the most critical and under-secured components linking EVs, EVSEs, and the grid.

CAN as an Exposed Interface

Several charging standards, including CHAdeMO, GB/T, and some NACS implementations, extend CAN signaling outside the vehicle, exposing it at the charging port [4] [11] [12]. This creates a direct cyber-physical bridge between external infrastructure and in-vehicle ECUs.

CAN's Structural Weaknesses

CAN lacks authentication, encryption, message integrity, and sender identification, leaving it fundamentally unable to verify the legitimacy of transmitted frames. These structural weaknesses enable a wide range of attacks, including spoofing, replay, false-data injection, and denial-of-service, demonstrated extensively across prior studies [13] [14] [15] [16] [17] [18] [19].

CAN Compromise as a Gateway to Grid-Level Risk

Manipulated CAN data can drive converters into unsafe operating regions [20], falsify battery telemetry [13] [14], disrupt charging-station control logic [11], and even propagate malicious load modulation into the grid [19]. As EV adoption increases and CAN-based interfaces remain exposed, the risks associated with unsafe or malicious CAN messages grow accordingly.

Motivation for Hardware-Enforced CAN Safety

Software-only CAN intrusion-detection systems are fundamentally insufficient because a compromised ECU can still transmit arbitrary frames, software defenses can be bypassed or disabled, and EVCS power-electronics controllers require deterministic, cycle-accurate safety enforcement. At the same time, no existing approach provides the real-time, deterministic limit checking of voltage, current, and power requests needed for safe operation, nor do current solutions offer hardware-accelerated validation of CAN identifiers and payloads, isolation from compromised EVs or EVSE components, protection of converters and battery interfaces, or prevention of malicious load modulation that could propagate into the electric grid.

Motivation for this thesis

The rapid expansion of EV charging infrastructure, combined with the increasing exposure of CAN-based communication pathways, creates a pressing need for secure, low-latency, hardware-enforced protections at the EV–EVSE interface. This motivates the central contribution of this thesis: a hardware-accelerated CAN limit-checking module that enforces safety constraints at the boundary between EV and EVSE. By validating CAN messages in hardware before they reach the power-electronics controller, the proposed accelerator provides a trusted, deterministic, and tamper-resistant mechanism for ensuring safe charging behavior.

RELATED WORK

Electric vehicle charging systems have become a focal point for cybersecurity research as electrification accelerates and charging infrastructure becomes deeply integrated with transportation and grid operations. Prior work spans protocol-level security, EVSE and backend vulnerabilities, CAN-based automotive attacks, hardware-level defenses, and grid-level cyber-physical risks. This chapter reviews the state of the art across these domains, highlighting the strengths and limitations of existing approaches and identifying the gap that motivates the hardware-accelerated CAN limit-checking module developed in this thesis.

Protocol-Level Security in EV Charging Systems

OCPP Security Weaknesses and Proposed Enhancements

The Open Charge Point Protocol is the dominant communication standard between charging stations and central management systems. Alcaraz et al. identify structural weaknesses in widely deployed versions such as OCPP 1.6, including insufficient authentication, weak TLS configurations, and susceptibility to man-in-the-middle attacks [8]. Their analysis shows that compromised OCPP endpoints can manipulate charging rates, falsify metering data, or disrupt smart-charging schedules, with potential grid-level consequences. Proposed mitigations include stronger certificate-based authentication, secure commissioning, and end-to-end message integrity protections.

ISO 15118 and PLC-Layer Vulnerabilities

ISO 15118 introduces Plug-and-Charge functionality and high-bandwidth communication over power-line communication. Multiple studies demonstrate that real-world implementations

remain vulnerable. The *Brokenwire* attack shows that CCS charging sessions can be aborted at distances up to 47 meters by injecting a crafted wireless interference signal that disrupts PLC communication [9]. Other work highlights risks such as eavesdropping on unencrypted PLC traffic and session hijacking [11]. These findings motivate physical-layer hardening, improved PLC interference detection, and more robust session-management logic.

EVCSMS and Backend Platform Vulnerabilities

Nasr et al. provide one of the most comprehensive analyses of EV Charging Station Management Systems (EVCSMS), revealing widespread high-severity vulnerabilities, including SQL injection, cross-site scripting, server-side request forgery, and cross-site request forgery, that enable remote compromise of charging stations and their management platforms [10]. Because EVCSMS platforms control authentication, billing, scheduling, and remote operation, these weaknesses allow attackers to manipulate charging behavior, disable stations, or take full administrative control. Their disclosure resulted in multiple CVEs, underscoring the immaturity of backend EVCS security.

Vailoces et al. similarly show that insecure APIs, outdated OCPP deployments, and cloud misconfigurations expose EVSE networks to account takeover, remote charger manipulation, and denial-of-service conditions [11]. These studies collectively demonstrate that backend and cloud-connected EVCS components remain a major attack surface.

Cyber-Physical Risks in DC Fast Charging Systems

Idaho National Laboratory conducted one of the most detailed experimental analyses of DC fast-charging vulnerabilities. Their work shows that once an attacker gains access, either

physically or through a compromised EV, the internal communications network of a DCFC can be manipulated using off-the-shelf tools [5]. Attackers can disrupt coordination among power-electronics modules, falsify HMI charging data, or force abrupt load transitions, producing unstable power factor, elevated current THD, and erratic load behavior.

A key finding is that many DCFCs rely on a single internal communications network linking the HMI, power-electronics controllers, cooling systems, and protocol modules. This architecture means that compromising any one component, including through a malicious EV, can provide a pathway to manipulate charger behavior or propagate malware between vehicles and infrastructure [5].

Importantly, INL demonstrates that CHAdeMO-based charging exposes a CAN bus pathway between the EV and the charger, enabling firmware extraction, reflashing, and message manipulation directly over CAN. This establishes CAN as a critical cyber-physical vulnerability in modern DC fast-charging systems.

Grid-Level Cyber-Physical Threats from EVCS Compromise

Load-Altering Attacks Using Public Data

Acharya et al. show that publicly available data, such as smartphone-app charging-station usage and utility-published grid topology, can be combined to infer where high-wattage EV loads are located and when they are active [6]. Using this information, attackers can design data-driven load-altering attacks capable of destabilizing grid frequency by coordinating malicious charging behavior across many EVs.

MaDEVIoT Attacks and Urban Grid Instability

A related study introduces *MaDEVIoT* attacks, demonstrating that coordinated cyberattacks on Internet-connected EVCSs can manipulate large amounts of charging demand simultaneously, creating abrupt load changes capable of destabilizing urban power systems [7]. Using Manhattan as a case study, the authors show that compromising a single EVCS operator could trigger cascading outages by 2030.

EVs as Distributed Attack Vectors

Ahmed and Dow highlight how EVs, especially when integrated into vehicle-to-grid (V2G) programs, can serve as distributed attack vectors capable of propagating malware between charging stations and utilities [19]. Their analysis shows that compromised EVs can coordinate load-altering attacks, falsify charging requests, or disrupt grid-interactive services.

CAN-Based Vulnerabilities in EV Charging Ecosystems

CAN Exposure Through Charging Standards

Multiple studies show that several charging standards, including CHAdeMO, GB/T, and some NACS implementations, extend CAN signaling outside the vehicle, exposing it at the charging port [4] [11] [12] [21]. This creates a direct cyber-physical bridge between external infrastructure and in-vehicle ECUs.

Shen et al. demonstrate that physical-layer spoofing of CP/CC signals can enable CAN injection attacks that bypass higher-layer authentication entirely [12]. Their PORTulator device shows that analog authentication mechanisms in major charging standards can be trivially spoofed, enabling forged CAN messages to reach safety-critical ECUs.

CAN as a Vehicle-Level Attack Surface

A large body of automotive security research demonstrates that CAN's lack of authentication, encryption, and sender verification enables full vehicle compromise.

- Koscher et al. show that once an attacker reaches the CAN bus, they can control braking, steering, engine behavior, and instrument clusters [22].
- Checkoway et al. demonstrate remote CAN compromise via Bluetooth, telematics, and infotainment systems [23].
- Keen Security Lab shows that BMW's infotainment and telematics systems can be exploited to inject arbitrary CAN frames across multiple domains [24].

These studies establish that CAN is a systemic vulnerability in modern vehicles.

CAN-Based Attacks Relevant to EV Charging

Several works directly connect CAN vulnerabilities to EV charging:

- Acharya et al. show that CAN manipulation can serve as an entry point for demand-side grid attacks [13].
- Hamdare et al. highlight that CAN-based chargers inherit CAN's lack of authentication and integrity [4].
- Chandwani et al. show that CAN manipulation can drive onboard chargers (OBCs) into unsafe operating regions, causing over-voltage, over-current, or thermal stress [20].
- Sandia National Laboratories identify CAN as an exposed interface in CHAdeMO, Tesla, and GB/T charging systems [25].

Collectively, these studies demonstrate that CAN compromise can propagate from the vehicle into the EVSE and ultimately into the grid.

Intrusion Detection and Prevention for CAN Networks

Software-Based CAN Authentication and IDS

Several works propose software-only mechanisms to authenticate CAN messages or detect anomalies:

- Lin and Sangiovanni-Vincentelli propose pairwise symmetric keys, message counters, and selective MACs to authenticate CAN traffic under strict bandwidth constraints [16].
- Khandelwal and Shreejith propose a real-time zero-day intrusion detection system implemented on FPGA, using timing and payload analysis to detect spoofing attacks [17].
- Basnet and Ali propose deep-learning-based IDS for 5G-enabled EVCS environments, detecting false-data injection and DDoS attacks [26].

While these approaches improve detection, they remain vulnerable because a compromised ECU can still transmit arbitrary frames.

Hardware-Based CAN Security

Hardware-level approaches offer stronger guarantees:

- Elend and Adamson propose a security-enhancing CAN transceiver that enforces whitelisting, error-flag injection, arbitration-loss detection, and rate limiting directly at the physical layer [27].
- Araujo-Filho et al. propose a low-latency intrusion-prevention system capable of detecting anomalies within the first few data bytes of a frame and forcing CAN error states to discard malicious transmissions before completion [15].
- Iehira et al. demonstrate a combined bus-off + spoofing attack that bypasses software-based IDS, motivating hardware-level defenses [18].

These works show that hardware-based CAN protection is feasible and effective, but none address the EVCS-specific need for deterministic limit checking of charging parameters.

EVCS-Specific Cybersecurity Frameworks and Analyses

Early EVCS Security Frameworks

Falk and Fries provide one of the earliest systematic examinations of EVCS cybersecurity, identifying foundational requirements such as mutual authentication, integrity protection, secure firmware updates, and non-repudiation [28]. Their work highlights the need for secure communication across the EV–EVSE–backend chain.

Reeh et al. develop a structured scoring methodology for EVCS cyber-physical risks, showing that even low-cost attacks such as replay or ARP spoofing can disrupt charging coordination or overload distribution equipment [29].

System-Level EVCS Security Models

Pratt and Carroll propose the Vehicle Charging Infrastructure Security (VCIS) framework, emphasizing isolation of communication pathways, continuous monitoring, and coordinated response across OEMs, EVSE vendors, and utilities [21].

Sandia National Laboratories provide a comprehensive federal threat model showing that compromised EVSE can manipulate authorization, firmware, metering, or load-management signals, potentially affecting grid stability at both distribution and transmission levels [25].

False-Data Injection and Federated Learning Defenses

Chen and Guo analyze false-data injection (FDI) attacks targeting EV charging operations in AC/DC hybrid distribution systems, showing how manipulated measurements can

disrupt both vehicle safety and grid stability [30]. They propose a federated-learning-based anomaly detector deployed across distributed EVSE edge devices.

Summary and Research Gap

Existing research demonstrates that EV charging systems face a multilayered security challenge: core communication protocols such as OCPP and ISO 15118 contain exploitable weaknesses, while EVCS management platforms expose large remote attack surfaces that adversaries can leverage for widespread compromise. At the physical layer, DC fast-charging systems rely on insecure internal networks and often expose CAN pathways directly to external interfaces, making CAN a systemic vulnerability across both vehicles and EVSE. Although numerous software-based intrusion-detection mechanisms have been proposed, they cannot prevent a compromised ECU from transmitting malicious frames, nor can they guarantee the deterministic, cycle-accurate behavior required by EVCS power-electronics controllers. Hardware-based CAN defenses exist, but they are designed for automotive domains and are not tailored to the timing, safety, and isolation requirements of EV charging infrastructure. Critically, no existing solution provides real-time, deterministic, hardware-enforced limit checking of charging parameters at the EV–EVSE boundary, leaving a significant gap in the protection of both vehicle systems and grid-interactive charging equipment.

This gap motivates the central contribution of this thesis: a hardware-accelerated CAN limit-checking module that enforces safety constraints for EVCS power-electronics controllers, providing a trusted boundary between EV and EVSE.

EXPLINATION OF CAN PROTOCOL

The Controller Area Network (CAN) is a serial communication protocol developed to support reliable, deterministic data exchange in distributed embedded systems. Originally introduced by *Bosch*, a leading German engineering and automotive-technology company, for automotive applications, CAN has since become a foundational technology across transportation, industrial automation, and safety-critical control domains [31]. Its design emphasizes robustness through multi-master access, non-destructive arbitration, synchronized bit-level communication, and comprehensive error-handling mechanisms [31]. The CAN 2.0B standard defines the protocol's physical signaling and data-link behavior, while later ISO standards specify the electrical characteristics, wiring requirements, and transceiver behavior needed for practical deployment [31]. The following sections summarize the key elements of the CAN physical layer, frame structure, error-handling mechanisms, and arbitration strategy as presented in the standard.

Physical Layer

The CAN physical layer defines how bits are encoded, timed, and synchronized on the shared bus. CAN uses Non-Return to Zero (NRZ) encoding with two logical states, dominant and recessive, where dominant bits override recessive bits, enabling multi-master operation and arbitration [31]. Bit timing is divided into four segments (synchronization, propagation, phase 1, phase 2), allowing nodes to compensate for propagation delays and maintain synchronization [31]. To ensure periodic transitions for clock alignment, CAN employs bit stuffing, inserting a complementary bit after five identical bits [31]. The bit time must be long enough to

accommodate propagation delays across the entire network, including controller, transceiver, and cable delays [31].

Transceivers interface the CAN controller with the differential bus, providing drive capability, protection, and error detection [31]. Bus length and bit rate are tightly coupled, and proper termination, typically 120-Ω resistors at both ends for high-speed CAN, is required to prevent reflections [31]. ISO 11898-2, ISO 11898-3, and SAE J2411 define the electrical characteristics and topologies for high-speed, fault-tolerant, and single-wire CAN networks [31].

Message Frame Formats

CAN communication is organized into structured message frames. The primary frame type is the data frame, which includes the Start of Frame (SOF), Arbitration Field, Control Field, Data Field (0–8 bytes), CRC Field, ACK Field, and End of Frame (EOF) [31]. CAN supports both standard (11-bit) and extended (29-bit) identifiers [31]. Arbitration occurs during the identifier field using dominant/recessive logic, ensuring that the highest-priority message wins without corrupting the bus [31].

Standard Data Frame

A standard CAN data frame (Figure 3) begins with a SOF bit, which is dominant and provides a hard-synchronization point for all nodes on the bus [32]. The SOF is followed by the Arbitration Field, consisting of the 11-bit identifier and the Remote Transmission Request (RTR) bit. The RTR bit distinguishes data frames (RTR = dominant) from remote frames (RTR = recessive) [32].

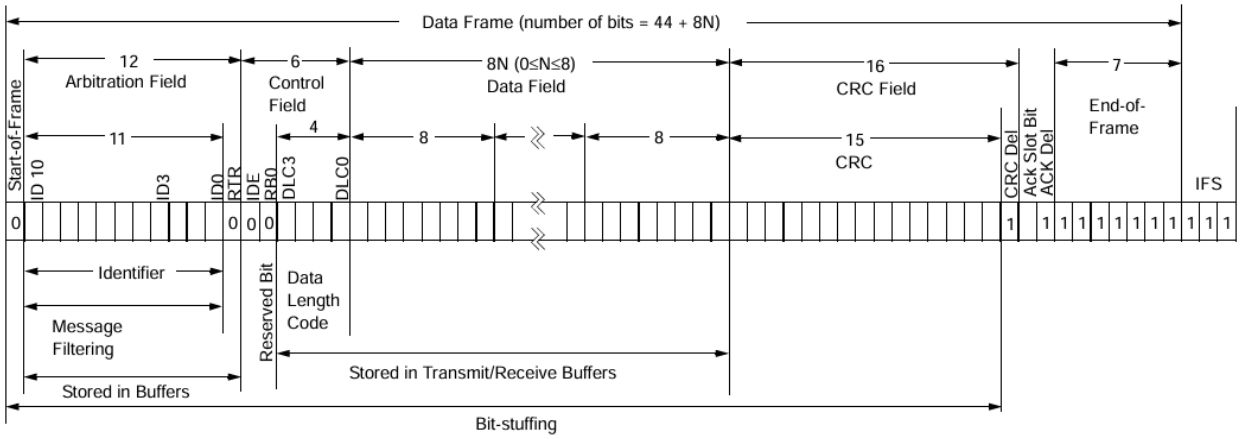

Figure 3: Standard data frame [32].

Next is the Control Field, which contains six bits: the Identifier Extension (IDE) bit, a reserved bit (RB0), and the Data Length Code (DLC). For standard frames, the IDE bit must be dominant, indicating that the frame uses the 11-bit identifier format. The DLC specifies the number of data bytes (0–8) included in the frame [32].

The Data Field follows, containing 0–8 bytes of application data. After the data field, the Cyclic Redundancy Check (CRC) field provides error detection using a 15-bit CRC sequence and a recessive CRC delimiter [32].

The Acknowledge (ACK) Field consists of two bits: the ACK Slot and the ACK Delimiter. During the ACK Slot, the transmitter sends a recessive bit; any node that has received the frame correctly overwrites this with a dominant bit, acknowledging successful reception regardless of whether the node accepts the message into its buffers [32].

Finally, the frame ends with the End-of-Frame (EOF) sequence of seven recessive bits, followed by the interframe space before the next transmission may begin [32].

Extended CAN Data Frame

In addition to the standard 11-bit identifier format, CAN 2.0B also supports extended data frames, which use a 29-bit identifier to provide a significantly larger addressing space (Figure 4) [32]. An extended frame begins with the dominant SOF bit, followed by a 32-bit Arbitration Field. The first 11 bits of this field contain the most significant portion of the identifier (Base ID), followed by the Substitute Remote Request (SRR) bit, which is always recessive, and the IDE bit, which is recessive to indicate that the frame uses the extended format [32]. These two recessive bits ensure that standard frames always win arbitration when both formats are present on the bus, since a standard frame transmits a dominant IDE bit at the same position [32]. After the SRR and IDE bits, the remaining 18 bits of the identifier (Extended ID) are transmitted, followed by the RTR bit, which distinguishes data frames (dominant) from remote frames (recessive) [32].

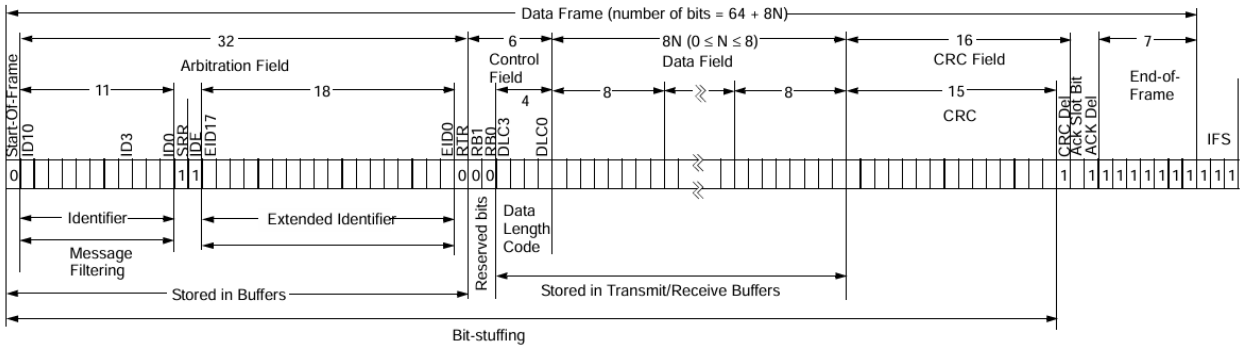

Figure 4: Extended data frame [32].

The Control Field in an extended frame is structurally identical to that of a standard frame, consisting of two dominant reserved bits and the four-bit DLC that specifies the number of data bytes (0–8) [32]. The remainder of the extended frame, including the Data Field, CRC

sequence and delimiter, Acknowledge Field, End-of-Frame, and intermission, is formatted the same as in the standard data frame [32]. By splitting the 29-bit identifier into an 11-bit base and an 18-bit extension, the CAN protocol ensures that the IDE bit appears in the same bit position for both frame types, enabling seamless coexistence of standard and extended frames on the same network [32].

Error Detection and Fault Confinement

CAN incorporates multiple layers of error detection, including bit monitoring, bit stuffing checks, frame format checks, CRC verification, and acknowledgment checks [31]. When a node detects an error, it transmits an error flag, causing all nodes to discard the frame and wait for retransmission [31]. Nodes maintain Transmit Error Counters (TEC) and Receive Error Counters (REC) to track their error status [31]. Depending on these counters, a node may be in an error-active, error-passive, or bus-off state [31]. A bus-off node disconnects itself from the network to prevent a faulty device from degrading overall communication. This fault-confinement mechanism is a key contributor to CAN's robustness in safety-critical environments [31].

Arbitration and Medium Access

CAN uses a multi-master, non-destructive arbitration mechanism to control access to the bus. When the bus becomes idle, any node may begin transmitting [31]. If multiple nodes transmit simultaneously, arbitration occurs bit-by-bit during the identifier field. Because dominant bits override recessive bits, a node that transmits a recessive bit but reads a dominant bit loses arbitration and immediately stops transmitting [31]. The message with the numerically

lowest identifier wins and continues uninterrupted [31]. This arbitration method ensures deterministic access, bounded latency for high-priority messages, and efficient bandwidth utilization even under heavy network load [31].

OVERVIEW OF EV CHARGING STATION COMPONENTS AND EXPERIMENTAL SETUP

EV Charging Station Architecture

Electric vehicle charging stations are sophisticated cyber-physical systems that integrate power-electronic hardware, embedded controllers, communication interfaces, and cloud-connected management platforms. Their design reflects the dual role EVCSs play at both the grid edge and the network edge, serving simultaneously as power-conversion equipment and as networked computing devices [33]. This position makes EVCSs essential to the broader clean-energy transition, particularly as bidirectional charging enables electric vehicles to operate as distributed energy resources (DERs), while also exposing them to operational and cybersecurity risks that must be carefully managed [33].

Because these systems combine electrical, computational, and communication subsystems, it is useful to decompose an EVCS into its major functional domains. The Pacific Northwest National Laboratory (PNNL) taxonomy organizes EVCS functionality into ten high-level control categories, each corresponding to a major subsystem or component class within a modern charging station.

Safety-Critical Components

Safety functions are primarily implemented within the power electronics subsystem, which includes the AC/DC converter, DC/DC stages, contactors, relays, and associated sensing hardware. These components enforce electrical protection and ensure compliance with safety standards. Key capabilities include ground-fault detection, overcurrent protection, arc-fault

detection, insulation monitoring, thermal monitoring of cables and power modules, and emergency-stop mechanisms. Pilot-signal supervision, monitoring the proximity pilot and control pilot, ensures that charging halts immediately if connector integrity is lost [33]. These components form the foundational protection layer of the EVCS and operate in concert with higher-level functions that govern user access, authorization, and charging initiation.

Authorization and User-Access Components

Authorization functions are handled by the system control unit, user-interface hardware, and the charging-station management system (CSMS). These components regulate access to the charger through mechanisms such as RFID readers, NFC payment terminals, mobile-app authentication, and ISO 15118 Plug-and-Charge certificate-based authentication. Some systems also support AutoCharge, which authenticates based on the EV's MAC address. Local fallback authorization allows charging to continue when CSMS connectivity is lost, ensuring service continuity [33]. Once a charging session is authorized, the EVCS must accurately measure and account for delivered energy.

Billing and Metering Components

Accurate billing is enabled by the revenue-grade energy meter, integrated current and voltage sensors, and the CSMS. These components measure delivered energy, apply tariff logic, and generate transaction records. When the primary meter is unavailable, the EVCS may approximate energy transfer using power-electronics telemetry to maintain billing continuity [33]. These components ensure regulatory compliance and financial accuracy. Accurate metering depends on reliable communication between the EVCS, the vehicle, and backend systems, making the communication subsystem another critical component.

Network and Communication Components

Communication subsystems enable the EVCS to interface with EVs, backend systems, and site-level controllers. These include support for protocols such as OCPP, OCPI, ISO 15118, and DIN SPEC 70121, as well as networking interfaces such as Ethernet, Wi-Fi, and cellular modems. Network services, including DNS, NTP, VPN, and TLS, provide secure and reliable connectivity. Some installations employ redundant or multipath networking to enhance resilience [33]. These communication pathways support not only user-facing functions but also internal coordination between the supervisory controller and the power-electronics subsystem.

Load-Management and Power-Control Components

Load-management functions are implemented through the power converter, site energy controller, and EVCS supervisory controller. These components regulate charging power to prevent overloads and optimize energy usage. Capabilities include smart-charging algorithms, frequency-watt and droop-response controls, ride-through and trip-off thresholds, and automatic restoration of charging after disturbances [33]. These functions enable EVCSs to integrate with microgrids, demand-response programs, and renewable-energy systems. As EVCSs evolve toward grid-interactive operation, many systems now incorporate bidirectional power-transfer capabilities

Bidirectional Power Transfer (BPT) and DER Components

Chargers supporting vehicle-to-grid (V2G), vehicle-to-home (V2H), or vehicle-to-building (V2B) operation incorporate additional DER-oriented components. These include isolation mechanisms to prevent unintentional backfeeding, DER-control interfaces, and power-electronics controls for active/reactive power, ramp-rate limits, and power-factor settings.

Grid-forming and grid-following modes allow the charger to either establish local voltage and frequency or synchronize with an existing grid waveform [33]. These components enable EVs to function as flexible grid resources. Supporting these advanced functions requires continuous monitoring and diagnostic capabilities within the EVCS.

Fault-Detection and Diagnostic Components

Diagnostic components include sensors, embedded controllers, and data-logging subsystems that continuously monitor voltage, current, temperature, and internal states. These components record errors, support predictive maintenance, and trigger automatic shutdown in response to critical faults [33]. They are essential for maintaining reliability and reducing downtime. Because EVCSs are widely deployed and remotely managed, these diagnostic functions must integrate with supervisory and cloud-based control systems.

Remote Monitoring and Control Components

Remote-management capabilities are enabled by the CSMS, communication modules, and supervisory controller. These components allow operators to perform remote diagnostics, issue control commands (start, stop, reset), and monitor charger health and performance in real time [33]. They are particularly important for large-scale EVCS deployments where on-site maintenance is impractical. The distributed and networked nature of these systems introduces cybersecurity requirements that span hardware, firmware, and software layers.

Cybersecurity Components

Cybersecurity controls span hardware, firmware, and software layers. Key components include secure key storage, certificate-management systems, TLS/VPN communication modules,

secure-boot mechanisms, firmware-integrity verification, authentication and access-control systems, remote attestation, and firewall configurations [33]. These components protect EVCSs from cyber threats that could compromise safety, billing integrity, or grid stability. Coordinating all of these subsystems requires a supervisory control unit that manages communication, safety, and power-electronics behavior.

Supervisory Control Unit

The system control unit serves as the supervisory controller that coordinates all EVCS subsystems. While it does not execute every control function directly, it orchestrates the behavior of specialized components such as the power electronics, communication modules, safety hardware, metering systems, and DER controllers [33]. This architecture ensures modularity, scalability, and clear separation of responsibilities across the EVCS.

Understanding the roles and interactions of these components provides the foundation for constructing a representative experimental environment. To study these internal control pathways and evaluate hardware-accelerated CAN safety mechanisms, this work relied on research platforms and software models provided by PNNL. These platforms enabled controlled observation of EVCS communication behavior and informed the design of the hardware accelerator described in the following sections.

PNNL's Role and Provided Research Platforms

Pacific Northwest National Laboratory supported this research through its ongoing program in electric vehicle charging infrastructure cybersecurity. As part of this collaboration, PNNL provided access to specialized EVCS research platforms designed to expose the internal

communication pathways of modern charging stations and to enable controlled experimentation with power-electronics control behavior. These platforms were developed to support national-scale EVCS security research and reflect real-world architectures used in both Level-2 and DC fast-charging systems.

Two complementary resources were supplied. The first was a compact EVCS subsystem that provided direct visibility into the interactions between a supervisory controller, a power-electronics controller, and associated sensing hardware. This platform enabled detailed observation of I^2C-based control signaling and informed the selection of the power-controls microcontroller as the most suitable integration point for a hardware-accelerated CAN safety mechanism.

The second resource was a DC fast-charging emulator consisting of a power-electronics box, a programmable DC load, and a pair of Python-based control scripts that model the behavior of the system controller and the power-electronics controller. Although project funding ended before the physical hardware could be incorporated into the testbench, the software components were delivered early and served as the behavioral specification for the C-based power-electronics controller used in this work.

Together, these PNNL-provided platforms formed the foundation of the experimental environment. They supplied realistic control-path behavior, representative CAN-based command traffic, and a clear architectural model of how supervisory and power-electronics controllers interact within an EVCS. This context guided the design of the hardware accelerator and shaped the structure of the experimental setup described in the following sections.

Emulator #1: EVCS Box for Signal-Path Discovery

The first research platform provided by PNNL was a compact Level-2 EVCS subsystem designed to expose the internal communication pathways between the supervisory controller and the power-electronics hardware. Unlike a full charging station, this emulator emphasized transparency rather than power capability. Its architecture included a Raspberry Pi - based supervisory controller, a *Vector* vSECC communication module, a *TI* BQ25720EVM linear voltage controller, a DFRobot wattmeter, and a dedicated power and signal distribution board, all connected through standard CP/PP/PE signaling.

This platform offered a unique advantage: the I²C bus connecting the power-electronics controller to the linear voltage controller was fully accessible. Monitoring this bus made it possible to observe the real timing and structure of control messages without needing to reverse-engineer firmware or rely on simulated behavior. By sniffing the I²C traffic, it was possible to identify when the system issued start-charging commands, how target voltages and currents were encoded, and how the supervisory controller sequenced its interactions with the *TI* converter.

These observations provided the first concrete insight into how an EVCS power-electronics subsystem behaves internally. They also revealed that the power-controls microcontroller, responsible for interpreting CAN-based requests and translating them into I²C-level commands, was the most practical and security-relevant integration point for a hardware-accelerated CAN safety mechanism. To validate this, a proof-of-concept demonstration was developed in which a single board computer from *Resilient Computing*, called RadPC,

monitored the I²C bus and successfully decoded charging-session transitions and power-setpoint commands in real time (Figure 5).

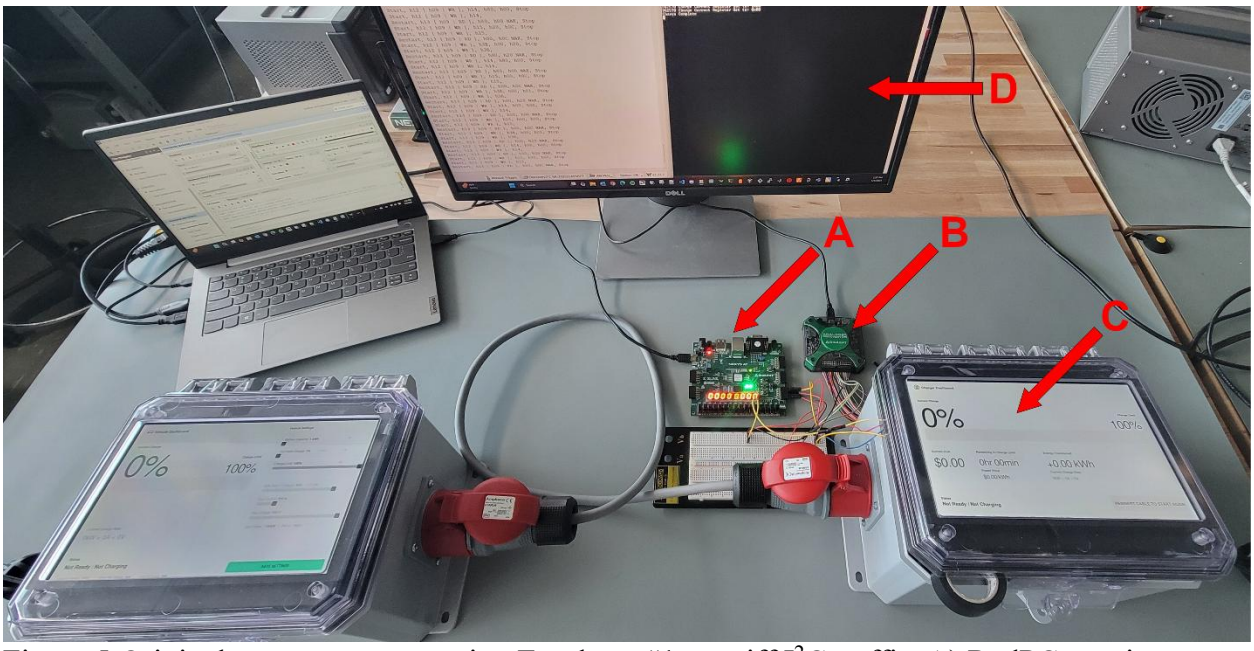

Figure 5:Original prototype setup using Emulator #1 to sniff I²C traffic. A) RadPC running on a Nexys A7 board. B) Analog Discovery II used to monitor I2C bus for comparison to RadPC sniffing. C) EVCS emulator. D) Serial monitor used to monitor RadPC sniffing.

The clarity gained from this emulator established the foundation for the remainder of the experimental environment. While this first platform illuminated the low-level control interactions within a Level-2 EVCS, the second PNNL emulator extended this perspective to the more complex control architecture of a DC fast-charging system.

Emulator #2: DCFC Power Electronics Box and Software Control Model

The second research platform provided by PNNL extended the perspective gained from the Level-2 EVCS subsystem into the domain of direct-current fast charging. Whereas the first emulator exposed low-level control signaling within a relatively simple architecture, the DCFC

Power Electronics Box was designed to replicate the full control stack of a modern fast-charging system, including supervisory coordination, power-electronics regulation, and realistic load behavior. This emulator consisted of three primary components: the *BelayBox* supervisory controller, the PE Box containing the power-electronics controller and converter hardware, and a programmable DC load that emulated an EV battery. Together, these elements formed a complete DCFC control loop capable of initiating charging sessions, interpreting CAN-based power requests, and regulating output voltage and current through a *TI* BQ25720EVM converter.

Although project funding ended before the physical hardware could be incorporated into the accelerator testbench, PNNL delivered the software components that drive the emulator's control architecture. These included two Python-based controllers, each running on a Raspberry Pi equipped with an RS485-based CAN interface. The System Controller script generated CAN messages that requested specific voltage and current targets, mirroring the behavior of a supervisory controller in a commercial DCFC. The Power-Electronics Controller script interpreted these CAN requests, executed the charging-session state machine, and issued I^2C commands to the *TI* converter to regulate output power. Together, these scripts provided a complete and executable model of the DCFC control pathway.

These software models became central to the development of the experimental environment. Because they captured the exact message formats, timing behavior, and state transitions used in the physical emulator, they served as the behavioral specification for the C-based power-electronics controller implemented on a RadPC Hydra single board computer. This allowed the accelerator to be evaluated within a realistic control context even without access to the physical DCFC hardware. The emulator's software therefore played a pivotal role:

it bridged the gap between the simplified Level-2 subsystem examined in the first emulator and the more complex, high-power control architecture targeted by the hardware accelerator.

This software-defined control model also provided a natural transition into the next stage of the experimental setup, where the Python-based PE controller was re-implemented in C and integrated with RadPC Hydra to form the firmware environment in which the CAN limit-checking accelerator operates.

Power-Electronics Controller Simulation

The software environment accompanying the second PNNL emulator provided a complete behavioral model of a DC fast-charging control loop, and this model became the foundation for the power-electronics controller used in this research. The Python-based System Controller and Power-Electronics Controller scripts captured the exact CAN message formats, timing behavior, and state transitions used in the emulator's physical hardware. Because these scripts implemented the same supervisory logic and converter-control routines found in commercial DCFC systems, they offered a realistic and well-defined specification for the controller that would interface with the hardware accelerator.

Re-implementing this control logic in C for RadPC created a deterministic and self-contained firmware environment that mirrored the behavior of the emulator's PE controller. The C-based controller interprets CAN-based voltage and current requests, executes the same charging-session state machine defined in the Python model, and generates the internal control actions that would normally be issued over I^2C to the *TI* BQ25720EVM. Although the physical converter hardware was not part of the testbench (Figure 6), the controller preserves the structure

and timing of these interactions, allowing the accelerator to be evaluated under realistic operating conditions.

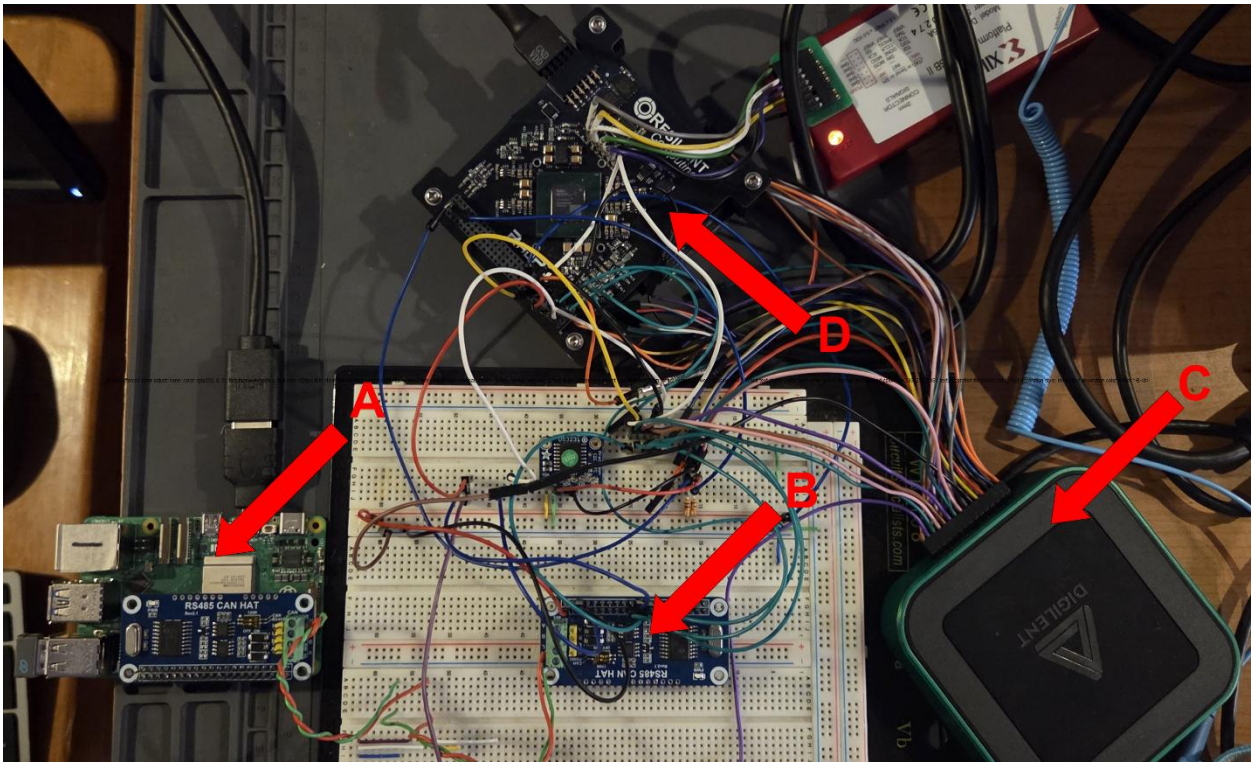

Figure 6: External testbench used to validate the C-based power-electronics controller and CAN-monitoring accelerator. The setup includes a Raspberry Pi (A) with an RS485 CAN HAT generating supervisory CAN traffic, a second RS485 CAN HAT (B) connected to the RadPC MCU for CAN reception, an Analog Discovery 3 (C) used to observe SPI communication between the MCP2515 and the accelerator, and a RadPC Hydra (D) executing the controller firmware.

This simulation served two critical purposes. First, it ensured that the accelerator was tested against a control loop representative of an actual EVCS power-electronics subsystem rather than synthetic or simplified traffic. Second, it allowed RadPC to function simultaneously as the supervisory MCU and the host for the hardware accelerator, enabling tight integration between firmware and hardware. By grounding the controller in PNNL's reference model, the

simulation provided a faithful and reproducible environment for evaluating the accelerator's behavior within a realistic EVCS control architecture.

Hardware Accelerator Platform (RadPC Hydra)

RadPC Hydra served as the hardware platform that hosted both the C-based power-electronics controller and the CAN limit-checking accelerator. Its architecture provided the combination of determinism, fault tolerance, and FPGA flexibility required to integrate a hardware accelerator directly into the control loop defined in the previous section. Implementing the controller logic on RadPC created a unified environment in which firmware and hardware could interact at cycle-level granularity, enabling realistic evaluation of the accelerator's behavior within an EVCS-style control stack.

The RadPC-SBC-100 is built around an industrial-grade Xilinx Artix-7 200T FPGA and implements a set of redundant 32-bit RISC-V soft cores designed for radiation-tolerant operation. These cores are managed by patented recovery algorithms that automatically detect, isolate, and repair single-event effects, allowing the system to behave like a conventional embedded processor while maintaining high reliability. This architecture has been validated through multiple space missions, including deployments in low-Earth orbit and a lunar lander demonstration, making it a robust platform for safety-critical embedded applications.

Although RadPC Hydra was originally designed for radiation-tolerant space applications, the accelerator developed in this work does not rely on the platform's single-event-effect mitigation features. Instead, the design leverages RadPC's other major architectural advantage: its ability to host tightly integrated, memory-mapped hardware accelerators within the same FPGA fabric as the RISC-V soft cores. This capability allows custom logic to appear to the

firmware as a conventional peripheral with deterministic register-level access, eliminating the need for external buses or nondeterministic communication paths. By mapping the CAN-safety accelerator directly into RadPC's address space, the controller firmware can configure limits, forward parsed frame fields, and retrieve evaluation results with cycle-level precision. While the radiation-hardening features are not required for EVCS-style terrestrial deployments, RadPC's unified processor-plus-accelerator architecture provides an ideal environment for evaluating hardware-enforced safety rules and demonstrates how such accelerators could be incorporated into future embedded control platforms

RadPC's PC104 expansion connector enabled physical integration with the CAN interface used in the experimental setup. A custom adapter PCB allowed an RS485-based CAN HAT to be connected directly to RadPC (Figure 7), providing the physical CAN transceiver path required for communication with the simulated EVCS system controller. This configuration placed RadPC in the same architectural role as the power-electronics controller in a commercial charging station, completing the hardware foundation for the accelerator evaluation.

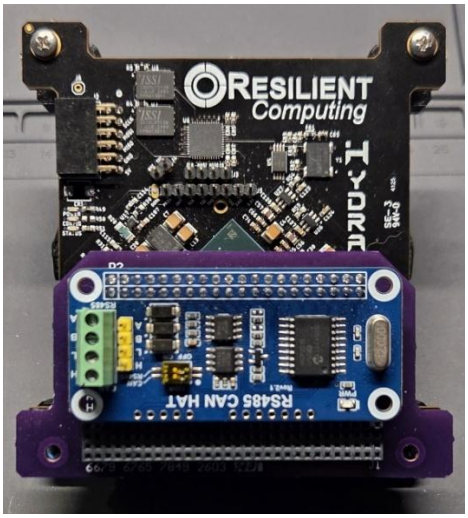

Figure 7: Resilient Computing's RadPC-SBC-100 with the RS485 CAN Hat connected via a custom PCB.

CAN Bus Integration and Test Environment

The overall communication architecture used in the experimental setup is shown in Figure 8, illustrating the interaction between the Raspberry Pi system controller, the MCP2515-based CAN interfaces, and the RadPC Hydra hosting the hardware accelerator.

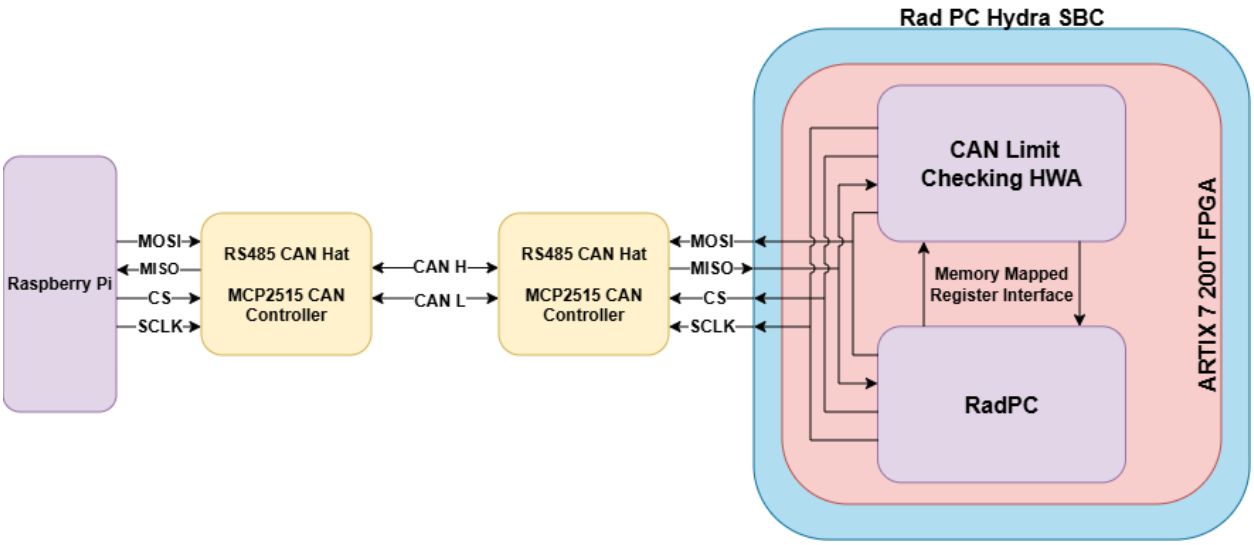

Figure 8: Conceptual block diagram of the experimental setup used to test the CAN limit checking hardware accelerator.

Integrating RadPC Hydra into a realistic EVCS communication loop required a physical CAN interface that matched the architecture used in both PNNL emulators. To accomplish this, RadPC was connected to an RS485-based CAN HAT that incorporates the *Microchip* MCP2515, a widely used stand-alone CAN controller that communicates with its host processor over SPI. The MCP2515 provides the CAN protocol engine, transmit and receive buffers, filtering logic, and interrupt signaling, while the HAT's RS485 transceiver handles the physical-layer differential signaling required for the CAN bus.

Because RadPC Hydra uses a custom RISC-V soft-core architecture and does not run a conventional operating system, a fully software-based MCP2515 driver was developed early in the project to enable initial CAN-bus testing on the RadPC Hydra. This driver implemented register access, controller initialization, bit-timing configuration, and both transmit and receive routines, allowing RadPC to participate in CAN communication before any hardware acceleration work began. When the hardware accelerator and FPGA-based receive pipeline were later introduced, the MCP2515's receive path was replaced entirely by hardware logic, enabling incoming CAN frames to be captured, parsed, and rule-checked directly within the FPGA fabric. The original software library remained in use for CAN transmission, where its deterministic SPI routines and simple control flow were sufficient for the final design. This division of responsibilities allowed the system to retain a lightweight, reliable software TX path while leveraging hardware acceleration for all RX-side processing.

A custom adapter PCB mechanically and electrically interfaced the RS485 CAN HAT with RadPC's expansion connector, routing SPI, interrupt, and power lines while maintaining proper signal integrity. The CAN-H and CAN-L lines from this interface were wired directly to a second RS485 CAN HAT attached to a Raspberry Pi 5 (Figure 9). The Raspberry Pi executed PNNL's System Controller Simulation, generating realistic CAN-based voltage and current requests that followed the same message structure and timing as the DCFC emulator's supervisory controller.

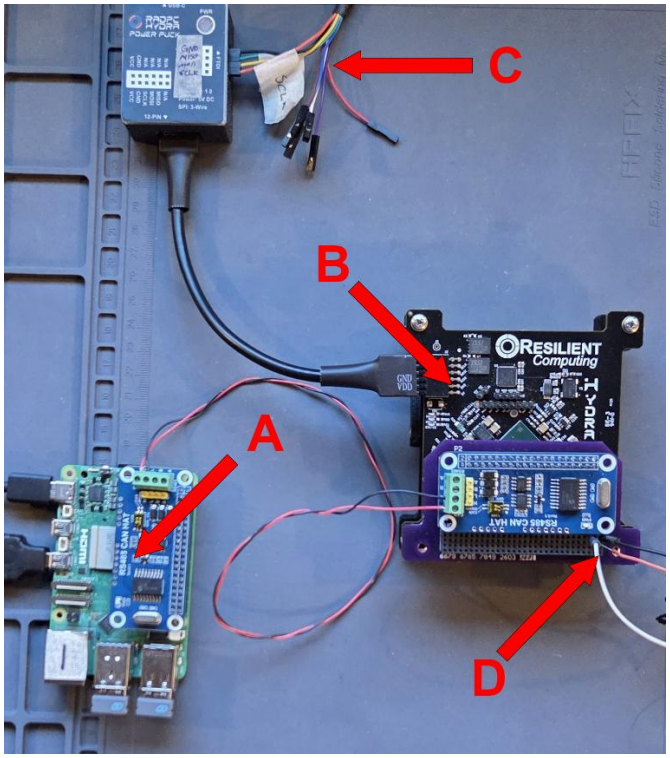

Figure 9: Final experimental testbed used for system-integration verification. The setup includes the Raspberry Pi with RS485 CAN HAT (A) generating supervisory CAN frames, a RadPC Hydra with its own RS485 CAN HAT (B) receiving and processing frames through the accelerator, the RadPC programming and power interface (C), and a UART connection to a host PC (D) for serial-terminal monitoring.

This configuration created a closed, reproducible communication loop in which RadPC received supervisory CAN commands, evaluated them using the hardware accelerator, and processed them through the C-based power-electronics controller. By combining a hardware MCP2515 interface, a custom RadPC driver, and PNNL's system-controller simulation, the test environment accurately reflected the communication behavior of a commercial EVCS power-electronics subsystem.

SYSTEM ARCHITECTURE

System Overview

The CAN-limit-checking subsystem is a dedicated hardware accelerator designed to validate, constrain, and safely process power-electronics control messages carried over the CAN bus. Its purpose is to ensure that voltage, current, and power requests remain within a configurable safety envelope before they reach the RadPC MCU, enabling deterministic enforcement of electrical and operational limits in real time. The subsystem operates autonomously, reacting to incoming CAN frames, evaluating them through a structured sequence of rule checks, and presenting a validated, bounded version of the message to the MCU through a memory-mapped register interface.

The accelerator is composed of several tightly integrated hardware components, each responsible for a specific stage of the data-processing pipeline:

- SPI Master – A synchronous byte-level serial engine that handles all low-level SPI signaling to the MCP2515. It generates the SPI clock, shifts data in and out, asserts chip-select, and guarantees protocol-correct timing for every byte transfer.
- MCP2515 Control FSM – A higher-level controller that sequences SPI transactions to retrieve CAN frames from the MCP2515. It issues the appropriate read commands, collects the CAN ID, DLC, and data bytes, and forwards them to the rule engine. It also manages SPI-bus ownership and ensures that frame retrieval is atomic and deterministic.
- Rule Engine – The subsystem's decision-making core. It parses the incoming frame, applies spoofing and whitelist rules, checks voltage, current, and power limits, performs clamping when necessary, and assembles the processed output. It also computes

requested power using a reciprocal-based arithmetic method and generates safety flags and fault codes.

- Reciprocal Lookup Table – A synchronous ROM containing precomputed fixed-point reciprocals used to replace hardware division during power-limit evaluation. It enables fast, deterministic computation of allowable current bounds derived from configured power limits.
- Register Bank – The hardware–software boundary. It stores configuration registers written by the MCU and latches processed outputs from the rule engine. It exposes these values to the MCU through a stable memory-mapped interface and asserts an interrupt when a new validated frame is ready.
- Top-Level CAN-Limit-Checking Module – The integration layer that connects all components. It routes SPI signals, coordinates the MCP2515 controller and rule engine, manages configuration distribution, and ensures that the accelerator behaves as a single cohesive peripheral from the MCU's perspective.

The overall architecture of the CAN limit-checking subsystem is shown in Figure 10, illustrating how the SPI master, MCP2515 controller, rule engine, reciprocal LUT, and register bank interact within the top-level module. Together, these components form a deterministic, pipelined processing chain: the MCP2515 signals the arrival of a new frame, the SPI master retrieves it under the direction of the MCP2515 controller, the rule engine evaluates and clamps the data using the reciprocal LUT, and the register bank presents the final results to the MCU. This architecture ensures that every CAN message is validated in hardware with predictable latency, providing a robust safety layer for power-electronics control.

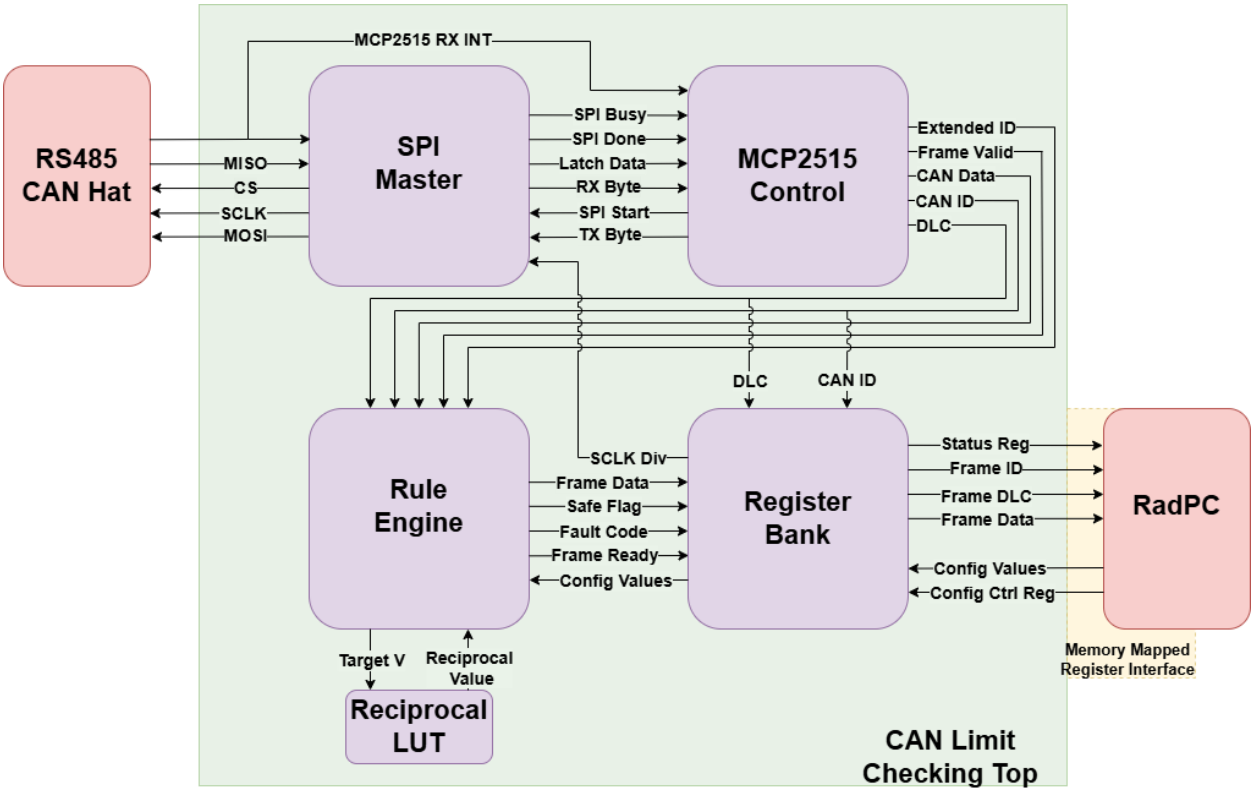

Figure 10: Conceptual block diagram showing the components of the CAN limit checking hardware accelerator.

SPI Master Component

The SPI Master component provides the low-level serial interface required to communicate with the MCP2515 CAN controller. It is responsible for generating the SPI clock, shifting transmit and receive data, asserting and deasserting chip-select, and coordinating byte-level handshaking with the MCP2515 control logic. Unlike the MCP2515 control component, which is protocol-aware, the SPI Master is intentionally protocol-agnostic: it performs only the electrical and timing functions of SPI Mode 0, leaving higher-level sequencing to the upstream controller. This separation ensures that the SPI Master remains a deterministic,

reusable hardware block capable of supporting any MCP2515 command sequence issued by the accelerator.

Component Interface

The SPI Master interfaces with three groups of signals: the system clock and reset, the control inputs from the MCP2515 control component, and the physical SPI pins connected to the MCP2515. The control interface includes a start signal, a transmit byte, and auxiliary indicators such as the extended-frame flag and interrupt-clear request, which are passed through for debugging and system-level observability. When a transaction is initiated, the SPI Master asserts its busy flag, drives chip-select low, and begins shifting out the provided transmit byte while simultaneously sampling the MISO line. As each byte completes, the component asserts a one-cycle done-byte pulse and raises the latch-data signal to indicate that the received byte is valid. Once the final byte in the sequence has been transmitted, the SPI Master deasserts chip-select, raises the done flag for one cycle, and returns to the idle state. Through these interfaces, the SPI Master provides a clean, byte-oriented handshake that the MCP2515 control component uses to retrieve CAN frames deterministically.

Internal Data Flow

Internally, the SPI Master implements a shift-register-based data path driven by a programmable clock divider. When a transaction begins, the transmit byte is loaded into an 8-bit shift register, and the receive register is cleared. The component then asserts chip-select and enters the shifting phase, during which the SPI clock toggles at a rate determined by the 16-bit divider value supplied by the MCU. On each rising edge of the internal SPI clock, the most significant bit of the transmit register is driven onto MOSI. On each falling edge, the component

samples the MISO line and shifts the received bit into the least significant position of the receive register. After eight such cycles, the receive register contains the full byte returned by the MCP2515, and the transmit register has been shifted out completely. The component then asserts the latch-data signal, increments its internal byte counter, and prepares for the next transmit byte. This process repeats until the MCP2515 control component has completed its command sequence. Once the final byte has been shifted, the SPI Master deasserts chip-select, pulses the done signal, and resets its internal counters in preparation for the next transaction.

Decision-Making Behavior

The decision-making behavior of the SPI Master is governed by a compact finite-state machine that ensures correct sequencing of load, shift, and completion phases. The component remains in the idle state until a start request is asserted, at which point it loads the transmit byte and asserts chip-select. During the shifting phase, the FSM advances only when the clock divider indicates that a full SPI clock edge has occurred, preventing timing irregularities and ensuring that each bit is sampled and driven at the correct moment. After eight bits have been shifted, the FSM transitions to a byte-completion state, where it asserts the done-byte and latch-data signals and increments the byte counter. The FSM continues this cycle until the upstream controller indicates that the final byte of the transaction has been transmitted. At that point, the FSM deasserts chip-select, pulses the done signal, and returns to idle. This structured decision-making process guarantees that every SPI transaction is executed with fixed timing, correct bit ordering, and clean byte-level boundaries, enabling the MCP2515 control component to reliably reconstruct CAN frames.

MCP2515 Control Component

The MCP2515 Control Component is responsible for retrieving complete CAN frames from the MCP2515 stand-alone CAN controller and presenting the decoded identifier, DLC, data bytes, and format information to the rule engine. Unlike the SPI Master, which provides a generic byte-level interface, the MCP2515 control block implements a protocol-aware finite-state machine (FSM) that issues the MCP2515's READ RX BUFFER command, sequences the required dummy reads, reconstructs the CAN identifier, and asserts a clean, single-cycle frame-valid strobe when a full frame has been captured.

This component forms the first stage of the accelerator's processing pipeline. It ensures that every CAN frame is retrieved deterministically, with correct byte ordering, and without requiring firmware involvement. The MCP2515 control block therefore acts as the hardware boundary between the external CAN bus and the internal safety-rule evaluation logic.

Component Interface

The MCP2515 control component interfaces with the MCP2515 CAN controller, the SPI master, and the downstream rule engine, acting as the bridge between the external CAN bus and the accelerator's internal processing pipeline. On the receive side, the component monitors the MCP2515's interrupt pin, which signals that a new frame is available in RXB0. Once the interrupt is detected, the component assumes full control of the SPI master, issuing the appropriate read command and retrieving each byte of the frame through a sequence of dummy SPI transactions. Throughout this process, the component relies on the SPI master's handshake signals to determine when a transmitted byte has completed and when a received byte is ready to be latched.

As the frame is retrieved, the MCP2515 control block assembles the identifier, DLC, and data bytes into internal registers and exposes them to the rule engine through dedicated output ports. The reconstructed identifier is presented on a 29-bit bus, with the component asserting an extended-ID flag when the EXIDE bit in the SIDL register indicates a 29-bit frame. The DLC is provided as a four-bit value, and the eight data bytes are delivered in two 32-bit words corresponding to bytes 0–3 and bytes 4–7. Once all fields have been captured, the component asserts a single-cycle frame-valid signal, indicating that the rule engine may safely consume the newly received frame. In parallel, the component generates a one-cycle clear-IRQ pulse to notify the MCU that the MCP2515's RX0IF flag should be cleared before the next frame arrives. Through these interfaces, the MCP2515 control component ensures that each CAN frame is delivered to the accelerator in a complete, correctly ordered, and timing-deterministic manner.

Internal Data Flow

The MCP2515 control component implements a deterministic, linear data-retrieval pipeline that begins when the MCP2515 asserts its interrupt line to indicate that a new CAN frame is available in receive buffer RXB0. Once the interrupt is detected, the component issues the MCP2515's READ RXB0SIDH command and then performs a sequence of dummy SPI reads to retrieve each field of the CAN frame in the order defined by the MCP2515 register map. The SIDH and SIDL bytes are captured first, followed by the EID8 and EID0 registers, allowing the component to reconstruct either an 11-bit or 29-bit identifier depending on the EXIDE bit in SIDL. After the identifier fields are read, the component retrieves the DLC byte and then the eight data bytes, storing them into two 32-bit internal registers corresponding to bytes 0–3 and bytes 4–7. Each byte is latched on the cycle in which the SPI master asserts its data-ready signal,

ensuring correct alignment between the MCP2515's output stream and the accelerator's internal register structure. Once all fields have been captured, the component assembles the final identifier, extracts the DLC, and asserts a single-cycle frame-valid pulse to signal that a complete frame is ready for rule-engine evaluation. The component also generates a one-cycle clear-IRQ pulse to notify the MCU that the MCP2515 interrupt flag should be cleared before the next frame is received.

Decision-Making Behavior

The decision-making behavior of the MCP2515 control component centers on coordinating SPI transactions, interpreting MCP2515 register fields, and determining when a frame is complete and ready for downstream processing. The component does not evaluate safety rules or modify frame contents; instead, it ensures that the MCP2515 is accessed correctly and that the retrieved frame is structurally coherent. Its finite-state machine advances only when the SPI master reports that a byte has been fully received, preventing partial or misaligned data from propagating into the rule engine. During the identifier-assembly stage, the component inspects the EXIDE bit in the SIDL register to determine whether the frame uses the standard 11-bit format or the extended 29-bit format, and it reconstructs the identifier accordingly. Once all fields have been read, the component decides that the frame is complete and asserts the frame-valid signal for exactly one clock cycle, guaranteeing atomic delivery of the frame to the rule engine and register bank. Finally, the component asserts the clear-IRQ pulse to prompt the MCU to clear the MCP2515's RX0IF flag, ensuring that subsequent frames can be retrieved without interrupt contention. Through this sequencing, the MCP2515 control block provides a

reliable, deterministic interface between the external CAN controller and the accelerator's internal processing pipeline.

Rule Engine Component

The rule engine serves as the accelerator's central evaluation block, responsible for interpreting each received CAN frame and determining whether it complies with the system's configured safety and authorization policies. Whereas the MCP2515 control component retrieves and assembles raw frame fields, the rule engine interprets those fields, examining the identifier, requested voltage and current, and computed power, and applies hardware-level checks to verify that the message is authorized and safe. The component operates as a deterministic, multi-stage pipeline driven by a finite-state machine, allowing it to enforce whitelist membership, identifier format requirements, and voltage, current, and power limits entirely in hardware. By performing these evaluations without firmware involvement, the rule engine provides a tamper-resistant decision-making layer that forms the core of the accelerator's safety and security guarantees.

Component Interface

The rule engine receives its inputs directly from the MCP2515 control component, including the reconstructed CAN identifier, the extended-ID flag, the DLC, and the eight data bytes. These inputs arrive atomically when the MCP2515 control block asserts its frame-valid signal, ensuring that the rule engine always processes complete and correctly aligned frames. The component also receives a set of configuration registers written by the MCU, which define the allowable voltage, current, and power ranges, as well as whether rules should apply only to extended identifiers. On the output side, the rule engine provides the processed data bytes, the

calculated power value, the safe-flag classification, and a five-bit fault code. Once evaluation is complete, the component asserts a single-cycle frame-ready pulse, signaling the register bank to latch the results and notify the MCU. Through this interface, the rule engine forms the central enforcement point for all safety and authorization policies within the accelerator.

Internal Data Flow

Internally, the rule engine implements a multi-stage processing pipeline driven by a finite-state machine. When a new frame arrives, the component first extracts the individual data bytes and reconstructs the target voltage and current fields for messages with identifier 0x401, which represent supervisory control requests. These values are stored in internal registers and used throughout the subsequent evaluation stages. The component then checks whether the frame should be subject to rule enforcement, based on the extended-ID configuration and the frame's identifier type. If the frame passes this initial filter, the rule engine compares the identifier against a hardware-encoded whitelist of allowed CAN IDs. Only whitelisted identifiers are permitted to proceed to the safety-limit checks.

For TARGET messages, the rule engine evaluates the requested voltage and current against the configured minimum and maximum limits. If either field falls outside the allowable range, the component clamps the value to the nearest limit and records the corresponding fault condition. After voltage and current checks, the rule engine computes the requested power by multiplying the target voltage and current. This value is then compared against the configured power limits. To support power-based clamping, the component uses a reciprocal lookup table to compute allowable current values derived from the power limits and the requested voltage. These computed limits are latched and combined with the configured current limits to determine the

final allowable current range. If the requested power exceeds or falls below the configured bounds, the rule engine clamps the current accordingly and records a power-limit fault.

Once all checks are complete, the component assembles the output data bytes, substituting clamped values where necessary. The processed bytes are combined into a 32-bit word for output, and the safe-flag and fault-code registers are updated to reflect the results of the evaluation. The rule engine then asserts its frame-ready signal, indicating that the processed frame is available for the register bank and MCU.

Decision-Making Behavior

The decision-making behavior of the rule engine is governed by a deterministic state machine that ensures each frame is evaluated in a fixed sequence. The component begins in the idle state, waiting for the frame-valid signal from the MCP2515 control block. Once a frame is detected, the rule engine transitions through parsing, spoof-checking, whitelist evaluation, and identifier validation. These early stages determine whether the frame is authorized to proceed to the safety-limit checks. If any violation is detected, such as a non-extended frame when extended-only mode is enabled, or an identifier not present in the whitelist, the component immediately marks the frame as unsafe, records the appropriate fault code, and bypasses the remaining checks.

For authorized TARGET messages, the rule engine evaluates voltage, current, and power in a structured sequence. Each check is gated by the absence of prior errors, ensuring that only the first detected violation determines the fault code. Voltage and current checks compare the requested values against configured limits and clamp them if necessary. Power evaluation uses the reciprocal lookup table to compute allowable current ranges derived from the configured

power limits. The rule engine then determines whether the requested power is within bounds and clamps the current accordingly. After all checks, the component assembles the final output bytes, updates the safe-flag and fault-code registers, and asserts the frame-ready signal to complete the evaluation cycle. A complete summary of rule conditions and their corresponding clamp or mute behaviors is provided in Table 1.

Table 1: Summary of rule conditions, corresponding accelerator actions, and resulting fault-bit assignments.

Rule / Condition	What is Checked	Action Taken	Output Behavior	Fault Code
Non-extended ID when ext-only enabled	i_cfg_ext_only = 1 and i_is_extended = 0	Mute frame	safe_flag = 0, data ignored	"00001"
ID not in whitelist	i_can_id ∉ WHITELIST	Mute frame	safe_flag = 0, data ignored	"00010"
Voltage out of range	V > Vmax or V < Vmin	Clamp voltage	clamped V written to output	"00100"
Current out of range	I > Imax or I < Imin	Clamp current	clamped I written to output	"01000"
Power out of range	P > Pmax or P < Pmin	Clamp current (via reciprocal)	clamped I written to output	"10000"

Reciprocal Lookup Table Component

The reciprocal lookup table provides a hardware-efficient method for computing the reciprocal of the target-voltage value used in the rule engine's power-based current-limit calculations. Instead of performing a division operation, which is expensive in FPGA fabric and would introduce variable latency, the accelerator uses a precomputed table stored in a VHDL package. The reciprocal LUT maps each possible 16-bit voltage value to a corresponding fixed-point reciprocal, allowing the rule engine to derive allowable current limits from configured power limits using only multiplication and shifting. This approach ensures

deterministic timing and avoids the resource overhead associated with implementing a hardware divider.

Component Interface

The reciprocal LUT receives the system clock and a 16-bit voltage input from the rule engine. The voltage value represents the requested or clamped target voltage extracted from a TARGET message. On each rising clock edge, the component indexes into the precomputed lookup table defined in reciprocal_pkg and outputs the corresponding 16-bit reciprocal value. The interface is intentionally minimal: the component does not perform any validation or range checking, as the rule engine guarantees that the voltage input is well-formed and within the domain of the table. The LUT therefore acts as a synchronous, single-cycle read port into a constant ROM structure.

Internal Data Flow

Internally, the component consists of a single clocked process that reads the reciprocal value associated with the provided voltage index. The lookup table itself is implemented as a constant array within the reciprocal_pkg package, generated offline to ensure that each entry corresponds to a fixed-point approximation of $1/V$scaled by 2^{24}. When the rule engine supplies a new voltage value, the LUT uses that value as an address into the table and retrieves the corresponding reciprocal on the next rising clock edge. This synchronous behavior ensures that the reciprocal value is stable and aligned with the rule engine's multi-stage evaluation pipeline, particularly during the power-limit and current-limit calculations.

The reciprocal lookup table stores a 16-bit fixed-point approximation of $1/V$, scaled by $2^{24}$to preserve fractional precision while keeping the LUT output compact. The table covers the

full range of voltage values used in the EVCS control loop (0–5050), and scaling the reciprocal by 2^{24}ensures that even very small reciprocal values, corresponding to high voltage requests, retain sufficient resolution for downstream calculations. Each LUT entry is therefore a 16-bit representation of $\lfloor 2^{24}/V \rfloor$, which is multiplied by the 32-bit configured power limits to produce a 48-bit intermediate value. The rule engine then extracts bits 39 down to 24 of this product to form a precise 16-bit current limit derived from the power constraint. This approach avoids the resource cost and multi-cycle, data-dependent latency of a hardware divider, providing constant-time, deterministic behavior. The LUT requires only 5,051 entries × 16 bits (≈10 kB), which is negligible compared to the Artix-7 200T FPGA's 13,140 Kb of block RAM. Given the ample BRAM available, trading a small amount of memory for speed, precision, and fixed-latency operation makes the LUT-based reciprocal method the most efficient and reliable choice for real-time power-limit evaluation.

Decision-Making Behavior

The reciprocal LUT does not implement any branching or conditional logic; its behavior is entirely determined by the voltage input and the contents of the lookup table. Each clock cycle, the component simply outputs the reciprocal associated with the current voltage index. This deterministic, memory-based approach eliminates the variability and complexity of arithmetic division, allowing the rule engine to compute power-derived current limits using only multiplication and bit extraction. By providing a constant-latency reciprocal operation, the LUT contributes to the accelerator's overall timing predictability and supports real-time enforcement of safety constraints within the EVCS control loop.

Register Bank Component

The register bank provides the structured memory-mapped interface through which the RadPC MCU accesses the results of each processed CAN frame. Unlike the other components in the accelerator pipeline, the register bank does not participate in frame evaluation or safety-rule enforcement. Instead, it serves as the boundary between the hardware accelerator and the MCU, capturing the metadata and processed data associated with each frame and presenting them in a stable, software-readable format. Although the register map includes configuration registers, these values are fixed at synthesis time in the current implementation. As a result, the register bank's primary runtime responsibility is to store the accelerator's outputs and manage the interrupt mechanism that notifies the MCU when new data is available.

Component Interface

The register bank connects to both sides of the accelerator pipeline. On the configuration side, it receives several 32-bit words from the MCU that encode the voltage (Figure 11) and current limits (Figure 12), power limits (Figure 13, Figure 14), and control bits (Figure 15) such as the extended-ID-only flag, the interrupt-clear request, and SPI clock divider. These values are unpacked and routed to the SPI Master, rule engine, and other hardware blocks that depend on them.

Configuration 1 Register - Voltage Limits

31	30	29	28	27	26	25	24
V_max (15)	V_max (14)	V_max (13)	V_max (12)	V_max (11)	V_max (10)	V_max (9)	V_max (8)
23	**22**	**21**	**20**	**19**	**18**	**17**	**16**
V_max (7)	V_max (6)	V_max (5)	V_max (4)	V_max (3)	V_max (2)	V_max (1)	V_max (0)
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**
V_min (15)	V_min (14)	V_min (13)	V_min (12)	V_min (11)	V_min (10)	V_min (9)	V_min (8)
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
V_min (7)	V_min (6)	V_min (5)	V_min (4)	V_min (3)	V_min (2)	V_min (1)	V_min (0)

Figure 11: Memory mapped voltage limit configuration register defining the minimum and maximum allowable voltage values. The accelerator compares incoming CAN frame voltage requests against these thresholds to classify frames as safe or unsafe.

Configuration 2 Register - Current Limits

31	30	29	28	27	26	25	24
I_max (15)	I_max (14)	I_max (13)	I_max (12)	I_max (11)	I_max (10)	I_max (9)	I_max (8)
23	**22**	**21**	**20**	**19**	**18**	**17**	**16**
I_max (7)	I_max (6)	I_max (5)	I_max (4)	I_max (3)	I_max (2)	I_max (1)	I_max (0)
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**
I_min (15)	I_min (14)	I_min (13)	I_min (12)	I_min (11)	I_min (10)	I_min (9)	I_min (8)
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
I_min (7)	I_min (6)	I_min (5)	I_min (4)	I_min (3)	I_min (2)	I_min (1)	I_min (0)

Figure 12: Memory-mapped current-limit configuration register specifying the allowable current range for incoming CAN requests. These limits are used by the rule engine to enforce hardware-level current-safety constraints.

Configuration 3 Register - Power Limit Low

31	30	29	28	27	26	25	24
P_Low (31)	P_Low (30)	P_Low (29)	P_Low (28)	P_Low (27)	P_Low (26)	P_Low (25)	P_Low (24)
23	**22**	**21**	**20**	**19**	**18**	**17**	**16**
P_Low (23)	P_Low (22)	P_Low (21)	P_Low (20)	P_Low (19)	P_Low (18)	P_Low (17)	P_Low (16)
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**
P_Low(15)	P_Low (14)	P_Low(13)	P_Low(12)	P_Low (11)	P_Low (10)	P_Low(9)	P_Low (8)
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
P_Low7)	P_Low (6)	P_Low (5)	P_Low(4)	P_Low (3)	P_Low (2)	P_Low(1)	P_Low (0)

Figure 13: Memory-mapped lower power-limit register defining the minimum allowable power request. The accelerator computes power from voltage and current fields and flags frames whose calculated power falls below this threshold.

Configuration 4 Register - Power Limit High

31	30	29	28	27	26	25	24
P_High (31)	P_High (30)	P_High(29)	P_High (28)	P_High (27)	P_High (26)	P_High (25)	P_High(24)
23	**22**	**21**	**20**	**19**	**18**	**17**	**16**
P_High(23)	P_High(22)	P_High(21)	P_High (20)	P_High (19)	P_High (18)	P_High (17)	P_High (16)
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**
P_High (15)	P_High (14)	P_High(13)	P_High(12)	P_High (11)	P_High (10)	P_High (9)	P_High (8)
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
P_High (7)	P_High(6)	P_High(5)	P_High (4)	P_High(3)	P_High (2)	P_High (1)	P_High(0)

Figure 14: Memory-mapped upper power-limit register defining the maximum allowable power request. Frames whose computed power exceeds this limit are classified as unsafe and reported to the MCU.

Config Control - Software to Hardware

31	30	29	28	27	26	25	24
Clk_Div (15)	Clk_Div (14)	Clk_Div (13)	Clk_Div (12)	Clk_Div (11)	Clk_Div (10)	Clk_Div (9)	Clk_Div (8)
23	**22**	**21**	**20**	**19**	**18**	**17**	**16**
Clk_Div (7)	Clk_Div (6)	Clk_Div (5)	Clk_Div (4)	Clk_Div (3)	Clk_Div (2)	Clk_Div (1)	Clk_Div (0)
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**
Unused	Unused	Unused	Unused	Unused	Unused	Unused	Unused
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Unused	Unused	Unused	Unused	Unused	Unused	Is_Ext_ID	Clear_IRQ

Figure 15: Memory-mapped configuration control register used to configure accelerator operation. The register contains the 16-bit SPI clock-divider field, the extended-identifier enable bit, and the interrupt-clear bit, allowing the RadPC MCU to control timing behavior and acknowledge accelerator interrupts.

On the reporting side, the register bank receives the processed frame metadata from the rule engine, including the safe/unsafe classification, the fault code, the CAN identifier, the DLC, and the eight data bytes. These values are written into dedicated registers whenever the rule engine asserts its frame-valid signal. The corresponding memory-mapped structures are shown in Figures 16–20: the status register (Figure 16), the frame-identifier register (Figure 17), the DLC register (Figure 18), and the two data registers containing bytes 0–3 and bytes 4–7 (Figure 19, Figure 20). The component also manages an interrupt latch that notifies the MCU when new data is available, holding the interrupt high until the MCU explicitly acknowledges it through the configuration-control register. Through this interface, the register bank provides a clean separation between real-time hardware processing and MCU-driven supervisory control.

Status Register - Hardware to Software

31	30	29	28	27	26	25	24
Unused	Unused	Unused	Unused	Unused	Unused	Unused	Unused
23	22	21	20	19	18	17	16
Unused	Unused	Unused	Unused	Unused	Unused	Unused	Unused
15	14	13	12	11	10	9	8
Unused	Unused	Unused	Unused	Unused	Unused	Unused	Unused
7	6	5	4	3	2	1	0
Unused	Err (4)	Err (3)	Err (2)	Err (1)	Err (0)	Frm_Safe	IRQ

Figure 16: Memory-mapped status register containing frame-classification results, fault codes, and interrupt flags. The RadPC MCU reads this register to determine whether the most recent frame was safe and to identify any rule-violation conditions.

Frame Meta Data - ID

31	30	29	28	27	26	25	24
Unused	Unused	Unused	CAN ID (28)	CAN ID (27)	CAN ID (26)	CAN ID (25)	CAN ID (24)
23	22	21	20	19	18	17	16
CAN ID (23)	CAN ID (22)	CAN ID (21)	CAN ID (20)	CAN ID (19)	CAN ID (18)	CAN ID (17)	CAN ID (16)
15	14	13	12	11	10	9	8
CAN ID (15)	CAN ID (14)	CAN ID (13)	CAN ID (12)	CAN ID (11)	CAN ID (10)	CAN ID (9)	CAN ID (8)
7	6	5	4	3	2	1	0
CAN ID (7)	CAN ID (6)	CAN ID (5)	CAN ID (4)	CAN ID (3)	CAN ID (2)	CAN ID (1)	CAN ID (0)

Figure 17: Memory-mapped frame-identifier register storing the 29-bit CAN ID of the most recently processed frame. The accelerator writes this field after rule-engine evaluation, allowing the MCU to inspect the identifier of safe or unsafe frames.

Frame Meta Data - DLC

31	30	29	28	27	26	25	24
Unused	Unused	Unused	Unused	Unused	Unused	Unused	Unused
23	22	21	20	19	18	17	16
Unused	Unused	Unused	Unused	Unused	Unused	Unused	Unused
15	14	13	12	11	10	9	8
Unused	Unused	Unused	Unused	Unused	Unused	Unused	Unused
7	6	5	4	3	2	1	0
Unused	Unused	Unused	Unused	DLC (3)	DLC (2)	DLC (1)	DLC (0)

Figure 18: Memory-mapped data-length-code register storing the DLC value of the most recent CAN frame. This register allows the MCU to verify the frame length and correlate it with the received data bytes.

Frame Data 0

31	30	29	28	27	26	25	24
Data_3 (7)	Data_3 (6)	Data_3 (5)	Data_3 (4)	Data_3 (3)	Data_3 (2)	Data_3 (1)	Data_3 (0)
23	22	21	20	19	18	17	16
Data_2 (7)	Data_2 (6)	Data_2 (5)	Data_2 (4)	Data_2 (3)	Data_2 (2)	Data_2 (1)	Data_2 (0)
15	14	13	12	11	10	9	8
Data_1 (7)	Data_1 (6)	Data_1 (5)	Data_1 (4)	Data_1 (3)	Data_1 (2)	Data_1 (1)	Data_1 (0)
7	6	5	4	3	2	1	0
Data_0 (7)	Data_0 (6)	Data_0 (5)	Data_0 (4)	Data_0 (3)	Data_0 (2)	Data_0 (1)	Data_0 (0)

Figure 19: Memory-mapped data register containing bytes 0–3 of the most recently processed CAN frame. These values are written by the accelerator after SPI retrieval and used by the MCU for application-level processing.

Frame Data 1

31	30	29	28	27	26	25	24
Data_7 (7)	Data_7 (6)	Data_7 (5)	Data_7 (4)	Data_7 (3)	Data_7 (2)	Data_7 (1)	Data_7 (0)
23	**22**	**21**	**20**	**19**	**18**	**17**	**16**
Data_6 (7)	Data_6 (6)	Data_6 (5)	Data_6 (4)	Data_6 (3)	Data_6 (2)	Data_6 (1)	Data_6 (0)
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**
Data_5 (7)	Data_5 (6)	Data_5 (5)	Data_5 (4)	Data_5 (3)	Data_5 (2)	Data_5 (1)	Data_5 (0)
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Data_4 (7)	Data_4 (6)	Data_4 (5)	Data_4 (4)	Data_4 (3)	Data_4 (2)	Data_4 (1)	Data_4 (0)

Figure 20: Memory-mapped data register containing bytes 4–7 of the most recently processed CAN frame. Together with Figure 19, this register provides the full eight-byte payload for MCU consumption.

Internal Data Flow

Internally, the register bank consists of two active processes: a hardware-write path and an interrupt-control path. When the rule engine asserts its frame-valid signal, the hardware-write process clears the previous contents of the status and metadata registers and stores the new CAN identifier, DLC, and data bytes. The safe flag and fault code are written into the status register, allowing the MCU to determine whether the frame was accepted or rejected and why.

Because configuration values are fixed at compile time, the MCU-write path is limited to handling the interrupt-clear bit. When the MCU writes this bit, the register bank clears the interrupt latch and resets the “new frame available” flag in the status register. This ensures that the MCU receives exactly one interrupt per processed frame and prevents stale interrupt conditions from persisting across frames.

Decision-Making Behavior

The register bank's decision-making behavior is centered on ensuring that the MCU always receives a complete and consistent view of the most recently processed frame. The component writes new frame data only when the rule engine asserts its frame-valid signal, preventing partial or intermediate values from propagating to the MCU. The interrupt-control logic asserts the IRQ output whenever a new frame is written and holds it high until the MCU explicitly acknowledges it through the interrupt-clear bit in the configuration-control register. This handshake mechanism ensures reliable communication between the accelerator and the MCU, even if the MCU is delayed in servicing the interrupt.

Because configuration values are fixed at synthesis time, the register bank does not perform any runtime distribution of configuration parameters. Instead, it functions purely as a reporting and interrupt-management block, providing a deterministic and stable interface between the high-speed hardware pipeline and the MCU's supervisory software.

CAN Limit Checking Top Component

The top-level CAN limit checking component integrates all hardware blocks of the accelerator into a single cohesive module. It coordinates the SPI Master, MCP2515 control logic, rule engine, and register bank, forming the complete data path from CAN-bus reception to MCU-visible results. While each subcomponent performs a specific function, byte-level SPI communication, frame retrieval, rule evaluation, or register storage, the top-level module defines how these blocks interact, ensuring that data flows deterministically through the pipeline and that the MCU receives a consistent view of each processed frame. By encapsulating the full

accelerator architecture, the top-level component provides the structural backbone that connects the MCP2515 CAN controller on one side and the RadPC MCU on the other.

Component Interface

The top-level module interfaces directly with both the MCP2515 CAN controller and the RadPC MCU. On the CAN-side interface, it exposes the SPI signals required to communicate with the MCP2515, including MOSI, MISO, SCLK, and chip-select, as well as the MCP2515's interrupt line. These signals feed into the SPI Master and MCP2515 control components, which together retrieve each CAN frame and present its fields to the rule engine.

On the MCU-side interface, the top-level module provides access to the memory-mapped registers implemented in the register bank. These include the status register, frame identifier, DLC, and the two data registers containing bytes 0–3 and bytes 4–7. The top-level module also forwards the calculated requested power for TARGET messages and asserts an interrupt line to notify the MCU when a new frame has been processed. Although configuration registers are present in the register map, their values are fixed at synthesis time in the current design; the only runtime MCU interaction is the interrupt-clear bit used to acknowledge new frame data.

Internal Data Flow

The internal data flow of the top-level component follows a strictly ordered pipeline. When the MCP2515 asserts its interrupt line, the MCP2515 control component initiates a sequence of SPI transactions through the SPI Master to retrieve the frame's identifier, DLC, and data bytes. These fields are then passed to the rule engine, which evaluates the frame against the configured safety and authorization rules. The rule engine outputs the processed data bytes, the safe/unsafe classification, the fault code, and the calculated power value for TARGET messages.

Once the rule engine asserts its frame-ready signal, the register bank captures the processed frame information and updates the memory-mapped registers that the MCU reads. The register bank also asserts the interrupt latch, which propagates to the MCU through the top-level module's IRQ output. This pipeline ensures that each frame moves cleanly from the MCP2515 to the MCU with deterministic timing and without software intervention in the intermediate stages.

Decision-Making Behavior

The top-level component does not perform rule evaluation or frame interpretation itself; instead, its decision-making behavior centers on orchestrating the interactions between subcomponents. It ensures that the SPI Master is driven only when the MCP2515 control logic requests a new byte, that the rule engine receives complete and correctly aligned frame fields, and that the register bank updates its stored values only when a frame has been fully processed. The top-level module also manages the propagation of configuration values synthesized into the design, routing them to the rule engine and SPI Master as static inputs. Finally, it coordinates the interrupt-acknowledgment mechanism by passing the MCU's interrupt-clear bit to the register bank, ensuring that each processed frame generates exactly one interrupt. Through this structured coordination, the top-level component provides the control framework that binds the accelerator's hardware blocks into a unified, reliable system.

SYSTEM VERIFICATION

Verification Objectives

The verification process described here was designed to demonstrate that the CAN limit-checking hardware accelerator operates correctly, deterministically, and safely under realistic EVCS communication conditions. The objective was to confirm that the accelerator enforces all configured safety constraints, correctly mediates communication between the MCP2515 CAN controller and the RadPC MCU, and reliably handles both valid and malformed CAN traffic. Verification focused on functional correctness, rule-evaluation accuracy, fault-handling behavior, and timing consistency.

Functional Correctness

The accelerator must correctly process all incoming CAN frames retrieved from the MCP2515 and expose validated results to the RadPC MCU through the memory-mapped register interface. Verification confirms that the system:

- validates CAN identifiers against a compile-time whitelist,
- enforces configured voltage, current, and power limits,
- clamps out-of-range values to the nearest safe bound,
- generates appropriate fault codes,
- asserts a hardware “safe/unsafe” flag,
- triggers an interrupt when a validated frame is ready, and
- never drops, duplicates, or corrupts frames under normal operating conditions.

Safety-Rule Enforcement

Voltage, current, and power limits are configured through compile-time registers and must be applied deterministically to every received frame. Verification ensures that:

- out-of-range voltage values are clamped, marked unsafe, and accompanied by a fault code,
- out-of-range current or power values are clamped, marked unsafe, and accompanied by a fault code,
- frames with unknown or non-whitelisted CAN identifiers are marked unsafe, assigned a fault code, and ignored by the rule engine, and
- frames whose identifier format does not match the expected standard-frame configuration (i.e., frames received with the IDE bit set) are marked unsafe, assigned an extended-frame fault code, and ignored.

Normal-Operation Behavior

Under valid operating conditions, the accelerator must:

- pass all valid frames unchanged,
- avoid false positives (valid frames marked unsafe),
- avoid false negatives (unsafe frames marked safe),
- maintain deterministic processing behavior, and
- process all frames without loss or duplication.

These behaviors establish the baseline for safe integration with the power-electronics controller.

Timing Consistency

Although no strict real-time deadlines are imposed in the current design, the accelerator's performance requirements are defined entirely by the MCP2515 interface rather than by high-throughput computation. The MCP2515 transfers each received CAN frame over SPI at up to 10 MHz, and at the 125 kbps CAN bus rate used in the experimental setup, the minimum inter-frame spacing is significantly larger than the accelerator's internal processing time. Because of this margin, the hardware does not need to achieve extreme throughput; it simply must ensure that rule evaluation and register updates complete before the next SPI transaction begins. The timing-consistency tests therefore focus on verifying that this fixed-latency pipeline behaves predictably under all validated frame conditions. Consequently, verification measures:

- the consistency of MCP2515 interrupt-to-processing latency,
- the stability of SPI transaction duration,
- the repeatability of rule-engine evaluation time, and
- the determinism of register-bank update and interrupt assertion.

To quantify the timing margin, the accelerator's end-to-end processing latency was measured across all verified frames. At the 32 MHz system clock used in the experimental setup, the rule-engine pipeline requires approximately 15 clock cycles from the MCP2515 interrupt to the assertion of the frame-ready signal, corresponding to a processing time of roughly 470 ns per CAN frame. This equates to a theoretical throughput of more than 2 million frames per second, far exceeding the physical limits of the 125 kbps CAN bus, which can deliver at most approximately 1,000 frames per second. The accelerator therefore provides over three orders of magnitude of timing headroom relative to the maximum possible CAN-frame arrival rate, ensuring that every frame is processed well before the MCP2515 can deliver the next one.

The verification process also includes internal signal observation using a *Vivado* Integrated Logic Analyzer (ILA). The ILA is used to capture cycle-accurate timing of SPI transactions and to confirm correct loading of the CAN identifier, DLC, and data registers within the RadPC Hydra FPGA fabric. These measurements provide internal visibility into the accelerator's behavior and support validation of both timing determinism and correct register-update sequencing.

System-Integration Behavior

Verification also confirms correct interaction between the accelerator, RadPC, and the external CAN-bus environment. The RadPC MCU retrieves processed frames from the register bank, logs results over UART, and provides compile-time configuration of rule parameters. The verification environment includes a Raspberry Pi running the System Controller Simulation, MCP2515-based RS485 CAN HATs, a RadPC Hydra, a UART terminal for logging, and an Analog Discovery 3 for SPI waveform capture and CAN-message validation (Figure 21).

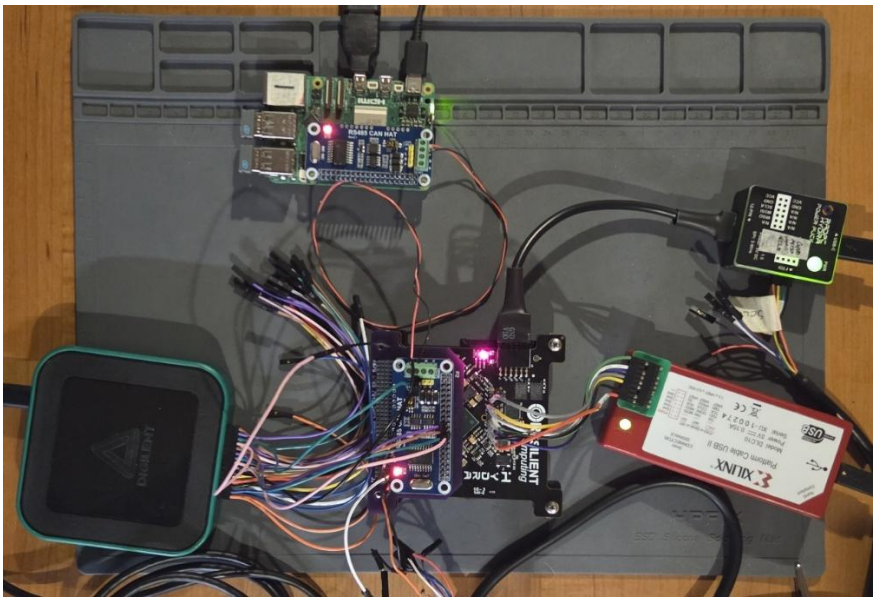

Figure 21: Verification testing setup.

Verification Scope and Assumptions

The verification activities in this chapter focus on demonstrating the functional correctness, safety-rule enforcement, and timing consistency of the CAN limit-checking hardware accelerator when integrated into the RadPC Hydra platform and exercised through the experimental CAN-bus environment described in Chapter 5. The scope of verification is limited to behaviors that the accelerator is explicitly designed to provide and does not extend to full EVCS-level power-electronics behavior or grid-scale system interactions.

Scope of Verification

The following aspects of system behavior are included within the scope of verification:

- Correct acquisition and parsing of CAN frames retrieved from the MCP2515 via the SPI master.
- Validation of CAN identifiers against the compile-time whitelist and correct handling of unknown or unexpected IDs.
- Enforcement of configured voltage, current, and power limits, including clamping behavior and generation of corresponding fault codes.
- Correct operation of the rule engine, including safe/unsafe flag generation, fault-code assignment, and correct assembly of processed output frames.
- Correct register-bank behavior, including latching of processed frame data, exposure of results to the RadPC MCU, and assertion of the “frame ready” interrupt.
- Fault-handling behavior, including detection of malformed frames, non-matching frame configurations, and out-of-range parameters.

- Timing consistency, including repeatability of SPI transactions, rule-engine evaluation, and register-update latency, sufficient to establish a baseline for future real-time integration.
- System-integration behavior, including correct interaction between the accelerator, the RadPC MCU, the MCP2515 CAN controller, and the Raspberry Pi-based System Controller Simulation.

Out-of-Scope Items

The following behaviors are outside the scope of this verification effort:

- Full EVCS power-electronics behavior, including converter dynamics, thermal limits, or closed-loop control.
- Grid-level or fleet-level cybersecurity impacts, including load-altering attacks or coordinated EVCS manipulation.
- ISO 15118, OCPP, or backend-server security, which are unrelated to the CAN-level protections implemented here.
- Long-term reliability, environmental robustness, or EMI/EMC performance, which require specialized hardware testing.
- Hard real-time deadlines, as the current design does not impose strict latency constraints, though timing consistency is evaluated.

Assumptions

Verification is performed under the following assumptions:

- The MCP2515 CAN controller operates correctly and delivers well-formed CAN frames to the accelerator except when intentionally subjected to malformed-frame tests.

- The compile-time configuration registers accurately represent the intended safety limits and frame formats.
- The Raspberry Pi System Controller Simulation generates representative supervisory CAN traffic consistent with the DCFC control model.
- The RadPC MCU firmware correctly reads the register bank, logs results over UART, and does not introduce additional timing variability.
- The *Vivado* ILA provides accurate internal signal capture for verifying timing behavior and confirming correct population of CAN-related registers inside the FPGA.
- The Analog Discovery 3 provides accurate SPI waveform capture and CAN-message validation for timing and correctness measurements.
- The CAN bus operates under nominal electrical conditions, and physical-layer disturbances are not part of this verification effort.

These boundaries ensure that verification focuses on the accelerator's intended function: deterministic, hardware-enforced validation and safe processing of CAN-based power-electronics control messages.

Verification Methodology

The verification methodology is designed to evaluate the CAN limit-checking hardware accelerator under controlled and repeatable conditions that reflect realistic EVCS supervisory-control behavior. The approach combines black-box functional testing, white-box rule-evaluation checks, timing-consistency measurements, and targeted fault-injection experiments. All tests are executed using the experimental setup described in Chapter 5, with the Raspberry Pi generating supervisory CAN traffic, the MCP2515 CAN controller delivering

frames to the accelerator, and the RadPC MCU retrieving processed results through the memory-mapped register interface.

Test Categories

Verification is organized into four complementary categories that collectively assess functional correctness, safety-rule enforcement, timing behavior, and system integration.

- Black-box functional tests evaluate the accelerator as a peripheral device by sending known CAN frames from the Raspberry Pi and observing the processed outputs retrieved by the RadPC MCU. These tests confirm correct parsing, clamping, safe/unsafe flag generation, interrupt behavior, and register-bank updates without relying on internal visibility into the rule engine.
- White-box rule-evaluation tests use controlled input frames and compile-time configuration values to validate the internal decision-making logic of the rule engine. These tests confirm that voltage, current, and power limits are applied correctly; that clamping behavior matches the configured bounds; and that fault codes correspond to the correct violation type.
- Fault-injection tests intentionally introduce malformed, unsafe, or non-compliant CAN frames to verify that the accelerator correctly identifies and handles error conditions. These tests include out-of-range voltage and current values, power-limit violations, unknown CAN identifiers, and frames that do not match the expected format or DLC.
- Timing-consistency measurements assess the repeatability of the accelerator's processing latency. Although no strict real-time deadlines are imposed, timing stability is essential

for future integration with power-electronics control loops. Measurements include SPI transaction duration, rule-engine evaluation time, and interrupt-to-register-update latency.

Test Procedure

Each verification test follows a structured procedure to ensure repeatability and traceability:

1. Configure compile-time rule parameters, including voltage, current, and power limits, as well as the CAN-ID whitelist and expected frame formats.
2. Generate a controlled CAN frame on the Raspberry Pi, using either valid supervisory-control values or intentionally malformed or unsafe parameters for fault-injection tests.
3. Capture SPI activity between the MCP2515 and the accelerator using the Analog Discovery 3 to confirm correct frame retrieval and byte-level sequencing.
4. Capture internal FPGA signals using the *Vivado* ILA to verify correct timing and loading of the CAN ID, DLC, and data registers, and to correlate internal register-update events with external SPI activity.
5. Observe accelerator outputs through the RadPC MCU's register-bank reads and UART logs, including clamped values, safe/unsafe flags, fault codes, and interrupt behavior.
6. Compare observed behavior against expected results derived from the configured limits, rule definitions, and fault-handling requirements.
7. Repeat tests across multiple trials to evaluate timing consistency and ensure that no frames are dropped, duplicated, or inconsistently processed.

Evaluation Criteria

A test is considered successful when:

- Valid frames are processed without modification, marked safe, and delivered to the MCU with no loss or duplication.
- Unsafe frames are correctly identified, clamped when appropriate, marked unsafe, and assigned the correct fault code.
- Frames with unknown IDs or invalid formats are ignored and marked unsafe without propagating unsafe data to the MCU.
- Interrupts are asserted exactly once per processed frame.
- Timing measurements remain consistent across repeated trials, with no anomalous latency spikes or variability that would impact future real-time integration.

This methodology provides a comprehensive evaluation of the accelerator's correctness, safety behavior, and integration readiness.

Test Cases and Expected Results

Verification was carried out through a structured set of test cases designed to evaluate the accelerator's behavior under normal operation, boundary conditions, and fault-injection scenarios. Each test case specifies the input CAN frame, the relevant configuration parameters, the expected hardware-accelerator behavior, and the observable outputs retrieved by the RadPC MCU. Together, these tests provide comprehensive coverage of the accelerator's functional, safety, and timing-consistency requirements.

Normal-Operation Test Case

Valid frame with whitelisted ID

- Input: CAN ID matches whitelist; voltage, current, and computed power all within configured limits.
- Expected hardware behavior:
    - Frame passes unchanged.
    - Status Register Frame Safe (bit 1) = ‘1’
    - Status Register Error Code (bits 6-2) = “00000”
    - Status Register Frame Ready (bit 0) = ‘1’
    - Data registers contain the exact received bytes.
    - Interrupt asserted to MCU.
    - Register bank updated exactly once.
- Measured outcome:
    - UART log shows unmodified data.
    - AD3 confirms consistent SPI timing.
    - No dropped or duplicated frames.
    - ILA shows correct sequential latching of ID, DLC, and data bytes.

Limit-Violation Test Cases

Over-Voltage Condition

- Input: Voltage field exceeds configured maximum.
- Expected hardware behavior:
    - Voltage clamped to V_max.

  - Status Register Frame Safe (bit 1) = ‘0’
  - Status Register Error Code (bits 6-2) = “00100” (Out-of-range voltage fault)
  - Status Register Frame Ready (bit 0) = ‘1’
  - Data registers contain clamped voltage value.
  - Interrupt asserted.
- Measured outcome:
  - Register bank shows clamped voltage.
  - UART log shows unsafe flag, correct message for fault code “00100”, and correct clamped voltage value.
  - ILA confirms correct register behavior.

Under-Voltage Condition

- Input: Voltage field below configured minimum.
- Expected hardware behavior:
  - Voltage clamped to V_min.
  - Status Register Frame Safe (bit 1) = ‘0’
  - Status Register Error Code (bits 6-2) = “00100” (Out-of-range voltage fault)
  - Status Register Frame Ready (bit 0) = ‘1’
  - Data registers contain clamped voltage value.
- Measured outcome:
  - Register bank shows clamped value.
  - UART log shows unsafe flag, correct message for fault code “00100”, and correct clamped voltage value.

  - ILA confirms correct register behavior.

Over-Current Condition

- Input: Current field exceeds configured maximum.
- Expected hardware behavior:
  - Current clamped to I_max.
  - Status Register Frame Safe (bit 1) = ‘0’
  - Status Register Error Code (bits 6-2) = “01000” (Out-of-range current fault)
  - Status Register Frame Ready (bit 0) = ‘1’
  - Data registers contain clamped current value.
- Measured outcome:
  - Register bank shows clamped current.
  - UART log shows unsafe flag, correct message for fault code “01000”, and correct clamped current value.
  - ILA confirms correct register behavior.

Over-Power Condition

- Input: Computed power (requested V × I) exceeds configured upper power limit.
- Expected hardware behavior:
  - Current required to reach safe power level computed using reciprocal lookup table.
  - Current clamped to reach safe power level.
  - Status Register Frame Safe (bit 1) = ‘0’
  - Status Register Error Code (bits 6-2) = “10000” (Out-of-range power fault)

  - Status Register Frame Ready (bit 0) = ‘1’
  - Data registers contain clamped current value.
- Measured outcome:
  - Register bank shows clamped current.
  - UART log shows unsafe flag, correct message for fault code “10000”, correct requested power, and correct clamped current value so power is in range.
  - ILA confirms correct register behavior.

<u>Identifier-Validation Test Case</u>

Unknown CAN ID

- Input: CAN ID not present in whitelist.
- Expected hardware behavior:
  - Status Register Frame Safe (bit 1) = ‘0’
  - Status Register Error Code (bits 6-2) = “00010” (Non-whitelist ID fault)
  - Status Register Frame Ready (bit 0) = ‘1’
  - Data registers still latch the received bytes, but MCU ignores data.
  - No clamping applied.
- Measured outcome:
  - UART log shows unsafe flag, correct message for fault code “00010”
  - ILA confirms correct register behavior.

Frame-Format Validation Test Case

Unexpected Standard/Extended Format Mismatch

- Input: Frame whose IDE bit does not match the expected format (e.g., standard frame received when extended ID is required).
- Expected hardware behavior:
    - Status Register Frame Safe (bit 1) = ‘0’
    - Status Register Error Code (bits 6-2) = “00001” (Frame configuration fault)
    - Status Register Frame Ready (bit 0) = ‘1’
    - Data registers latch the received bytes but are flagged unsafe.
- Measured outcome:
    - UART log shows unsafe flag, correct message for fault code “00001”
    - ILA confirms correct register behavior.

Timing-Consistency Test Cases

Repeated Valid-Frame Timing Test

- Input: 100+ identical valid frames sent at fixed intervals.
- Expected hardware behavior:
    - Fixed SPI transaction duration.
    - Fixed rule-engine evaluation time.
    - Fixed latency from MCP2515 interrupt falling edge to accelerator frame-ready rising edge.
    - Status Register Frame Ready bit toggles with deterministic periodicity.
- Measured outcome:

- ILA shows identical cycle counts across all captures.
- No jitter or variation observed.

System-Integration Test Case

Supervisory Control Sequence Test

- Input: Realistic sequence of voltage/current requests generated by the Raspberry Pi System Controller Simulation.
- Expected hardware behavior:
  - All valid frames processed normally.
  - Unsafe frames flagged with correct fault codes (0x01–0x11).
  - Register bank (Figures 14–18) updated once per frame.
  - No dropped frames, no duplicated interrupts.
- Measured outcome:
  - UART logs match expected supervisory-control behavior.
  - AD3 confirms correct SPI sequencing.
  - MCU successfully reads all register fields for each frame.

Results and Discussion

The verification results demonstrate that the CAN-limit-checking hardware accelerator performs its intended functions reliably and deterministically across all tested scenarios. The accelerator consistently enforced safety constraints, correctly classified valid and invalid frames, and maintained stable timing behavior throughout repeated trials. The configuration parameters

used for all tests are listed in Table 2, and the system was configured to accept only extended CAN identifiers to enable format-violation testing.

Table 2: Configuration parameters used for validation testing.

Configuration Values

Parameter	Value
Voltage High [dV]	4995
Voltage Low [dV]	1000
Current High [dA]	2498
Current Low [dA]	1000
Power High [dW]	1230000
Power Low [dW]	105000

Functional Correctness

Across all normal-operation tests, the accelerator processed valid supervisory-control frames exactly as expected. Frames with whitelisted identifiers and in-range voltage, current, and power values passed through the rule engine unchanged, were marked safe, and triggered a single interrupt per frame. The STATUS register reflected:

- Frame Safe (bit 1) = 1
- Error Code (bits 6–2) = 00000 (no-fault code)
- Frame Ready (bit 0) = 1

The external logic-analyzer capture in Figure 22 confirms correct reception of the CAN frame and proper SPI byte ordering. The lower trace shows the CAN frame as observed on the bus, while the upper trace shows the MCP2515's SPI transfer of the same frame to the accelerator. The alignment between these traces verifies correct frame retrieval and byte ordering.

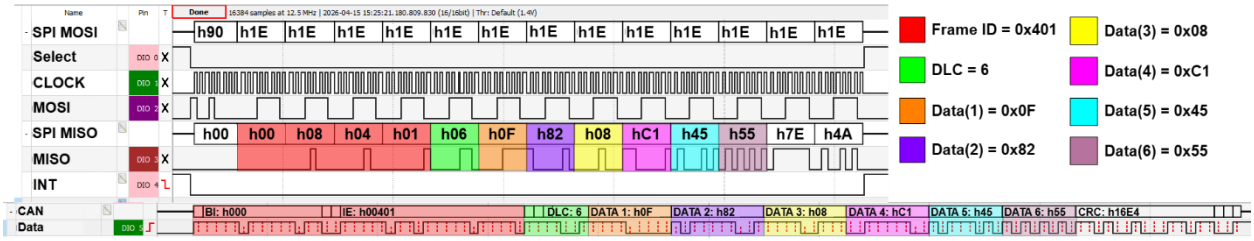

Figure 22: External logic-analyzer capture showing a complete safe CAN frame and its corresponding MCP2515 SPI transaction. The lower trace shows the CAN frame as observed directly on the bus, including the standard identifier, DLC value, and six data bytes. The upper trace shows the MCP2515's SPI transfer of the same frame to the accelerator, with the ID, DLC, and data fields visible in the decoded SPI stream. The alignment between the CAN-bus capture and the SPI transaction confirms correct frame retrieval and byte-ordering behavior at the hardware interface.

The internal ILA capture in Figure 23 shows the sequential latching of the DLC, data bytes, and 29-bit identifier into the register bank. The STATUS register indicates a safe frame, and the UART output matches the expected decoded values. Together, these captures confirm that the SPI master, MCP2515 controller, rule engine, and register bank operate correctly under nominal conditions.

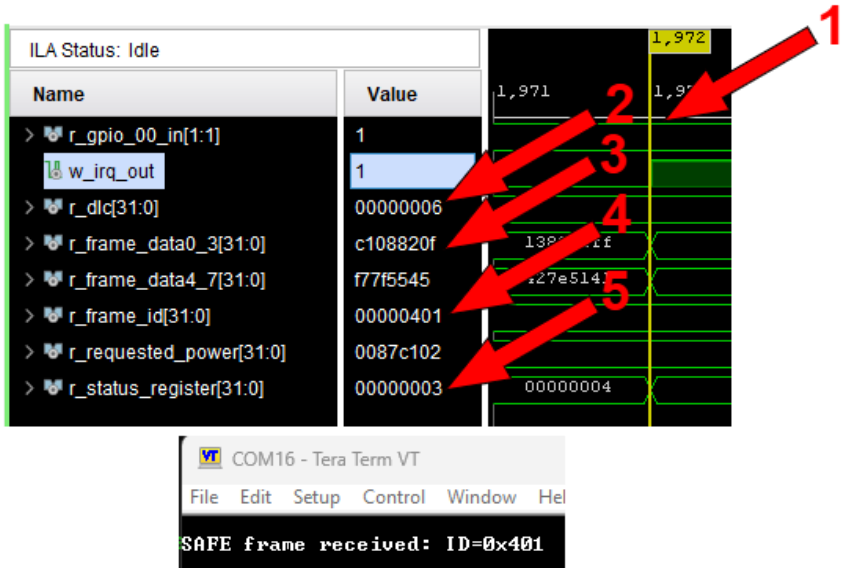

Figure 23: Internal ILA capture demonstrating correct register-loading behavior for the safe frame shown in Figure 22. The ILA trace captures the rising edge of the frame-ready interrupt (1), followed by the sequential latching of the DLC register (2), data-byte registers (3), and the 29-bit CAN identifier (4) into the accelerator's register bank. The status register (5) indicates that the frame-ready interrupt is set and that the frame is classified as safe. The corresponding UART output (bottom) shows the decoded frame as reported by the RadPC MCU, confirming agreement between internal FPGA state and external system-level behavior.

Safety-Rule Enforcement

Tests involving out-of-range voltage, current, and power values confirmed that the accelerator's clamping and fault-handling mechanisms operate as designed. In each case, the STATUS register reflected:

- Frame Safe (bit 1) = 0
- Frame Ready (bit 0) = 1

The specific error-code patterns were:

- Voltage faults: 00100 (voltage-limit fault code)
- Current faults: 01000 (current-limit fault code)
- Power faults: 10000 (power-limit fault code)

Voltage Violations

The external captures in Figure 24 and Figure 25 show CAN frames whose voltage fields exceed or fall below the configured limits. In both cases, the MCP2515 correctly forwards the frame to the accelerator, and the SPI transaction matches the expected byte ordering.

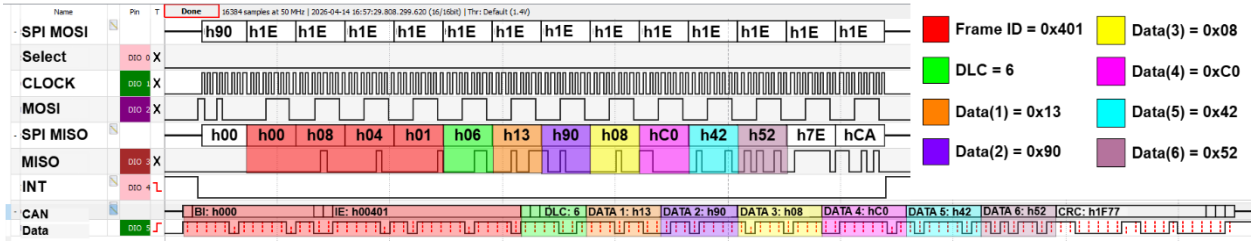

Figure 24: External logic-analyzer capture showing a high-voltage request CAN frame and its corresponding MCP2515 SPI transaction. The lower trace shows the CAN frame as observed on the bus, including the standard identifier, DLC value, and six data bytes representing a voltage request that exceeds the configured safety limit. The upper trace shows the MCP2515's SPI transfer of the same frame to the accelerator, with the ID, DLC, and data fields visible in the decoded SPI stream. The alignment between the CAN-bus capture and the SPI transaction confirms correct retrieval of the out-of-range request and proper byte ordering at the hardware interface.

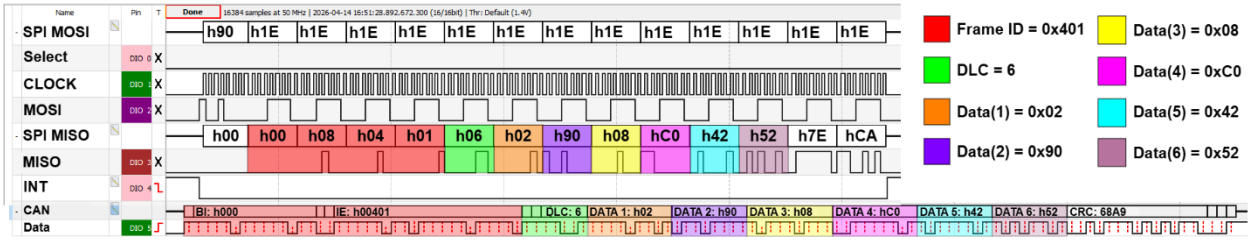

Figure 25: External logic-analyzer capture showing a low-voltage request CAN frame and the corresponding MCP2515 SPI transaction. The lower trace shows a CAN frame whose data bytes encode a voltage request below the configured minimum safety threshold. The upper trace shows the MCP2515's SPI transfer of the same frame to the accelerator, with the identifier, DLC, and data fields visible in the decoded SPI stream. The alignment between the CAN-bus capture and the SPI transaction confirms correct retrieval of the out-of-range voltage request and proper byte-ordering behavior at the hardware interface.

The internal ILA captures in Figure 26 and Figure 27 confirm that:

- the voltage field is clamped to V_max or V_min,
- the STATUS register asserts the voltage-limit fault code (00100),
- the unsafe flag is set,
- and the register bank latches the clamped value exactly once.

UART logs match the unsafe classification and report the correct clamped voltage.

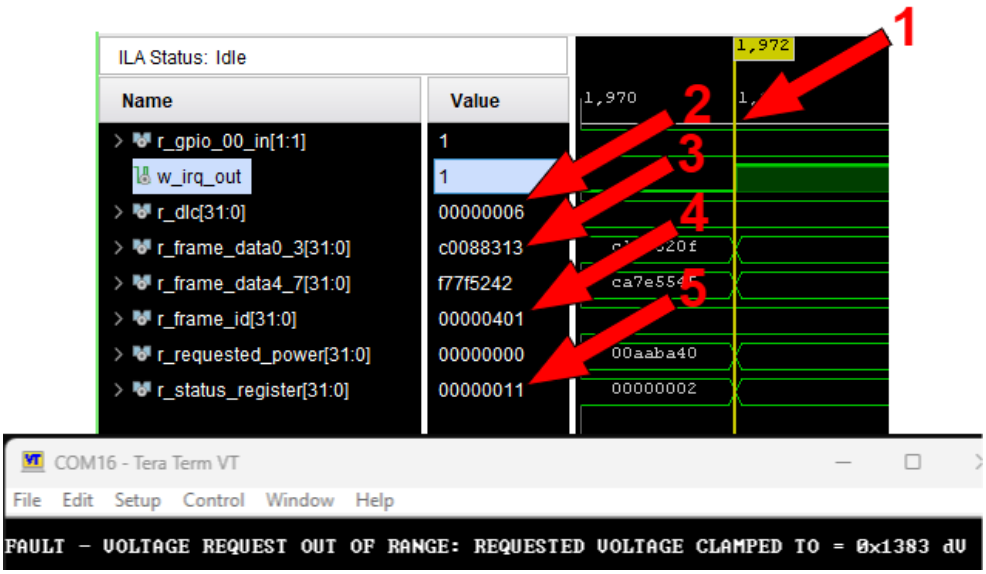

Figure 26: Internal ILA capture demonstrating correct register-loading and fault-classification behavior for the high-voltage request frame shown in Figure 24. The ILA trace captures the rising edge of the frame-ready interrupt (1), followed by the sequential latching of the DLC register (2), data-byte registers (3), and the 29-bit CAN identifier (4) into the accelerator's register bank. The status register indicates that the frame-ready interrupt is set, the frame is classified as unsafe, and the voltage-limit fault code is asserted (5). The corresponding UART output (bottom) shows the decoded unsafe frame and the reported error code, confirming agreement between internal FPGA state and external system-level behavior.

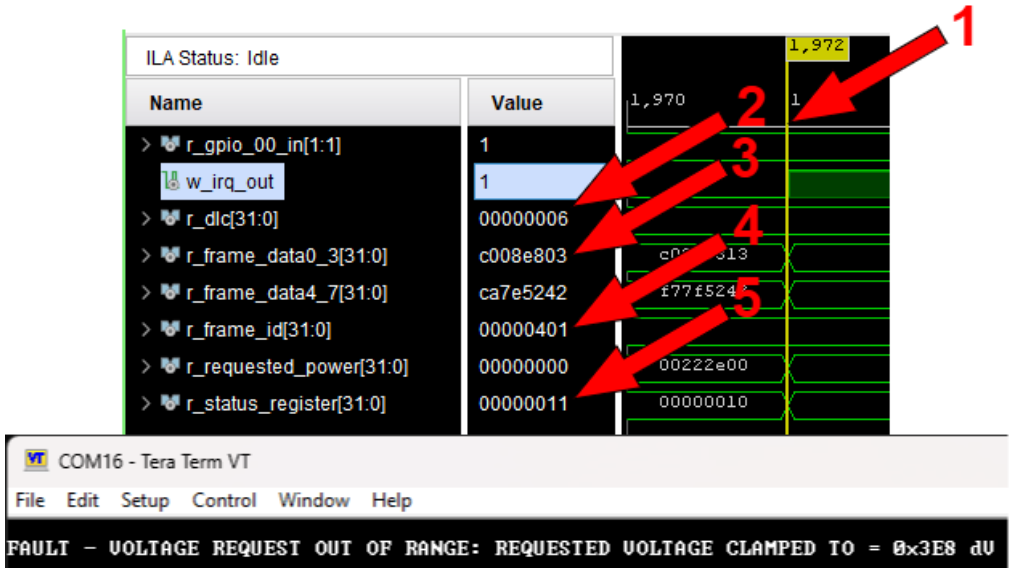

Figure 27: Internal ILA capture demonstrating correct register-loading and fault-classification behavior for the low-voltage request frame shown in Figure 25. The ILA trace captures the rising edge of the frame-ready interrupt (1), followed by the sequential latching of the DLC register (2), data-byte registers (3), and the 29-bit CAN identifier (4) into the accelerator's register bank. The status register indicates that the frame-ready interrupt is set, the frame is classified as unsafe, and the voltage-limit fault code is asserted (5). The corresponding UART output (bottom) shows the decoded unsafe frame and the reported error code, confirming agreement between internal FPGA state and external system-level behavior.

Current Violations

The external capture in Figure 28 shows a CAN frame requesting a current above the configured maximum. The SPI waveform confirms correct retrieval of the out-of-range request.

The internal ILA capture in Figure 29 shows:

- clamping of the current field to I_max,
- assertion of the current-limit fault code (01000),
- and correct latching of all fields into the register bank.

UART output confirms the unsafe classification and the clamped current value.

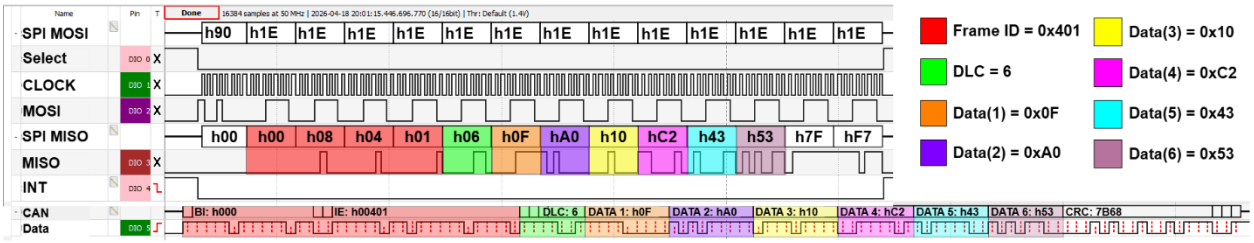

Figure 28: External logic-analyzer capture showing a high-current request CAN frame and the corresponding MCP2515 SPI transaction. The lower trace shows a CAN frame whose data bytes encode a current request exceeding the configured safety limit. The upper trace shows the MCP2515's SPI transfer of the same frame to the accelerator, with the identifier, DLC, and data fields visible in the decoded SPI stream. The alignment between the CAN-bus capture and the SPI transaction confirms correct retrieval of the out-of-range current request and proper byte ordering at the hardware interface.

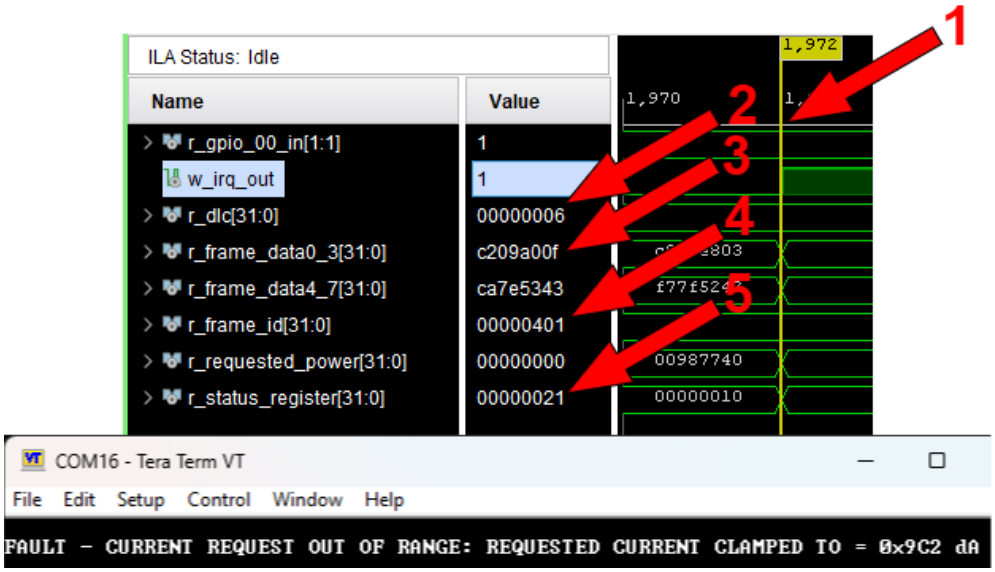

Figure 29: Internal ILA capture demonstrating correct register-loading and fault-classification behavior for the high-current request frame shown in Figure 28. The ILA trace captures the rising edge of the frame-ready interrupt (1), followed by the sequential latching of the DLC register (2), data-byte registers (3), and the 29-bit CAN identifier (4) into the accelerator's register bank. The status register (5) indicates that the frame-ready interrupt is set, the frame is classified as unsafe, and the current-limit fault code is asserted. The UART output (bottom) shows the decoded unsafe frame and the reported error code, confirming agreement between internal FPGA state and external system-level behavior.

Power Violations

The external capture in Figure 30 shows a CAN frame whose voltage and current fields produce a power request above the configured limit. The SPI waveform confirms correct retrieval of all required fields.

The internal ILA capture in Figure 31 shows:

- computation of requested power,
- clamping of the current field to enforce the power limit,
- assertion of the power-limit fault code (10000),
- and correct latching of the adjusted current value.

UART logs show the unsafe flag, the correct fault code, the computed power, and the clamped current.

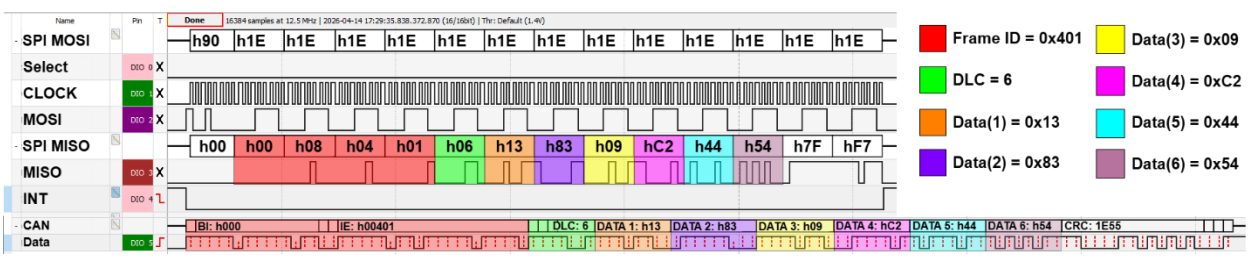

Figure 30: External logic-analyzer capture showing a high-power request CAN frame and the corresponding MCP2515 SPI transaction. The lower trace shows a CAN frame whose encoded voltage and current values produce a power request exceeding the accelerator's configured power limit. The upper trace shows the MCP2515's SPI transfer of the same frame, with the identifier, DLC, and data fields visible in the decoded SPI stream. The capture verifies that the MCP2515 delivers the full frame correctly and that the accelerator receives all fields required to compute and evaluate the power-limit condition.

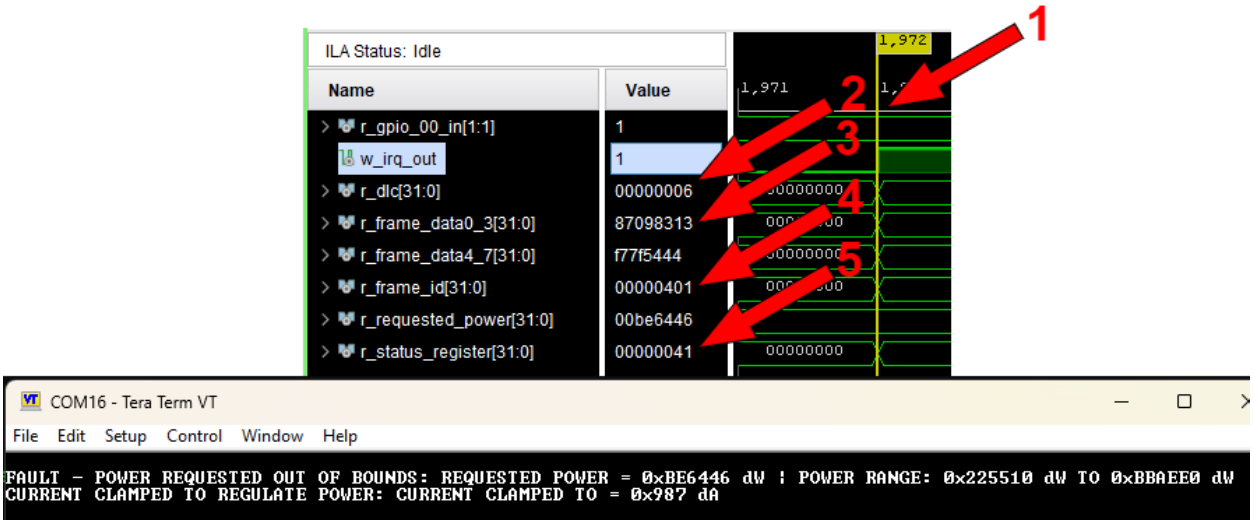

Figure 31 Internal ILA capture demonstrating correct register-loading and fault-classification behavior for the high-power request frame shown in Figure 30. The ILA trace captures the rising edge of the frame-ready interrupt (1), followed by the sequential latching of the DLC register (2), data-byte registers (3), and the 29-bit CAN identifier (4) into the accelerator's register bank. The accelerator computes the requested power from the received voltage and current fields and classifies the frame as unsafe when the calculated value exceeds the configured power threshold. The status register (5) asserts the power-limit fault code, and the UART output (bottom) reports the unsafe frame and associated error code, confirming agreement between internal FPGA state and external system-level behavior.

Identifier and Format Validation

Frames with unknown or non-whitelisted CAN identifiers were consistently rejected. The external capture in Figure 32 shows a non-whitelisted identifier being forwarded by the MCP2515. The internal ILA capture in Figure 33 shows:

- the frame being marked unsafe,
- the STATUS register asserting the ID-violation fault code (00010),
- and the MCU receiving the correct unsafe-frame report.

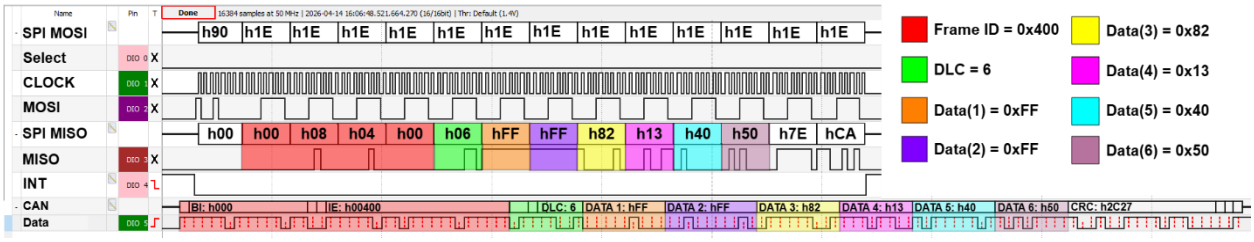

Figure 32: External logic-analyzer capture showing a non-whitelisted CAN identifier and the corresponding MCP2515 SPI transaction. The lower trace shows a CAN frame with an identifier not included in the accelerator's configured whitelist. The upper trace shows the MCP2515's SPI transfer of the same frame, with the ID, DLC, and data fields visible in the decoded SPI stream. The capture verifies that the MCP2515 delivers the frame correctly and that the accelerator receives the full identifier for whitelist evaluation.

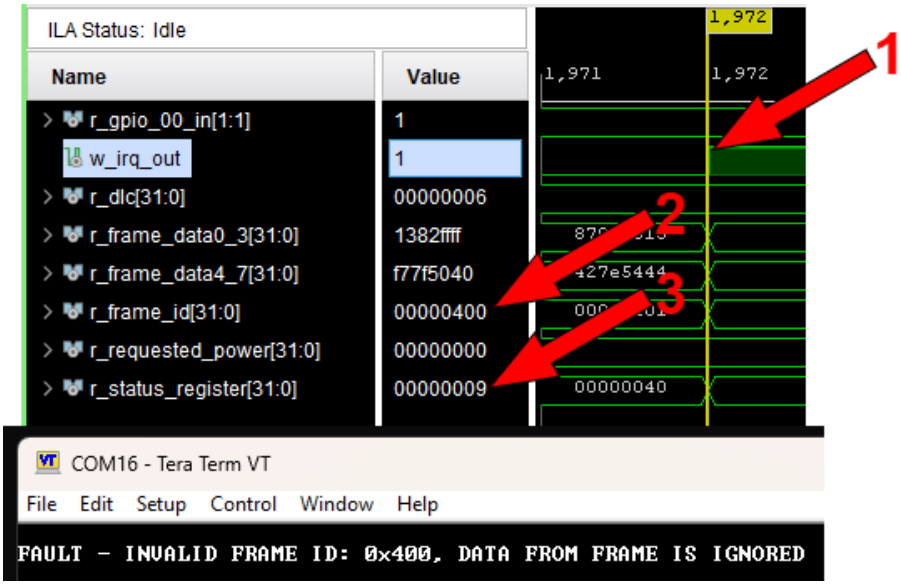

Figure 33: Internal ILA capture demonstrating correct rejection of the non-whitelisted CAN identifier shown in Figure 32. The ILA trace shows the frame-ready interrupt (1) followed by the sequential latching of the DLC and data fields, along with the non-whitelisted identifier (2). The status register (3) indicates that the frame-ready interrupt is set, the frame is classified as unsafe, and the ID-violation fault code is asserted. The UART output (bottom) shows the corresponding unsafe-frame report, confirming that the accelerator's whitelist logic and system-level reporting behave consistently.

Frames whose identifier format did not match the expected extended-ID configuration were also reliably detected. The external capture in Figure 34 shows a standard-frame message (IDE = 0) being forwarded to the accelerator. The internal ILA capture in Figure 35 shows:

- attempted loading of the standard-frame fields,
- immediate detection of the format mismatch,
- assertion of the format-violation fault code (00001),
- and correct reporting to the MCU.

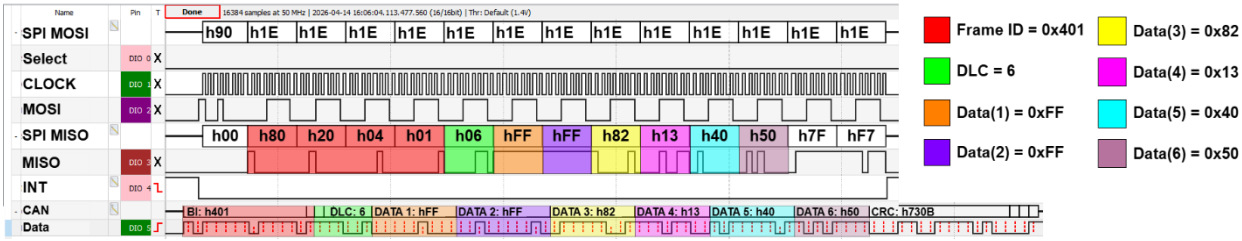

Figure 34: External logic-analyzer capture showing a standard-frame CAN message received when an extended-frame (IDE-bit asserted) format is expected. The lower trace shows a CAN frame transmitted with the IDE bit not asserted, indicating a standard 11-bit identifier. Because the accelerator is configured to accept only extended 29-bit identifiers for this test, the frame constitutes a format-mismatch violation. The upper trace shows the MCP2515's SPI transfer of the same frame to the accelerator, with the standard-ID field visible in the decoded SPI stream. The capture confirms that the MCP2515 correctly forwards the standard frame and that the accelerator receives the full identifier and data fields for validation against the expected extended-frame format.

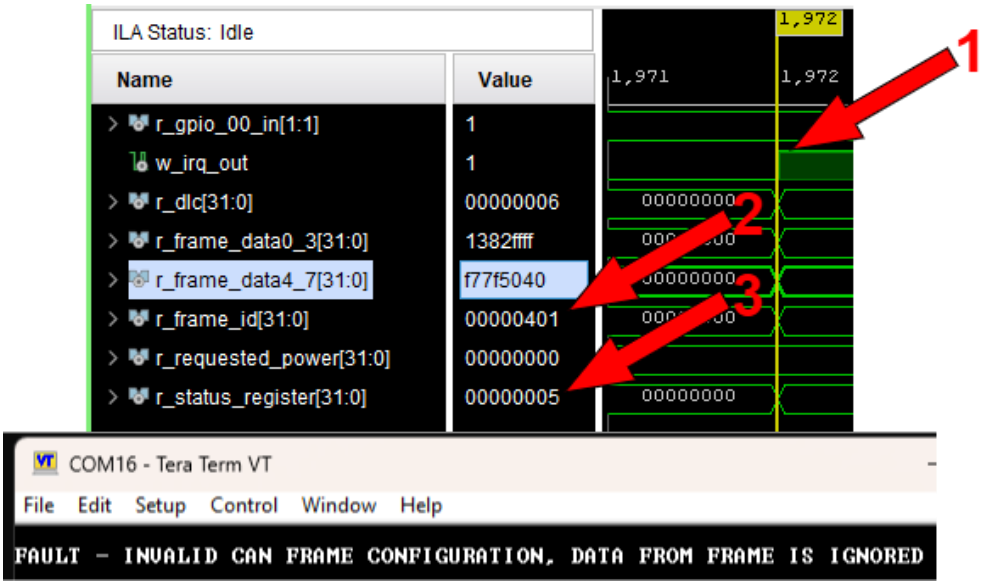

Figure 35: Internal ILA capture demonstrating correct detection and rejection of a standard-frame message when an extended-frame (IDE-bit asserted) format is expected. The ILA trace captures the rising edge of the frame-ready interrupt (1), followed by the attempted loading of the standard 11-bit identifier, DLC, and data fields into the accelerator's register bank (2). Because the IDE bit is not asserted, the accelerator classifies the frame as unsafe and asserts the extended-frame format-violation fault code. The status register (3) reflects the unsafe classification, and the UART output (bottom) shows the corresponding unsafe-frame report, confirming agreement between internal FPGA state and external system-level behavior.

Timing Consistency

Timing-consistency measurements showed highly repeatable behavior across all trials. The internal ILA capture in Figure 36 demonstrates:

- a fixed number of cycles between the MCP2515 interrupt and the accelerator's frame-ready signal,
- consistent ordering of ID, DLC, and data-byte latching,
- and no timing anomalies even during back-to-back frame tests.

These results confirm that the accelerator provides deterministic timing suitable for integration with real-time power-electronics control loops.

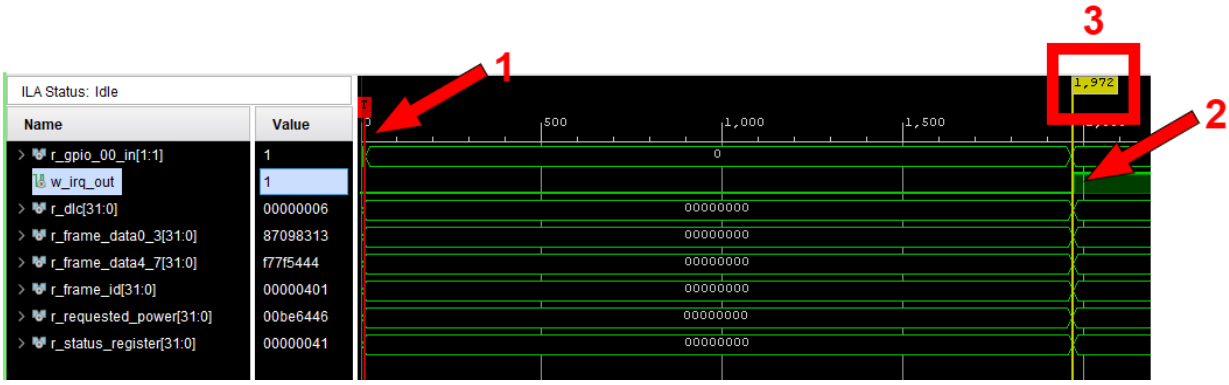

Figure 36: Internal ILA capture demonstrating consistent interrupt-to-data-ready latency across all verified frames. The trace shows the falling edge of the MCP2515 frame-ready interrupt (1) followed by the rising edge of the accelerator's frame-data-ready signal (2). The number of clock cycles between these two events (3) remains constant across all ILA captures, indicating that the SPI-to-register pipeline, rule-engine evaluation, and register-bank update sequence execute with fixed latency. This repeatable cycle count confirms that the accelerator provides deterministic timing behavior suitable for integration into real-time EVCS control loops.

System-Integration Behavior

End-to-end tests using the Raspberry Pi System Controller Simulation confirmed that the accelerator integrates cleanly into the broader EVCS supervisory-control workflow. Mixed-traffic tests demonstrated correct behavior across valid frames, boundary-value frames, and fault-injection frames. The Analog Discovery 3 confirmed correct SPI sequencing throughout all tests, and the MCU successfully read all register fields for each processed frame.

Summary of Findings

The verification results collectively demonstrate that the CAN limit-checking hardware accelerator meets all functional and safety objectives defined earlier in this chapter. The accelerator reliably enforces configured safety limits, correctly identifies malformed or unauthorized frames, and maintains deterministic behavior across repeated trials. These findings validate the accelerator's suitability as a hardware-enforced safety layer for EVCS power-electronics controllers and establish a strong foundation for future work involving

real-time integration, expanded rule sets, and deployment in higher-power charging environments.

FUTURE WORK

The work presented in this thesis establishes a hardware-accelerated foundation for enforcing safety and security on the internal CAN networks of electric vehicle charging stations. While the accelerator has been fully validated in a controlled test environment, several opportunities remain to extend the design toward a deployable, production-grade system. These future directions focus on integrating the accelerator into the broader EVCS control architecture, strengthening its role within a hardened supervisory platform, and expanding its capabilities to meet the operational demands of modern charging infrastructure.

Full EVCS Control-Loop Integration

A natural next step for this work is the integration of the hardware accelerator into a complete EVCS control loop. The current test environment emulates the System Controller and power-electronics controller using a Raspberry Pi and a RadPC Hydra MCU, which provides a controlled and reproducible platform for functional verification. While this setup is effective for validating the accelerator's behavior in isolation, it does not capture the full range of interactions, timing constraints, and subsystem dependencies present in a production-grade charging station. Integrating the accelerator into a full EVCS control architecture would enable evaluation under realistic operating conditions and reveal system-level behaviors that cannot be observed in a simplified testbench.

A complete EVCS control loop includes several tightly coupled components: the supervisory controller responsible for coordinating charging sessions and enforcing high-level policies; the power-electronics controller responsible for regulating voltage, current, and power

delivery; the cooling and safety subsystems that respond to thermal and electrical conditions; and the protocol-translation modules that implement CHAdeMO, CCS, or ISO 15118 communication. Each of these components exchanges time-sensitive information over the internal CAN network, and their interactions influence both the frequency and structure of CAN traffic. Embedding the accelerator into this environment would allow assessment of its performance when subjected to realistic message rates, subsystem interactions, and asynchronous events.

System-level integration would also expose the accelerator to operational scenarios that are difficult to reproduce in a laboratory setting. These include rapid power-setpoint changes during dynamic charging, thermal derating events triggered by cooling-system feedback, handshake sequences during charging initiation and termination, and error-recovery or retry logic invoked during communication faults. Evaluating the accelerator under these conditions would provide insight into its behavior during transient events, its ability to maintain deterministic timing under load, and its interaction with the PE controller's real-time control loop. Such testing is essential for demonstrating that the accelerator can operate reliably within the complex, multi-component environment of a commercial EVCS.

A system-level integration effort would therefore represent a major step toward deployment, enabling validation of the accelerator's safety guarantees, timing behavior, and interoperability within a complete charging-station architecture. It would also provide the foundation for integrating the accelerator with more advanced controller platforms, including the malware-tolerant MCU architecture described in related work.

Integration with the CyberShield Malware-Tolerant Architecture

A significant opportunity for future work is the integration of the CAN safety accelerator with *CyberShield*, the malware-tolerant execution platform developed at Montana State University (Figure 37) [34]. *CyberShield* is a hardware-based malware-mitigation architecture that uses randomized instruction codes within functionally equivalent processor cores to detect when malicious code is injected. The system is built around multiple redundant cores that execute the same program using different instruction-code assignments, ensuring that any unauthorized modification to the instruction stream produces detectable divergence. All binaries must be compiled, encrypted, and obfuscated using a custom toolchain, then decrypted and loaded into each core through a trusted execution environment. During runtime, *CyberShield* continuously verifies that the instruction streams across cores remain consistent with the obfuscation key; any divergence triggers an immediate halt, malware removal, and rollback to a known-good execution point. Integrating the CAN safety accelerator with *CyberShield* would create a unified platform that combines hardware-level message validation with hardware-level malware detection, strengthening the resilience of EVCS supervisory controllers against both cyber-physical manipulation and firmware-level compromise.

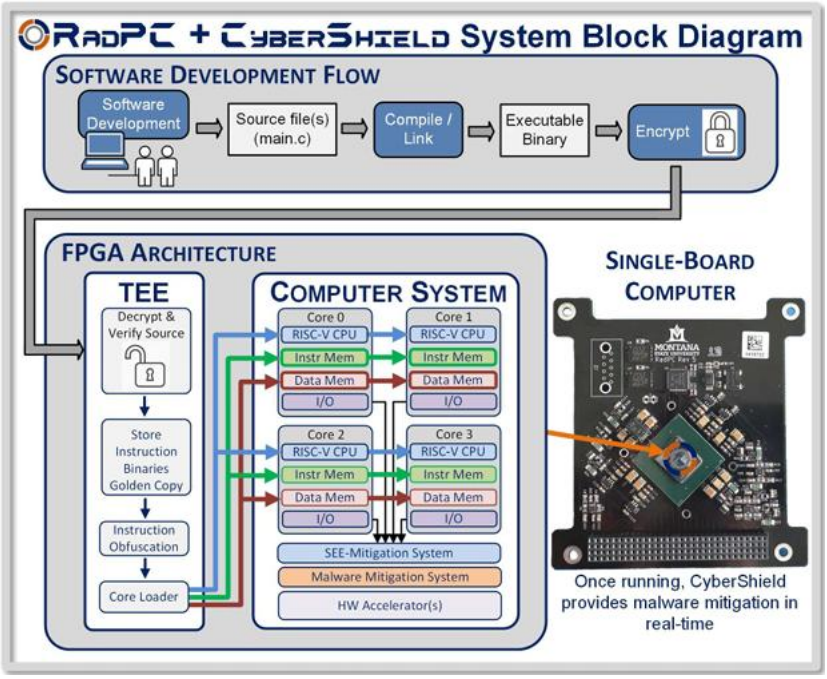

Figure 37: System block diagram of *CyberShield* [34].

Integrating the hardware accelerator into this architecture would create a multi-layered defense system in which *CyberShield* protects the supervisory firmware from tampering, while the accelerator enforces real-time validation of CAN identifiers and power-limit constraints. This pairing would allow the EVCS power-electronics controller to maintain safe operation even under sophisticated cyber-physical attacks. For example, if an attacker attempted to modify the PE controller firmware to bypass limit checking or inject unsafe setpoints, *CyberShield's* redundant-core verification would detect the divergence, while the accelerator would continue to block unsafe CAN frames at the hardware boundary.

A combined *CyberShield*-accelerator system would provide a hardened EVCS controller capable of resisting firmware-level compromise, protocol-level manipulation, and unsafe CAN-borne commands simultaneously. This integration represents a critical step toward deploying the accelerator in commercial EV charging stations, where adversaries may have both

physical access and the ability to manipulate internal communication networks. Evaluating the combined system under adversarial conditions, such as firmware tampering, spoofed CAN traffic, and coordinated subsystem manipulation, would provide strong evidence of its suitability for real-world deployment.

Runtime-Configurable Rule Engine and Dynamic Safety Policies

A further opportunity for future work is extending the accelerator from a statically configured rule engine into a dynamically reconfigurable safety module capable of adapting to different charger models, power-module configurations, and evolving safety requirements. In the current design, identifier whitelists and voltage, current, and power limits are fixed at software compile time. While this approach simplifies verification and ensures deterministic behavior, commercial EVCS deployments will inevitably require periodic updates to these ranges, for example, adjusting over- and under-voltage thresholds as hardware ages, adding new allowable operating envelopes, or modifying current and power limits to match firmware revisions or grid-service participation. Introducing runtime configurability, however, also introduces a new vulnerability: the configuration interface itself becomes part of the attack surface. Any mechanism that allows field updates to safety-critical parameters must therefore be designed with strong authentication, integrity protection, and isolation from untrusted domains. One approach is to implement a protected configuration interface accessible only to a trusted execution domain, such as the *CyberShield* architecture described previously, ensuring that only authenticated and integrity-verified updates can modify safety-critical parameters. This would allow the accelerator to enforce hardware-level safety guarantees while still accommodating the operational flexibility required in modern EVCS deployments.

A dynamically configurable rule engine would also enable more advanced safety policies. For example, the accelerator could enforce charger-specific derating curves, module-dependent current limits, or time-varying constraints driven by thermal conditions or grid-service participation. Incorporating these capabilities would allow the accelerator to serve not only as a static safety filter but as an adaptive hardware co-processor that participates directly in the charger's real-time control strategy. Developing a secure, runtime-configurable rule engine would therefore represent a meaningful evolution of the accelerator architecture. It would bridge the gap between a research prototype and a deployable safety component capable of supporting the diverse and dynamic requirements of commercial EV charging systems.

Extending Rule Coverage and Parallelizing Evaluation

Additional safety rules could be incorporated into the accelerator to broaden its protection scope and support more sophisticated EVCS behaviors. The current architecture is modular, and new rules can be added by extending the state machine or by introducing additional combinational checks that operate on the parsed voltage, current, and power fields. For example, one valuable extension would be a voltage-slew-rate rule that detects unusually rapid changes in requested voltage—an indicator of potentially malicious behavior such as repeatedly commanding sharp voltage drops to destabilize the grid or force protective shutdowns. Other candidates include rules for detecting inconsistent multi-frame supervisory messages, enforcing minimum dwell times between successive limit changes, or validating cross-field relationships such as ensuring that current requests scale reasonably with voltage. As the rule set grows, the design could also benefit from parallelizing rule checks, allowing multiple constraints to be evaluated concurrently rather than sequentially within the state machine. This would reduce

per-frame latency, simplify timing closure for more complex rule sets, and enable future accelerators to support richer, protocol-aware safety policies without sacrificing determinism.

CONCLUSION

Electric vehicle charging stations are rapidly becoming critical components of the modern power grid, yet their internal communication networks continue to rely on the unauthenticated and inherently vulnerable CAN protocol. This thesis addressed a key gap in EVCS cybersecurity by developing a hardware-accelerated CAN-safety monitor capable of enforcing identifier validation and real-time limit checking for voltage, current, and power. The goal was to create a deterministic, tamper-resistant, and low-latency protection mechanism that operates independently of firmware integrity and provides a final line of defense against unsafe or malicious CAN-borne commands.

The system architecture introduced in this work integrates a dedicated rule engine, reciprocal lookup table, register bank, and MCP2515 control logic into a cohesive hardware accelerator positioned between the CAN controller and the power-electronics MCU. This design enables the accelerator to intercept every CAN frame, validate its structure and identifier, compute derived quantities such as power, and enforce safety constraints before the frame reaches the control loop. By offloading these operations into hardware, the accelerator eliminates timing variability, reduces attack surface, and ensures that safety-critical checks cannot be bypassed through firmware compromise.

Verification results demonstrate that the accelerator performs these functions reliably across a wide range of operating conditions. Logic-analyzer captures confirm correct SPI communication, byte ordering, and frame acquisition. Internal ILA traces show consistent register-loading behavior, correct classification of safe and unsafe frames, and proper assertion of fault codes for identifier violations, format mismatches, and limit-exceeding requests.

Timing-consistency analysis further shows that the latency between the MCP2515's frame-ready interrupt and the accelerator's data-ready signal remains constant across all test cases, indicating deterministic behavior suitable for real-time EVCS control loops. Together, these results validate the functional correctness, safety-rule enforcement, and temporal stability of the accelerator.

The contributions of this thesis demonstrate that hardware-enforced CAN safety is both feasible and effective for EV charging systems. By providing a deterministic and tamper-resistant validation layer, the accelerator strengthens the resilience of EVCS power-electronics controllers against spoofed messages, malformed frames, and unsafe command injection. As charging infrastructure continues to scale and cyber-physical threats grow more sophisticated, embedding dedicated security co-processors into EVCS architectures offers a promising path toward safer and more robust charging networks.

This work establishes a foundation for future system-level integration, including deployment within full EVCS control loops and incorporation into hardened execution platforms such as the *CyberShield* malware-tolerant MCU. These directions point toward a comprehensive, multi-layered defense architecture capable of protecting both the firmware and the communication pathways of next-generation charging stations.

REFERENCES CITED

[1] E. Wood, B. Borlaug, M. Moniot, D. Lee, Y. Ge, F. Yang and Z. Liu, "The 2030 National Charging Network: Estimating U.S. Light-Duty Demand for Electric Vehicle Charging Infrastructure," 2023. [Online]. Available: https://docs.nrel.gov/docs/fy23osti/85654.pdf. [Accessed 3 March 2026].

[2] U.S. Department of Energy, Alternative Fuels Data Center, "Electric Vehicle Charging Infrastructure Trends form the Alternative Fueling Station Locator: Second Quarter 2024," U.S. Department of Energy, Washington, D.C., 2024.

[3] D. Drake, H. S. Mavikumbure and V. C. a. M. Manic, "Cyber-Physical Security Trends of EV Charging Systems: A Survey," November 2024. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10906000. [Accessed 11 Februrary 2026].

[4] S. Hamdare, O. Kaiwartya, M. Aljaidi, M. Jugran, Y. Cao, S. Kumar, M. Mahmud, D. Brown and J. Lloret, "Cybersecurity Risk Analysis of Electric Vehicles Charging Stations," *Sensors,* vol. 23, no. 15, pp. 6716-6716, July 2023.

[5] K. Rhode, "Cyber Security of DC Fast Charging: Potential Impacts to the Electric Grid," U.S. Department of Energy - Office of Scientific and Technical Information, 2019.

[6] S. Acharya, Y. Dvorkin and R. Karri, "Public Plug-in Electric Vehicles + Grid Data: Is a New Cyberattack Vector Viable?," *IEEE Transactions on Smart Grid,* vol. 11, no. 6, 2020.

[7] S. Acharya, H. Khan, R. Karri and a. Y. Dvorkin, "MaDEVIoT: Cyberattacks on EV Charging Can Disrupt Power Grid Operation," *IEEE Power & Energy Society Innovative Smart Grid Technologies Conference (ISGT),* pp. 1-5, 2024.

[8] C. Alcaraz, J. Lopez and S. Wolthusen, "OCPP Protocol: Security Threats and Challenges," *IEEE Transactions on Smart Grid,* vol. 8, no. 5, pp. 2452-2459, 2017.

[9] S. Köhler, R. Baker, M. Strohmeier and a. I. Martinovic, "Brokenwire: Wireless Disruption of CCS Electric Vehicle Charging,," in *Proceedings 2023 Network and Distributed System Security Symposium*, 2023.

[10] T. Nasr, S. Torabi, E. Bou-Harb, C. Fachkha and C. Assi, "Power jacking your station: In-depth security analysis of electric vehicle charging station management systems," *Computers & Security,* vol. 112, p. 102511, 2022.

[11] G. Vailoces, A. Keith, A. Almehmadi and K. El-Khatib, "Securing the Electric Vehicle Charging Infrastructure: An In-Depth Analysis of Vulnerabilities and Countermeasures," in *Proceedings of the International ACM Symposium on Design and Analysis of Intellegent Vehicular Networks and Applications*, 2023.

[12] H. Sh, Y. He, S. Song, J. Zhuge and a. J. Mao, "Physical-Layer Signal Injection Attacks on EV Charging Ports: Bypassing Authentication via Electrical-Level Exploits," arXiv.org, 2025. [Online]. Available: https://arxiv.org/abs/2506.16400?utm_source=chatgpt.com. [Accessed 14 March 2026].

[13] S. Acharya, Y. Dvorkin, H. Pandzic and R. Karri, "Cybersecurity of Smart Electric Vehicle Charging: A Power Grid Perspective," *IEEE Access,* vol. 8, pp. 214434-214453, 2020.

[14] L. Pawlik, J. L. Wilk-Jakubowski, P. T. Grabski and G. Wilk-Jakubowski, "Securing the Electrified Future: A Systematic Review of Cyber Attacks, Intrusion and Anomaly Detection, and Authentication in Electric Vehicle Charging Infrastructure," *Energies,* vol. 18, no. 18, p. 4847, 2025.

[15] P. F. D. Araujo-Filho, A. J. Pinheiro, G. Kaddoum, D. R. Campelo and F. L. Soares, "An Efficient Intrusion Prevention System for CAN: Hindering Cyber-Attacks With a Low-Cost Platform," *IEEE Access,* vol. 9, pp. 166855-166869, 2021.

[16] C.-W. Lin and A. Sangiovanni-Vincentelli, "Cyber-Security for the Controller Area Network (CAN) Communication Protocol," in *International Conference on Cyber Security*, Alexandria, VA, 2012.

[17] S. Khandelwal and S. Shreejith, "Real-Time Zero-Day Intrusion Detection System for Automotive Controller Area Network on FPGAs," in *IEEE 34th International Conference on Application-specific Systems, Architectures and Processors*, Porto, Portugal, 2023.

[18] K. Iehira, H. Inoue and K. Ishida, "Spoofing attack using bus-off attacks against a specific ECU of the CAN bus," in *15th IEEE Annual Consumer Communications & Networking Conference*, Las Vegas, NV, 2018.

[19] S. Ahmed and F. M. Dow, "Electric vehicle technology as an exploit for cyber attacks on the next generation of electric power systems," in *Proc. of the 2016 4th International Conference on Control Engineering*, 2016.

[20] A. Chandwani, S. Dey and A. Mallik, "Cybersecurity of Onboard Charging Systems for Electric Vehicles - Review, Challenges, and Countermeasures," *IEEE Access,* vol. 8, pp. 226982-226998, 2020.

[21] R. M. Pratt and T. E. Carroll, "Vehicle Charging Infrastructure Security," in *IEEE International Conference on Consumer Elecytronics (ICCE)*, 2019.

[22] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham and S. Savage, "Experimental Security Analysis of a Modern Automobile," in *IEEE Symposium on Security and Privacy*, 2010.

[23] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, A. C. Karl Koscher, F. Roesner and T. Kohno, "Comprehensive Experimental Analyses of Automotive Attack Surfaces," in *Proceedings of the USENIX Security Symposium*, 2011.

[24] Keen Security Lab, "Experimental Security Assessment of BMW Cars: A Summary Report," Tencent Keen Security Lab, 2018.

[25] J. e. a. Johnson, "Cybersecurity for Electric Vehicle Charging Infrastructure," Sandia National Laboratories, 2022.

[26] M. Basnet and M. Ali, "Exploring cybersecurity issues in 5G enabled electric vehicle charging station with deep learning," *IET Generation, Transmission & Distribution,* vol. 15, no. 24, pp. 3435-3449, 2021.

[27] B. Elend and T. Adamson, "Cyber security enhancing CAN transceivers," in *Proc. 16th Int. CAN Conf*, Nuremberg, Germany, 2017.

[28] S. Fries and R. Falk, "Electric Vehicle Charging Infrastructure – Security Considerations and Approaches," *ResearchGate,* 2012.

[29] D. Reeh, F. C. Tapia, Y.-W. Chung, B. Khaki, C. Chu and R. Gadh, "Vulnerability Analysis and Risk Assessment of EV Charging System under Cyber-Physical Threats," *IEEE Xplore,* 2019.

[30] W. Chen and J. Guo, "Federated learning-based cyber attack detection on Electric Vehicles (EVs) in AC/DC hybrid distribution systems," *Journal of Engineering and Applied Science,* vol. 72, no. 1, 2025.

[31] M. D. Natale, H. Zeng, P. Giusto and A. Ghosal, Understanding and Using the Controller Area Network Communication Protocol, New York, NY: Springer Science+Business Media, 2012.

[32] Microchip Technology Inc., "MCP2515 Stand-Alone CAN Controller with SPI Interface," Microchip Technology Inc., 2005.

[33] T. E. Carroll, S. Smith and B. LeMeres, "Taxonomy of High-Level Controls for EV Charging Stations," Pacific Northwest National Laboratory, Richland, WA, 2024.

[34] G. Perkins, "Providing Secure Boot of Obfuscated Opcodes for CyberShield Using a Trusted Execution Envirnoment," 2025. [Online]. Available: https://www.cs.montana.edu/izurieta/thesis/PerkinsThesis.pdf.