Instituto Tecnológico
y de Estudios Superiores de Occidente

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación del 29 de noviembre de 1976.

Departamento de Electrónica, Sistemas e Informática

Especialidad en Sistemas Embebidos

Safe and secure UDS-based flash programming via CAN bus

TRABAJO RECEPCIONAL que para obtener el **GRADO** de
ESPECIALISTA EN SISTEMAS EMBEBIDOS

Presenta: **JOSUE DAVID MAYA PADILLA**

Asesor **HÉCTOR ANTONIO RIVAS SILVA**

Tlaquepaque, Jalisco. enero de 2022.

Safe and secure UDS-based flash programming via CAN bus

Maya Padilla, Josue D.
Especialidad en Sistemas Embebidos
Instituto Tecnologico y de Estudios Superiores de Occidente
Tlaquepaque, Jalisco
josue.maya@iteso.mx

***Abstract*— The bootloader is a firmware that helps update the application program of a microcontroller unit. In the automotive industry, a safe and secure bootloader must be implemented considering the standards that ensure the quality control of the system. In this paper, the development of a UDS bootloader via CAN bus was described according to the automotive standard ISO 26262 and automotive SPICE by using an adaptation of the V-model development cycle and considering the following sections: requirements, architecture, design and implementation, testing, and integration. For the system validation, software and system tests were executed in a controlled environment. The next step involves the execution of tests using an automotive environment.**

***Keywords*—** *CAN, UDS, safe, bootloader, flashing*

I. INTRODUCTION

In the automotive industry, a bootloader is constantly used for diagnostics and reprogramming of any automotive engine control unit (ECU) [1]. A bootloader is a firmware, independent of the user application that starts running when the system is turned on [2], [3]. This firmware allows communication with the ECU and its components, such as sensors or actuators. A bus control area network (CAN) bootloader must be designed based on the memory architecture of each microcontroller to identify the sections where the new firmware will be uploaded. Communication between the host (who will provide the new firmware) and the ECU, can define issues such as security and instruction management [4], [5]. Moreover, the architecture of the bootloader firmware must be defined to communicate with multiple parts of the microcontroller and the ECU system. The system is defined in steps, which include its architecture, design, and implementation [1], [6]; i.e., the system is described from the project concept to explain the product's functionality.

For instance, a car may have 40 computers on average, but some include up to 100, including different types of sensors [7]; The body control module (BCM) is one example. BCM is responsible for diagnosing and managing the vehicle's body sensors and actuators, such as turn signals, optical horn, and high beams. The BCM must meet quality standards to maintain a safe system according to the Automotive Security Integrity Level (ASIL).

ASIL is a deterministic factor in automotive firmware; therefore, it must comply with a specific level of quality control depending on the component of the automotive system. A firmware is considered safe and secure if it complies with the ISO 26262 standard; this is an interpretation of the generic IEC 61508 functional safety standard for product development. According to ISO 26262, the firmware development should be segmented, complying in each phase with guidelines that can mitigate software issues and bugs [8]. The steps needed for software development are divided into 5 categories: requirements, architecture, design and implementation, testing, and integration [9].

The development of quality-assured bootloader firmware that meets automotive standards is a necessary system in the industry, however, firmware with free or complete documentation is not available. For this reason**,** this paper describes the development of a bus CAN bootloader based on a unified diagnostics services (UDS) protocol following automotive standards to provide a safe and secure system.

The paper is organized as follows: Section II describes the methodology. Section III shows prototype components. Section IV presents the requirements. Section V depicts the system and firmware architecture. Section VI details the development and implementation of firmware. Lastly, Section VII demonstrates that the firmware is safe and secure.

II. SYSTEMS DEVELOPMENT LIFE CYCLE

The methodology used was the V development cycle (V-Model); this model establishes the stages in the development of a product, starting with the concept idea of the system and ending with the way the system will work and its real capabilities [10]. The parts of this development model are described in "Fig. 1".

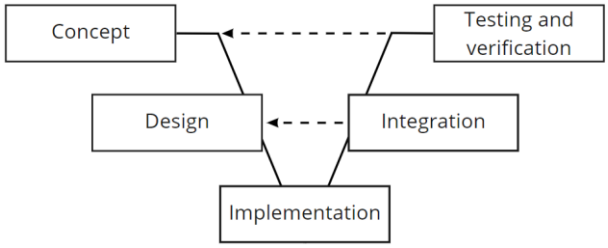

Fig. 1. Generic development cycle of the V-Model.

The ISO 26262 and Automotive SPICE standards were used for the prototype development. Both standards provide development models based on the V-Model cycle. [8].

A. *ISO 26262*

This is a standard for functional safety in vehicles that establishes the restrictions and guidelines for the development of an automotive system [9]; it contemplates the hardware and software development stages of the product. “Fig. 2” presents the sections considered in the standard.

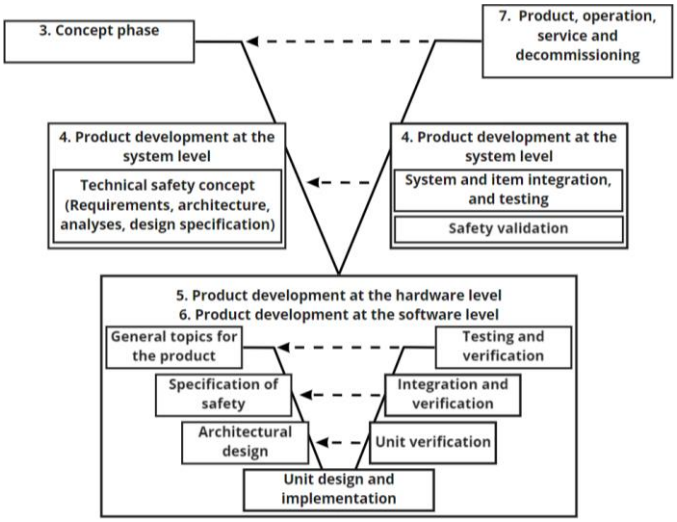

Fig. 2. V-Model cycle according to stages of the ISO:26262 [9].

B. *Automotive SPICE*

This standard is used to perform process evaluation of the development of embedded automotive systems. It contains indicators for the interpretation of the standard. The standard is separated into 3 processes: primary life cycle, organizational life cycle, and supporting life cycle. Each process is divided by process categories and contains activities according to each category. The primary life cycle process and the following categories: system and software engineering [11] are shown in "Fig. 3".

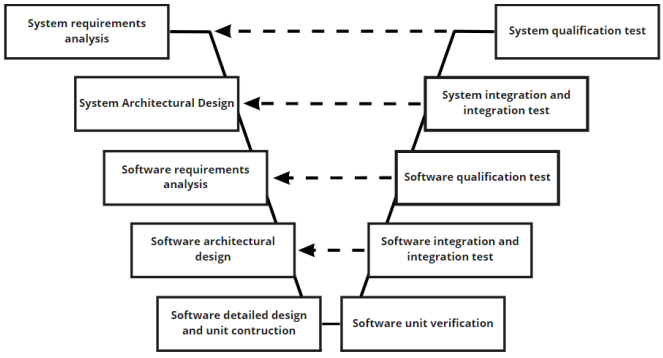

Fig. 3. V-Model cycle according to stages of the Automotive SPICE 3.1[11].

According to both standards, the life cycle implemented in the development of this project was defined in 5 stages, as shown in "Fig. 4". This model represents the stages considering the complete system, i.e., hardware and software were integrated in each step, therefore, the system and software stages will be represented together, however, they will be designed individually.

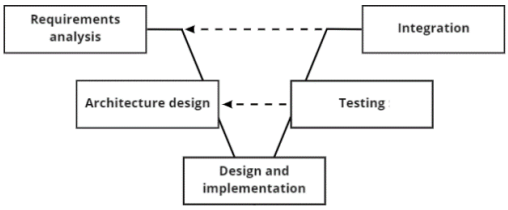

Fig. 4. The adopted and implemented V-model cycle.

III. PROTYPE DESCRIPTION

The bootloader shall be capable of reprogramming the program memory (i.e., flash memory) of a microcontroller unit (MCU) by means of a specified type of data transmission (e.g., CAN, Ethernet, UART) [12]; therefore, the bootloader shall be a separate firmware from the application program and shall be stored in a flash memory region reserved specifically for this firmware.

Three types of embedded system booting are available: flash boot, ROM boot, and SRAM boot. The first is the common boot mode, where the program or application is in the flash memory. The second type, ROM boot is used when configurations are needed in the initialization of the application [2].

In some embedded systems, the memory space used to store the bootloader firmware is not defined, so the flash memory where the application program is stored, must be segmented by the firmware and it needs be specifically defined where the bootloader and the application will start the memory region. "Fig. 5" shows how the same memory region can be divided for a bootloader; in other MCUs, the vendor defined exclusive regions for bootloader storage (the memory regions are exposed in the data sheet of each microcontroller). This type of MCU can be seen in "Fig. 6".

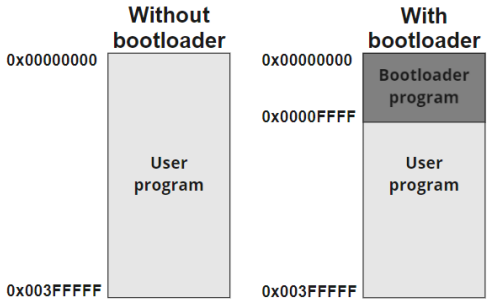

Fig. 5. Segmented flash memory. A Texas Instruments MCU of the Hercules family was taken as reference [13].

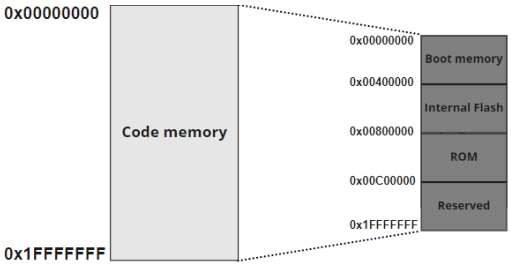

Fig. 6. Segmented flash memory. A Microchip MCU of the SAM family was taken as a reference [14].

The prototype bootloader can update the ECU firmware of a BCM through a standard connection in automotive systems. The bootloader can change the application that controls the lighting sequence of a vehicle.

For the prototype, the hardware components and tools described in Table I were used; these components are generic, and necessary for any system that requires a CAN bus bootloader. Therefore, these may vary depending on the system where it is implemented. The selection of the specific component will be explained in section V.

TABLE I. COMPONENT'S PROTOTYPE.

Component
Button
LEDs or lights
EEPROM or any flash memory
Transceiver
MCU
Bus CAN analyzer
CAN adapter RS232 to OBDII

IV. REQUIREMENTS

The first stage is requirements analysis; this is the process in which requirements are established according to stakeholder needs, physical possibilities, and mapping with the product concept [15]. This process shall be concluded before the initialization of the architecture design process.

In this process, the requirements of the system and software were defined. These requirements are written in accordance with the International Council on Systems Engineering (INCOSE) [15]. Although both types of requirements are described, these were not built in one instance; they were developed as shown in "Fig. 3".

The requirements shall follow the next characteristics according to INCOSE [16]:

1) *necessary*
2) *implementation independent*
3) *unambiguous*
4) *complete*
5) *singular*
6) *feasible*
7) *verifiable*
8) *correct*
9) *confirming*

Table II is a comparison between a requirement that meets the characteristics of the guidelines for writing requirements and one that was written incorrectly. Both requirements were used to define the bootloader.

TABLE II. COMPARISON OF GOOD VS BAD REQUIREMENTS.

Requirement	Characteristic by INCOSE								
	1	2	3	4	5	6	7	8	9
The system shall update the flash memory of the ECU in an automobile's BCM	✓	✓	✓	✓	✓	✓	✓	✓	✓
The system will update any part of the car	×	×	×	×	×	×	×	×	×

B. System Requirements

The system requirements analysis consists of structuring the system according to the needs of stakeholders, and these shall lead the design system. The requirements specific expected behaviors, the limitations of the system, and the components.

The requirements can express hardware or software objectives. Table III shows some requirements that were proposed for the system development; these were analyzed with INCOSE guidelines, and their specification is in accordance with ISO 26262 and Automotive SPICE. Table III indicates if the requirement will be verifiable by hardware, software, or both.

TABLE III. SYSTEM REQUIREMENTS (SYSREQ).

ID	Requirements	Verif. HW	Verif. SW
SyR01	The bootloader shall be implemented on a microcontroller with ARM architecture.	✓	
SyR02	The microcontroller shall have a Cortex-M series architecture for embedded purposes.	✓	
SyR03	The system shall have a transceiver to CAN communication with baud rate up to 250 kbps	✓	✓
SyR04	The transceiver shall be the model ATA6561	✓	
SyR05	The system shall have a bootloader to upgrade the system flash memory		✓
SyR06	Bootloader shall read the flash memory		✓

C. Software Requirements

Software requirements are guidelines for the software development of a system. These requirements are the translation of the system requirements that are related to the software; if the system requirements are related to both hardware and software, only those requirements needed to fulfill the software shall be defined.

Software requirements shall demonstrate their relationship with system requirements; each software requirement shall be traced to at least one system requirement. Table IV presents examples of software requirements that can be traced to the software related requirements in Table III. The last column of Table IV indicates the system requirement to which the software requirement is related.

TABLE IV. SOFTWARE REQUIREMENTS (SWREQ).

ID	Requirements	Traceability with SysReq
SwR01	The "CANDRV" component must export an interface called "Init" to establish the initial configuration of the CAN communication.	SyR03
SwR02	The system shall have a component to write, read and erase data from flash memory.	SyR05
SwR03	The "FLASHABS" component shall export an interface called "WriteApp" to store the "BuffMem" buffer at the "DirMem" address of the flash memory.	SyR05
SwR04	The "FLASHABS" component must export an interface called "GetInfo" to obtain the information stored in the address "Dir" of length "LenDir".	SyR06

V. Architecture

The purpose of the architecture stage is to design and identify which requirements correspond to system and software elements. The steps in this process are:

1) *Define elements of the architecture*
2) *Asign position to elements*
3) *Behavior analysis of the elements*

B. System Architecture

The architecture system design was defined based on the system requirements. The system elements were identified with the system requirements following the steps proposed; after that, the elements were placed in the component diagram (Fig. 7), where the relationship between the elements, as well as their position in the system is shown.

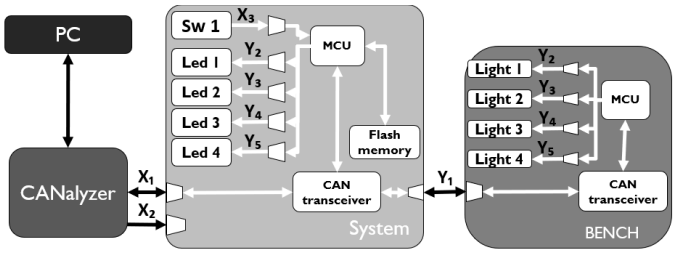

Fig. 7. System architecture.

Then, P-Diagram was used to analyze the behavior of the system elements. With this diagram, the architecture developer can establish the ideal behaviors of the system:

1) *Levels of requirements (e.g., performance)*
2) *Inputs*
3) *Ideal outputs and error outputs*

"Fig. 8" is an example P-diagram of the LED element used in the system. The diagram shows the desired characteristics of the element according to the system requirements.

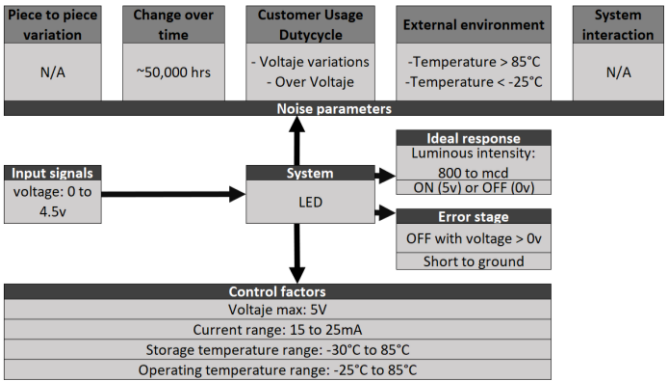

Fig. 8. LED Diagram.

C. Software Architecture

The software architecture design was based on the AUTOSAR architecture model [17], which is used in the automotive industry as a software development standard. According to AUTOSAR structure, the proposed architecture established limitations between the layers, which are shown in "Fig. 9" [17].

USES	System services/OS	Memory Services	Crypto Services	Communication services	I/O Hw Abs	Memory Hw abs	Communication Hw abs	Microcontroller Drivers	Memory Drivers	Communication Drivers	I/O Drivers
Application	✓	✓	✓	✓	✓	×	×	×	×	×	×
System services/OS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Memory Services	✓	✓	✓	×	×	✓	×	×	×	×	×
Crypto Services	✓	✓	✓	×	×	×	×	×	×	×	×
Communication services	✓	✓	✓	✓	✓	×	✓	×	×	×	×
I/O Hw Abs	✓	×	✓	×	✓	×	✓	✓	×	✓	✓
Memory Hw abs	✓	✓	×	×	×	✓	✓	×	✓	✓	×
Communication Hw abs	✓	×	×	✓	×	×	✓	×	×	✓	✓
Microcontroller Drivers	✓	×	×	×	◊	×	×	◊	×	×	◊
Memory Drivers	✓	×	×	×	×	◊	×	×	×	×	×
Communication Drivers	✓	×	×	×	×	×	◊	×	×	×	✓
I/O Drivers	✓	×	×	×	◊	×	×	◊	×	×	◊

Fig. 9. Layer limitation matrix.

The software architecture developed is shown in "Fig. 10" according to the AUTOSAR layers [17].

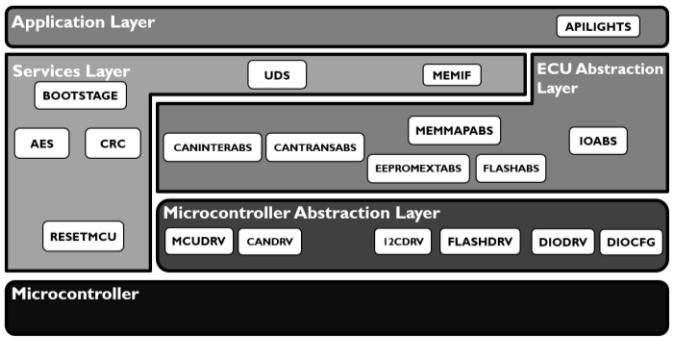

Fig. 10. Software architecture.

VI. Design and Implementation

According to the life cycle in "Fig. 4", the next stage is design and implementation. In this step, two products must be created: Software design specification and software implementation.

A. CAN

This is a multi-master communication protocol, e.g., a device can send or receive messages from all devices in the network [18]; if more than one device sends a message at the same time, the highest priority message is determined by the message ID; if the message ID is closer to 0x0h, the message is of lower priority.

A CAN message sends no more than eight bytes of data. When the device needs to send messages with more than eight bytes of data, the driver must be configured to send and receive multi-frame messages. Table V describes the first three bytes according to the type of message where the type is indicated in first byte of the message. The SN field keeps a rolling counter that starts at 0x21h and ends at 0x2F; if is necessary, rollover to 0x20h. For the message type "FlowControl", the FS field indicates state of receiver drive and this field have tree states:

continue to send, wait, and overflow. The block size of frames (Fig. 11) is saved in the BS field. Finally, STmin specifies the minimum time gap allowed between two messages.

TABLE V. FRAME FOR MULTI-FRAME CAN MESSAGES.

Message type	Bytes			
	1		2	3
	Bits 7-4	Bits 3-0		
SingleFrame	0	DataLength	-	-
FirstFrame	1	DataLength		-
ConsecutiveFrame	2	SN	-	-
FlowControl	3	FS	BS	STmin

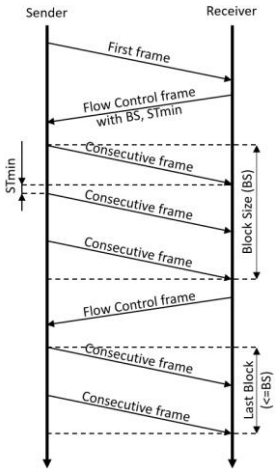

Fig. 11. Can multiple frame transmission.

B. *Unified Diagnostic Services (UDS)*

UDS is a protocol used in the automotive industry by diagnostic systems [19]; the protocol is used to communicate with the ECUs and control specific functions. This protocol requires a client to request services and a server who responds the client's request.

Table VI shows the UDS protocol services used for this system, The id services and the description of each service are displayed in Table VI.

TABLE VI. SERVICES USED FROM UDS PROTOCOL.

***Function*/ (id)Sub-function**	**Id (hex)**	**Description**
DiagnosticSessionControl (0x01) programmingSession	0x10	Enable diagnostic sessions to activate a group of services/functionalities in ECU.
ECUReset (0x01) hardReset (0x02) softReset	0x11	Service to request a system reset by client.
ReadMemoryByAddress	0x23	Request memory data from the system by client. Requests need a specific address and size of memory.
SecurityAccess (0x01) requestSeed (0x02) sendKey	0x27	Permit or restrict access data and/or diagnostic services for security.
RoutineControl (0x01) startRoutine (0x02) stopRoutine	0x31	Execute a sequence of steps.
RequestDownload	0x34	Start data transfer to system.
TransferData	0x36	Transfer data from client to system.
RequestTransferExit	0x37	Indicate finish transfer data.

C. *Secuency*

"Fig. 12" explains, with a sequency diagram, the bootloader access process when the system is powered or rebooted. In the first step, the boot loader checks whether the flag indicating startup boot is set; this flag must be set when the application program is executed.

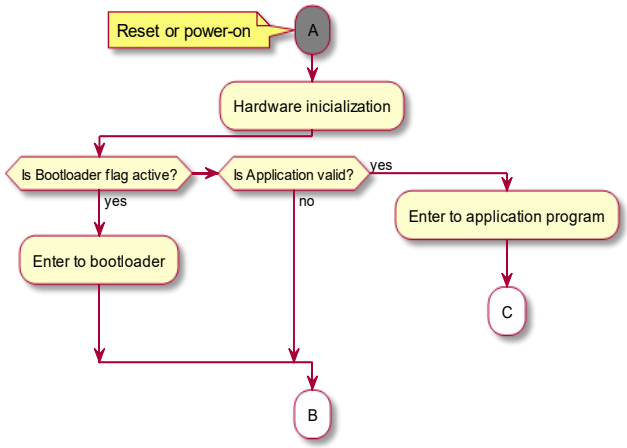

Fig. 12. Boot of the system.

The client and server were defined according to UDS protocol. The CAN diagnostic (PC/CANalyzer) is used as the client and the ECU as the server, thus, When the server is running the application program, the client can send a message to interrupt the application and set the flag that tells the ECU to restart (Fig. 13) and run the bootloader (Fig. 12).

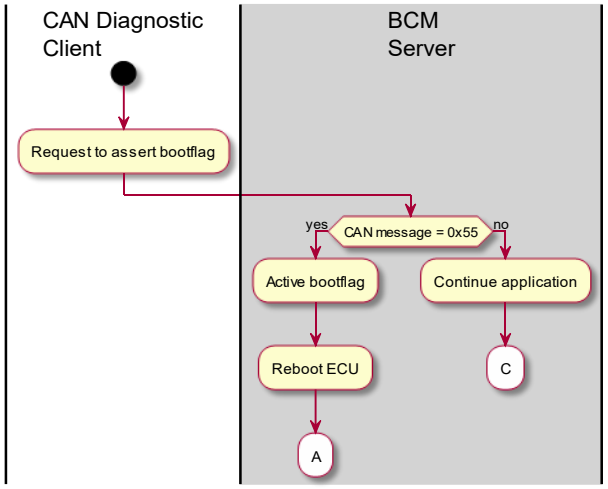

Fig. 13. Activate boot-flag and start bootloader process.

Table VII includes the specific instructions to flash the ECU, these instructions are executed in the bootloader program only. If the instructions are executed in the application program, the ECU shall response with a negative response according with codes by UDS. The software needs a mechanism to security protection, this mechanism must prevent unauthorized writing and reading of the flash memory. The following steps are the instructions for flashing ECU [20]:

1) The client request to ECU enter in programming session

2) Client request a seed for get key, it shall match the server key.

3) Client sends key generate to server. On server, it must generate same key received by client and check match.

4) Optional: When the client needs to save a copy of the application memory, the client can request the server to read flash memory.

5) The server clears the application's memory region at the client's request

6) The client sends the request to download the new application to the server's application memory.

7) Sends the application program through multiple CAN messages. The messages shall be send as displayed on "Fig. 11."

8) Server receives the last message with exit indication. This instruction ends transmision and saves application in the allocated memory region

9) Check that the CRC application program is valid and start the application.

TABLE VII. SERVICES SEQUENCE TO FLASHING ECU.

No	*Instruction*	Request (hex) Client	Response Server
1	Program mode	10 02	50 02
2	Request seed for unlock ECU	27 01	67 01 XX…
3	Send key generate	27 02 XX…	67 02
4	Read memory program	23 XXXXXX…	63 XXXXXX…
5	Clear memory	31 01 FF 00	71 00
6	Request download new firmware	34 XXXXXX…	74 XX
7	Transfer new firmware	36 XXXXXX…	76 XX XX…
8	Exit transfer data	37	77
9	CRC	31 01 FF 01	71 01 XXXX

VII. TEST PLAN

This is the last stage in life cycle shown in "Fig. 4". In this stage, the software and the system were tested with test cases designed from the requirements.

A. Software Test Plan

According to ISO 26262 and Automotive SPICE, in a software test plan 2 types of tests can be designed and executed: software unit verification, and software integration verification [9], [12]. Software unit verification provides test cases for each software unit compliant with the software design and software requirements. Software integration verification integrates the software units and implements test cases for the verification of the interaction and behavior of the software, according to the software architecture and requirements.

In this software testing plan, only software integration tests were implemented, and scenarios were designed to meet the requirements and the software architecture. Some of the proposed cases are shown in Table VIII with their respective result.

TABLE VIII. SOFTWARE TEST CASE FROM SOFTWARE TEST PLAN.

Test description	Result
Multiple messages must be sent over the CAN bus with a time difference of more than 500 microseconds.	Pass
Multiple messages must be received over the CAN bus with a time difference of more than 500 milliseconds.	Pass
The ECU shall be reset with a CAN message.	Pass
The pins shall be set in the following sequence: PA05, PC09, PD25, and PD21.	Pass

B. System Test Plan

ISO 26262 and Automotive SPICE mention that for a system to function adequately, it must be tested by integrating hardware and software, a stage known as system integration test. These test cases should be adapted to the architecture and system requirements.

Table IX shows the result of some example test cases for system integration. These scenarios were tested in the prototype environment where a SAMv71 development board was used [2].

TABLE IX. EXAMPLE OF SYSTEM TESTS CASES FROM SYSTEM TEST PLAN.

Test description	Result
When the button is pressed, the current pin shall be turned off and the next pin shall be turned on according to the following sequence: PA05, PC09, PD25, and PD21.	Pass
When an erroneous key is sent, the system shall not allow access to the bootloader and shall continue to the program application.	Pass
The system shall be able to reprogram the system 5 times continuously with different firmware.	Pass
The client must read the system program memory, store the binary, and reprogram the bootloader with the saved binary.	Pass

VIII. CONCLUSIONS

This paper describes the development of a safe and secure bootloader via CAN bus. The development of the software conformed to automotive standards, the stages of requirements, architecture, design and implementation, and testing were developed. The testing stage ensured that the prototype complied with the established requirements, thus demonstrating that the software is safe and secure. In continuation, the system could be implemented in an automotive system to validate the functionality of the bootloader firmware with any ECU.

REFERENCES

[1] Y. Kang, J. Chen and B. Li, "Generic Bootloader Architecture Based on Automatic Update Mechanism," 2018 IEEE 3rd International Conference on Signal and Image Processing (ICSIP), 2018, pp. 586-590, doi: 10.1109/SIPROCESS.2018.8600478.

[2] X. Li et al., "Design and Verification of MCU Chip Bootloader," 2020 IEEE 9th Joint International Information Technology and Artificial Intelligence Conference (ITAIC), 2020, pp. 395-402, doi: 10.1109/ITAIC49862.2020.9339147.

[3] Du, L., “UDS in CAN flash programming”, in Materials Science and Engineering Conference Series, 2019, vol. 490, no. 7, p. 072060. doi:10.1088/1757-899X/490/7/072060.

[4] W. Zhiqiang, "Design and implementation of Bootloader base on CAN bus," 2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC), 2020, pp. 43-46, doi: 10.1109/ITNEC48623.2020.9084675.

[5] D. HU, D. HOU, K. GUO and C. SUN, "Design and implementation of Diagnostic system for Integrated body Controller Based on CAN bus," 2019 Chinese Automation Congress (CAC), 2019, pp. 1175-1178, doi: 10.1109/CAC48633.2019.8996400.

[6] S. Mischie and R. Pazsitka, "Designing a MSP430 Bootloader," 2019 International Conference on Applied Electronics (AE), 2019, pp. 1-4, doi: 10.23919/AE.2019.8866999.

[7] B. Kinniry, “More Auto Computers Means More Complicated, Costly and Longer Repairs”, ceinetwork.com

https://www.ceinetwork.com/cei-blog/auto-computers-means-complicated-costly-longer-repairs/, (accessed Sep 26, 2021)

[8] D. D. Ward and I. Ibarra, "Development phase in accordance with ISO 26262," 8th IET International System Safety Conference incorporating the Cyber Security Conference 2013, 2013, pp. 1-6, doi: 10.1049/cp.2013.1718.

[9] Road vehicles — Functional safety, ISO/FDIS 26262, 2018.

[10] Pendrill, Richard, “Automation of UDS-based flashing for software testing puposes in CANoe”, M.S. thesis, Dept. Indistrial Electrical Engineering and Automation, Lund University, Lnd, Swiden, 2016.

[11] VDA QMC Working Group 13 / Automotive SIG, “Automotive SPICE Porcess Assessment Model”, 2017.

[12] Pendrill, Richard, “Automation of UDS-based flashing for software testing puposes in CANoe”, M.S. thesis, Dept. Indistrial Electrical Engineering and Automation, Lund University, Lnd, Swiden, 2016.

[13] Texas Instruments., “TMS570LC4357 Hercules™ Microcontroller Based on the ARM® Cortex®-R Core”, Acceded: September 26, 2021, Avaliable: https://www.ti.com/lit/ds/symlink/tms570lc4357.pdf?ts=1636382658633&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FTMS570LC4357

[14] Microchip Technology Inc., “SAM E70/S70/V70/V71 Family - Datasheet” Acceded: September 26, 2021, Avaliable: https://ww1.microchip.com/downloads/en/DeviceDoc/SAM-E70-S70-V70-V71-Family-Data-Sheet-DS60001527E.pdf

[15] INCOSE Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities, 4th ed., International Council on System Engineering (INCOSE), Seattle, WA, USA, 2015

[16] International Council on System Engineering (INCOSE), “Guide for Writing Requirements, 2012.

[17] AUTOSAR, “Layared Software Architecture”, Acceded: September 26, 2021, Avaliable: https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf

[18] S. Dekanic, R. Grbic, T. Maruna and I. Kolak, "Integration of CAN Bus Drivers and UDS on Aurix Platform," 2018 Zooming Innovation in Consumer Technologies Conference (ZINC), 2018, pp. 39-42, doi: 10.1109/ZINC.2018.8448921.

[19] Road vehicles — Unified diagnostic services (UDS), ISO 14229, 2006.

[20] D. Bogdan, R. Bogdan and M. Popa, "Design and implementation of a bootloader in the context of intelligent vehicle systems," 2017 IEEE Conference on Technologies for Sustainability (SusTech), 2017, pp. 1-5, doi: 10.1109/SusTech.2017.8333509.

[21] Microchip Technology Inc., “SAM V71 Xplained Ultra - User Guide” Acceded: September 26, 2021, Avaliable: http://ww1.microchip.com/downloads/en/devicedoc/atmel-42408-samv71-xplained-ultra_user-guide.pdf