Faculty of Engineering
Department of Electrical and Electronics Engineering

EE 401 FINAL REPORT

Fall 2024

Test Software and Bootloader Development for an ECU

Submitted by

Hasan Alp Doyduk
S025015

İsmail Akbaş
S024094

Supervisor

Asst. Prof. Umut Başaran

APPROVAL PAGE

Faculty of Engineering
Department of Electrical and Electronics Engineering

EE 401 FINAL REPORT

Fall 2024

Hasan Alp Doyduk
İsmail Akbaş

Test Software and Bootloader Development for an ECU

Jury Members:

Supervisor	:	**Asst. Prof. Umut Başaran**	______________________
Jury Member 1	:	**Prof. Dr. Çiğdem Eroğlu Erdem**	______________________
Jury Member 2	:	**Dr. Hamza Makhamreh**	______________________

ABSTRACT

This report presents the development of test software and a bootloader system for an automotive ECU (Electronic Control Unit). The project stages include component selection, schematic design, PCB layout design and software development. These stages enable the establishment of necessary communication between hardware elements using various communication protocols. Additionally, a communication interface must be designed to allow testing and control of the ECU over serial or CAN communication protocols. Furthermore, these stages make it possible to implement a bootloader that provides secure and efficient firmware updates for the ECU.
The project faces challenges such as ensuring reliable data transfer over the CAN bus, achieving compatibility between the bootloader and ECU firmware, and developing robust error-handling mechanisms. Despite these challenges, the project aims to deliver a modular and scalable system for ECU testing and firmware updates.
Future work will expand to include additional automotive scenarios and optimize the bootloader's integration with diagnostic tools for real-time debugging. The successful completion of this project will contribute significantly to the reliability and functional safety of automotive systems, aligning with industry trends in smart and connected vehicles.

ACKNOWLEDGMENT

We would like to extend our heartfelt gratitude to Asst. Prof. Umut Başaran for invaluable guidance, unwavering support, and insightful advice throughout this journey.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

CAN	Control Area Network
DRC	Design Rule Check
ECU	Electrical Control Unit
ERC	Electrical Rule Check
LDO	Low Dropout Regulator
MCU	Microcontroller Unit
TVS	Transient Voltage Suppressor

1 INTRODUCTION

The increasing integration of electronics in the automotive industry has established ECUs as a foundational component of modern vehicles. ECUs manage critical operations such as engine control, braking systems and in-vehicle communications ensuring that vehicles function with precision, safety and efficiency. The growing complexity of ECU software, driven by advancements in automotive technology necessitates innovative methods for software testing and firmware updates to uphold system reliability and safety [1, 2].

Bootloaders have become essential components for safe and efficient firmware management, providing a critical bridge between software and hardware. By enabling seamless firmware updates through communication interfaces such as CAN, bootloaders eliminate the need for traditional programmers, simplifying the process of patching bugs, addressing security vulnerabilities, and introducing new features. This approach not only minimizes system downtime but also ensures accessibility without requiring extensive knowledge of the device. To further enhance efficiency, incremental update mechanisms have been introduced, significantly reducing update times and memory consumption compared to earlier methods that required complete rewrites. Additionally, studies highlight the importance of modular bootloader designs and robust error-handling strategies in maintaining system reliability during firmware updates [1, 3].

The need for rigorous testing methodologies is equally significant. Software problems can be found more precisely and consistently with automated tools for assessing ECU performance. These technologies improve testing and minimize human error by verifying real-time outputs using predefined reference signals. Improved test automation ensures that ECUs operate dependably in various scenarios while also increasing development productivity [2, 4].

Effective communication protocols play a crucial role in testing and updating ECUs. CAN bus is mostly used in the automotive industry since its robustness and reliability. Serial communication often employed during the development phase complements CAN by offering an easy yet efficient method for interfacing with ECUs. This combination ensures flexibility and compatibility with existing automotive communication standards [5,6].

This project focuses on developing a comprehensive framework for test software and a bootloader system tailored to automotive ECUs. The proposed bootloader incorporates incremental updates to improve firmware management efficiency, while the test software is designed to simulate realistic automotive scenarios for evaluating ECU functionality. This dual approach addresses key challenges in ECU development by ensuring both reliability and adaptability. Additionally, the solutions align with industry trends toward modular and scalable systems preparing the foundation for future innovations [1,7].

By integrating these technologies, this project aims to improve the efficiency of firmware updates and the reliability of testing processes in automotive systems. The results are expected to contribute to the evolution of connected and autonomous vehicles, offering scalable solutions for increasingly software-driven automotive environments [2,4,7].

2 METHODOLOGY

This study focuses on test software and the development of a bootloader system for an automotive ECU including a communication framework to verify functionality and facilitate secure firmware updates. The research methodology combines experimental and quantitative approaches to methodically design, implement, and evaluate the system's components, ensuring reliability and compliance with automotive specifications. The methodology tries to explain each phase of the project to ensure reproducibility and clarity for future development.

The bootloader system is designed to support incremental updates, a modern approach that optimizes memory usage and reduces the time required for firmware modifications. Instead of traditional methods, which require rewriting the entire memory, incremental updates focus on specific sections; therefore, improving efficiency and minimizing disruption. Previous studies indicate the importance of modular designs to ensure compatibility between hardware platforms while maintaining reliability during the update process. The bootloader primarily utilizes the Controller Area Network (CAN) protocol for its robustness, reliability, and ability to handle real-time data transmission. In addition to CAN, serial communication, such as UART, has been integrated into the bootloader to enhance flexibility during initial testing and debugging phases. Serial communication provides a simpler and more accessible interface, particularly beneficial during early development when lower hardware complexity and ease of use are required. [1,3].

In parallel, the test software is designed to simulate realistic automotive scenarios, enabling a detailed assessment of the ECU functionality. Automating test processes ensures reproducibility and accuracy, reducing potential human errors during validation. Such automated frameworks have been shown to improve the reliability of system evaluations, particularly in handling time-sensitive and complex ECU operations [2].

The tools utilized in this project have been carefully selected to meet the requirements of hardware design, software development and testing. KiCad circuit design has developed significantly;

schematic diagrams have been finished, and work is now being done to complete the PCB layout. Market research has guided the selection of automotive-specific components, including microcontrollers, voltage regulators and transceivers, all chosen for their compliance with AEC-Q standards and reliability in automotive environments.

For software development, MATLAB Simulink is integrated with the MPLAB X Integrated Development Environment (IDE), which allows for efficient code generation and distribution. This integration allows for optimized and correct code transfer to the microcontroller using MPLAB-specific blocks within Simulink. By successfully programming basic features such as LED blinking or digital port HIGH and LOW signals, this method has validated the compatibility of the instruments and the feasibility of the design. These developments provide the basis for more complex operations, including serial communication (UART), which is currently the primary focus of the project. Although the full implementation of serial communication is not complete, it serves as a stepping stone towards adopting CAN communication in later stages.

2.1 Problem Formulation

The main challenge in this project is to design and implement a system capable of testing and updating the firmware of an automotive ECU through serial or CAN communication protocols. The ECU must be thoroughly tested under various operating conditions to ensure its functionality and reliability. Additionally, the bootloader must enable secure and efficient firmware updates, facilitating the integration of new features and compliance with automotive safety standards. Several critical considerations arise during the development process. The communication interface must support both low-speed serial communication and high-speed, real-time CAN communication. It is essential to maintain data integrity during firmware updates, ensuring no corruption occurs in the ECU's memory. Furthermore, the system must include robust error-detection and recovery mechanisms to address potential communication failures or power disruptions.

2.2 Proposed Solution

The proposed solution involves developing a modular system that integrates test software and a bootloader for ECUs. When the system is examined under two main headings hardware and software, a PCB is designed in the hardware section that can provide lossless communication by supporting both serial and CAN protocols, while the software section is designed to interact with the ECU through a specially created communication interface that supports both serial and CAN protocols. The test software simulates real-world automotive scenarios to verify the performance of the ECU, while the bootloader enables safe and efficient firmware updates. The system architecture is shown in Figure1.

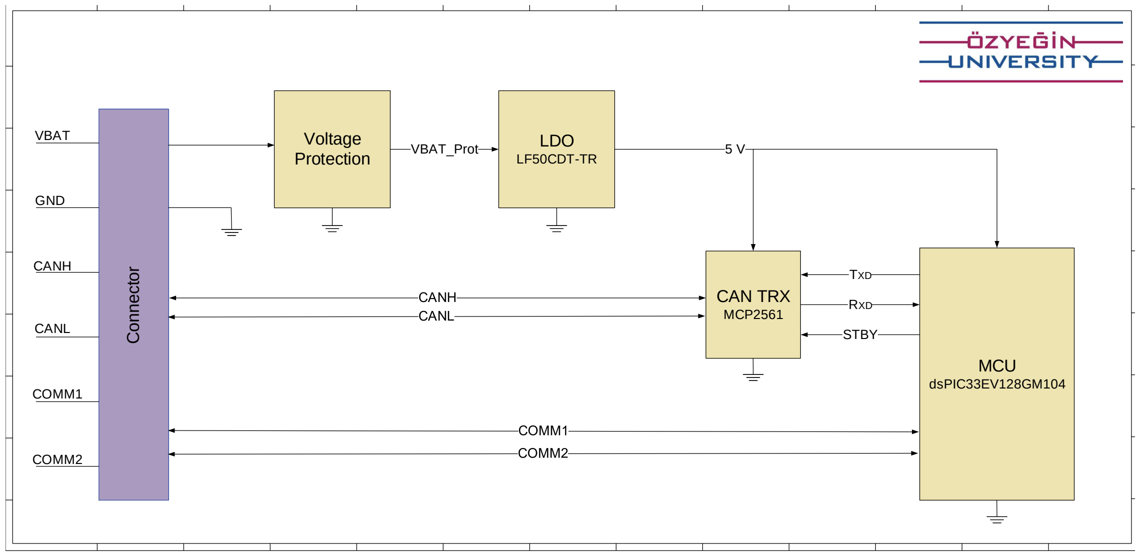

Figure 1: System block diagram.

2.3 Design Methods and Tools

The design of this project includes the detailed planning and implementation of hardware and software components. The hardware design involves selecting components based on automotive standards, creating schematic diagrams, and designing a compact PCB layout using KiCad. The software development process incorporates the combined use of MATLAB Simulink and MPLAB X applications. C code is generated by running Simulink models designed for various scenarios, considering the PCB's hardware features. Using the MPLAB X IDE, necessary modifications were made to this code, which is embedded into the microcontroller.

2.3.1 Component Selection

The components in the hardware were selected to fulfill key automotive requirements, ensuring reliability and flexibility under various conditions. These components were selected to operate efficiently while being powered by the vehicle's battery voltage, which ranges from 9 V to 16 V, and to withstand a wide temperature range of -40°C to +85°C, ensuring durability in harsh automotive environments. Additionally, the necessary components for implementing serial communication and the CAN protocol have been selected.

Furthermore, all components were required to have AEC-Q certification, which is an industry-standard qualification for automotive-grade components. The AEC-Q certification ensures that the components can endure the unique stressors of automotive environments, such as vibration, thermal cycling, and electromagnetic interference. Specifically, this certification guarantees that the components meet reliability standards for long-term operation under extreme conditions, providing the level of robustness necessary for integration into safety-critical automotive systems. By adhering to AEC-Q standards, the hardware design aligns with the high-quality requirements of the automotive industry, ensuring compatibility and compliance with industry regulations.

- **Low Dropout Regulator**

  LDOs are voltage regulators that provide a stable output voltage when the input voltage is higher than the output. Their main advantage is the ability to operate with a minimal difference between the input and output voltage, known as the dropout voltage. LDOs are used to power low-power devices such as microcontrollers, sensors and communication modules, offering stable voltage with minimal heat dissipation.

  As shown in the Table 1, the LF50CDT-TR LDO was selected from among many options for its features, which are the most suitable for the project. LF50CDT-TR is selected to step down the battery voltage to a stable 5 V DC voltage and it provides output current up to 500 mA. It supplies the microcontroller and CAN transceiver with this output current [8].

Table 1: LDO comparison table.

	Microchip		**Texas Instruments**
	MCP1755S	MIC29302	
Features	**MCP1755/MCP1755S**	**MIC2915X/30X/50X/75X**	**TPSA84A**
Input Voltage	3.6 to 16 V	3 to 26 V (For applications with input voltage 6V or below, see MIC37xxx LDOs.)	Without BIAS: 1.4 V to 6.5 V With BIAS: 1.1 V to 6.5 V
Output Voltage	1.8 to 5 V	5 V	Adjustable operation: 0.8 to 5.15 V ANY-OUT™ operation: 0.8 to 3.95 V
Output Current	300 mA	500 µA (max)	0 to 3 A
Temperature Range	-40°C to 125°C	-40°C to 125°C	-40°C to 125°C
Dropout Voltage	Low (0.5 V)	Low	180 mV

	STMicroelectronics	**STMicroelectronics**	**Analog Devices**
		LF50CDT-TR	
Features	**LDFM50-Q**	**LFXX**	**LT3010-5**
Input Voltage	3 to 16 V	-0.5 to 40 V	3 to 80 V
Output Voltage	5 V	1.5 to 12 V	5 V
Output Current		500 mA	
Temperature Range	-40°C to 125°C	-40°C to 125°C	-40°C to 125°C
Dropout Voltage	400 mV	0.45 V	300 mV

- **CAN Transceiver**

A CAN transceiver enables communication between microcontrollers and other devices in a CAN system. It converts digital signals into differential signals for CAN communication and vice versa, ensuring proper voltage levels and electrical isolation. The transceiver manages the physical layer, facilitating data transmission and reception over twisted-pair cables, and plays a key role in ensuring reliable data transfer within automotive systems. The differential signaling used in the CAN protocol helps maintain signal integrity in noisy automotive environments.

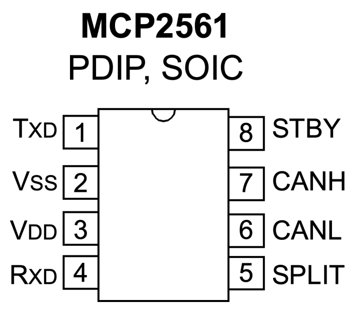

Figure 2: MCP2561 package type.

In this project, the MCP2561 CAN transceiver, as shown in the Figure 2, has been selected. It supports speeds up to 1 Mb/s and operates within a voltage range of 1.8 V – 5.5 V, ensuring reliable communication across a wide temperature range [9].

- **Microcontroller**

The dsPIC33EV128GM104-I/PT is a 16-bit Digital Signal Controller (DSC) from Microchip, capable of processing speeds up to 70 MHz. It features 128 KB of Flash memory and 8 KB of RAM, offering ample storage for complex tasks. The microcontroller supports multiple communication interfaces including CAN, I2C, SPI, and UART, making it highly adaptable for automotive and industrial systems [10].

In this project, the dsPIC33EV128GM104-I/PT acts as the central processing unit, handling communication, control, and data processing, ensuring reliable operation under various conditions. Its high performance and versatility make it ideal for managing real-time tasks, particularly in automotive systems where CAN communication is essential. The compact 44-pin package ensures its suitability for space-limited applications while maintaining robust functionality.

- **Protection Circuit**

Protection circuits using Schottky and TVS diodes protect electronic components from voltage spikes and transients. Schottky diodes provide fast switching and low forward voltage, redirecting current to protect against reverse voltage. TVS diodes clamp high-energy voltage spikes, such as those from electrostatic discharge or lightning, to safe levels. These diodes are essential in automotive and industrial systems, ensuring reliable and long-lasting circuitry.

The protection circuit, with SS8PH10HM3-A/H and SMAJ36CA-HT components, shields against surges and voltage fluctuations. SS8PH10HM3-A/H ensures reverse voltage protection, while SMAJ36CA-HT clamps spikes, stabilizing the battery voltage and protecting downstream components [11, 12].

- **Connector**

  The TSM-106-01-T-DV connector serves as the primary interface for the system, providing essential connections for power, ground, and communication. It supplies the battery input voltage, GND reference, and CAN communication signals (CANH and CANL), ensuring seamless integration with the vehicle's electrical and communication networks. The connector features a 2.54 mm pitch size, allowing for easy plug-in and plug-out, which facilitates quick installation and replacement [13].

2.3.2 Circuit Schematics

After selecting all the necessary components, schematic diagrams were created using KiCad by referencing the application circuits provided in the components' datasheets. A hierarchical structure was adopted to make the schematics more organized and easier to understand. In the project's block diagram, as shown in Figure 3, a root file was created to encompass the other schematic files.

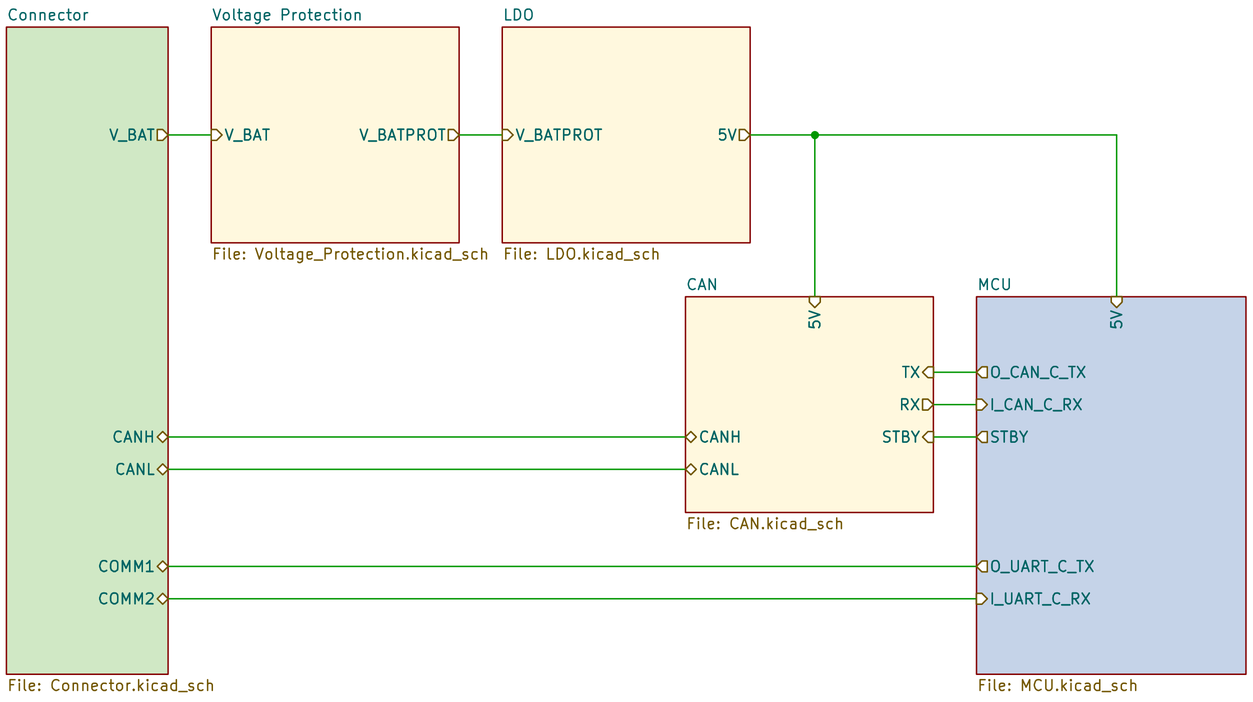

Figure 3: Root hierarchy.

- **LDO Hierarchy**

  The stabilization of input and output power was achieved by placing two decoupling capacitors, as recommended in the LF50CDT-TR datasheet. These capacitors help filter noise and voltage spikes, ensuring a stable voltage supply to the powered components. A 0.1 µF input capacitor (C1) filters high-frequency noise and stabilizes the input voltage, while a 2.2 µF output capacitor (C2) ensures stability and improves transient response. The design, as shown in Figure 4, minimizes component count and board space, using ceramic capacitors with low ESR to maintain stability and performance, with the 2.2 µF capacitor being sufficient for stability, simplifying the design and reducing costs.

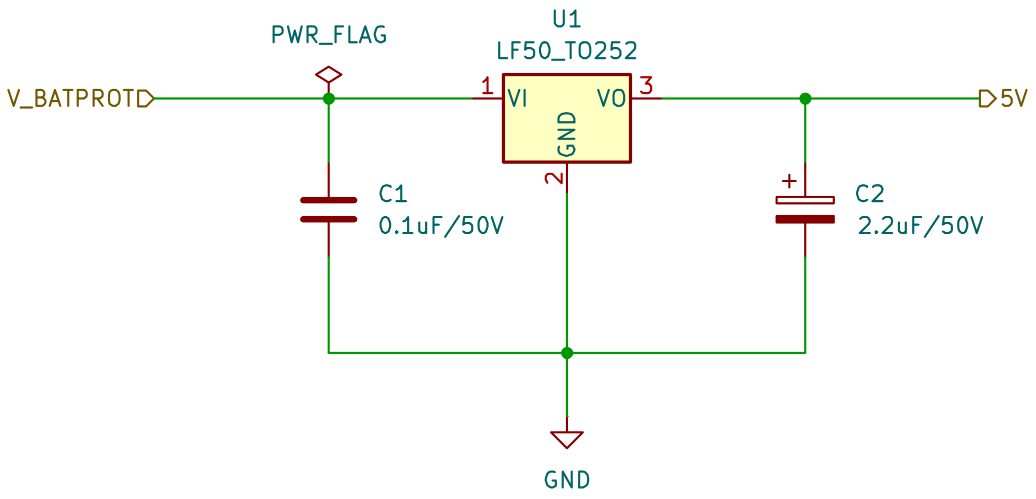

Figure 4: Lf50CDT-TR circuit schematic.

- **CAN Hierarchy**

  The MCP2561 is a high-speed CAN transceiver designed to interface between a CAN protocol controller and the physical two-wire CAN bus. In the circuit schematic, as shown in Figure 5, the transceiver is connected to the CAN bus via the CANH and CANL pins, which are the differential signal lines for data transmission. The VDD pin is connected to the power supply, and the VSS pin is connected to the ground. The TXD and RXD pins interface with the CAN controller, facilitating data transmission and reception. The SPLIT pin, if used, provides a split termination voltage for the CAN bus, enhancing signal integrity. The application circuit also includes decoupling capacitors placed close to the VDD and VSS pins to filter out high-frequency noise and stabilize the power supply voltage. Additionally, resistors may be used for setting the bus termination and biasing, ensuring proper signal levels and impedance matching.

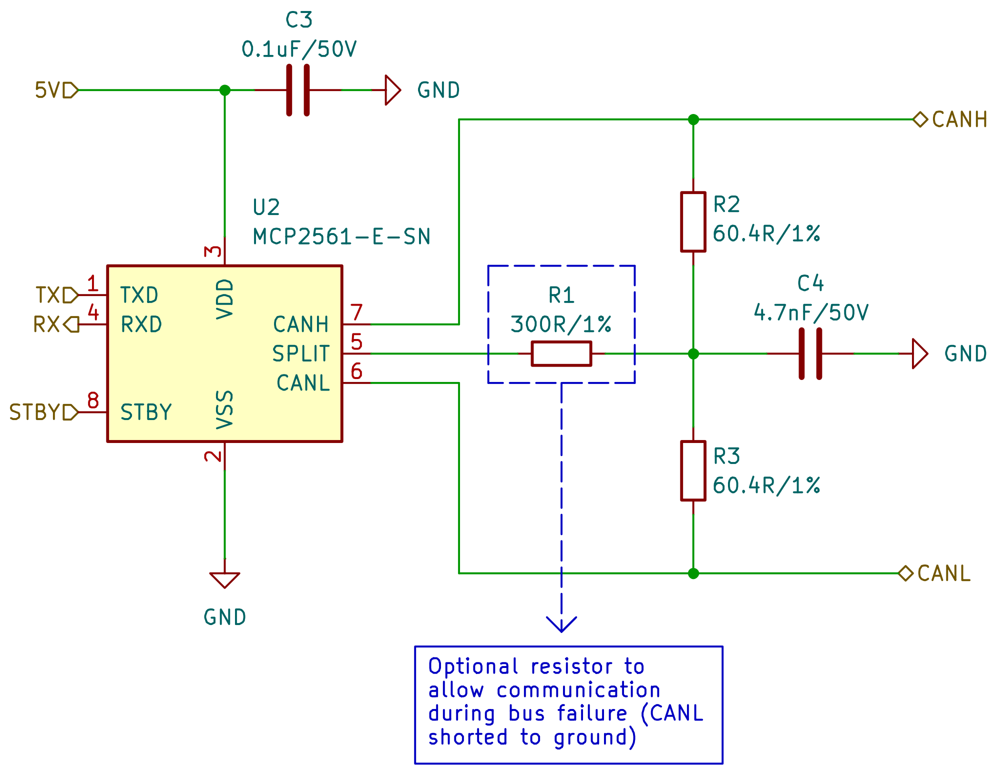

Figure 5: MCP2561 circuit schematic.

- **MCU Hierarchy**

The MCU schematic, as shown in Figure 6, includes several key components essential for its operation. It features a CX3225CA 12MHz oscillator that provides the necessary clock signals to ensure the MCU operates at the correct frequency, facilitating accurate timing for various tasks. The analog input, PWM output, and digital output pins were used to test the MCU. Additionally, the schematic includes programming pins, which are critical for loading the firmware onto the MCU and enabling communication during the development and debugging phases.

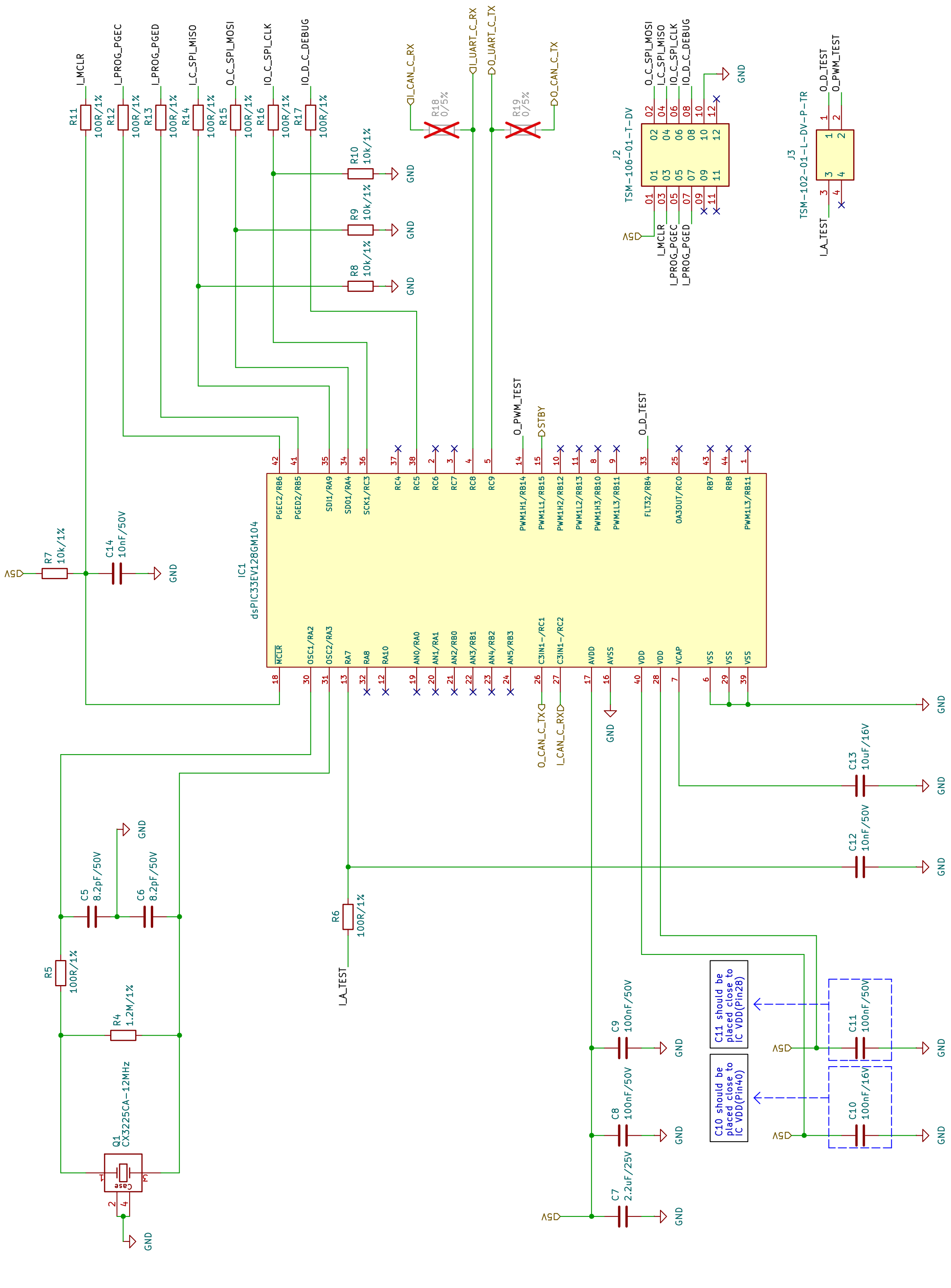

Figure 6: MCU circuit schematic.

- **Voltage Protection Hierarchy**

  The voltage protection hierarchy includes SMAJ36CA-HT, a unidirectional TVS diode designed for electronic components by clamping high-voltage spikes and dissipating excess energy. With a standoff voltage of 36V, it ensures that voltage levels do not exceed safe thresholds, providing protection against electrical overstress. The schematic, as shown in Figure 7, also includes SS8PH10HM3-A/H, a Schottky diode used for fast switching and protection, typically in power supply applications, and provides low forward voltage drop and efficient current conduction. These components work together to ensure the reliability and longevity of the system by preventing damage from voltage spikes and maintaining stable operation.

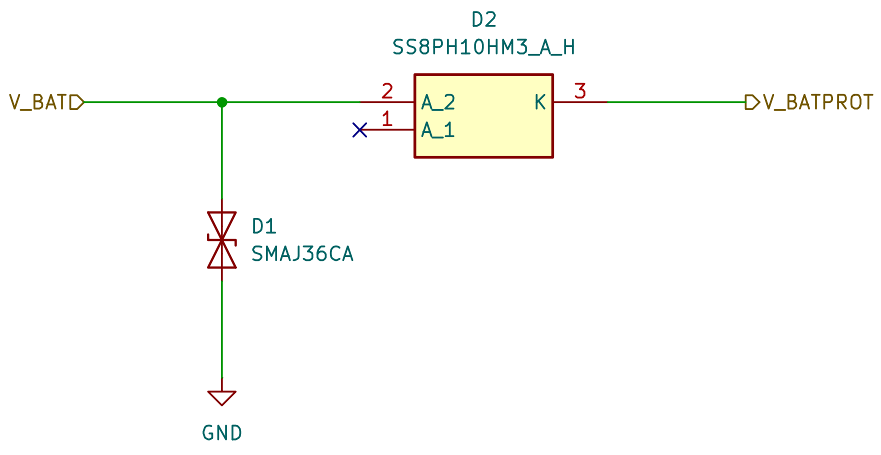

Figure 7: Voltage protector circuit schematic.

- **Connector Hierarchy**

  In the connector schematic, as shown in Figure 8, there are no other circuit components besides the connector and its connections. The battery and ground connections are the power connections, while CANL, CANH, COMM2, and COMM1 are the signal connections. The placement of the pins has been arranged in such a way that minimizes the overlap of traces in the PCB layout section.

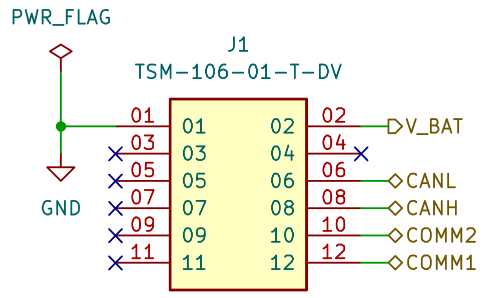

Figure 8: Connector circuit schematic.

2.3.3 PCB Layout

Components in the PCB layout, as shonw in Figure 9, placed using KiCad application. Before starting the layout, ERC was applied to the schematic circuits, and the identified errors were corrected. After all the errors were fixed, the layout phase was initiated. The layout specifications were finalized before placing the components and making the connections:

- A 4-layer structure was chosen, consisting of signal, power, ground, and signal layers. The main reasons for choosing a 4-layer PCB structure are that 4-layer PCBs provide better signal integrity during transmission and reduce electromagnetic interference (EMI). This structure allows for the effective routing of high-speed signals, enabling the design of more complex and faster-operating circuits. Additionally, the extra layers improve heat distribution, which leads to more efficient operation and a longer lifespan of the components.

- The CANBUS traces were routed to be as close to each other as possible. The characteristic impedance of the CAN traces was maintained at 120 ohms. To prevent signal delay or degradation, vias were avoided in the CAN lines.

- The decoupling capacitors on the power lines and the analog input capacitors have been placed close to the power pins of the respective components and the analog input pins.

- The TVS diode in the voltage protection circuit has been placed near the battery voltage connection on the connector.

- The via ports have a radius of 0.8 mm, and the holes are set to 0.4 mm, with the necessary adjustments made in the KiCad board layout menu.

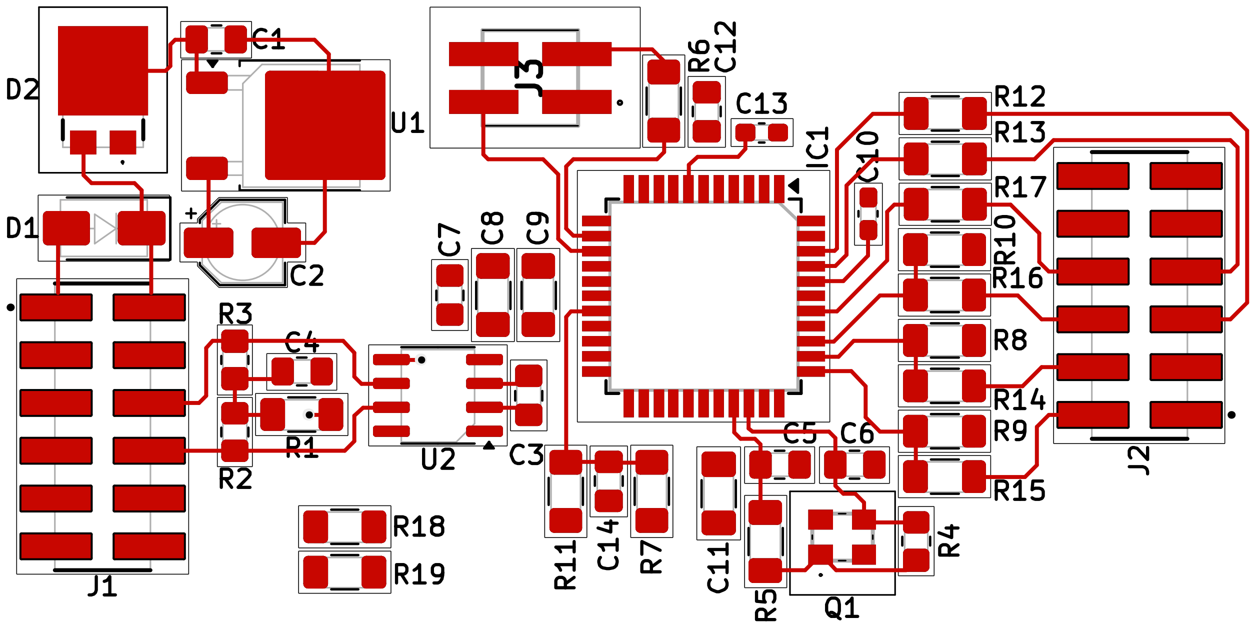

Figure 9: PCB layout overview.

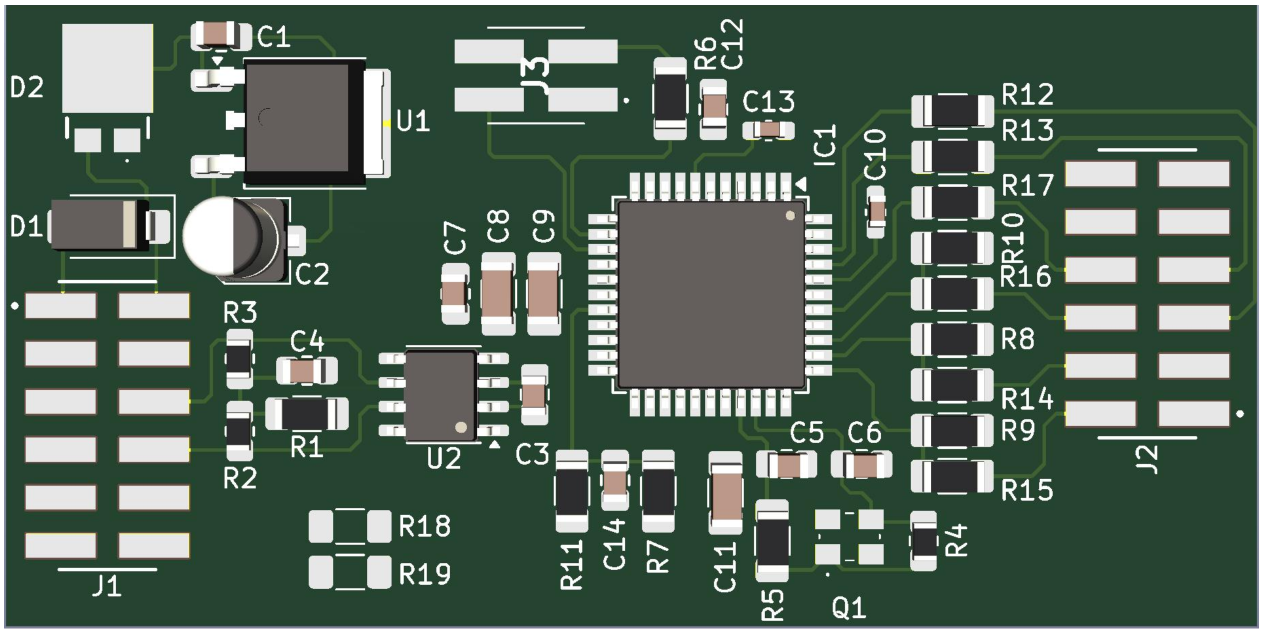

Figure 10: 3D view of the PCB.

The PCB layout in KiCad has progressed to the component placement stage, with all components positioned appropriately to optimize space and ensure adherence to design requirements. While the connections between components have not yet been established, the 3D view of the layout, as shown in Figure 10, provides a clear visualization of the board's structure. In this view, the relative positioning of components, such as the microcontroller, connectors, and transceivers, can be observed, highlighting the spatial arrangement and alignment on the PCB. This representation is particularly useful for evaluating the physical layout and ensuring that component placement adheres to design constraints, such as clearance and accessibility, before proceeding with routing the connections.

2.3.4 Software Development

The software development process involves creating MATLAB Simulink models to simulate the behavior of the ECU under different scenarios. C code is generated by running these models, which is modified as necessary using MPLAB X IDE and embedded into the microprocessor. Although the Simulink model can be directly embedded into the processor, or the C code can be written solely using MPLAB X IDE and then embedded, these two practical approaches simplify the steps and provide the opportunity for better control and verification.

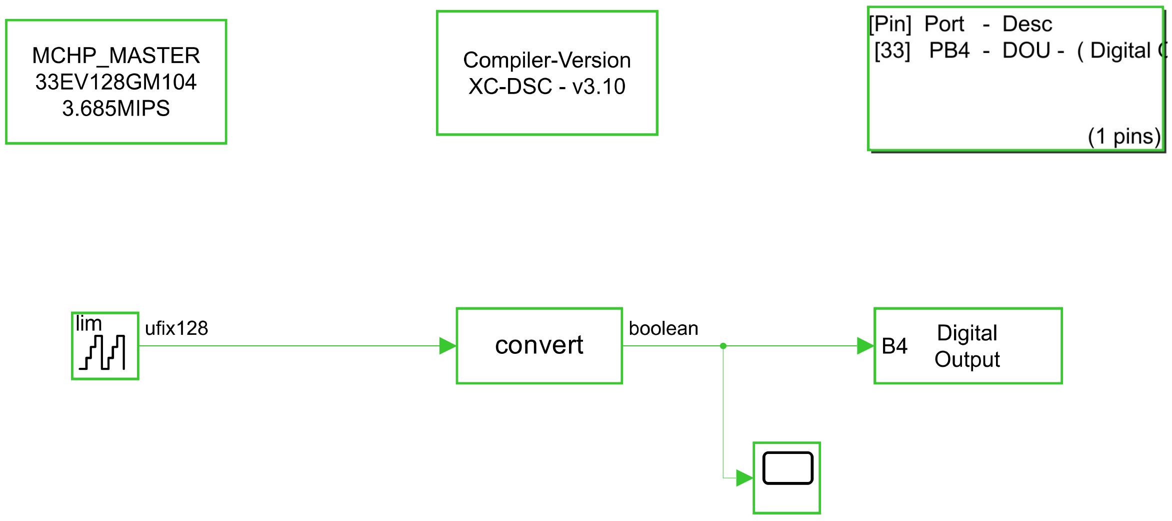

Figure 11: Digital output Simulink model.

The simple Simulink model as shown in Figure 11 allows switching the RB4 digital output pin (pin 33) of the microcontroller between LOW and HIGH states. As seen in the figure, the model was created using MPLAB Simulink blocks, and the microcontroller and compiler information were input. The output voltage graphs of the pin are displayed in the figure below. This model's purpose is to verify the proper functioning of both the hardware and software. After running the model, the generated C code was embedded into the microcontroller through MPLAB X IDE, and the functionality was confirmed by measuring pin 33 with a multimeter.

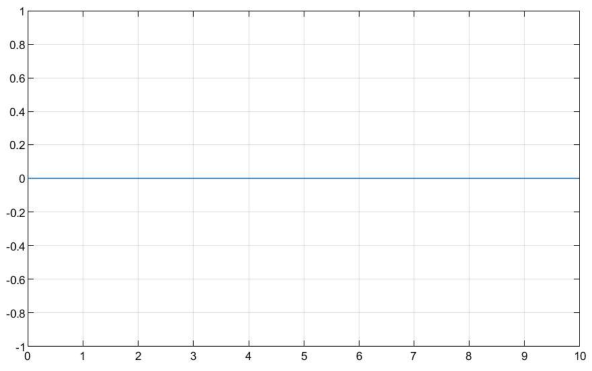

Figure 12: 0V digital output graph.

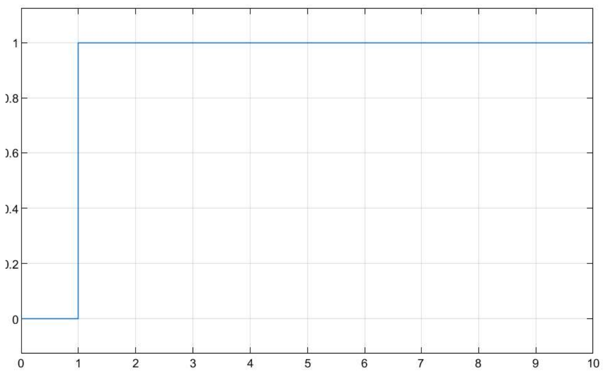

Figure 13: 5V digital output graph.

3 RESULTS AND DISCUSSIONS

The project "Test Software and Bootloader Development for an ECU" has achieved important progress in hardware design and software development, laying the foundation for further integration and testing. Although the system is still in development, initial results highlight its potential for enabling reliable ECU testing and firmware updates.

3.1 Results

The components were selected based on their purposes and environmental conditions, and schematic diagrams were designed based on application circuits in their datasheets, with the PCB layout following these connections. A C code was generated using Simulink and MPLAB X IDE to toggle the microcontroller's digital output pin between LOW and HIGH, enabling the voltage change to be observed with a multimeter. As the PCB was not ready for testing, a test circuit has been assembled with the microcontroller and necessary components. This test circuit has been used to be able to start with the software development in advance. By embedding the code into the microcontroller, the voltage change on the digital output pin was monitored using a multimeter.

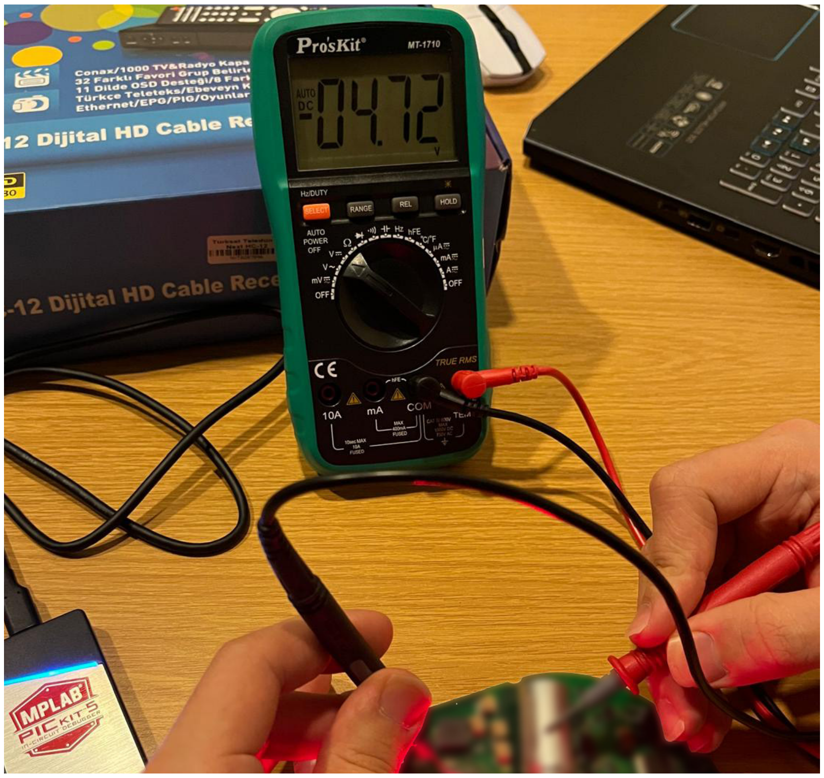

Figure 14: Measured digital output at Pin 33 (RB4) with a multimeter when the software applies Logic High to Pin 33 (RB4).

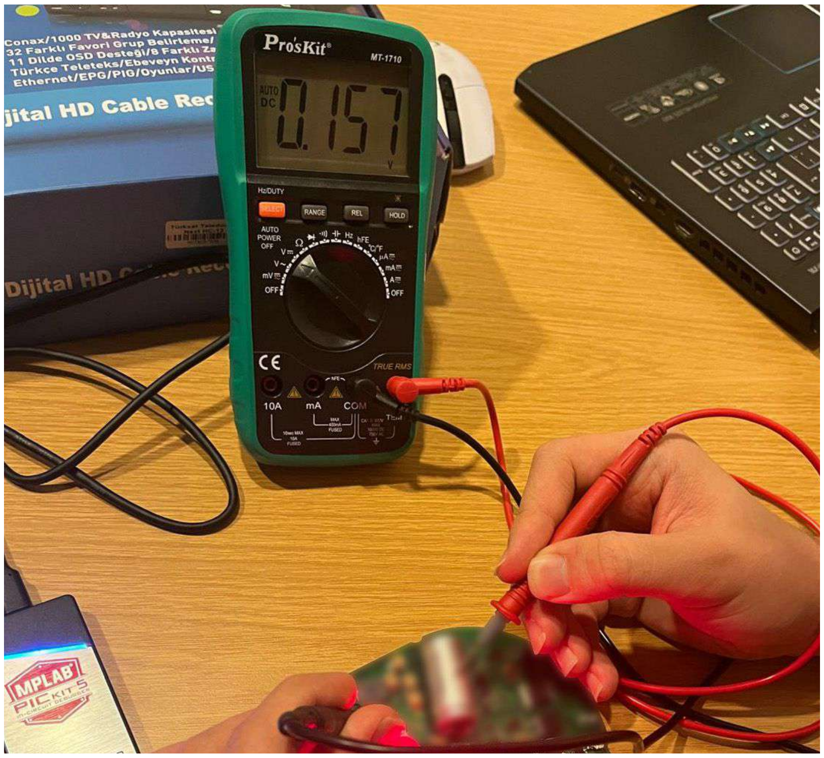

Figure 15: Measured digital output at Pin 33 (RB4) when the software applies Logic Low to Pin 33 (RB4).

As seen in the figure, when the digital output pin is in the HIGH state, an approximate voltage of 5 V supplied by the microcontroller in the test circuit, powered at 5V, is measured on this pin. In the LOW state, a value close to 0 V is observed. This proves that the C code generated using MATLAB Simulink and MPLAB X IDE is functioning correctly.

3.2 Discussion

No tests have yet been performed on the PCB that is under development in the hardware design phase of the project. In future stages, tests will be conducted on this board using the microcontroller's digital output, analog input, and PWM output pins. At this point, the results have only validated the correctness of the software written using two parallel applications. In addition to the tests conducted with a multimeter, the schematic design created using the KiCad application has successfully passed the ERC test. Similarly, the PCB layout designed using KiCad successfully passed the DRC test. It can be stated that all parts of the hardware design, except for the production phase, have been completed.

4 CONCLUSIONS AND FUTURE WORKS

4.1 Conclusion

This project has successfully demonstrated the foundational development of a scalable and modular system comprising test software and a bootloader for automotive ECUs. By adopting a systematic approach to hardware and software design, critical milestones such as the selection of components compliant with industry standards, schematic design, PCB layout, and the validation of initial functionalities through simulations and practical tests have been successfully achieved.

4.2 Future Works

The steps required for the full completion of the project are as follows:

- **PCB Testing:** Conduct thorough testing of the manufactured PCB to ensure its functionality and compliance with the design specifications.
- **Serial Communication Development:** Finalize and debug the serial communication system to establish a reliable connection between the test software and the ECU.
- **Integration of CAN Protocol:** Implement and test the CAN communication protocol for seamless and high-speed interaction in accordance with automotive standards.
- **Software Enhancements:** Develop software to facilitate testing on additional hardware ports, such as analog inputs and PWM outputs, expanding the scope beyond digital outputs.
- **System Validation:** Perform system-level testing to simulate real-world automotive scenarios, ensuring the overall functionality and robustness of the ECU testing and bootloader system.
- **Bootloader Implementation:** Enhance the bootloader to support secure incremental updates and incorporate robust error-handling mechanisms for reliable firmware updates.

REFERENCES

[1] Z. Ji, Z. Xiangyu, and P. Yong, "Implementation and research of bootloader for automobile ecu remote incremental update," 01 2015.

[2] M. Conrad, S. Sadeghipour, and H.-W. Wiesbrock, "Automatic evaluation of ecu software tests," *SAE Transactions*, vol. 114, pp. 583–592, 2005. [Online]. Available: http://www.jstor.org/stable/44682468

[3] Y. Kang, J. Chen, and B. Li, "Generic bootloader architecture based on automatic update mechanism," in *2018 IEEE 3rd International Conference on Signal and Image Processing (ICSIP)*. IEEE, 2018, pp. 586–590.

[4] E. G. Leaphart, S. E. Muldoon, and J. N. Irlbeck, "Application of robust engineering methods to improve ecu software testing," *SAE Transactions*, pp. 1007–1018, 2006.

[5] D. Bogdan, R. Bogdan, and M. Popa, "Design and implementation of a bootloader in the context of intelligent vehicle systems," in *2017 IEEE Conference on Technologies for Sustainability (SusTech)*. IEEE, 2017, pp. 1–5.

[6] J. Van den Herrewegen, "Automotive firmware extraction and analysis techniques," Ph.D. dissertation, University of Birmingham, 2021.

[7] J. Zhang, Y. Lv, and Z. Liao, "Research on automotive ecu remote update and it's security," in *Journal of Physics: Conference Series*, vol. 1074, no. 1. IOP Publishing, 2018, p. 012133.

[8] STMicroelectronics, *LFXX: Very low drop voltage regulator with inhibit function*, 2017, https://www.st.com/resource/en/datasheet/lfxx.pdf.

[9] Microchip Technology Inc., *MCP2561 High-Speed CAN Transceiver*, 2013, https://ww1.microchip.com/downloads/aemDocuments/documents/OTH/ProductDocuments/DataSheets/20005167C.pdf.

[10] Microchip Inc., *dsPIC33EVXXXGM00X/10X Family Data Sheet*, 2024, available: https://ww1.microchip.com/downloads/aemDocuments/documents/OTH/ProductDocuments/DataSheets/dsPIC33EVXXXGM00X-10X-Family-Data-Sheet-DS70005144H.pdf.

[11] Littlefuse, *SMAJ Series*, 2022, https://www.littelfuse.com/assetdocs/tvs-diodes-smaj-datasheet?assetguid=13c2a823-03b8-4d1f-9ddc-9b44670aed9d.

[12] Vishay, *SS8PH9, SS8PH10: High Current Density Surface-Mount High Voltage Schottky Rectifier*, 2020, https://www.vishay.com/docs/88989/ss8ph10.pdf.

[13] S. Inc., *TSM-1XX-XX-XXX-XX-X-XXX-X*, 2024, https://suddendocs.samtec.com/prints/tsm-1xx-xx-xxx-xx-x-xxx-x-mkt.pdf.

APPENDICES

Digital Output Test Code main.c

```
/*
 * Academic License - for use in teaching, academic research, and meeting
 * course requirements at degree granting institutions only.  Not for
 * government, commercial, or other organizational use.
 *
 * ----------------------------------------------------------------------
 * MPLAB Device Blocks for Simulink v3.58 (26-Nov-2024)
 *
 *   Product Page:  https://www.mathworks.com/matlabcentral/fileexchange/71892
 *           Forum: https://forum.microchip.com/s/sub-forums?&subForumId=a553l
 * 000000J2rNAAS&forumId=a553l000000J2pvAAC&subForumName=MATLAB
 *           Wiki:  http://microchip.wikidot.com/simulink:start
 * ----------------------------------------------------------------------
 * File: digital_output_2_0_main.c
 *
 * Code generated for Simulink model 'digital_output_2_0'.
 *
 * Model version                  : 1.4
 * Simulink Coder version         : 24.2 (R2024b) 21-Jun-2024
 * C/C++ source code generated on : Fri Jan  3 12:54:11 2025
 */

/* Set Fuses Options */
```

```
#pragma config FNOSC = FRC
#pragma config OSCIOFNC = ON, PLLKEN = ON
#pragma config FWDTEN = OFF
#pragma config PWMLOCK = OFF
#pragma config DMTEN = DISABLE

#define MCHP_isMainFile
#include "digital_output_2_0.h"
#include "digital_output_2_0_private.h"

/* Microchip Global Variables */
/* Solver mode : MultiTasking */
int main()
{
  /* Initialize model */

  /* Configure Pins as Analog or Digital */
  /* Configure Remappables Pins */

  /* Configure Digitals I/O directions */
  TRISB = 0xFFEF;                      /* Port input (1) / output (0) */

  /* Initialize model */
  digital_output_2_0_initialize();

  /* Configure Timers */
  /* --- TIMER 1 --- This timer is enabled at end of configuration functions. */
  T1CON = 0;                           /* Stop Timer 1 and resets control register */
  _T1IP = 2;                           /* Set timer Interrupt Priority */
  _T1IF = 0;                           /* Reset pending Interrupt */
  _T1IE = 1;                                /* Enable Timer Interrupt. */
  PR1 = 0x0E64;                             /* Period */

  /* Enable Time-step */
  TMR1 = 0x0E63;                            /* Initialize Timer Value */
  T1CONbits.TON = 1;                        /* Start timer 1. Timer 1 is the source
  *                                         trigger for the model Time-step */

  /* Main Loop */
  for (;;) ;

  /* Terminate model */
  digital_output_2_0_terminate();
}                                           /* end of main() */

/*
 * File trailer for generated code.
 *
 * [EOF]
 */
```

Digital Output Test Code 2

```
/*
 * Academic License - for use in teaching, academic research, and meeting
 * course requirements at degree granting institutions only.  Not for
 * government, commercial, or other organizational use.
 *
 * ------------------------------------------------------------------
 * MPLAB Device Blocks for Simulink v3.58 (26-Nov-2024)
 *
 *    Product Page: https://www.mathworks.com/matlabcentral/fileexchange/71892
 *    Forum: https://forum.microchip.com/s/sub-forums&subForumId=a553l000000J2
 * rNAAS&forumId=a553l000000J2pvAAC&subForumName=MATLAB
 *    Wiki:  http://microchip.wikidot.com/simulink:start
 * -------------------------------------------------------------------
 * File: digital_output_2_0.c
 *
 * Code generated for Simulink model 'digital_output_2_0'.
 *
 * Model version                  : 1.4
 * Simulink Coder version         : 24.2 (R2024b) 21-Jun-2024
 * C/C++ source code generated on : Fri Jan  3 12:54:11 2025
 */

#include "digital_output_2_0.h"
#include "rtwtypes.h"

/* Block signals and states (default storage) */
DW_digital_output_2_0_T digital_output_2_0_DW;

/* Real-time model */
static RT_MODEL_digital_output_2_0_T digital_output_2_0_M_;
RT_MODEL_digital_output_2_0_T *const digital_output_2_0_M = &digital_output_2_0_M_;

/* Model step function for TID0 */
void digital_output_2_0_step0(void)    /* Sample time: [0.001s, 0.0s] */
{
  /* (no output/update code required) */
}

/* Model step function for TID1 */
```

```
void digital_output_2_0_step1(void)    /* Sample time: [1.0s, 0.0s] */
{
  uint8_T rtb_Output;
  boolean_T rtb_DataTypeConversion;

  /* UnitDelay: '<S1>/Output' */
  rtb_Output = digital_output_2_0_DW.Output_DSTATE;

  /* Switch: '<S4>/FixPt Switch' incorporates:
   *  Constant: '<S4>/Constant'
   *  UnitDelay: '<S1>/Output'
   */
  digital_output_2_0_DW.Output_DSTATE = 0U;

  /* DataTypeConversion: '<Root>/Data Type Conversion' */
  rtb_DataTypeConversion = (rtb_Output != 0);

  /* S-Function (MCHP_Digital_Output_Write): '<S2>/Digital Output Write' */
  LATBbits.LATB4 = rtb_DataTypeConversion;
}

/* Model initialize function */
void digital_output_2_0_initialize(void)
{
  /* Registration code */

  /* Set task counter limit used by the static main program */
  (digital_output_2_0_M)->Timing.TaskCounters.cLimit[0] = 1;
  (digital_output_2_0_M)->Timing.TaskCounters.cLimit[1] = 1000;

  /* Start for S-Function (MCHP_Master): '<Root>/Microchip Master' */
  /* S-Function "Microchip MASTER" initialization Block: <Root>/Microchip Master */
}

/* Model terminate function */
void digital_output_2_0_terminate(void)
{
  /* (no terminate code required) */
}

/* Scheduler */
void __attribute__((__interrupt__,__auto_psv__)) _T1Interrupt(void)
{
  {
    struct {
      unsigned int Flags1 : 1;
    } static volatile Overrun __attribute__ ((near)) ;

    struct {
      unsigned int Flags1 : 1;
    } static volatile event __attribute__ ((near)) ;

    struct {
      uint_T Task1;                 /* 1.0s periodic task. Max value is 1000 */
    } static taskCounter __attribute__ ((near)) = {
      .Task1 = 1                    /* Offset is 0 (1000 + 1 - 0 including pre
      decrement */
    };

    _T1IF = 0;                      /* Re-enable interrupt */
```

```
/* Check subrate overrun, set rates that need to run this time step*/
taskCounter.Task1--;                 /* Decrement task internal counter */
if (taskCounter.Task1 == 0) {        /* task dropped on overload */
  taskCounter.Task1 = (uint16_T) 1000;/* 1.0s periodic task. Max value is 1000 */
  event.Flags1 = 1U;                 /* Flag tag to be executed */
}

/* ---------- Handle model base rate Task 0 ---------- */
digital_output_2_0_step0();

/* Get model outputs here */
if (_T1IF ) {
  return;                            /* Will re-enter into the interrupt */
}

/* Re-Enable Interrupt. IPL value is 2 at this point */
_IPL0 = 1;                           /* Enable Scheduler re-entrant
*                                     interrupt. Lower IPL from 2 to 1 */
_IPL1 = 0;

/* Step the model for any subrate */
/* ---------- Handle Task 1 ---------- */
if (Overrun.Flags1) {
  /* Priority to higher rate steps interrupted */
  return;
}

while (event.Flags1) {               /* Execute task tid 1 */
  Overrun.Flags1 = 1U;
  event.Flags1 = 0U;
  digital_output_2_0_step1();
```

```
      /* Get model outputs here */
    }

    Overrun.Flags1 = 0U;

    /* Disable Interrupt. IPL value is 1 at this point */
    _IPL1 = 1;                                /* Disable Scheduler Interrupts.
                                                 Rise IPL from 1 to 2 */
    _IPL0 = 0;
  }
}

/*
 * File trailer for generated code.
 *
 * [EOF]
 */
```