Politecnico di Torino

**The Department of Electronics and Telecommunications (DET)**

**Master of Science Thesis in Electronic Engineering (Embedded Systems)**

A.Y. 2024/2025

October 2024

# Development of an embedded system for electric vehicles with particular attention to hardware-software integration

**Enhancing Electric Vehicle Experience Through Engine Sound Simulation**

Supervisor:
Professor Mihai Teodor Lazarescu

Candidate:
Ali Fakour Razeghi

## Abstract

The rapid adoption of electric vehicles (EVs) has introduced numerous environmental benefits, including reduced emissions and lower operational noise. However, the silent operation of EVs presents a unique challenge: the absence of the auditory cues traditionally provided by internal combustion engine (ICE) vehicles. Engine noise, though often considered a byproduct, plays a crucial role in enhancing the driver's experience and providing auditory feedback essential for vehicle control. The lack of such feedback in EVs leads to reduced sensory involvement, diminished situational awareness, and decreased control confidence for drivers. This thesis addresses these gaps by developing a synthetic engine sound system tailored specifically for EVs, aimed at restoring an engaging driving experience through carefully designed auditory stimuli.

The proposed system utilizes the SPC58EC80-DISP board, an advanced automotive microcontroller, to interface with various EV sensors and communication buses, including the Controller Area Network (CAN) bus, analog-to-digital converters (ADC), and pulse-width modulation (PWM) signals. Real-time data such as speed, throttle position, and motor load are collected and processed to simulate ICE engine sounds corresponding to the vehicle's current operational state. This data is then forwarded to a miniPC, such as a Raspberry Pi, which generates realistic engine sounds that dynamically adjust based on driving conditions like acceleration, deceleration, and cruising. The system's design ensures precise synchronization of auditory feedback with vehicle behavior, thereby contributing to a more intuitive, immersive, and satisfying driving experience.

A notable feature of the system is its bidirectional communication capability, which ensures that the synthesized sound output is always synchronized with the vehicle's real-time state. Data flows continuously between the vehicle sensors and the sound generation unit, allowing instant adjustments based on driver actions and changes in torque demand. Moreover, the system supports feedback to vehicle control units, preserving critical functions such as torque management and stability control, thereby enhancing driver experience without compromising safety or overall vehicle performance. This tight integration between auditory output and vehicle dynamics ensures that the driver receives continuous, context-appropriate auditory cues, creating a cohesive and responsive experience.

This thesis outlines the methodology for designing, developing, and testing the synthetic engine sound system in detail. The hardware setup encompasses real-time data collection from multiple EV sensors, while software development focuses on extracting, processing, and integrating this information to produce responsive and contextually appropriate sound synthesis. Communication protocols are implemented to maintain precise synchronization between the vehicle and the sound generator, ensuring realistic and immersive auditory feedback. The testing phase includes functional, environmental, and user experience assessments to validate the system's performance under different driving conditions. The results demonstrate that this integrated system effectively bridges the sensory gap inherent in EVs, significantly enhancing driver engagement, improving situational awareness, and potentially improving vehicle control. This research thus provides a novel and effective solution to a significant challenge posed by the transition from ICE vehicles to quieter EVs, contributing to both safety and driving enjoyment.

**Keywords**: Electric vehicles (EVs), Auditory feedback, Internal combustion engine (ICE), Engine noise, Driving experience, SPC58EC80-DISP board, Data collection, Sensors, Communication buses, Controller Area Network (CAN bus), Analog-to-digital converters (ADC), Pulse-width modulation (PWM), MiniPC, Raspberry Pi, Sound synthesis, Torque control, Real-time adjustments, Vehicle control, Immersive auditory experience, Sound generation, Software development, Communication protocols, Vehicle performance

"To my beloved parents, whose unwavering love, endless support, and boundless sacrifices have guided me to this moment. This work stands as a testament to your belief in me and the foundation of values you've instilled. With all my gratitude and love, I dedicate this to you."

# Acknowledgements

First and foremost, I would like to express my sincere gratitude to my supervisor, **Professor Mihai Teodor Lazarescu**, for his guidance and support throughout the preparation of this thesis. His constructive feedback, particularly regarding the academic structuring of this work, was invaluable in ensuring that it met the expected standards. His balanced and straightforward approach made the process more effective and helped me stay on track during challenging phases of the research.

I am also deeply grateful to **Francesco Spagnolo**, CEO of **2Electron**, for his generous support and for providing the essential materials and components that made this project possible. His commitment to innovation and encouragement throughout my research were fundamental in helping me achieve my objectives.

In addition, I would like to thank my colleagues at **2Electron** for their kind support and collaboration throughout the course of this project. Their technical expertise, patience, and valuable input significantly contributed to overcoming various challenges and enhanced my overall experience. Working alongside such knowledgeable and dedicated professionals has been an incredibly inspiring aspect of this journey.

Finally, I wish to acknowledge the broader **2Electron** team, whose collaborative work environment and commitment to fostering research provided an ideal setting for my thesis. The resources and support offered by **2Electron** were critical in making this work a success.

# Table of Contents

1 Introduction ... 1
1.1 Significance of Research ... 2
1.2 Research Objectives ... 3
1.3 Brief Overview of Methodology ... 4
1.4 Comprehensive Literature Review ... 5
1.4.1 Previous Work on Sound Synthesis for Electric Vehicles ... 5
1.4.2 Embedded Systems Used in Automotive Applications ... 9
1.4.3 Challenges and Future Trends ... 11
2 Background and Analysis ... 13
2.1 Key Requirements for Automotive Embedded Systems ... 13
2.2 Analysis of Alternative Embedded Systems ... 14
2.2.1 NXP S32K Series ... 14
2.2.2 Texas Instruments TMS570 ... 16
2.2.3 Renesas RH850 ... 18
2.2.4 The SPC58EC80-DISP Board: Justification for Selection ... 20
2.3 Justification ... 21
3 Methodology ... 23
3.1 System Design and Integration ... 23
3.1.1 Architecture Development ... 23
3.1.2 EV System Integration ... 25
3.1.3 Hardware Configuration ... 27
3.2 Data Acquisition and Transmission ... 29
3.2.1 Firmware Development ... 30
3.2.2 Communication Protocols ... 33
3.2.3 Data Handling and Filtering ... 36
3.3 Testing and Validation ... 41
3.3.1 Functional Testing ... 41

User Experience Testing ........ 44
3.3.2 Environmental Testing ........ 46
3.3.3 Iterative Refinement ........ 49
3.3.4 Long-term Reliability ........ 51
4 Hardware Overview ........ 55
4.1 MCU Key Features ........ 56
4.2 Power Supply ........ 63
4.3 CAN and ISO CAN-FD ........ 64
4.3.1 CAN Transceivers (U1 and U2) ........ 64
4.3.2 Bus Termination and Filtering ........ 65
4.3.3 Test Points (TP1, TP2, TP3, and TP4) ........ 65
5 Software Overview ........ 67
5.1 SPC5Studio Overview ........ 67
5.1.1 Creating a New Application ........ 68
5.1.2 Add Components to the Project ........ 71
5.1.3 Generate Application Code ........ 72
5.1.4 Compile Your Application ........ 73
5.2 UDE STK Overview ........ 74
5.2.1 Key Features and Overview of UDE STK 5.2 ........ 74
5.2.2 Application Areas and Benefits ........ 77
6 Development of the Embedded System for Electric Vehicles ........ 79
6.1 State Machine Design for Sound Synthesis ........ 80
6.2 Initial Configuration ........ 84
6.2.1 Clock Distribution ........ 84
6.2.2 Low Level Driver Component Register-Level Abstraction (RLA) ........ 85
6.2.3 Pin Mapping ........ 86
6.2.4 SARADC Configuration ........ 87
6.2.5 UART Configuration ........ 90
6.2.6 DSPI Configuration ........ 92

6.3 CAN Bus Overview and Implementation ..... 95
6.3.1 CAN Bus as a Communication System ..... 95
6.3.2 Physical Structure of the CAN Bus ..... 95
6.3.3 Working Mechanism ..... 96
6.3.4 Advantages of the CAN Bus ..... 96
6.3.5 Use Cases in Vehicles ..... 96
6.3.6 CAN Message Characteristics ..... 97
6.3.7 CAN Implementation Codes ..... 99
7 Results and Discussion ..... 102
7.1 Functional, User Experience, and Environmental Testing ..... 103
7.1.1 Functional Testing Outcomes ..... 103
7.1.2 User Experience Testing Outcomes ..... 103
7.1.3 Environmental Testing Outcomes ..... 104
7.2 System Performance Analysis ..... 104
7.2.1 Real-Time Responsiveness ..... 104
7.2.2 Sound Quality and Synchronization ..... 104
7.3 Comparison With Relevant References ..... 105
8 Conclusion and Future Improvements ..... 107
8.1 Summary of Main Results ..... 107
8.2 Limitations ..... 108
8.3 Implications for the Industry ..... 109
8.4 Future Improvements ..... 109
9 Bibliography ..... 111

# List of Figures

Figure 1-1 Acoustic Vehicle Alerting System (AVAS). .......... 6
Figure 3-1 Overall system architecture .......... 24
Figure 3-2 Unsmoothed accelerator pedal signals captured by ADC inputs .......... 37
Figure 3-3 Smoothed accelerator pedal signals captured by ADC inputs .......... 38
Figure 3-4 Data transmission latency measurement .......... 42
Figure 4-1 DISP discovery board-top side .......... 56
Figure 4-2 SPC58EC80 Block diagram .......... 61
Figure 4-3 Peripheral allocation .......... 62
Figure 4-4 Power supply circuit of the SPC58EC80-DISP board .......... 63
Figure 4-5 CAN and ISO CAN-FD interface circuit of the board .......... 66
Figure 5-1 Project name window .......... 68
Figure 5-2 Application name window .......... 69
Figure 5-3 Select platform window .......... 70
Figure 5-4 Add components to project .......... 71
Figure 5-5 Application generation summary .......... 72
Figure 5-6 Compile application .......... 73
Figure 6-1 Finite state machine flowchart .......... 80
Figure 6-2 Clock tree .......... 84
Figure 6-3 Low Level Driver Component RLA .......... 85
Figure 6-4 Pin mapping .......... 86
Figure 6-5 SARADC configuration parameters .......... 88
Figure 6-6 UART configuration parameters .......... 90
Figure 6-7 SPI communication signals .......... 93
Figure 6-8 CAN twisted pair wire typical color .......... 95
Figure 6-9 CAN driver structure .......... 99
Figure 6-10 CAN TX message frame structure .......... 100
Figure 6-11 CAN RX message frame structure .......... 101

# List of Tables

Table 2-1 NXP semiconductors S32K1xx microcontroller family characteristics ............... 14
Table 2-2 Texas Instruments TMS570LS0x32 Microcontroller characteristics ................... 16
Table 2-3 Renesas RH850/F1L microcontroller family characteristics ............................. 18
Table 2-4 STMicroelectronics SPC5 microcontroller family characteristics ...................... 20
Table 2-5 Comparative table among MCUs................................................................... 21

# Acronyms

ADC: Analog-to-Digital Converter

ADAS: Advanced Driver Assistance Systems

CAN: Controller Area Network

DMA: Direct Memory Access

DSPI: Deserial/Serial Peripheral Interface

EV: Electric Vehicle

FSM: Finite State Machine

I2C: Inter-Integrated Circuit

ICE: Internal Combustion Engine

ISO CAN-FD: CAN with Flexible Data-Rate

LIN: Local Interconnect Network

PWM: Pulse Width Modulation

RTOS: Real-Time Operating System

SARADC: Successive Approximation Register Analog-to-Digital Converter

SoC: State of Charge

SoC: System on Chip

SPI: Serial Peripheral Interface

UART: Universal Asynchronous Receiver-Transmitter

# 1 Introduction

The emergence of electric vehicles (EVs) has significantly transformed the automotive industry, offering a quieter, more energy-efficient driving experience compared to traditional internal combustion engine (ICE) vehicles. However, this quietness, while beneficial in reducing noise pollution, presents a unique challenge: the absence of auditory feedback that drivers have relied upon for decades. In ICE vehicles, the engine noise serves as a critical source of sensory input, providing valuable information about the vehicle's speed, acceleration, and general performance. This auditory feedback helps drivers maintain control and enhances their engagement with the vehicle.

In contrast, the nearly silent operation of EVs creates a sensory gap, leading to a diminished connection between the driver and the vehicle's operational state. The lack of engine sound not only reduces the emotional engagement that comes with driving but can also hinder a driver's intuitive ability to gauge vehicle behavior, potentially impacting driving performance and safety. Addressing this problem is the primary focus of this thesis: developing a system to synthesize realistic engine sounds for EVs, thereby restoring an essential layer of feedback and enhancing the overall driving experience.

At the heart of this system is the SPC58EC80-DISP board, a versatile and powerful microcontroller platform specifically designed for automotive applications. Built on a robust 32-bit Power Architecture® MCU, the SPC58EC80-DISP is well-suited for tasks that require high reliability and real-time performance—both of which are essential in the context of modern EV systems. The board features multiple communication interfaces such as CAN, LIN, FlexRay, UART, SPI, I2C and Ethernet, as well as ADCs, PWMs, and general-purpose I/Os, making it highly adaptable to various automotive control systems.

This microcontroller board serves as the backbone of the proposed sound synthesis system. By collecting and processing real-time data from the EV's sensors and communication buses, the SPC58EC80-DISP enables accurate monitoring of vehicle conditions. The CAN bus interface plays a pivotal role in gathering key data, such as vehicle speed, throttle position, and torque demand. In addition, ADC channels will capture essential analog signals, such as accelerator pedal position.

The data collected from these sources will then be transmitted to a miniPC, such as a Raspberry Pi, which will handle the sound synthesis process. Using advanced sound generation algorithms, the miniPC will convert the data into realistic, responsive engine sounds that correspond to the current driving conditions. These synthesized sounds will be output through speakers strategically placed in the vehicle's cabin, providing drivers with an immersive auditory experience that mimics the familiar engine sounds of ICE vehicles. This feedback not only enhances driving enjoyment but also serves as a sensory tool for better vehicle control.

One of the key innovations in this system is its bidirectional communication capability. The SPC58EC80-DISP board not only transmits sensor data to the miniPC but also receives feedback or commands that are essential for maintaining vehicle performance, such as torque adjustments. This closed-loop communication ensures that the synthesized sounds are always in sync with the vehicle's operational state, thus enhancing the overall driving experience without compromising critical vehicle functions.

In summary, this thesis presents a novel solution to the sensory gap inherent in EVs. By leveraging the SPC58EC80-DISP board's capabilities for real-time data collection and processing, combined with advanced sound synthesis techniques, this system will offer drivers an immersive and responsive auditory experience, contributing to both driving pleasure and control precision in electric vehicles.

## 1.1 Significance of Research

The significance of this research lies in addressing a critical sensory deficiency in modern electric vehicles. As the automotive industry shifts towards EVs, the traditional sensory cues associated with driving are being lost, potentially affecting driver engagement and safety. This research aims to bridge this gap by providing an innovative solution that restores auditory feedback, a fundamental component of the driving experience. By synthesizing engine sounds that are responsive to real-time vehicle dynamics, this system will enhance both the emotional connection between the driver and the vehicle and improve intuitive driving control.

Furthermore, this research has broader implications for the automotive industry, where enhancing the driving experience is becoming increasingly important in the competitive EV market. By developing a system that reintroduces auditory cues, manufacturers can make EVs more appealing to a wider range of drivers, including those who may be hesitant to switch from ICE vehicles due to the perceived loss of driving enjoyment. In addition, the proposed system could contribute to improved safety by providing drivers with another layer of real-time feedback that aids in decision-making.

The research also highlights the capabilities of the SPC58EC80-DISP board in automotive applications, demonstrating how versatile microcontroller platforms can be leveraged to solve emerging challenges in the industry. This approach can serve as a reference for future developments in EV technology, where real-time data processing and sensory feedback systems will play a crucial role in defining the user experience. Ultimately, this work contributes to the evolution of electric vehicles, making them not only environmentally friendly but also enjoyable and safe to drive.

## 1.2 Research Objectives

The specific objectives of this research are as follows:

1. **Develop a Sound Synthesis System for EVs**: The primary goal is to design and implement an advanced sound synthesis system that generates realistic engine sounds for electric vehicles. This system will utilize real-time data collected from various vehicle sensors to provide auditory feedback that mimics the experience of ICE vehicles. Such sound cues are crucial for both driver immersion and safety, addressing the unique challenge of the inherently quieter EV driving experience.

**Significance**: By synthesizing engine-like sounds, this research aims to enhance the driving experience, providing intuitive auditory cues that aid driver awareness, especially during acceleration, deceleration, and gear changes. The added auditory feedback can help improve the overall driving experience for users transitioning from ICE vehicles to EVs.

2. **Integrate Real-Time Data Collection and Processing**: Utilize the SPC58EC80-DISP microcontroller to gather and process critical sensor data from the vehicle's CAN bus and ADC channels. This data will be used to accurately capture and represent the operational state of the vehicle, such as speed, pedal position, and other driving conditions.

**Significance**: The real-time nature of this system ensures that synthesized engine sounds correspond precisely to the vehicle's current status. Effective real-time processing and data handling are essential to guarantee the realism and responsiveness of the generated sound, making it a seamless part of the driving experience.

3. **Establish Bidirectional Communication for Synchronization**: Implement a closed-loop communication system between the SPC58EC80-DISP microcontroller and a miniPC. This bidirectional setup will enable continuous synchronization between the vehicle's conditions and the synthesized auditory feedback, ensuring that any changes in the vehicle's operational parameters are reflected immediately in the sound output.

**Significance**: Ensuring that the synthesized sounds are always in sync with the vehicle's conditions is vital for driver confidence and situational awareness. This closed-loop communication system reduces latency and potential discrepancies, providing a robust connection between vehicle behavior and auditory feedback.

4. **Enhance Driver Experience and Safety**: Evaluate the impact of synthesized sounds on driver engagement, vehicle control, and safety. The study will focus on how auditory feedback influences driver behavior, improves responsiveness, and contributes to a more immersive and safer driving experience.

**Significance**: Properly designed auditory feedback can contribute significantly to enhancing situational awareness for drivers, especially in the context of quiet electric vehicles. By improving driver engagement and offering audible indicators of speed and vehicle behavior, the system can help reduce driver errors and increase safety, especially in urban environments where pedestrians and other road users may not hear EVs approaching.

5. **Demonstrate the Applicability of the SPC58EC80-DISP Board**: Showcase the SPC58EC80-DISP microcontroller's capabilities in real-time data acquisition, sound synthesis, and closed-loop control, highlighting its effectiveness for automotive applications beyond traditional use-cases.

**Significance**: Demonstrating the applicability of the SPC58EC80-DISP board in this complex automotive scenario underlines its versatility and robustness. This research aims to position the microcontroller as a viable platform for advanced real-time embedded systems, potentially encouraging its adoption in similar automotive or industrial projects, thus expanding its usage footprint.

## 1.3 Brief Overview of Methodology

The proposed methodology outlines a systematic approach for designing and implementing a sound synthesis system for EVs. The process involves the integration of hardware and software components to collect real-time vehicle data, synthesize engine-like sounds, and ensure seamless communication between the EV's systems and the sound generation unit. This methodology is divided into four key phases: system design and integration, data acquisition and transmission, sound synthesis development, and testing and validation. Each phase is designed to ensure the system's performance, reliability, and user experience meet the required standards.

- **System Design and Integration**: This phase involves developing a modular system architecture that integrates both hardware (SPC58EC80-DISP board, sensors, actuators) and software (firmware for the SPC58EC80-DISP board). The architecture must facilitate efficient interaction between components, including robust communication pathways such as CAN or UART to ensure seamless data flow.
- **Data Acquisition and Transmission**: Firmware will be developed for the SPC58EC80-DISP to collect data from vehicle sensors and communicate with a miniPC for sound synthesis. Low-level programming in C will be used to ensure real-time data handling and communication reliability, while robust communication protocols like CAN or SPI will be implemented to manage data flow effectively.
- **Sound Synthesis Development**: The miniPC, such as a Raspberry Pi, will process the data collected from the vehicle and use advanced sound generation algorithms to create realistic engine sounds. These algorithms will simulate different engine modes (e.g., idle, acceleration, deceleration) to enhance the auditory experience in alignment with real-time vehicle conditions.

- **Testing and Validation**: The system will undergo rigorous functional, user experience, and environmental testing to validate its performance and reliability. Functional tests will ensure accurate data transmission and synchronization between the EV and the synthesized sounds. User experience testing will focus on evaluating the realism and satisfaction of the generated sounds under various driving conditions. Environmental testing will ensure the robustness of the system across different conditions such as temperature and humidity.

This iterative process will help ensure that the final system is both reliable and effective in enhancing the driving experience of electric vehicles by reintroducing a critical sensory component that is missing in EVs compared to traditional ICE vehicles.

## 1.4 Comprehensive Literature Review

The use of sound synthesis in electric vehicles has emerged as a significant area of interest in recent years, driven by the need to enhance the driver experience, improve vehicle integration, and address safety concerns associated with the nearly silent operation of these vehicles. Unlike traditional internal combustion engine vehicles, which provide drivers with rich auditory feedback that supports both engagement and intuitive vehicle control, EVs often lack these sensory cues. To address this gap, various approaches to synthetic sound generation have been developed, focusing on creating realistic engine sounds that replicate the auditory characteristics of ICE engines. These efforts aim to improve driver engagement, restore a familiar auditory experience, ensure safety by making EVs more audible to pedestrians and other road users, and integrate seamlessly with existing embedded systems in automotive applications.

This section provides a comprehensive review of the different techniques and systems that have been developed for sound synthesis in EVs, including previous work on sound synthesis for electric vehicles, the role of embedded systems in automotive applications, and the challenges and future trends in this field. The review will highlight the evolution of sound synthesis methods, the integration of embedded systems to facilitate sound generation, and the ongoing challenges in achieving realistic, adaptable, and regulation-compliant solutions that enhance both driver experience and public safety.

### 1.4.1 Previous Work on Sound Synthesis for Electric Vehicles

The use of sound synthesis in electric vehicles is a relatively new field, emerging in response to the unique auditory challenges posed by the lack of engine noise in EVs. Previous research has focused on various aspects of sound synthesis, including safety, driver experience, and technical implementation. This literature review examines these areas to provide context for the present study.

**1. Safety Considerations**

Early studies on sound synthesis for EVs emphasized the safety implications of a nearly silent vehicle, especially for pedestrians and cyclists. Research by (Hella GmbH & Co. KGaA, 2021) examined the development of Acoustic Vehicle Alerting Systems (AVAS), which are designed to produce artificial sounds that help alert pedestrians to the presence of an EV. This work demonstrated that specific frequencies and sound patterns can effectively increase pedestrian awareness without being overly intrusive, offering a solution that balances safety needs with sound pollution concerns.

Further investigations by (Nikolaos Kournoutos & Jordan Cheer, 2019) examined the use of directional speakers and adaptive sound levels to provide effective warnings while minimizing noise pollution. Their findings suggested that sound synthesis could be tailored to the environmental context, enhancing safety without compromising the quiet nature of EVs. This approach allows the warning sounds to adapt to varying noise conditions, thereby reducing unnecessary sound output in quieter environments.

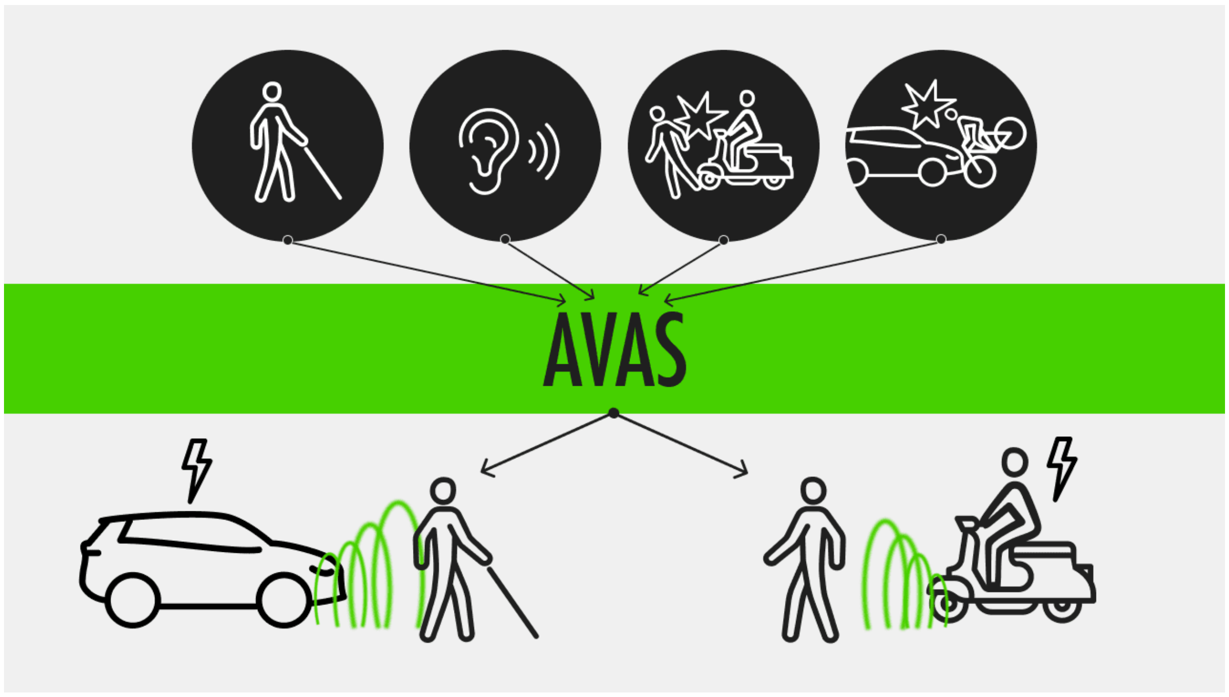


*Figure 1-1 Acoustic Vehicle Alerting System (AVAS). The AVAS system generates sound for nearly silent electric vehicles to improve the safety of vulnerable road users such as pedestrians, cyclists, and children.*

Figure 1-1 is depicting the Acoustic Vehicle Alerting System (AVAS), a system implemented to improve safety around electric vehicles and hybrid vehicles, which are typically much quieter compared to traditional internal combustion engines. The diagram shows key motivations and effects of AVAS, represented with different icons:

- **Visual Impairment Assistance**: AVAS aids visually impaired individuals by emitting artificial vehicle sounds, allowing them to better detect and navigate around electric vehicles.
- **Improved Auditory Detection**: AVAS addresses the challenge of low-noise EVs by adding audible cues, ensuring that pedestrians and other road users can hear approaching vehicles, enhancing awareness.
- **Preventing Accidents with Pedestrians and Cyclists**: Quieter EVs pose a risk to pedestrians and cyclists who may not hear them, increasing the likelihood of accidents. AVAS helps mitigate this risk by making EVs audible in low-speed environments.
- **Reducing Overall Accident Risk**: The implementation of AVAS aims to reduce collisions by ensuring both pedestrians and other vulnerable road users, such as cyclists, have clear auditory alerts of approaching electric vehicles.

The bottom part of the image visually represents how AVAS emits sounds, ensuring that pedestrians and individuals using scooters or other small vehicles can detect and avoid electric vehicles effectively, especially in urban areas or while navigating streets.

## 2. Driver Experience and Engagement

A significant body of research has focused on how sound synthesis can improve driver experience and engagement. A study by (Tsugi Studio, 2018) on real-time engine sound synthesis demonstrated that adding realistic engine sounds enhances driver engagement by mimicking the auditory cues traditionally provided by internal combustion engine vehicles. The study found that incorporating responsive, synthesized engine sounds improved drivers' perception of acceleration and vehicle performance, creating a more immersive and satisfying driving experience.

Similarly, (Valter Prpic, Elena Gherri, & Luisa Lugli, 2024) explored the psychological impact of synthesized engine sounds on drivers, particularly in how auditory feedback influences speed perception and vehicle control. Their findings showed that synthesized sounds could reduce driver anxiety in high-speed situations by providing crucial auditory cues. Without these sounds, drivers often underestimate their speed, potentially leading to faster driving. This research underscores the importance of realistic sound feedback in helping drivers maintain control and feel more attuned to the vehicle's dynamics, contributing to a more controlled and satisfying driving experience.

**3. Technical Implementation and Sound Design**

From a technical perspective, previous research has explored various methods for generating synthesized sounds in electric vehicles. (Lazaro, et al., 2022) investigated the use of granular synthesis for EV sound design, focusing on how specific parameters can shape the driver's emotional response. Their study found that granular synthesis is particularly effective for producing dynamic and responsive sounds that adapt in real time to changes in vehicle speed and throttle position, providing an immersive and engaging experience. This approach highlights the potential of granular synthesis to enhance driver satisfaction through a more authentic auditory experience that mirrors the vehicle's operation.

Another notable contribution in sound synthesis for electric vehicles comes from **Ansys Sound: ASD GeneBOX**, which focuses on integrating real-time data from the CAN bus to modulate sound synthesis parameters. This approach allows synthesized sounds to be closely synchronized with the vehicle's operational state, ensuring a coherent and realistic auditory experience that enhances driver engagement. By dynamically adjusting sound based on real-time data, ASD GeneBOX demonstrates the importance of responsive sound design in creating an authentic experience that mirrors the driving conditions and vehicle behavior.

**4. Challenges in EV Sound Generation**

Generating engine-like sounds for electric vehicles presents several challenges, both technical and perceptual. One major challenge is the **realism** of synthesized sounds. Achieving a sound that feels authentic to drivers and replicates the auditory cues provided by traditional engines is complex, particularly since electric motors operate differently from internal combustion engines. Techniques such as granular synthesis and physical modeling need to be carefully tuned to ensure that the generated sounds closely mirror the dynamic changes experienced during driving.

One significant challenge in generating sounds for electric vehicles is managing noise pollution while meeting regulatory standards, particularly in urban environments. (Cesbron, et al., 2021) explored how different road surfaces impact electric vehicle noise emissions at urban speeds, underscoring the complexities of designing sounds that enhance safety without contributing to excessive noise pollution. Their findings illustrate the importance of developing sound synthesis systems that comply with noise regulations while preserving the quiet characteristics of EVs, especially in densely populated areas where noise management is essential.

Latency is also a critical issue in EV sound generation. The system must process real-time data from vehicle sensors and generate corresponding auditory feedback without perceptible delay. Any lag between vehicle actions and the associated sound can disrupt the driver's experience and potentially reduce trust in the system. Therefore, low-latency sound processing algorithms and efficient hardware-software integration are necessary to provide a seamless driving experience.

Another area of exploration in EV sound synthesis is the personalization of auditory experiences to cater to individual driver preferences. (Chang K., Cho G., Song W., & Kim M., 2022) investigated the design of personalized EV driving sounds that adapt based on the driver's emotional state, using real-time emotion recognition to adjust sound profiles dynamically. This approach allows the auditory feedback in EVs to be more responsive and tailored to the driver's mood, enhancing user satisfaction and engagement. By aligning sound synthesis with the driver's emotional needs, this study demonstrates the potential of customized soundscapes to create a more enjoyable and immersive driving experience.

**5. Challenges and Future Directions**

Despite the advancements made in sound synthesis for EVs, several challenges remain. One major challenge is achieving a balance between providing sufficient auditory feedback for driver engagement while minimizing noise pollution. Research by (Cesbron, et al., 2021) highlighted the complexities involved in designing EV sounds that are informative yet unobtrusive, particularly in urban environments where noise regulations are strict. Their study on the influence of road surfaces on EV noise emissions underscores the importance of carefully crafted soundscapes that enhance driver awareness without contributing excessively to urban noise.

Another area for future exploration is the personalization of synthesized sounds. Studies by (Chang K., Cho G., Song W., & Kim M., 2022) have begun to explore how sound profiles could be customized based on real-time emotion recognition, allowing soundscapes to adapt to the driver's mood and preferences. This level of customization has the potential to improve driver satisfaction significantly, making EVs more appealing to a broader audience by offering a more personalized and engaging auditory experience.

### 1.4.2 Embedded Systems Used in Automotive Applications

Embedded systems play a crucial role in modern automotive applications, contributing to safety, efficiency, and overall vehicle performance. The increasing complexity of vehicle electronics has led to the adoption of sophisticated embedded platforms that integrate various subsystems, including engine control, infotainment, and advanced driver assistance systems (ADAS). This literature review focuses on the use of embedded systems in automotive applications, highlighting their evolution, current trends, and relevance to sound synthesis in electric vehicles.

**1. Evolution of Embedded Systems in Automotive Applications**

The role of embedded systems in automotive applications has grown considerably over the past few decades. Initially, automotive electronics focused on basic tasks such as electronic ignition and fuel injection control, limited by the technology of the time and operating in isolation. However, advancements in semiconductor technology during the 1960s and beyond enabled the development of more capable automotive systems.

The introduction of powerful microcontrollers in the 1990s, particularly those based on 32-bit architectures, allowed embedded systems to support more complex functions. This new level of processing power enabled advancements in engine management and real-time diagnostics, providing vehicles with greater efficiency and performance capabilities.

In the 2000s, embedded systems became more interconnected as technologies like the CAN bus allowed different electronic control units (ECUs) to communicate seamlessly within a vehicle. This networking capability enabled the integration of multiple subsystems, enhancing both the functionality and reliability of modern vehicles.

**2. Real-Time Processing and Reliability**

Modern automotive embedded systems must meet strict real-time and reliability requirements, particularly for safety-critical applications. Real-time operating systems (RTOS) play a crucial role in automotive ECUs, providing the deterministic response times required for functions such as anti-lock braking systems (ABS) and airbag deployment. Standards like AUTOSAR have defined RTOS frameworks specifically for automotive applications to ensure the precision and reliability needed for these critical tasks. The SPC58EC80-DISP, featuring a 32-bit Power Architecture® e200z4 dual-core CPU, is an example of a microcontroller capable of supporting such demanding requirements. With interfaces for CAN FD, Ethernet, FlexRay, and LIN, it enables real-time data acquisition and decision-making, making it well-suited for applications such as sound synthesis and ADAS.

**3. Communication Protocols and Data Handling**

Communication protocols are fundamental to the effective operation of automotive embedded systems. The CAN bus has long been the standard for in-vehicle communication due to its robustness and efficiency in managing real-time data. More recently, advanced protocols like FlexRay and Ethernet have gained traction in the automotive industry, offering higher data rates and supporting more complex interactions between ECUs.

The SPC58EC80-DISP board, used in this research, leverages these advanced communication protocols to facilitate data collection and distribution for the sound synthesis system. By utilizing CAN for real-time sensor data acquisition and Ethernet for high-speed communication with external processing units, this embedded system can deliver the performance required for seamless auditory feedback.

**4. Embedded Systems in Sound Synthesis**

The use of embedded systems in sound synthesis for EVs is a relatively recent development. Modern microcontrollers, particularly those based on ARM Cortex architectures, are increasingly used to generate engine-like sounds in EVs, providing real-time auditory feedback that responds to changes in vehicle speed and load. These microcontrollers leverage digital signal processing (DSP) algorithms to create realistic and responsive sounds, enhancing both driver experience and pedestrian safety. Techniques such as granular synthesis and physical modeling have proven effective on microcontrollers equipped with DSP capabilities, enabling dynamic sound generation that adapts to driving conditions.

In the context of this thesis, the SPC58EC80-DISP board functions as the main controller, gathering sensor data and interfacing with a sound synthesis unit. With its high processing power and diverse communication interfaces, the board ensures that synthesized sounds are precisely aligned with the vehicle's operational state, delivering real-time auditory feedback that contributes to a more immersive driving experience.

### 1.4.3 Challenges and Future Trends

The development of sound synthesis systems for electric vehicles presents several challenges that must be addressed to achieve widespread adoption and effectiveness. One major technical challenge lies in achieving a balance between realistic auditory feedback for driver engagement and compliance with regulatory noise standards, especially in urban environments. Synthesized engine sounds must be informative yet unobtrusive, requiring the use of sophisticated algorithms to create dynamic soundscapes that adapt to changing driving conditions. Additionally, these soundscapes must be carefully crafted to prevent excessive noise pollution, which is increasingly regulated in densely populated areas.

Another critical challenge is managing latency in sound generation. To ensure a seamless driving experience, the system must generate auditory feedback with minimal delay. This requires efficient real-time processing capabilities, integrating data from vehicle sensors and producing sound outputs that align precisely with vehicle dynamics. Any lag between vehicle actions and corresponding auditory feedback can reduce driver trust and diminish the sense of control, making low-latency systems crucial for effective sound synthesis.

Security is also a growing concern as sound synthesis systems become more interconnected with other vehicle subsystems. Ensuring robust communication security to protect against cyber threats is vital, particularly as vehicles become more connected through external networks and protocols. The complexity of ensuring the integrity of data transmission while maintaining system responsiveness presents a significant challenge that developers must overcome.

In terms of future trends, the integration of artificial intelligence (AI) and machine learning (ML) is expected to revolutionize sound synthesis in EVs. These technologies can be used to personalize the driving experience by analyzing driver behavior and preferences. For instance, AI-driven sound synthesis systems could adjust auditory feedback based on individual driving styles, providing more personalized and engaging auditory experiences. Machine learning could also be used for predictive maintenance, using real-time data analytics to anticipate component wear or failures, allowing for proactive interventions that enhance both system reliability and sound accuracy.

Another emerging trend is the development of adaptive sound systems that respond not only to vehicle dynamics but also to external environmental factors. By incorporating data on ambient noise levels, these systems can dynamically modify sound profiles to enhance clarity and maintain auditory feedback quality in various driving environments. This adaptability ensures that the generated sounds remain effective under different conditions, such as heavy urban traffic or quiet rural settings.

Additionally, advancements in embedded systems are enhancing the capabilities of sound synthesis units. Microcontrollers, such as the SPC58EC80-DISP board, with high processing power and diverse communication interfaces, are enabling more sophisticated data integration and processing, facilitating the real-time generation of high-quality, immersive auditory feedback. The integration of DSP (Digital Signal Processing) capabilities directly into microcontrollers is also allowing for more complex and realistic sound modeling techniques, such as granular synthesis and physical modeling, which enhance the authenticity of synthesized sounds.

The future of sound synthesis in EVs will also likely involve increased collaboration between automotive manufacturers and regulatory bodies to establish standardized requirements for vehicle auditory feedback. As the demand for personalized, yet regulation-compliant, auditory experiences grow, sound synthesis technologies must evolve to meet both consumer expectations and legislative standards.

## Conclusion

The literature on sound synthesis for electric vehicles provides a comprehensive foundation for understanding the various dimensions of this emerging field. Safety, driver experience, and technical implementation are key areas that have been extensively studied, highlighting both the potential benefits and the challenges of introducing synthesized sounds in EVs. Additionally, the review of embedded systems in automotive applications underscores their essential role in enabling sophisticated vehicle functions, such as real-time data processing and communication, which are critical to the successful implementation of a sound synthesis system. This thesis builds on previous research by focusing on real-time data integration and the use of advanced sound generation techniques to enhance driver engagement and safety, addressing the sensory gap left by the absence of traditional engine sounds in electric vehicles.

# 2 Background and Analysis

The use of embedded systems in modern EVs is pivotal for real-time data collection, communication between various vehicle subsystems, and efficient processing of sensor data. Such systems must be robust, compliant with automotive standards, and capable of handling real-time operations with minimal latency. In this chapter, we will analyze different embedded system platforms based on their ability to meet the demands of automotive applications, specifically focusing on communication protocols, real-time data collection, and processing capabilities. The chapter concludes by presenting the SPC58EC80-DISP board as the superior choice for this project due to its automotive-specific design and features.

## 2.1 Key Requirements for Automotive Embedded Systems

Before discussing alternative platforms, it’s essential to define the key requirements for an embedded system in EVs, particularly those focused on real-time data collection and processing:

- **Automotive communication protocols**: The system must support protocols such as CAN, LIN, and potentially FlexRay or Ethernet to enable communication between ECUs and sensors.
- **Real-time processing**: The system must handle data in real-time, ensuring minimal latency in processing and acting on the sensor data to improve vehicle safety and performance.
- **Reliability and durability**: Automotive-grade systems must withstand harsh environments, including temperature extremes, vibrations, and EMI.
- **Low power consumption**: EVs require power-efficient embedded systems to ensure they do not drain the vehicle's power reserves.
- **Scalability and flexibility**: The embedded system should allow easy integration with additional sensors or future system upgrades.
- **Automotive compliance**: The system must comply with automotive standards such as ISO 26262 (functional safety) and AEC-Q100 (qualification for integrated circuits).

## 2.2 Analysis of Alternative Embedded Systems

### 2.2.1 NXP S32K Series

The NXP S32K family of microcontrollers is designed for automotive applications, with a focus on safety, reliability, and support for automotive communication protocols. The main characteristics are shown in the Table 2-1.

| Category | Feature | S32K1XX Characteristics |
|---|---|---|
| **Core** | Architecture | Arm Cortex-M4F (32-bit) |
| | Clock Speed | Up to 112 MHz (HSRUN mode) |
| | Floating Point Unit (FPU) | Single Precision |
| **Memory** | Flash Memory | Up to 2 MB |
| | SRAM | Up to 256 KB with ECC |
| | FlexNVM | Up to 64 KB for EEPROM emulation |
| | Cache | 4 KB Code Cache |
| **Communication** | CAN | Up to 3 FlexCAN modules with optional CAN-FD |
| | LIN | Up to 3 LPUART modules supporting LIN protocol (v1.3 to v2.2A) |
| | SPI | Up to 3 Low-Power SPI (LPSPI) modules |
| | $I^2C$ | Up to 2 Low-Power $I^2C$ (LPI2C) modules |
| | Ethernet | 1 x 10/100 Mbps Ethernet (with IEEE 1588 support) |
| | FlexIO | Configurable for UART, $I^2C$, SPI, $I^2S$, PWM, etc. |
| **Analog** | ADC | Up to 2 x 12-bit ADCs (32 channels per module) |
| | Comparator | 1 x Analog Comparator with 8-bit DAC |
| **Timers** | Timers | Up to 8 FlexTimer modules (16-bit counters with PWM, IC/OC support) |
| | Low-Power Timers | 1 Low-Power Timer, 1 Low-Power Interrupt Timer, and 2 Programmable Delay Blocks (PDB) |
| **Safety** | ISO Compliance | ASIL B or D (depending on configuration) |
| | Watchdog Timers | Internal Watchdog and External Watchdog Monitor |
| | CRC Channels | Integrated CRC unit |
| | Error Correction | ECC on Flash and SRAM |
| **Power** | Low Power Modes | HSRUN, RUN, STOP, VLPR, and VLPS modes |
| | Voltage Range | 2.7V to 5.5V |
| **Debug** | Debug Interfaces | JTAG, Serial Wire Debug (SWD), and Trace support |
| **Package** | Available Packages | QFN (32-pin), LQFP (48/64/100/144/176-pin), MAPBGA (100-pin) |
| | Qualification | AEC-Q100 compliant for automotive applications |

*Table 2-1 NXP semiconductors S32K1xx microcontroller family characteristics*

- **Strengths**:
  - **Automotive protocols support**: The S32K series supports CAN, LIN, and FlexRay communication, which makes it suitable for real-time data exchange in EVs.
  - **Real-time processing**: Equipped with ARM Cortex-M processors, the S32K series is designed for low-latency operation and can handle real-time data processing efficiently.
  - **Automotive-grade**: The S32K microcontrollers are qualified for automotive use, with certifications for functional safety (ISO 26262).
  - **Low power consumption**: Designed to be energy-efficient, making them ideal for EV applications.
- **Weaknesses**:
  - **Development complexity**: Although the S32K series offers automotive-grade features, developing and optimizing code for real-time tasks can be more complex, requiring expertise in automotive toolchains and safety standards.
  - **Limited high-end processing**: While sufficient for most real-time data collection tasks, the processing power may not be as robust as more advanced systems for complex data analysis or advanced processing algorithms.
- **Conclusion**: The NXP S32K series is a strong contender for automotive applications, providing the necessary communication protocol support and reliability for data collection and processing, though it may be limited in more computationally demanding tasks.

### 2.2.2 Texas Instruments TMS570

The Texas Instruments TMS570 series is a family of ARM Cortex-R-based microcontrollers designed for safety-critical automotive applications, particularly those requiring high performance and reliability. The main characteristics are shown in the Table 2-2.

| Category | Feature | TMS570LS0432 Characteristics |
|---|---|---|
| **Core** | Architecture | Dual ARM Cortex-R4F running in lockstep |
| | Clock Speed | Up to 80 MHz |
| | Instruction Protection | ECC (Error Correction Code) on Flash and RAM |
| **Memory** | Flash Memory | 384 KB (Program Flash with ECC) |
| | SRAM | 32 KB with ECC |
| | Data Flash | 16 KB (Emulated EEPROM with ECC) |
| **Communication** | CAN | 2 x DCAN (CAN 2.0B compliant) |
| | LIN | 1 UART with LIN 2.1 support |
| | SPI | 2 Standard SPI modules + 1 Multibuffered SPI (MibSPI) |
| **Analog** | ADC | 12-bit ADC with 16 channels and 64 result buffers |
| **Timers** | High-End Timer | Next Generation High-End Timer (N2HET) with 19 programmable pins |
| | Quadrature Encoder | Enhanced QEP for position and motion control |
| **Safety** | ISO Compliance | ISO 26262 ASIL D compliant |
| | Watchdog Timers | Built-in Digital Watchdog Timer |
| | Error Monitoring | Error Signaling Module (ESM) with external error pin |
| | Self-Test | Built-In Self-Test (BIST) for CPU and RAM |
| **Power** | Low Power Modes | Supports low-power modes and optimized clock gating |
| | Voltage Range | Core: 1.14V–1.32V, I/O: 3.0V–3.6V |
| **Debug** | Debug Interfaces | JTAG, ARM CoreSight, IEEE 1149.1 boundary scan |
| **Package** | Available Packages | LQFP-100 |
| | Qualification | AEC-Q100 for automotive applications |

*Table 2-2 Texas Instruments TMS570LS0x32 Microcontroller characteristics*

- **Strengths**:
    - **Automotive safety**: The TMS570 series is compliant with ISO 26262 for functional safety, making it ideal for critical automotive systems.
    - **Real-time processing**: With the Cortex-R architecture, these microcontrollers are designed for real-time processing tasks and are capable of handling complex data processing with minimal latency.
    - **Communication support**: TMS570 microcontrollers support automotive communication protocols such as CAN, LIN, and FlexRay, ensuring seamless integration into EV networks.
    - **High performance**: Capable of handling more demanding processing tasks due to the real-time architecture and dual-core lockstep features for redundancy.
- **Weaknesses**:
    - **Higher power consumption**: While offering high performance, the TMS570 series generally consumes more power compared to lower-end automotive microcontrollers, which might be a disadvantage in EV applications where power efficiency is critical.
    - **Complexity**: Like the NXP S32K series, the TMS570 requires specialized knowledge and tools for development and is best suited for teams with significant automotive experience.
- **Conclusion**: The TMS570 excels in safety-critical applications with higher real-time processing demands. However, its higher power consumption and development complexity may limit its suitability for certain EV data collection tasks.

### 2.2.3 Renesas RH850

Renesas RH850 microcontrollers are known for their use in automotive applications requiring high performance, low power, and reliability. The main characteristics are shown in the Table 2-3.

| Category | Feature | RH850/F1L Characteristics |
|---|---|---|
| **Core** | Architecture | **G3KH core** |
| | Clock Speed | **Up to 240 MHz** |
| | Floating Point Unit (FPU) | Supported (Single Precision) |
| **Memory** | Flash Memory | **Up to 4 MB with ECC** |
| | SRAM | **Up to 120 KB with ECC** |
| | Retention RAM | Supported for standby modes |
| **Communication** | CAN | **2 x CAN-FD** |
| | LIN | Supported via LIN channels |
| | SPI | **Up to 6 channels** |
| | UART | Multiple channels |
| | Ethernet | Optional (10/100 Mbps) |
| **Analog** | ADC | **8 x 12-bit ADC channels** |
| | Comparators | Supported |
| **Timers** | General-Purpose Timers | Multiple timer channels |
| | Watchdog Timers | Dual Watchdog (Windowed and Interval) |
| | Real-Time Clock | Supported |
| **Safety** | ISO Compliance | **ISO 26262 ASIL D compliant** |
| | Fault Collection and Control Unit | Integrated |
| | Error Correction | ECC on Flash and RAM |
| **Power** | Low Power Modes | STOP, HALT, Retention Standby |
| | Voltage Range | 3.3V and 5V operation |
| **Debug** | Debug Interfaces | JTAG, On-chip Debug Unit (OCDU) |
| **Package** | Available Packages | QFP and BGA packages |
| | Qualification | AEC-Q100 for automotive applications |

*Table 2-3 Renesas RH850/F1L microcontroller family characteristics*

- **Strengths**:
    - **Power efficiency**: The RH850 series is highly optimized for low power consumption, making it ideal for EVs.
    - **Automotive-grade communication**: These microcontrollers support CAN, LIN, and Ethernet AVB, providing flexibility in data collection and real-time communication.
    - **Real-time capabilities**: The RH850 series offers robust real-time processing features, including high-speed data collection and processing, which is essential for EV applications.
    - **Automotive compliance**: The RH850 is ISO 26262 compliant, ensuring that it meets functional safety requirements for automotive systems.
- **Weaknesses**:
    - **Limited ecosystem**: While the RH850 provides excellent performance for real-time applications, its development ecosystem is somewhat limited compared to other platforms like NXP or TI, potentially slowing development.
    - **Moderate performance**: While efficient for many automotive tasks, the RH850 may not handle extremely complex real-time analytics or tasks requiring advanced processing power as well as high-end systems like TMS570.
- **Conclusion**: The Renesas RH850 strikes a good balance between power efficiency and real-time processing, making it a strong choice for data collection in EVs, though its ecosystem and development support may be limiting for more advanced tasks.

### 2.2.4 The SPC58EC80-DISP Board: Justification for Selection

The SPC58EC80-DISP board is part of STMicroelectronics' SPC5 family, designed specifically for automotive applications. It offers distinct advantages over the alternatives, particularly in terms of real-time processing, communication support, and automotive compliance. The main characteristics are shown in the Table 2-4.

| Category | Feature | SPC58EC80 Characteristics |
|---|---|---|
| **Core** | Architecture | Dual **e200z420n3** (Power Architecture) |
| | Clock Speed | Up to 180 MHz |
| | Instruction Cache | 8 KB per core |
| | Data Cache | 4 KB per core |
| | Floating Point Unit | Single Precision |
| **Memory** | Flash Memory | 4 MB Flash (with 128 KB data flash) |
| | SRAM | 384 KB general-purpose + 128 KB local data RAM (64 KB per core) |
| **Communication** | CAN | 8 x CAN-FD |
| | FlexRay | 1 x Dual Channel |
| | Ethernet | 1 MAC (10/100 Mbps, VLAN, AVB, and time-stamping support) |
| | LIN | 18 LINFlexD modules (UART-compatible) |
| | SPI | 8 x DSPI |
| | $I^2C$ | 1 channel |
| **Analog** | ADC | 5 x SAR ADC (3 x 12-bit, 1 supervisor 12-bit, 1 standby 10-bit ADC) |
| **Timers** | Timers | 64 x eMIOS (Enhanced Modular I/O Subsystem) |
| | System Timers | 8 x PIT (Periodic Interrupt Timer) |
| **Safety** | ISO Compliance | ISO 26262 ASIL B Capable |
| | Watchdog Timers | 3 Software Watchdog Timers |
| | CRC Channels | 2 CRC units |
| | Fault Collection and Control Unit | Integrated FCCU |
| **Power** | Low Power Modes | HALT, STOP, Standby with RTC, Smart Standby |
| | Supply Voltage | 3.3V / 5V |
| **Debug** | Debug Interfaces | High-speed SIPI/LFAST, Nexus Development Interface, JTAG |
| **Package** | Available Packages | eLQFP-176, eTQFP-144, eTQFP-100 |
| | Qualification | AEC-Q100 for automotive applications |

*Table 2-4 STMicroelectronics SPC5 microcontroller family characteristics*

To have a better understanding of characteristics of all mentioned MCUs, Table 2-5 shows and compares their characteristics.

| Category | SPC58EC80-DISP | S32K1XX | TMS570LS0432 | RH850/F1L |
|---|---|---|---|---|
| **Core** | Dual e200z420n3 (Power Arch.) | Arm Cortex-M4F | Dual Cortex-R4F (Lockstep) | G3KH Core |
| **Clock Speed** | 180 MHz | Up to 112 MHz | 80 MHz | 240 MHz |
| **Flash Memory** | 4 MB | Up to 2 MB | 384 KB | Up to 4 MB |
| **RAM** | 384 KB | Up to 256 KB | 32 KB | 120 KB |
| **CAN** | 8 x CAN-FD | Up to 3 x CAN-FD | 2 x CAN | 2 x CAN-FD |
| **FlexRay** | 1 x Dual Channel | No | No | Yes |
| **Ethernet** | 10/100 Mbps with VLAN | Optional | No | Optional |
| **ADC** | 5 x 12-bit SAR ADCs | 2 x 12-bit ADCs | 12-bit, 16 channels | 8 x 12-bit ADCs |
| **Safety Compliance** | ISO 26262 ASIL B capable | ASIL B/D capable | ISO 26262 ASIL D | ISO 26262 ASIL D |
| **Communication** | Comprehensive (UART, SPI, LIN, I²C, Ethernet) | Moderate (UART, SPI, I²C, CAN) | Limited (UART, CAN, SPI) | Strong (UART, SPI, CAN, Ethernet optional) |
| **Power Modes** | Advanced (HALT, STOP, Standby) | HSRUN, RUN, STOP modes | Basic low-power modes | Multiple low-power modes |

*Table 2-5 Comparative table among MCUs*

## 2.3 Justification

**Extensive Communication Capabilities:**

EV systems rely on CAN and advanced networking protocols to gather real-time data from multiple sensors and ECUs.

The SPC58EC80 supports 8 CAN-FD channels, FlexRay, and Ethernet, enabling seamless integration with the vehicle's complex communication network.

Alternatives like the S32K1XX and RH850/F1L offer fewer CAN-FD channels, and the TMS570LS0432 lacks CAN-FD entirely.

**High Processing Power:**

The dual-core architecture (e200z420n3) of the SPC58EC80 ensures robust real-time processing, crucial for handling sensor data and generating synthesized sound outputs without latency.

Although the RH850/F1L offers a higher clock speed (240 MHz), its single-core architecture may limit multitasking capabilities.

**Memory Resources:**

The SPC58EC80 provides 4 MB Flash and 384 KB RAM, sufficient for data-intensive tasks like processing sensor inputs and managing sound synthesis algorithms.

The TMS570LS0432 has significantly lower memory (384 KB Flash, 32 KB RAM), and the S32K1XX and RH850/F1L offer comparable but less versatile memory configurations.

Safety Features:

While the SPC58EC80 is ASIL B capable, it supports features like multiple watchdog timers and CRC units, which are sufficient for your non-safety-critical sound synthesis application. Achieving ASIL D is unnecessary for this use case.

Development Ecosystem:

SPC5Studio offers a robust development environment tailored for automotive applications, streamlining your project's development lifecycle.

**Conclusion**

The SPC58EC80-DISP stands out as the most suitable microcontroller for your project. Its:

Unmatched communication capabilities,

Sufficient processing power,

Ample memory resources, and

Automotive-focused design with a strong development ecosystem

make it the best choice for synthesizing engine sounds in EVs. This board not only meets your project's technical requirements but also provides scalability for future enhancements.

# 3 Methodology

The proposed methodology outlines a systematic approach for designing and implementing a sound synthesis system for EVs. The process involves the integration of hardware and software components to collect real-time vehicle data, synthesize engine-like sounds, and ensure seamless communication between the EV's systems and the sound generation unit. This methodology is divided into four key phases: system design and integration, data acquisition and transmission, sound synthesis development, and testing and validation. Each phase is designed to ensure the system's performance, reliability, and user experience meet the required standards.

## 3.1 System Design and Integration

### 3.1.1 Architecture Development

Design a modular system architecture incorporating both hardware (SPC58EC80-DISP board, sensors, actuators) and software (firmware, sound synthesis code). The architecture design forms the foundation of the system, ensuring all components are organized efficiently and interact seamlessly. The goal is to create a modular system architecture that integrates both hardware and software components, allowing flexibility, scalability, and ease of troubleshooting (Figure 3-1). Key considerations include:

- **Hardware Integration**: The primary hardware includes the SPC58EC80-DISP board, which serves as the central unit responsible for gathering data from the EV’s sensors and actuators. Additional sensors (e.g., for speed, throttle position, and braking) are integrated to capture real-time vehicle metrics. These sensors provide the necessary inputs for sound synthesis. The architecture should ensure easy hardware interfacing and reliable power management.

- **Modular Design**: A modular approach allows each component—data collection, communication, and sound synthesis—to function independently while interacting with the other modules. This makes it easier to update or troubleshoot individual parts without affecting the entire system. Modules can include the data acquisition module (for handling sensor inputs), the communication module (for transmitting data), and the sound generation module (for synthesizing and outputting sounds).

- **Software Integration**: On the software side, the system will incorporate low-level firmware running on the SPC58EC80-DISP to handle real-time data collection and communication with the sound synthesis software. The sound generation code will be executed on a miniPC, using advanced algorithms to synthesize engine sounds. The software must be designed to ensure seamless interaction between the hardware and the sound synthesis engine, maintaining synchronization with vehicle performance in real-time.

- **Scalability and Flexibility**: The system should be designed to accommodate future expansions or modifications. For instance, additional sensors or changes to sound synthesis algorithms can be incorporated without redesigning the entire system. Using a modular design also enables easier upgrades to either the hardware (e.g., a newer version of the SPC58EC80-DISP) or software (e.g., updated synthesis algorithms) without major architectural changes.
- **Communication Pathways:** The architecture must ensure robust communication between the SPC58EC80-DISP board and the miniPC. This can be achieved using a reliable communication protocol (such as CAN or UART), which allows for fast and secure data exchange. The architecture should also include error detection and handling mechanisms to ensure the system remains resilient to potential communication errors.

In summary, the architecture design is essential to ensuring that the system components work together efficiently, and that the system remains flexible, scalable, and easy to maintain. By designing with modularity in mind, each component can be independently developed, tested, and optimized, while still contributing to the overall system performance.

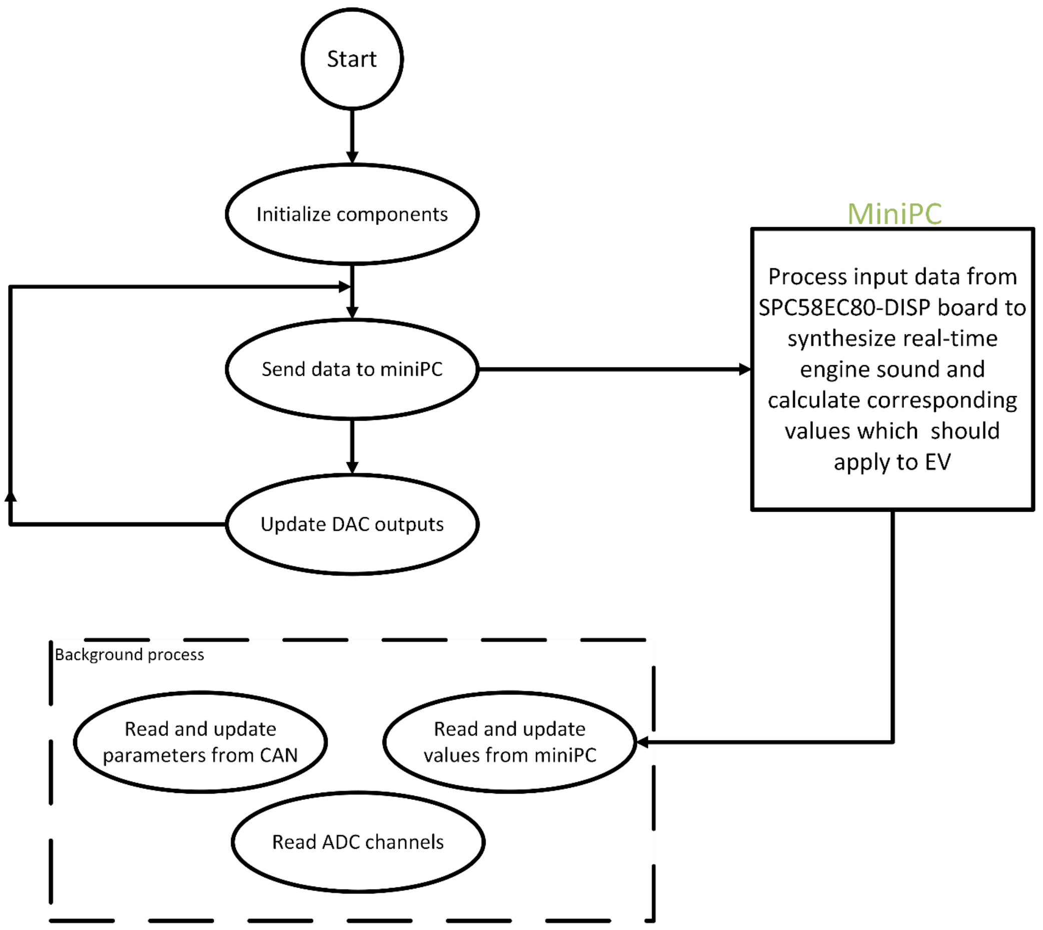

Start
Initialize components
Send data to miniPC
Update DAC outputs
MiniPC
Process input data from SPC58EC80-DISP board to synthesize real-time engine sound and calculate corresponding values which should apply to EV
Background process
Read and update parameters from CAN
Read and update values from miniPC
Read ADC channels

*Figure 3-1 Overall system architecture , showing the main loop of process and process which are running in background to have a real-time response to the system*

### 3.1.2 EV System Integration

Establish how the SPC58EC80-DISP board will interface with the EV’s CAN bus or other vehicle communication systems. Define the signals and parameters (e.g., speed, throttle position) to be monitored. The integration of the SPC58EC80-DISP board with the EV is a critical part of the system design, as it ensures that real-time vehicle data is accurately captured and utilized to drive the sound synthesis engine. This integration must account for the complexity of the EV’s electronic architecture, particularly the communication systems used within the vehicle, such as the CAN bus.

- **Interfacing with the EV’s CAN Bus:** Most modern electric vehicles use the CAN bus protocol to manage communication between various ECUs such as the engine control unit, battery management system, and transmission control unit. The CAN bus allows these components to exchange real-time data efficiently. The SPC58EC80-DISP board must interface with the EV’s CAN bus to collect data such as speed, throttle position, battery state of charge (SoC), motor torque, and other relevant parameters that influence vehicle dynamics.
    - **CAN Bus Protocol**: The CAN bus operates using a multi-master, message-oriented protocol. The SPC58EC80-DISP will need to be configured as a listener on this bus to retrieve the desired signals without interrupting or disturbing the communication flow between other ECUs. The firmware running on the SPC58EC80-DISP must be designed to parse and decode the CAN messages that correspond to vehicle metrics such as RPM, throttle, and acceleration. These signals will then be used as inputs to the sound synthesis engine to produce the desired audio effects that mimic traditional engine sounds.
    - **Signal Definition and Parameter Selection**: Defining which signals and parameters are relevant for sound synthesis is an important design step. For example, engine RPM is a key metric for sound synthesis as it directly influences the pitch and intensity of engine sounds. Other important signals may include gear selection, acceleration/braking status, and speed, as they contribute to the dynamic nature of the synthesized sound. The process of signal definition involves studying the CAN bus message structure, identifying key parameters, and implementing filters to extract this information in real-time.
    - **CAN Message Decoding**: Each signal or parameter carried over the CAN bus is encoded in a specific format, often requiring decoding algorithms to interpret the data correctly. The integration will involve setting up the SPC58EC80-DISP to decode relevant CAN identifiers (CAN IDs) and map them to the corresponding physical parameters of the vehicle. For instance, a specific CAN ID might represent vehicle speed, which can be converted from its raw binary form into a human-readable value that feeds into the sound synthesis algorithms.

- **Alternate Communication Protocols**: In some cases, depending on the EV’s architecture, other communication protocols like FlexRay, Ethernet, or LIN (Local Interconnect Network) may be in use. FlexRay, for example, offers higher data rates and redundancy, which might be required for more advanced vehicles or autonomous driving systems. The SPC58EC80-DISP must be flexible enough to adapt to different communication protocols in case the vehicle uses alternatives to CAN. In such cases, the communication interface needs to be adjusted accordingly, with proper decoders or transceivers added to interface with the specific vehicle bus system.
- **Wiring and Electrical Considerations**: Physical integration requires careful attention to wiring and electrical connections. The SPC58EC80-DISP board should be connected to the EV’s communication bus through the appropriate transceivers (e.g., CAN transceivers) to ensure stable data transfer. Proper grounding, shielding, and power supply considerations are also essential to avoid electromagnetic interference (EMI) that could corrupt the signals.
- **Calibration and Synchronization**: The signals coming from the EV need to be calibrated to match the expected input ranges of the sound synthesis algorithms. For instance, the range of throttle positions may need to be normalized before being used to modulate sound parameters. Additionally, synchronization between the data input and sound output is crucial to prevent latency or mismatches, which could degrade the user experience. Implementing buffering or time-stamping techniques can ensure real-time synchronization between the vehicle’s data and the sound engine's output.
- **Safety and Diagnostic Integration**: Safety is an essential aspect of EV system integration. The system should include **error detection mechanisms** that monitor the communication bus for faults, such as missing or corrupted CAN messages. Additionally, integrating diagnostic capabilities, such as logging errors or abnormal signals, will help in troubleshooting during development and testing. The SPC58EC80-DISP board can be programmed to report anomalies that could impact both the system's performance and safety.

In conclusion, integrating the SPC58EC80-DISP board with the EV's CAN bus and other communication systems is a multi-step process that requires precise understanding of the vehicle’s electronic architecture. This phase ensures the board collects accurate, real-time data for sound synthesis, which is vital for producing an authentic and responsive auditory experience that mirrors traditional engine behavior.

### 3.1.3 Hardware Configuration

Determine the placement of sensors, the wiring of the board, and power management. Choose supporting components, such as transceivers, for reliable communication between systems. In this phase, the physical setup and electrical connections of the system are designed to ensure the hardware functions as intended. This involves careful consideration of the placement of sensors, wiring, power management, and the selection of supporting components to facilitate reliable data collection and communication between systems.

- **Sensor Placement**: The strategic placement of sensors is crucial to ensure accurate and real-time data collection. Depending on the vehicle's architecture and the specific data required for sound synthesis, sensors must be positioned to measure parameters such as speed, throttle position, braking force, and motor load. For example:
    - **Speed sensors** can be placed on the wheels or connected to the vehicle's drivetrain to capture real-time velocity.
    - **Throttle position sensors** are typically attached to the throttle pedal or motor controller, providing data on how much power is being demanded by the driver.
    - **Brake sensors** detect when and how forcefully the brakes are applied, allowing for sound modulation (e.g., simulating deceleration sounds).
    - **Vibration sensors** or accelerometers may also be added to enhance sound realism by capturing physical vibrations related to vehicle dynamics, which can then be translated into auditory feedback.

Careful consideration must be given to environmental conditions, such as temperature and vibration, to ensure sensor durability and performance over time. Sensor calibration and alignment with vehicle components are also critical to ensure the data collected is accurate and directly correlates with vehicle behavior.

- **Wiring Design and Signal Integrity**: The wiring of the board to the sensors and other components needs to be designed with signal integrity in mind. Poor wiring or incorrect connections can introduce noise, latency, or signal loss, which can compromise system performance. Key considerations include:
    - **Cable Routing**: Wires should be routed to minimize their exposure to sources of EMI, such as the EV's motor or battery systems. Careful routing and shielding are needed to avoid signal degradation, especially for analog sensors that are sensitive to noise.
    - **Signal Grounding**: Proper grounding is essential to avoid ground loops and other electrical issues that can affect the integrity of the sensor data. Each component, including the SPC58EC80-DISP board and sensors, should share a common ground, and additional EMI shielding may be applied to the wiring if needed.

    - **Connector Types**: Choosing the right connectors (e.g., Molex, JST, or waterproof automotive connectors) ensures reliable, secure connections between the sensors and the main board. Weather-resistant connectors are essential in environments exposed to dust, moisture, or high temperatures, particularly for systems mounted outside the vehicle cabin.

- **Power Management**: Power management is another key element of hardware configuration. The SPC58EC80-DISP board and all connected sensors and components require stable power supplies. In an EV, the power management system must account for various power sources (e.g., the main vehicle battery, auxiliary systems) and their associated voltages. Considerations include:

    - **Voltage Regulation**: EV systems typically operate at higher voltages (e.g., 48V or more) for the main propulsion system, while the SPC58EC80-DISP board and sensors may require lower voltage levels (e.g., 3.3V or 5V). DC-DC converters or voltage regulators will be needed to step down the voltage appropriately for different components, ensuring stable operation without voltage spikes or drops that could damage the board or sensors.

    - **Power Distribution**: Power must be distributed evenly across the system, with consideration for current draw by each component. For instance, sensors may have varying power requirements depending on their type (e.g., analog vs. digital), and the wiring should be designed to handle these current levels safely.

    - **Power Protection**: Implementing fuses or current limiters ensures the system is protected from short circuits, overloads, or power surges. Additionally, the system should include reverse polarity protection to prevent accidental damage due to improper wiring.

- **Component Selection and Integration**: Choosing the right supporting components is crucial to ensure the entire system functions reliably and efficiently. Key supporting components include:

    - **Transceivers**: The SPC58EC80-DISP board will need CAN transceivers (or transceivers for the appropriate communication protocol) to facilitate data transmission between the vehicle's communication bus and the board itself. These transceivers convert the data from the differential CAN signals to digital signals that can be processed by the board's microcontroller.

    - **Signal Conditioning Circuits**: Signal conditioning circuits may be required to ensure that the sensor data is compatible with the board's ADCs or communication inputs. This can include amplifiers, filters (e.g., low-pass filters to remove noise), or level shifters to match signal levels with the board's requirements.

  - **Buffering and Isolation**: In cases where signals may need to travel long distances or interface with noisy systems, signal buffers or optocouplers may be used to ensure that signals are transmitted cleanly and are electrically isolated from the rest of the vehicle's systems. This reduces the risk of electrical noise or faults in one part of the system affecting other components.
  - **Cooling Solutions**: Depending on the power requirements and the environment in which the system operates, passive or active cooling systems (e.g., heatsinks or fans) may be necessary to dissipate heat generated by the board or components.

- **Mechanical Considerations**: The physical mounting of the SPC58EC80-DISP board, sensors, and other hardware must also be taken into account to ensure longevity and reliability. The system should be housed in a vibration-resistant enclosure, particularly in vehicles where components are exposed to mechanical stress. Enclosures should also be IP-rated for protection against dust and water ingress, particularly if mounted outside or near high-exposure areas of the vehicle.

In conclusion, a well-planned hardware configuration ensures the reliable operation of the system, with robust power management, signal integrity, and component integration being key factors. Each element, from sensor placement to wiring, plays a role in ensuring that the system captures accurate data, processes it efficiently, and communicates with the sound synthesis system effectively, all while remaining resilient to real-world conditions such as vibration and electromagnetic interference.

## 3.2 Data Acquisition and Transmission

The project involves developing low-level firmware for the SPC58EC80-DISP board to handle real-time data collection from electric vehicle sensors, including accelerometers and speed sensors. This firmware directly interfaces with hardware to collect, process, and transmit data in real-time, ensuring reliable system performance. Communication protocols such as CAN, UART, and SPI are implemented to transmit preprocessed sensor data to a miniPC for dynamic EV sound synthesis. To ensure robustness, error-checking mechanisms are integrated to maintain data integrity. Preprocessing of sensor data, including noise filtering and sensor calibration, is crucial to ensure only relevant and accurate information is transmitted, thereby optimizing the system’s efficiency and the quality of synthesized sound.

### 3.2.1 Firmware Development

Develop firmware for the SPC58EC80-DISP using low-level programming (C or assembly) to handle data collection from EV sensors (e.g., accelerometers, speed sensors). Optimize the firmware for real-time operation. Firmware development for the SPC58EC80-DISP board is one of the most critical steps in ensuring that the system can reliably collect, process, and transmit data in real-time from the electric vehicle's sensors. Since the firmware runs at a low level, it is responsible for interfacing with the hardware directly, handling tasks such as communication with the EV’s sensors, data acquisition, real-time processing, and efficient data transmission to the miniPC. The development of this firmware requires a deep understanding of embedded programming, real-time systems, and communication protocols.

- **Low-Level Programming in C/Assembly**: Firmware for the SPC58EC80-DISP board is typically written in C or assembly language due to the need for precise control over hardware resources. C is widely used in embedded systems because of its efficiency, low overhead, and ability to manipulate hardware registers directly. Assembly, on the other hand, offers even more fine-grained control and is sometimes used for highly optimized, time-critical sections of code, though it is less portable and harder to maintain.
    - **Peripheral Drivers**: One of the first tasks in firmware development is writing or configuring **drivers** to interface with the board’s various peripherals, such as ADCs, timers, PWM, and communication interfaces (CAN, UART, SPI, I2C). These drivers provide the foundation for communicating with external sensors and the vehicle's communication bus.
    - **Direct Memory Access (DMA)**: To optimize data handling and reduce the processor's load, the firmware should utilize DMA where possible. DMA allows peripherals to transfer data directly to and from memory without needing the CPU's intervention, freeing up processing power for other tasks like real-time data filtering.
    - **Hardware Abstraction Layer (HAL)**: Using a HAL or custom abstraction layer allows for more portable and maintainable code. The HAL can abstract away some of the board-specific details, enabling easier integration with other boards or future upgrades without rewriting large portions of the code.
- **Data Acquisition from EV Sensors**: The firmware must interface with various sensors (e.g., accelerometers, speed sensors, and throttle position sensors) to collect real-time data about the vehicle’s state. This data is essential for producing accurate sound synthesis in response to vehicle dynamics. Specific steps involved include:

  - **Polling vs. Interrupts**: Depending on the sensor type and system requirements, data can be collected via polling (regularly checking sensor states) or interrupts (where the sensor triggers an event when new data is available). For real-time performance and energy efficiency, using interrupts is typically preferred, as it reduces CPU load by only reacting when necessary.
  - **ADC Configuration**: For analog sensors like accelerometers, the firmware needs to configure the ADC to sample the sensor signals at a high enough frequency to capture fast changes in the vehicle's motion. The sampled data must then be processed, possibly using digital filtering to remove noise or smooth the signal before using it in the sound synthesis engine.
  - **Sensor Fusion**: If the system uses multiple sensors to gather vehicle data, the firmware may need to implement sensor fusion algorithms to combine inputs from different sources (e.g., combining speed sensor data with accelerometer data to enhance accuracy). This can improve the quality of the synthesized sounds by providing more detailed information about the vehicle's state.

- **Real-Time Operating System (RTOS) Considerations**: For more complex systems that require concurrent execution of tasks (e.g., data acquisition, communication), using an RTOS is beneficial. An RTOS ensures that time-critical tasks (such as sensor polling or CAN message handling) are given priority and executed with predictable timing. Key RTOS components include:
  - **Task Scheduling**: In an RTOS-based system, tasks such as data collection, communication with the miniPC, and error handling are divided into separate threads or tasks. The RTOS scheduler prioritizes these tasks to ensure that time-sensitive operations are completed within their deadlines.
  - **Inter-task Communication**: For tasks to share data efficiently, the firmware will need to implement inter-task communication methods, such as message queues or semaphores. This ensures that the data collected from sensors is passed to the sound synthesis or communication modules without delay or data corruption.
  - **Interrupt Handling**: Interrupts from external sensors or communication buses (e.g., CAN) need to be serviced quickly and efficiently. The firmware must be designed to handle these interrupts without significant overhead, ensuring that the real-time performance of the system is maintained.

- **Optimization for Real-Time Operation**: Since the system is designed to respond in real-time to changes in the vehicle's state, the firmware must be highly optimized to minimize latency and ensure that sound output remains synchronized with vehicle data.
    - **Minimizing Latency**: Reducing the time between data acquisition and sound generation is crucial for a seamless user experience. Interrupt-driven programming, efficient buffering of incoming data, and avoiding blocking code are essential to achieve low-latency operation.
    - **Code Optimization Techniques**: Low-level optimizations, such as reducing unnecessary loops, using fixed-point arithmetic instead of floating-point (to save processing power), and inlining frequently used functions, can further improve real-time performance. Additionally, careful memory management, such as placing frequently accessed variables in fast-access memory regions (e.g., registers or cache), will contribute to faster execution.
    - **Real-Time Debugging**: Real-time systems require special debugging techniques to ensure that the firmware performs as expected. Using tools like in-circuit debuggers (ICDs), logic analyzers, and profiling tools helps identify bottlenecks, measure response times, and ensure that tasks are meeting their real-time deadlines.
- **Communication with the MiniPC**: The firmware must also handle reliable communication between the SPC58EC80-DISP and the miniPC, which processes the sound synthesis algorithms. This may involve transmitting the processed sensor data over UART, CAN, or Ethernet depending on the chosen communication protocol. Key considerations include:
    - **Data Packing and Transmission**: To reduce communication overhead, the firmware should implement efficient data packing strategies, combining multiple sensor readings into a single message where possible. It’s also essential to implement error-checking mechanisms, such as Cyclic Redundancy Check (CRC), to ensure data integrity during transmission.
    - **Communication Protocol Optimization**: If the system uses a communication protocol like CAN, the firmware must handle CAN message prioritization and error handling to ensure that critical vehicle data is transmitted without delay. For Ethernet or UART communication, flow control mechanisms such as XON/XOFF or hardware-based flow control may be needed to prevent data loss during high-speed transmission.
- **Testing and Validation**: Firmware development also involves rigorous testing to ensure that the system behaves as expected under various conditions. Unit tests, hardware-in-the-loop (HIL) testing, and stress tests are performed to validate the firmware’s performance in real-time data collection, error handling, and communication.

In conclusion, developing firmware for the SPC58EC80-DISP board involves a combination of low-level programming, real-time optimization, and robust communication strategies to ensure reliable data acquisition and transmission. The firmware must be tightly integrated with the EV's sensors and communication buses to provide the real-time data necessary for dynamic sound synthesis, while ensuring minimal latency and maximum reliability.

### 3.2.2 Communication Protocols

Implement robust communication protocols (e.g., CAN, UART, or SPI) to transmit data from the SPC58EC80-DISP board to the miniPC. Include error-checking mechanisms to ensure data integrity. In any system where data needs to be transmitted between multiple components, particularly in embedded systems, selecting and implementing the right communication protocol is crucial for ensuring reliability, efficiency, and real-time performance. For this project, robust communication between the SPC58EC80-DISP board and the miniPC is essential to transmit real-time sensor data, enabling the dynamic synthesis of EV engine sounds. Several key aspects must be considered to establish a robust and reliable communication channel.

1. **Protocol Selection (CAN, UART, SPI)**: Based on the system's requirements, protocols like CAN, UART, or SPI can be implemented, depending on the type of data being transferred, the distance between components, and the required transmission speed.

   ➢ **CAN (Controller Area Network)**: CAN is widely used in automotive applications due to its robustness and ability to handle multiple nodes communicating on the same bus. It is particularly well-suited for environments where real-time data from multiple sources, such as sensors and control units, needs to be transmitted with minimal errors. Since the project involves data collection from the EV's sensors, which likely already use a CAN bus for communication, integrating the SPC58EC80-DISP into the existing CAN network is a logical choice. CAN also offers collision detection and priority-based message handling, ensuring that high-priority messages (e.g., data from critical sensors) are delivered first, with minimal latency.

   **CAN Communication Steps**:

   - **Message Framing**: CAN messages are transmitted in frames, which include the message ID, data, and error-checking bits (CRC). For efficient communication, the firmware must ensure that sensor data is packed into CAN frames without exceeding the bus's bandwidth limits.
   - **Priority Management**: Each CAN message has an ID that determines its priority. Critical vehicle data (such as speed or throttle position) should have a higher priority ID than less critical information (e.g., ambient temperature).

- **Error Handling**: CAN includes built-in error-detection mechanisms, such as bit-stuffing, CRC checks, and acknowledgment features, which automatically detect and correct transmission errors.

➢ **UART (Universal Asynchronous Receiver-Transmitter)**: UART is a simple, asynchronous communication protocol commonly used for point-to-point data transmission. It is ideal for low-speed, short-distance communication between the SPC58EC80-DISP and the miniPC. UART does not require a clock signal, making it less complex to implement compared to synchronous protocols like SPI.

**UART Communication Steps**:

- **Baud Rate Selection**: The baud rate (transmission speed) must be chosen carefully to balance the data rate with reliability. Higher baud rates offer faster communication but can lead to errors if the line quality is poor or if the processing overhead becomes significant.

- **Flow Control**: To prevent data loss during transmission, flow control mechanisms such as XON/XOFF (software flow control) or RTS/CTS (hardware flow control) can be implemented to ensure that the sender does not overwhelm the receiver with too much data at once.

- **Error Detection**: UART often implements parity bits for basic error detection. In systems where data integrity is critical, additional layers of error-checking, such as checksum or CRC, should be implemented in the higher layers of the communication protocol.

➢ **SPI (Serial Peripheral Interface)**: **SPI** is a high-speed, synchronous communication protocol ideal for short-range communication between devices. It offers much faster data transfer compared to UART, making it suitable for transmitting large amounts of sensor data quickly. SPI uses a **master-slave** architecture, with the SPC58EC80-DISP board typically acting as the master and the miniPC as the slave.

**SPI Communication Steps**:

- **Clock Configuration**: SPI uses a clock signal (SCK) to synchronize data transmission. The clock speed must be set appropriately to ensure that the miniPC can process incoming data without errors. The master (SPC58EC80-DISP) will control this clock, ensuring synchronized data transfer.

- **Full-Duplex Communication**: Unlike UART, SPI allows for full-duplex communication, meaning data can be sent and received simultaneously. This is beneficial in scenarios where bi-directional data transmission is required between the SPC58EC80-DISP and the miniPC, such as sending sensor data while receiving control commands.
- **Slave Select (SS)**: SPI uses the Slave Select (SS) line to choose the device it communicates with. In multi-device systems, proper management of the SS line ensures that only the intended slave device (miniPC) is communicating with the master.

2. **Data Integrity and Error Checking**: In automotive and embedded systems, data integrity is paramount, particularly when transmitting critical sensor information that directly influences the user experience. The communication protocol must include mechanisms to detect and recover from errors during transmission.

   ➢ **Cyclic Redundancy Check (CRC)**: CRC is a robust error-detection method that checks the integrity of data during transmission. In systems like CAN, CRC is embedded in each message frame. For UART or SPI, CRC can be manually implemented in the protocol layer, where a CRC code is calculated before sending the data and verified upon receipt. If a CRC mismatch is detected, the system can request a retransmission of the corrupted data.

   ➢ **Automatic Retransmission and Acknowledgment**: For protocols like CAN, automatic retransmission is built-in. If a transmission error is detected, the data frame is retransmitted until it is acknowledged successfully. For UART and SPI, this can be achieved by implementing acknowledgment (ACK/NACK) signals in the software. The receiver sends an ACK signal if the data is received correctly or a NACK signal if an error is detected, prompting a retransmission.

   ➢ **Error Logging and Recovery**: In real-time systems, tracking communication errors is critical for debugging and ensuring system reliability. Implementing an error logging mechanism within the firmware allows for monitoring error rates and diagnosing communication issues. If a specific protocol or communication line is experiencing frequent errors, the system can switch to a backup communication method (if available) or implement recovery protocols, such as reducing the baud rate to increase reliability.

3. **Real-Time Communication Considerations**: Given the real-time nature of the project, where sensor data is used to generate dynamic engine sounds in response to vehicle performance, minimizing latency and ensuring timely data transmission is critical. The communication protocol should be optimized to reduce transmission delays, ensuring that the system can respond in real-time to vehicle dynamics.

➢ **Buffering and Flow Control**: Implementing buffers within the firmware ensures that data is temporarily stored if the miniPC cannot process it immediately. The firmware should dynamically adjust buffer sizes to handle varying data loads, preventing overflow and data loss. Flow control mechanisms such as hardware flow control (RTS/CTS) for UART or slave management for SPI can also be used to prevent data loss when the receiver is overwhelmed.

➢ **Interrupt-Driven Communication**: To minimize latency, the communication system should be interrupt-driven, meaning that the processor is only interrupted when new data is available, rather than constantly polling the communication lines. For example, when using UART, an RX interrupt can trigger data reception when a new message arrives, allowing the system to handle communication asynchronously without consuming CPU cycles continuously.

4. **Synchronization and Timing**: Ensuring that data is synchronized between the SPC58EC80-DISP and the miniPC is essential for real-time applications. Misaligned or delayed data can lead to incorrect sound generation or unresponsive system behavior.

➢ **Timestamping Data**: One approach to maintain synchronization is to **timestamp** sensor data before sending it to the miniPC. The miniPC can then use these timestamps to align the incoming data with the appropriate sound generation routines, ensuring the engine sounds are accurately synchronized with vehicle dynamics.

➢ **Clock Synchronization**: In systems with synchronous communication (like SPI), keeping the clocks aligned is critical for data integrity. Even in asynchronous systems like UART, clock drift over time can cause communication errors. Periodic re-synchronization of clocks, either manually or automatically, helps prevent issues caused by desynchronized clocks.

### 3.2.3 Data Handling and Filtering

Design methods to preprocess data (e.g., filtering noise, calibrating sensors) before transmission, ensuring only relevant and accurate information is sent for sound synthesis. Accurate and reliable data transmission is a cornerstone of the overall system design, especially when dealing with sensor data in real-time applications such as EV sound synthesis. Before sensor data is transmitted from the SPC58EC80-DISP board to the miniPC, it is essential to preprocess the raw data to ensure that only relevant, accurate, and filtered information is sent. This preprocessing step helps to reduce noise, minimize unnecessary data transmission, and optimize the system's performance by ensuring that only critical information is utilized in the sound synthesis process.

#### 3.2.3.1 Noise Filtering and Data Smoothing

In any electronic system, noise from various sources (e.g., electromagnetic interference, temperature fluctuations) can degrade the accuracy of sensor data. Noise can lead to distorted or inconsistent values, which in turn can affect the sound synthesis algorithms. Therefore, implementing robust filtering techniques is essential to clean the data before it is used for real-time sound generation.

- **Low-Pass Filtering (LPF)**: Low-pass filters are commonly used to smooth out high-frequency noise from sensor signals. For example, the throttle position or speed sensors might produce noisy data due to mechanical vibrations or signal interference. By applying a low-pass filter, high-frequency noise is removed, allowing the system to focus on the actual vehicle dynamics. The cutoff frequency of the filter should be carefully selected based on the characteristics of the noise and the sensor data to avoid filtering out relevant information.
- **Moving Average Filter**: A simple and effective method for smoothing data is the **moving average filter**, which computes the average of a fixed number of previous data points. This helps to reduce short-term fluctuations and provide a more stable signal for further processing. Moving average filters are particularly useful in reducing rapid, unwanted changes in data while still maintaining the overall trend.

  For example, in an EV where acceleration and deceleration may produce rapid sensor readings, a moving average filter can smooth these readings to ensure the sound synthesis algorithm receives a steady input, resulting in smoother sound transitions.

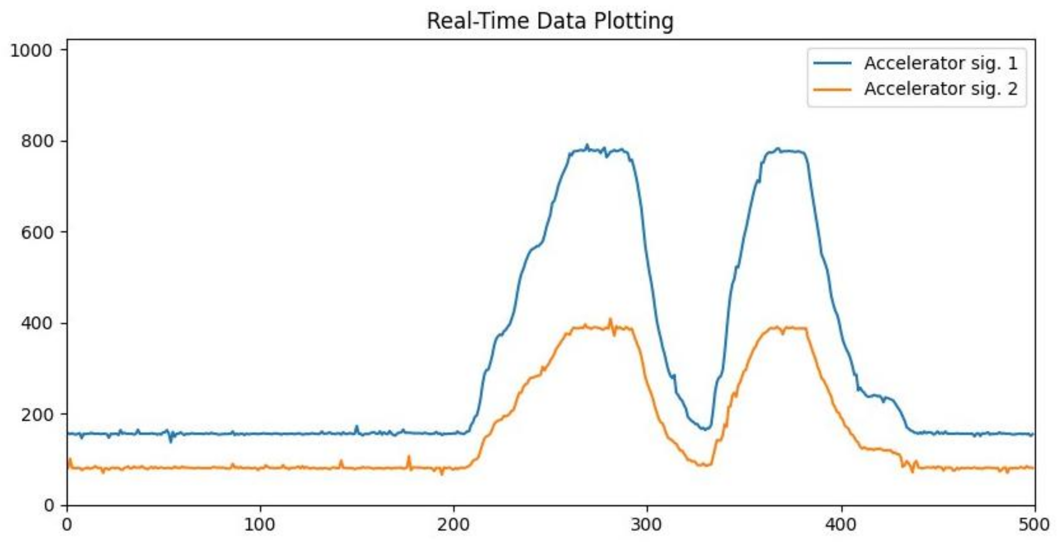


*Figure 3-2 Unsmoothed accelerator pedal signals captured by ADC inputs, showing some small distortions before applying smoothing techniques*

In the Figure 3-2, the signals from the accelerator pedal (labeled as "Accelerator sig. 1" and "Accelerator sig. 2") appear with noticeable irregularities or noise. These irregularities are due to sensor noise, vibrations, or quick, minor changes that aren't critical to capture in full detail.

In the Figure 3-3 (after applying a moving average filter), these same signals are smoother. The filter effectively reduces noise by averaging out rapid, minor fluctuations, leading to a clearer representation of the pedal's general trend.

The moving average filter works by taking the average of a specific number of consecutive data points (a "window") and then replacing the central point in that window with the average value. This process helps to remove short-term variations while retaining the overall trend of the signal. You can see that the filtered signals have fewer sharp changes and smoother edges. The moving average effectively reduces abrupt transitions and rounds off the signal edges, which can be particularly helpful in real-time processing to reduce noise.

In our specific application of synthesizing engine sounds for electric vehicles, using a moving average filter on the pedal signals is crucial to ensure that the generated sounds respond smoothly to pedal inputs. A noisy, unfiltered signal could result in jittery, unnatural sound synthesis, which would negatively impact the user experience. The smoothing provided by the moving average filter ensures that the resulting sound changes are more gradual and realistic, aligning better with the driver's expectations.

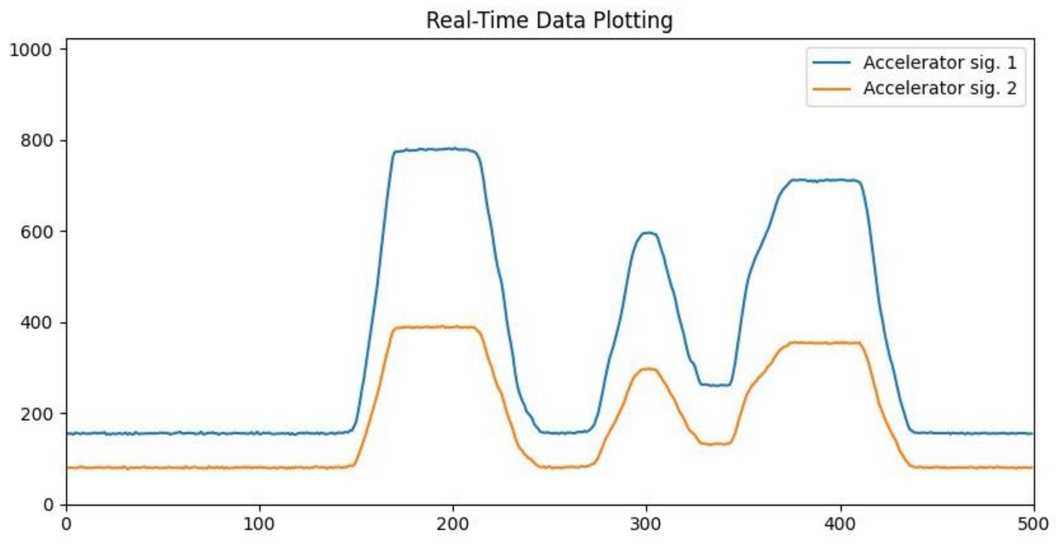


*Figure 3-3 Smoothed accelerator pedal signals captured by ADC inputs, showing a smooth signal after applying smoothing techniques*

- **Kalman Filter**: In more complex systems, a Kalman filter can be implemented for sensor data fusion and noise reduction. This algorithm is particularly effective for systems with multiple noisy inputs, as it continuously estimates the system's true state by minimizing the error between the predicted and actual measurements. The Kalman filter is beneficial for dynamic systems such as EVs, where parameters like speed, throttle, or battery charge fluctuate rapidly. The filter allows for real-time noise reduction and data smoothing while still accurately tracking rapid changes.

#### 3.2.3.2 Sensor Calibration and Data Normalization

Sensor readings often contain errors or offsets that must be corrected through calibration. Calibrating sensors ensures that the data transmitted is reliable and reflects the actual physical quantities being measured.

- **Zeroing and Offsetting**: Many sensors, especially those measuring physical parameters such as acceleration, torque, or speed, may exhibit small biases (offsets) in their readings. Before using the data for sound synthesis, the system should apply zeroing techniques to remove these biases. For example, when the vehicle is stationary, speed sensors might still produce small non-zero values due to electrical noise or mechanical imperfections. In such cases, zeroing the sensor ensures that the system treats zero speed as a true stationary state.

- **Gain Adjustment**: Some sensors may require gain adjustment to correct for errors in sensitivity. For instance, if a throttle sensor consistently underestimates or overestimates the throttle position, a gain correction factor can be applied to ensure that the sensor output accurately reflects the physical throttle state. Gain adjustments are critical when sensors from different manufacturers are used in the same system, as their response characteristics can vary.

- **Normalization of Sensor Data**: To ensure uniformity and facilitate sound synthesis, sensor data may need to be normalized. Normalization involves scaling the raw data to a fixed range, typically between 0 and 1. This step is particularly useful for sensors with varying output ranges. For example, the throttle position sensor might output values ranging from 0 to 1023, while the speed sensor might output values from 0 to 200. By normalizing both datasets to a common scale, the sound synthesis algorithm can process the inputs more effectively, leading to consistent and proportional sound output.

#### 3.2.3.3 Data Compression and Minimization

Efficient use of communication bandwidth between the SPC58EC80-DISP and the miniPC is critical, especially when handling real-time sensor data in an EV environment. Preprocessing the data to reduce its size, while preserving the key information needed for sound synthesis, can significantly improve system performance.

- **Thresholding and Deadbanding**: Thresholding is a technique that eliminates minor variations in sensor data that fall below a certain predefined threshold. For instance, slight variations in speed may not need to be reflected in the sound synthesis, as they would be imperceptible to the driver. By applying thresholding, these small variations are ignored, thus reducing the amount of data transmitted. Similarly, deadbanding can be used to ignore minor fluctuations around a setpoint (e.g., idle speed), further reducing unnecessary data transmission.

- **Event-Driven Transmission**: Instead of continuously sending sensor data, the system can implement an event-driven approach, where data is only transmitted when a significant change occurs. For instance, data from the throttle position sensor would only be sent when there is a noticeable change in the throttle's state. This reduces the amount of data sent, thus optimizing communication bandwidth and minimizing computational load on the miniPC.

- **Data Compression Algorithms**: For more complex systems where high-frequency data must be transmitted, applying data compression algorithms before transmission can help. Lossless compression techniques such as Run-Length Encoding (RLE) or Huffman coding can reduce the size of the data without losing critical information. This is particularly useful for sensors that produce large amounts of data, such as accelerometers or gyroscopes.

#### 3.2.3.4 Relevance Filtering

To ensure that only useful data is transmitted for sound synthesis, it is essential to filter out irrelevant or redundant sensor data. This ensures that the miniPC and the sound synthesis algorithm only process the necessary inputs, reducing computational load and improving real-time performance.

- **Contextual Data Filtering**: Depending on the vehicle's current operating mode, certain sensors might be more relevant than others. For instance, during steady highway cruising, the data from accelerometers or gyroscopes might not be as crucial for sound synthesis, whereas throttle position and speed sensors are. The system should be capable of dynamically adjusting the data filters based on the driving context, transmitting only the relevant information to optimize sound generation.

- **Anomaly Detection**: By implementing simple anomaly detection algorithms, the system can automatically detect and discard faulty or unexpected sensor readings. For example, if the throttle sensor suddenly reports an unusually high value while the speed remains constant, this could indicate a sensor malfunction. In such cases, the system should ignore the erroneous data to prevent it from disrupting the sound synthesis.

#### 3.2.3.5 Data Packaging and Protocol Compliance

Once the data has been filtered, smoothed, calibrated, and compressed, it must be appropriately packaged according to the chosen communication protocol (CAN, UART, or SPI) for transmission to the miniPC. Proper data packaging ensures that the system adheres to the communication protocol's requirements while minimizing transmission errors.

- **Message Structuring**: The filtered and processed data should be structured into predefined message formats, including header information, payload (sensor data), and error-checking mechanisms (such as CRC). Structuring messages properly ensures that the data is easily decoded at the receiving end.
- **Synchronization and Timing**: The data packets must be synchronized to ensure that the miniPC receives the data in the correct sequence and at the right time. Proper timestamping of data and packetizing it based on communication standards (such as CAN frames) ensures that the miniPC interprets the sensor data in a time-aligned manner, which is critical for real-time sound generation.

## 3.3 Testing and Validation

### 3.3.1 Functional Testing

Conduct testing to ensure the system is functioning correctly, focusing on data transmission accuracy, sound generation timing, and communication robustness. Functional testing is a critical step in the development of the sound synthesis system for EVs. This phase is designed to validate the system's core functionalities, ensuring that all components operate as intended and that the synthesized sound accurately reflects real-time vehicle data. The primary objectives of functional testing include verifying data transmission accuracy, assessing sound generation timing, and evaluating the robustness of communication between various system components.

#### 3.3.1.1 Data Transmission Accuracy

Ensuring the accuracy of data transmission is fundamental to the system's performance. The sound synthesis system relies heavily on real-time data from the SPC58EC80-DISP board, which collects various vehicle parameters such as speed, throttle position, and RPM. Accurate data transmission is essential for the miniPC to generate realistic engine sounds that align with the vehicle’s actual performance.

- **Data Integrity Checks**: To ensure the integrity of the data transmitted from the SPC58EC80-DISP to the miniPC, various integrity checks can be implemented. This includes error-checking protocols (e.g., checksums or cyclic redundancy checks) to detect any discrepancies in the data during transmission. By comparing the sent data with the received data, the system can identify any corruption or loss of information, allowing for corrective actions to be taken immediately.

- **Testing Data Latency**: Evaluating the latency of data transmission is also crucial. This involves measuring the time taken for data to travel from the SPC58EC80-DISP to the miniPC and be processed for sound generation. High latency can lead to noticeable delays in the audio output, diminishing the overall user experience.

  Figure 3-4 illustrates the time interval of 20ms for communication between the miniPC and the SPC58EC80-DISP board. The system should be tested under various operational conditions (e.g., idle, accelerating, cruising) to ensure that data latency remains within acceptable limits.

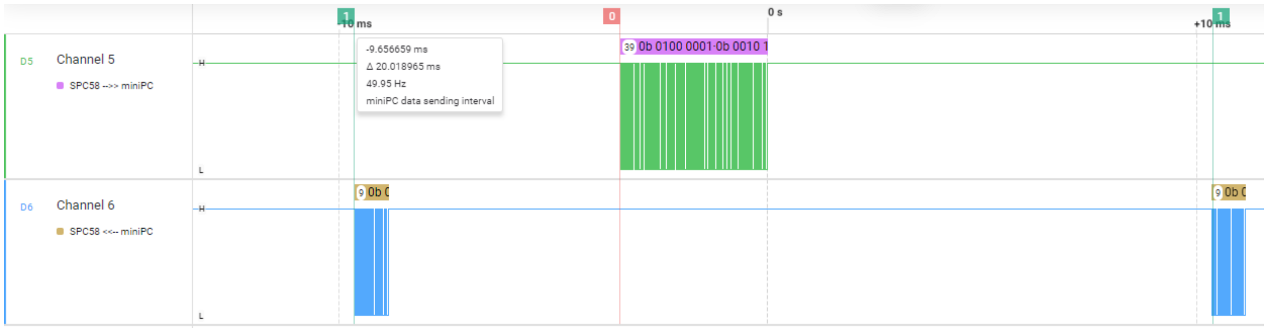


*Figure 3-4 Data transmission latency measurement. This figure shows communication signals captured by logic analyzer in the real project. The SPC58EC80-DISP board receives data every 20ms from miniPC.*

### 3.3.1.2 Sound Generation Timing

The timing of sound generation is vital to creating a realistic and engaging driving experience. The synthesized engine sounds must change dynamically and instantaneously based on real-time vehicle data.

- **Synchronization Tests**: Functional testing should include synchronization tests to ensure that sound generation occurs in perfect alignment with the incoming data. For instance, if the throttle position changes suddenly, the sound output should reflect that change without any lag. This requires precise timing measurements to verify that the sound synthesis algorithms respond accurately to real-time data inputs.

- **Dynamic Response Assessment**: Testing should also involve evaluating how the system reacts to rapid changes in driving conditions, such as quick accelerations or decelerations. The sound output must adapt fluidly to these changes, maintaining a continuous and coherent audio experience. This could be tested using various driving scenarios to gauge the responsiveness and timing of sound generation.

### 3.3.1.3 Communication Robustness

Robust communication between the SPC58EC80-DISP and the miniPC is essential for ensuring that the system remains functional even in the presence of potential disturbances or failures.

- **Stress Testing**: Functional testing should include stress tests that simulate extreme operating conditions, such as high vehicle speeds, rapid sensor updates, or interference from other electronic systems within the vehicle. The goal is to assess how the system handles these conditions and whether the communication remains stable and reliable.
- **Failover Mechanisms**: Implementing and testing failover mechanisms is also vital. In the event of communication failure or data loss, the system should have protocols to either restore communication or switch to a backup system without interrupting the audio output. Functional testing should involve simulating communication failures to verify that the system can effectively recover and continue operating.

#### 3.3.1.4 User Interaction Testing

An essential aspect of functional testing is evaluating how the system interacts with the driver. The synthesized sounds should not only be realistic but also enhance the driving experience without causing distraction or annoyance.

- **User Feedback Loop**: Incorporating a feedback loop during testing can provide valuable insights into user satisfaction with the audio output. Driver feedback can help identify areas for improvement in sound quality, volume levels, and responsiveness. By gathering data on user experiences, the system can be fine-tuned to meet driver expectations better.
- **Adjustability of Sound Parameters**: Functional testing should also include tests to ensure that sound parameters (such as volume, pitch, and type of sound) are adjustable and responsive to user input. For instance, if a driver prefers a sportier sound profile, the system should allow them to switch to a more aggressive engine simulation without compromising performance or quality.

#### 3.3.1.5 Compliance and Standards Testing

Finally, functional testing should ensure that the system complies with relevant automotive standards and regulations. This is crucial for ensuring the safety, reliability, and market readiness of the sound synthesis system.

- **Regulatory Compliance Checks**: Compliance with automotive standards (such as ISO or SAE) is necessary for the system to be considered safe and reliable. Testing should involve evaluating whether the system meets requirements related to audio output, electronic interference, and overall vehicle performance. This ensures that the system not only enhances the driving experience but does so within regulatory frameworks.

## User Experience Testing

Perform subjective testing with drivers to evaluate the realism and satisfaction of the generated sounds in various driving conditions (e.g., acceleration, braking, cruising). Users experience testing is a vital component of the evaluation process for the sound synthesis system in (EVs. This phase focuses on gathering subjective feedback from drivers regarding the realism, satisfaction, and overall impact of the generated engine sounds in various driving scenarios. The insights gained from this testing will inform refinements and enhancements to the system, ensuring it meets the expectations and preferences of end users.

### 3.3.1.6 Objective of User Experience Testing

The primary goal of user experience testing is to assess how well the synthesized sounds resonate with drivers and how effectively these sounds enhance their overall driving experience. Unlike objective measurements, which focus on technical performance metrics, user experience testing seeks to understand the emotional and sensory responses of drivers to the audio output.

- **Realism of Engine Sounds**: Participants will evaluate the authenticity of the synthesized sounds compared to traditional ICE sounds. Realistic sound generation is critical for providing drivers with auditory feedback that aligns with their expectations based on previous driving experiences. Drivers should feel that the sounds represent the vehicle's performance accurately, whether accelerating, cruising, or braking.
- **Satisfaction and Enjoyment**: Drivers will also assess their enjoyment of the generated sounds. A satisfying auditory experience can significantly enhance the perception of the vehicle's performance, making it feel more engaging and enjoyable to drive. This subjective feedback can help identify which sound profiles are preferred and which may be deemed unappealing or distracting.

### 3.3.1.7 Methodology for Testing

To conduct user experience testing effectively, a structured approach will be employed. This will include selecting a diverse group of drivers, defining testing parameters, and establishing a framework for gathering feedback.

- **Participant Selection**: A diverse group of drivers, encompassing various demographics (age, gender, driving experience, etc.), will be recruited for testing. This diversity ensures that feedback reflects a broad range of preferences and expectations, allowing for a more comprehensive understanding of user experiences.
- **Driving Scenarios**: Participants will be asked to drive the vehicle under different conditions, including acceleration, braking, and cruising at various speeds. This variety ensures that the sound synthesis system is evaluated across a spectrum of driving experiences, capturing how well it performs in real-world scenarios.

- **Evaluation Criteria**: During the test, participants will evaluate the sounds based on several criteria, including:
  - **Realism**: How closely the synthesized sounds mimic traditional engine sounds.
  - **Satisfaction**: Overall enjoyment of the audio output.
  - **Clarity**: How clearly the sounds convey the vehicle's performance characteristics.
  - **Distraction**: Whether the sounds are intrusive or distracting during driving.
  - **Emotional Impact**: The extent to which the sounds evoke emotional responses, such as excitement or comfort.

#### 3.3.1.8 Iterative Refinement:

The findings from user experience testing will be used to inform iterative refinements to the sound synthesis system. Feedback will drive adjustments to the sound profiles, ensuring that they align more closely with user preferences.

- **Sound Profile Adjustments**: Based on user feedback, sound profiles may be fine-tuned to enhance realism or address any dissatisfaction. This could involve adjusting sound parameters, applying different synthesis algorithms, or even redesigning the audio output strategy to better reflect user preferences.
- **Ongoing User Feedback Integration**: As the system evolves, incorporating ongoing user feedback will be essential to maintain alignment with driver expectations. Regular user experience assessments should be planned as part of the system lifecycle to ensure continuous improvement and adaptability to changing driver preferences.

## 3.3.2 Environmental Testing

Test the system in different environmental conditions (temperature, humidity) to validate the system's robustness in real-world driving scenarios. Environmental testing is a critical phase in the development of the sound synthesis system for EVs. This process aims to evaluate the system's robustness and performance across a variety of real-world conditions, ensuring it operates effectively in diverse environmental scenarios. Factors such as temperature, humidity, vibration, and external noise can significantly impact the performance of both the hardware and the sound synthesis algorithms. Thus, thorough environmental testing is essential to confirm that the system can maintain its functionality and audio quality in challenging conditions.

### 3.3.2.1 Objective of Environmental Testing

The primary goal of environmental testing is to ensure that the sound synthesis system remains reliable and consistent across different operating conditions. This involves assessing how environmental factors affect the performance of the SPC58EC80-DISP board, the miniPC, and the synthesized sounds.

- **System Robustness**: The system must demonstrate resilience to extreme environmental conditions, including high and low temperatures, varying humidity levels, and exposure to potential contaminants (like dust or moisture). A robust system will continue to function accurately and produce high-quality sound regardless of external conditions, which is critical for real-world application.

- **Audio Consistency**: Environmental conditions can alter sound propagation and perception. For example, changes in temperature or humidity can affect the acoustics inside the vehicle cabin, leading to variations in sound quality. Ensuring audio consistency under these varying conditions is essential for delivering a reliable driving experience.

### 3.3.2.2 Testing Parameters

To effectively conduct environmental testing, several key parameters will be evaluated:

- **Temperature Variations**: The system will be tested under a range of temperatures, from extreme cold (e.g., -20°C) to extreme heat (e.g., +60°C). The goal is to evaluate how these temperature extremes affect the electronics of the SPC58EC80-DISP board and miniPC, as well as the synthesis algorithms used for sound generation.

  - **Cold Conditions**: Testing in cold conditions helps identify potential issues such as slow boot times, reduced processing speeds, or audio output anomalies. It's essential to ensure that the system remains responsive and that the sound generation algorithms can handle low-temperature operation without degradation in performance.

    - **Hot Conditions**: High temperatures can lead to overheating of electronic components, which might result in thermal throttling or system failure. The testing will assess whether the system can sustain its performance without overheating and whether the sound output remains consistent during prolonged exposure to high temperatures.

- **Humidity Levels**: Humidity testing will examine how varying levels of moisture in the air (from low to high humidity environments) affect the performance of the system. High humidity can lead to condensation and potential short-circuiting of electronic components, while low humidity can result in static electricity build-up.

    - **Moisture Resistance**: The system should be assessed for its ability to withstand high humidity without compromising the electronic circuitry or sound generation quality. Protective measures such as enclosures or coatings may need to be tested for effectiveness in preventing moisture damage.

#### 3.3.2.3 Vibration Testing

As vehicles encounter various driving conditions, vibrations are an inherent part of the experience. Environmental testing should include assessments of how the system performs under different vibration frequencies and amplitudes, simulating real-world driving scenarios.

- **Simulated Driving Conditions**: The system should be subjected to vibration testing on simulators that replicate the various road conditions a vehicle may encounter, including smooth highways and rough terrains. This testing will help identify any potential issues related to the stability of connections, component reliability, or audio output quality during vibrations.

- **Long-term Durability**: Prolonged exposure to vibrations may cause components to loosen or degrade over time. Testing will evaluate the durability of the connections and components under continuous vibrations to ensure long-term reliability and consistent sound performance.

#### 3.3.2.4 External Noise Considerations

In addition to testing how the system generates sounds in different environmental conditions, it's crucial to evaluate how external noise impacts the perceived quality of the synthesized engine sounds.

- **Ambient Noise Levels**: Environmental testing will assess how various ambient noise levels (e.g., city traffic, highway sounds, and background chatter) affect the clarity and perception of the synthesized engine sounds. The sound synthesis system should be able to maintain audio quality and clarity in the presence of these external noises.

- **Adaptive Sound Generation**: The system could also incorporate algorithms that adapt the generated sounds based on the detected external noise levels. Testing should evaluate how effectively these adaptive algorithms function in real-world conditions, enhancing the overall driving experience.

#### 3.3.2.5 Data Collection and Analysis

Throughout the environmental testing process, systematic data collection and analysis are essential to draw meaningful conclusions regarding system performance.

- **Performance Metrics**: Metrics such as sound clarity, system response times, and temperature thresholds will be recorded during tests. This data will be crucial for identifying trends and potential areas for improvement.
- **Failure Mode Analysis**: Any failures or performance issues encountered during testing will be analyzed to understand their causes. This analysis will inform necessary design adjustments or enhancements to improve the system's resilience to environmental challenges.

#### 3.3.2.6 Iterative Improvements

The results of environmental testing will lead to iterative improvements in the sound synthesis system. Based on identified weaknesses or failures, the design can be refined to enhance robustness and audio quality.

- **Component Selection**: Choosing components that are more resilient to temperature fluctuations, humidity, and vibrations may be necessary based on testing outcomes. Improved materials or protective measures can help ensure that the system functions optimally under real-world conditions.
- **Design Modifications**: The overall system design may be modified based on testing results to enhance durability and performance. This could involve redesigning enclosures, improving heat dissipation methods, or implementing more effective noise-canceling technologies to maintain sound clarity.

### 3.3.3 Iterative Refinement

Analyze feedback from the testing phases and refine the system to address any issues related to sound quality, responsiveness, and integration with the EV's systems. Iterative refinement is a crucial phase in the development of the sound synthesis system for electric vehicles. This process involves a continuous cycle of analyzing feedback from various testing phases, identifying areas for improvement, and implementing changes to enhance the system's performance. The ultimate goal is to ensure that the synthesized sounds are of high quality, responsive to real-time data, and seamlessly integrated with the EV's existing electronic systems.

#### 3.3.3.1 Feedback Analysis

The iterative refinement process begins with a thorough analysis of the feedback collected during functional, user experience, and environmental testing phases.

- **Data Synthesis**: Feedback will be gathered from multiple sources, including user surveys, performance metrics, and observational data from test drives. This information will be synthesized to identify common themes, strengths, and weaknesses of the system. For instance, if multiple users report dissatisfaction with the realism of the engine sounds, it signals a critical area for improvement.
- **Prioritization of Issues**: Once feedback is analyzed, it’s essential to prioritize the identified issues based on their impact on user experience and system performance. Factors to consider include the frequency of the issue (how many users reported it), its severity (how much it affects the driving experience), and the feasibility of implementing a solution.

#### 3.3.3.2 Sound Quality Enhancement

Improving sound quality is a fundamental focus of the iterative refinement process. This may involve adjustments to the sound synthesis algorithms or the implementation of new audio processing techniques.

- **Algorithm Optimization**: Based on feedback regarding sound realism and clarity, the sound synthesis algorithms may need optimization. This could involve fine-tuning parameters such as frequency response, modulation depth, and attack/decay times to create more dynamic and lifelike engine sounds. Advanced techniques, such as granular synthesis or physical modeling, may also be explored to enhance sound realism.
- **Sound Profile Diversity**: Users may express a desire for more diverse sound profiles to match different driving scenarios (e.g., sport mode vs. eco mode). Iterative refinement will involve creating and testing additional sound profiles, ensuring that users have options that enhance their driving experience.

#### 3.3.3.3 Responsiveness Improvement

Responsiveness refers to the system's ability to accurately reflect changes in vehicle performance in real-time. Ensuring that the sound output is timely and reflective of the vehicle's current state is vital for a satisfying driving experience.

- **Latency Reduction**: Feedback regarding any noticeable delays between data input and sound output will be addressed by optimizing the data acquisition and processing pipeline. This may involve refining the firmware on the SPC58EC80-DISP board or optimizing the communication protocols to minimize latency.
- **Dynamic Sound Adjustment**: The system can be enhanced to allow for more rapid adjustments to the sound output based on real-time data. For instance, implementing adaptive algorithms that can respond to sudden changes in throttle position or speed will make the sound experience more engaging and realistic.

#### 3.3.3.4 Integration Optimization

A critical aspect of the iterative refinement process is ensuring that the sound synthesis system integrates seamlessly with the EV's existing electronic systems.

- **Interfacing Improvements**: Analyzing feedback related to communication issues between the SPC58EC80-DISP board and the miniPC will help identify any inefficiencies in interfacing. Improvements may include refining the communication protocols or optimizing data formats to ensure smooth and reliable data exchange.
- **System Compatibility Testing**: Ongoing testing will ensure that changes made during the refinement process do not negatively impact the performance of other electronic systems within the vehicle. Compatibility with various components, such as the vehicle's battery management system and other sensors, must be assessed to prevent conflicts and ensure overall system integrity.

#### 3.3.3.5 Continuous Testing and Validation

The iterative refinement process is cyclical, meaning that after implementing changes, the system must undergo further testing to validate improvements.

- **Validation Testing**: After refining the system based on feedback, additional rounds of functional and user experience testing will be conducted to ensure that the changes have effectively addressed previous issues. This testing phase is critical for confirming that sound quality, responsiveness, and integration have improved.
- **Documenting Changes**: Keeping detailed documentation of each iteration, including the changes made and their outcomes, is essential for tracking the development process. This documentation can serve as a valuable resource for future iterations and help maintain transparency throughout the development lifecycle.

### 3.3.4 Long-term Reliability

Evaluate system performance over extended testing sessions to ensure long-term reliability, sound consistency, and minimal latency in response to vehicle changes. Long-term reliability is a critical aspect of evaluating the sound synthesis system for electric vehicles. This phase focuses on assessing how well the system performs over extended periods, ensuring that it consistently delivers high-quality sound and responds accurately to vehicle changes throughout its operational lifespan. Ensuring long-term reliability is essential not only for user satisfaction but also for the overall safety and functionality of the vehicle.

#### 3.3.4.1 Objective of Long-term Reliability Testing

The primary objective of long-term reliability testing is to confirm that the sound synthesis system can maintain performance under continuous use and over varying operating conditions. This includes assessing sound consistency, system durability, and responsiveness to real-time vehicle data.

- **Sound Consistency**: The system should produce a consistent audio output over time, regardless of environmental fluctuations or wear on electronic components. Users expect the synthesized sounds to remain stable in quality, resembling realistic engine sounds throughout the lifespan of the vehicle.

- **Minimizing Latency**: As the system operates over extended periods, it’s essential to ensure that latency remains minimal. Increased latency can degrade the user experience and diminish the system's effectiveness in conveying real-time performance changes. The goal is to guarantee that the system remains responsive, with sound adjustments occurring instantly as vehicle conditions change.

#### 3.3.4.2 Testing Methodology

Long-term reliability testing will employ a structured methodology to ensure thorough evaluation.

- **Extended Testing Sessions**: The system will undergo extensive testing sessions that simulate prolonged use, incorporating various driving scenarios, ambient conditions, and user interactions. This approach helps identify how the system performs over time and under diverse conditions.

- **Continuous Monitoring**: Throughout the testing sessions, continuous monitoring of system parameters, including sound output quality, response times, and component temperatures, will be conducted. This data will provide insights into how the system holds up during extended use and whether any degradation occurs.

#### 3.3.4.3 Durability Assessment

Evaluating the durability of the hardware and software components of the sound synthesis system is crucial for ensuring long-term reliability.

- **Stress Testing**: The system will undergo stress testing to assess its ability to handle extreme conditions, such as high temperatures, humidity, and vibrations. These tests simulate potential real-world scenarios, allowing the identification of weaknesses in the system design that could lead to failure over time.

- **Component Lifespan Analysis**: Specific components, such as microphones, speakers, and circuit boards, will be evaluated for their lifespan and susceptibility to wear and tear. Understanding the durability of each component is essential for planning maintenance and replacements in the long-term use of the system.

#### 3.3.4.4 User Interaction and Feedback

Long-term reliability testing should also consider user interaction and feedback over time.

- **User Experience Monitoring**: Participants will be asked to provide feedback on their experiences with the sound synthesis system over extended use. This qualitative data can help identify any emerging issues or concerns that may not be apparent in initial testing phases.
- **Regular User Feedback Sessions**: Establishing regular sessions for users to share their experiences will enable continuous improvement and refinement of the system. Tracking changes in user perception over time can help maintain high satisfaction levels.

#### 3.3.4.5 Iterative Improvements Based on Findings

The insights gained from long-term reliability testing will drive iterative improvements to the sound synthesis system.

- **System Adjustments**: Based on testing outcomes, adjustments to the system may be necessary to enhance durability, sound consistency, and responsiveness. This could involve upgrading components, refining software algorithms, or implementing better protective measures against environmental factors.
- **Maintenance Protocols**: Establishing maintenance protocols based on the findings from long-term reliability testing will help ensure the system continues to perform optimally throughout its lifespan. This may include recommendations for routine checks, component replacements, or software updates.

# 4 Hardware Overview

The SPC58EC-DISP discovery plus board is a development platform designed to assess and create applications using the SPC58EC80E5 microcontroller, which comes in an eTQFP144 package. The SPC58EC80E5 microcontroller is intended for use in body, networking, and security-related applications.

The SPC58EC-DISP board is equipped with various interfaces such as an Ethernet controller, CAN-FD, FlexRay, LIN, UART, ADC, and a JTAG port, making it an ideal tool for evaluating the microcontroller and developing and debugging applications. It includes an integrated programmer and debugger (PLS-supported) for programming and debugging the microcontroller, with a section dedicated to enabling a USB virtual COM port. The board features connectors compatible with Arduino UNO R3 shields, simplifying functionality expansion by adding compatible shields or boards. Additionally, all GPIOs and key signals are accessible via a 4x37 0.1" pin array.

The SPC58EC80E5 is a high-performance automotive microcontroller (MCU) from the SPC58 C line of STMicroelectronics. It is designed to address complex body, networking, and security functions in automotive applications. The microcontroller is based on 32-bit Power Architecture technology, featuring dual e200z4 cores running at up to 180 MHz. It also includes Hardware Security Module (HSM) support, making it suitable for applications that require both high computational power and robust security features.

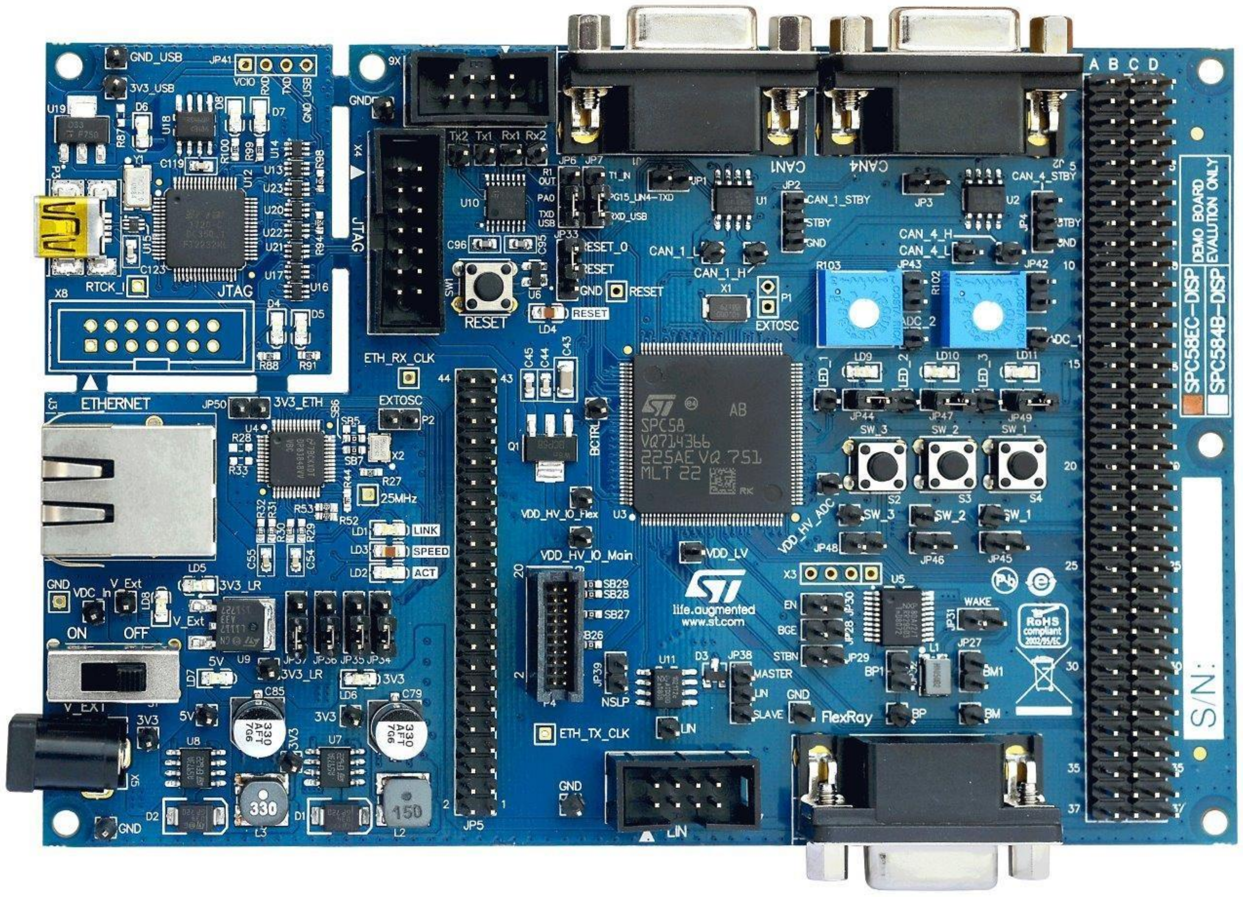


*Figure 4-1 DISP discovery board-top side*

## 4.1 MCU Key Features

### 1. Processor and Architecture

The SPC58EC80E5 utilizes two e200z4 cores for enhanced processing capability. The dual-core architecture allows for parallel processing, which improves the efficiency of tasks like automotive control systems, where real-time processing is critical. Each core operates at up to 180 MHz and supports Variable Length Encoding (VLE), a method for reducing code size by mixing 16-bit and 32-bit instructions, thereby optimizing memory usage.

In addition to the dual-core setup, a third e200z0 core is embedded in the Hardware Security Module (HSM), which is responsible for executing cryptographic operations and handling secure communications. This separation of the security functions from the main cores ensures that security-sensitive tasks are executed in an isolated environment, enhancing overall system security.

**2. Memory and Storage**

The microcontroller provides ample memory resources suitable for automotive applications:

- **4 MB of Flash memory** is dedicated to code storage, along with **128 KB of data Flash** that supports **EEPROM emulation**. The EEPROM emulation feature enables non-volatile data storage, crucial for storing configuration settings and other essential data during vehicle operation.

- **384 KB of general-purpose SRAM** offers sufficient temporary data storage for high-speed operations, and an additional **128 KB of local RAM** is available for each of the two main cores, enabling fast access to frequently used data.

- The **Hardware Security Module (HSM)** includes its own memory, with **176 KB of dedicated Flash memory** (144 KB for code and 32 KB for data). This memory is specifically for secure code and data storage, separate from the main Flash memory.

The Flash memory supports **read-while-write** operations, meaning the system can continue to read from Flash while it is being programmed or erased, which is important for real-time automotive systems where uninterrupted operation is critical.

**3. Communication Interfaces**

The SPC58EC80E5 is equipped with a comprehensive set of communication interfaces, making it ideal for complex automotive networks. These interfaces allow the microcontroller to communicate with other electronic control units in a vehicle:

- **Eight CAN-FD (Controller Area Network with Flexible Data-rate) channels** support high-speed communication, allowing for data rates up to 8 Mbps, which is essential for the fast-growing demands of in-vehicle networking, particularly in advanced driver-assistance systems (ADAS).

- **18 LINFlexD modules** provide communication over the LIN (Local Interconnect Network) and UART (Universal Asynchronous Receiver-Transmitter) protocols. LIN is typically used for lower-speed communications in body control systems, such as lighting, seat controls, and air conditioning.

- A **dual-channel FlexRay controller** enables robust, high-speed, time-deterministic communication, making it a popular choice for safety-critical applications such as braking and steering systems.

- **Ethernet support** at **10/100 Mbps** is compliant with IEEE 1588 for precise time-stamping, making it suitable for automotive Ethernet use cases like diagnostics, camera systems, and over-the-air software updates.

**4. Safety Features**

The SPC58EC80E5 is designed with safety in mind, adhering to **ASIL-B (Automotive Safety Integrity Level)** standards as outlined by the **ISO 26262** functional safety standard. It incorporates several key safety features:

- The **Fault Collection and Control Unit (FCCU)** is responsible for collecting fault information from various modules and initiating the appropriate responses, such as transitioning to a safe state in the event of a critical failure.

- The **Memory Error Management Unit (MEMU)** detects and reports memory errors, ensuring that corrupted data is handled properly. This is critical in automotive applications where memory errors could lead to unsafe conditions.

- The microcontroller includes **Cyclic Redundancy Check (CRC) units**, which verify data integrity to prevent transmission or storage errors. These CRC units are used to ensure that data has not been corrupted in transit or during processing.

- **End-to-end ECC (Error Correction Code)** is implemented across memories and bus communications, providing robust protection against data corruption by detecting and correcting single-bit errors.

**5. Power Management and Low-Power Modes**

In automotive environments, power efficiency is crucial, particularly in systems that operate continuously, even when the vehicle is not in active use. The SPC58EC80E5 provides several low-power modes, including:

- **HALT** and **STOP modes**, which reduce power consumption when certain functionalities are idle.
- **Standby mode** further minimizes power consumption by disabling most of the system except for critical wake-up circuitry.
- An **ultra-low-power standby mode** includes a real-time clock (RTC) and support for a **Smart Wake-up Unit** that monitors input pins to wake the system up based on predefined triggers.

These power-saving features make the microcontroller ideal for applications like body control modules and telematics, where power consumption must be minimized without sacrificing performance.

**6. Enhanced I/O and Peripherals**

The SPC58EC80E5 microcontroller provides a wide range of I/O options through its enhanced modular I/O subsystem (**eMIOS**):

- Up to **64 timed I/O channels** with **16-bit counter resolution** allow for precise pulse-width modulation (PWM) control, which is critical in applications like motor control, lighting, and HVAC systems.
- The microcontroller features **three 12-bit SAR (Successive Approximation Register) analog-to-digital converters (ADCs)** for accurate sensor data acquisition, along with an additional **10-bit SAR ADC** for supervisory functions.
- It also supports **Direct Memory Access (DMA)**, enabling faster data transfers between memory and peripherals without burdening the CPU.

The crossbar switch architecture allows multiple bus masters, such as the cores and peripherals, to access memory simultaneously without bottlenecking, improving the system's throughput.

### Conclusion

The SPC58EC80E5 microcontroller is an advanced, automotive-grade solution for body control, networking, and security applications. Its combination of high-performance dual-core architecture, extensive memory resources, multiple communication interfaces, and robust safety features make it suitable for modern automotive applications, especially in areas that demand real-time performance, data integrity, and security. The integrated Hardware Security Module ensures secure communication and data handling, while the extensive peripheral set, including CAN-FD, FlexRay, LIN, and Ethernet, makes it ideal for in-vehicle networking and control.

This microcontroller also excels in energy efficiency, with various low-power modes that are essential for applications where minimizing energy consumption is critical. Overall, the SPC58EC80E5 provides a scalable, reliable, and secure platform for advanced automotive systems.

Refer to Figure 4-2 for a block diagram of the SPC58EC80E5 and Figure 4-3 for its peripheral layout. These figures will help you visualize the interaction between its cores, memory, communication interfaces, and peripherals.

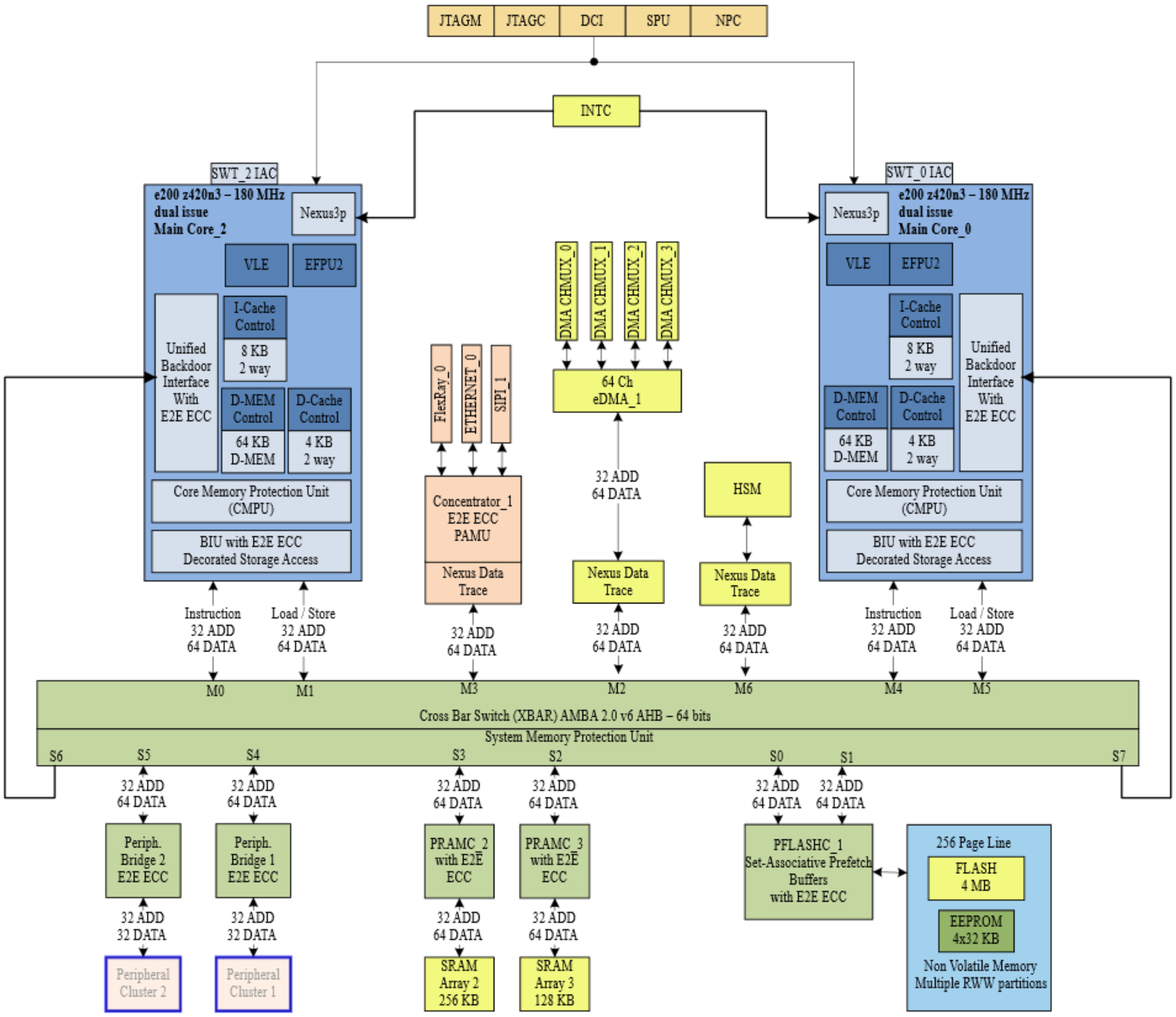
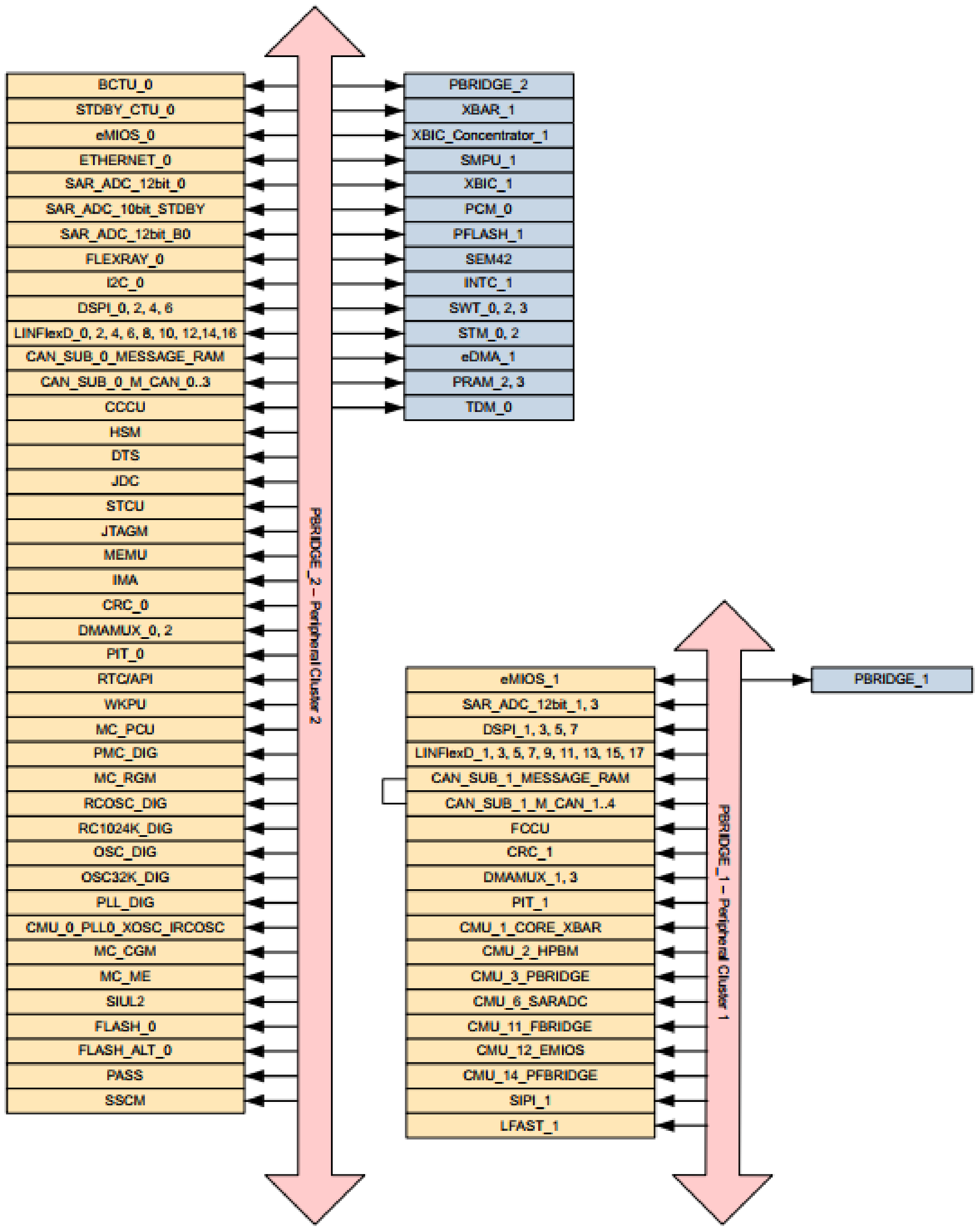


*Figure 4-2 SPC58EC80 Block diagram*



Note: In this diagram, ON-platform modules are shown in orange color and OFF-platform modules are shown in blue color.

*Figure 4-3 Peripheral allocation*

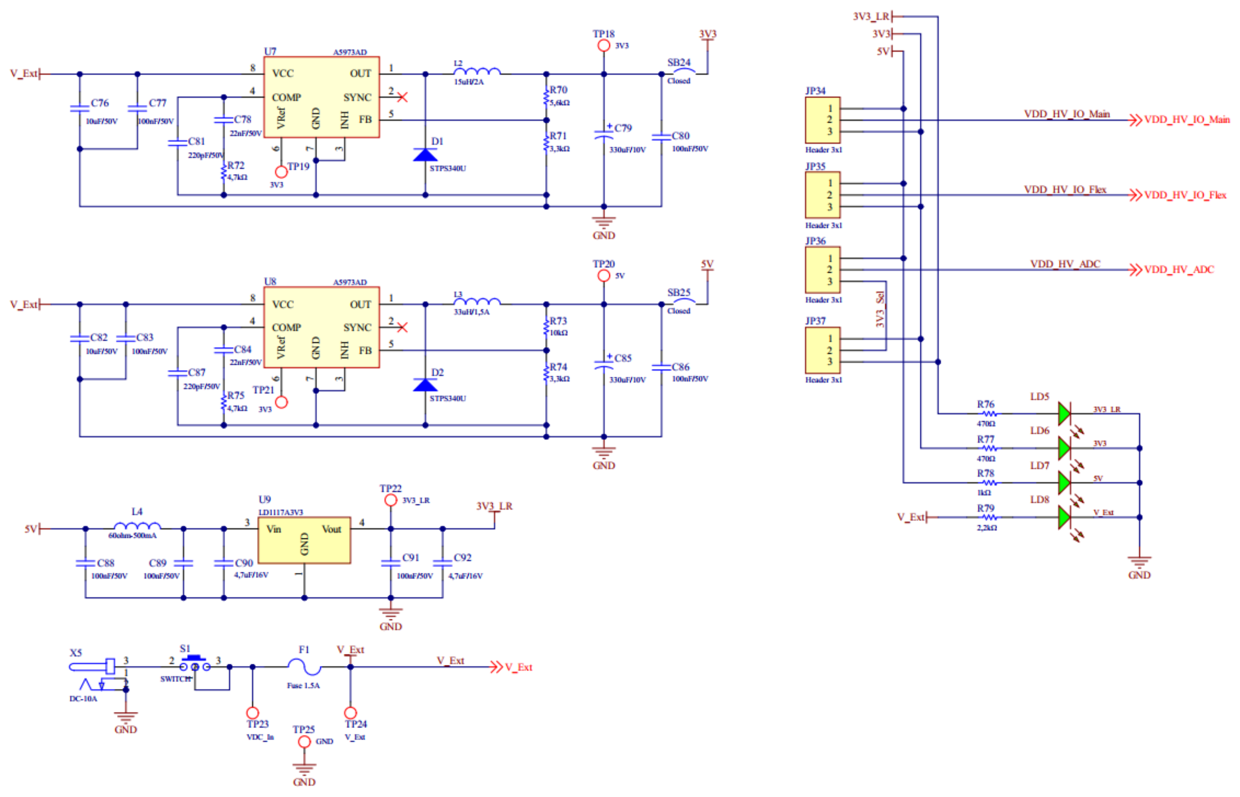


*Figure 4-4 Power supply circuit of the SPC58EC80-DISP board, providing 5V, 3.3V and linear regulated 3.3V*

## 4.2 Power Supply

The power supply circuit for the SPC58EC80-DISP board (Figure 4-4) is designed to provide flexible voltage regulation and distribution, supporting multiple voltage rails and external power input.

- **Primary Power Input and Protection**:
  - The circuit accepts an external DC input (V_Ext), which can be switched on/off and is protected by a fuse. This is the main power source.
- **Voltage Regulators**:
  - **Buck Regulators (U7 and U8)**: These two switching regulators convert V_Ext into stable 3.3V (3V3) and 5V (5V) outputs, which are used to power various parts of the board.
  - **Linear Regulator (U9)**: This LDO regulator provides an additional 3.3V output (3V3_LR) from the 5V rail, offering a low-noise option for components sensitive to power fluctuations.

- **Power Selection Jumpers (JP34 to JP37)**:
  - These headers allow users to select different voltage sources (3.3V, 5V, or V_Ext) for specific power domains, such as VDD_HV_IO_Main, VDD_HV_IO_Flex, and VDD_HV_ADC, providing flexibility in the power configuration.
- **Power Status Indicators**:
  - LEDs indicate the presence of the 3V3_LR, 3V3, 5V, and V_Ext voltages, allowing quick visual confirmation of each rail's status for easier troubleshooting.

The design provides regulated 3.3V and 5V outputs from an external DC source, with configurable power distribution and LED indicators for monitoring. It combines the benefits of efficient switching regulators and a low-noise linear regulator to meet the diverse power requirements of the board.

## 4.3 CAN and ISO CAN-FD

Figure 4-5 shows the CAN interfaces on the SPC58EC80-DISP board, which includes two independent CAN transceivers. Each section of the circuit handles communication on a separate CAN bus, allowing the board to interface with multiple CAN networks. Here’s a breakdown of the components and their functions:

### 4.3.1 CAN Transceivers (U1 and U2)

- MCP2562FD transceivers are used for both CAN channels. These transceivers support CAN FD (Flexible Data-rate), which allows for higher data transmission rates compared to standard CAN.
- **Pins**:
  - **TXD/RXD**: These pins connect to the microcontroller’s CAN TX and RX lines.
  - **CANH/CANL**: The differential high and low output pins are used for the CAN bus. These connect to the CAN bus lines and are responsible for transmitting and receiving differential signals.
  - **STBY (Standby)**: This pin controls whether the transceiver is in normal operation or standby mode, allowing power saving when the CAN interface is not in use. The standby pin can be grounded or pulled high through headers JP2 and JP4, making it configurable.

### 4.3.2 Bus Termination and Filtering

- **Termination Resistors (R5/R6 and R12/R13)**: Each CAN bus is terminated with 60.4Ω resistors to match the characteristic impedance of the CAN bus, which is typically 120Ω (parallel resistance of the two 60.4Ω resistors). This termination is necessary to prevent signal reflections on the bus.
- **Optional Jumpers (JP1 and JP3)**: These jumpers allow you to enable or disable termination on each CAN line, providing flexibility depending on whether the board is used as a single node or an endpoint on the bus.
- **Capacitors (C1, C2, C7, and C8)**: These 47pF capacitors are used for filtering high-frequency noise, helping to maintain signal integrity on the CANH and CANL lines.
- **Additional Capacitors (C5, C6, C11, and C12)**: Extra filtering capacitors, 4.7nF, connected to ground to further filter noise on the CAN lines.

### 4.3.3 Test Points (TP1, TP2, TP3, and TP4)

- These test points provide access to the CANH and CANL signals for both CAN buses. They can be used to monitor the differential CAN signals with an oscilloscope or other test equipment during debugging.

**D-Sub Connectors (J1 and J2)**

- **CAN Bus Output**: Both CAN networks are accessible through separate D-Sub 9-way connectors (J1 and J2), which are commonly used for CAN connections in industrial and automotive applications.
- **Pin Mapping**: The CANH and CANL lines are connected to the standard CAN pins on the D-Sub connectors. Additionally, external power (V_Ext) is also routed to these connectors, potentially for powering external CAN devices.

This CAN interface section enables the SPC58EC80-DISP board to communicate over two independent CAN channels. Each channel has its own transceiver (U1 and U2), with noise filtering and configurable termination to ensure robust communication. The D-Sub connectors provide a standard interface for connecting the board to external CAN networks, making this setup suitable for automotive and industrial CAN applications.

This design allows for flexibility in CAN configuration, with features like standby control, selectable termination, and accessible test points, making it versatile for a variety of CAN-based communication tasks.

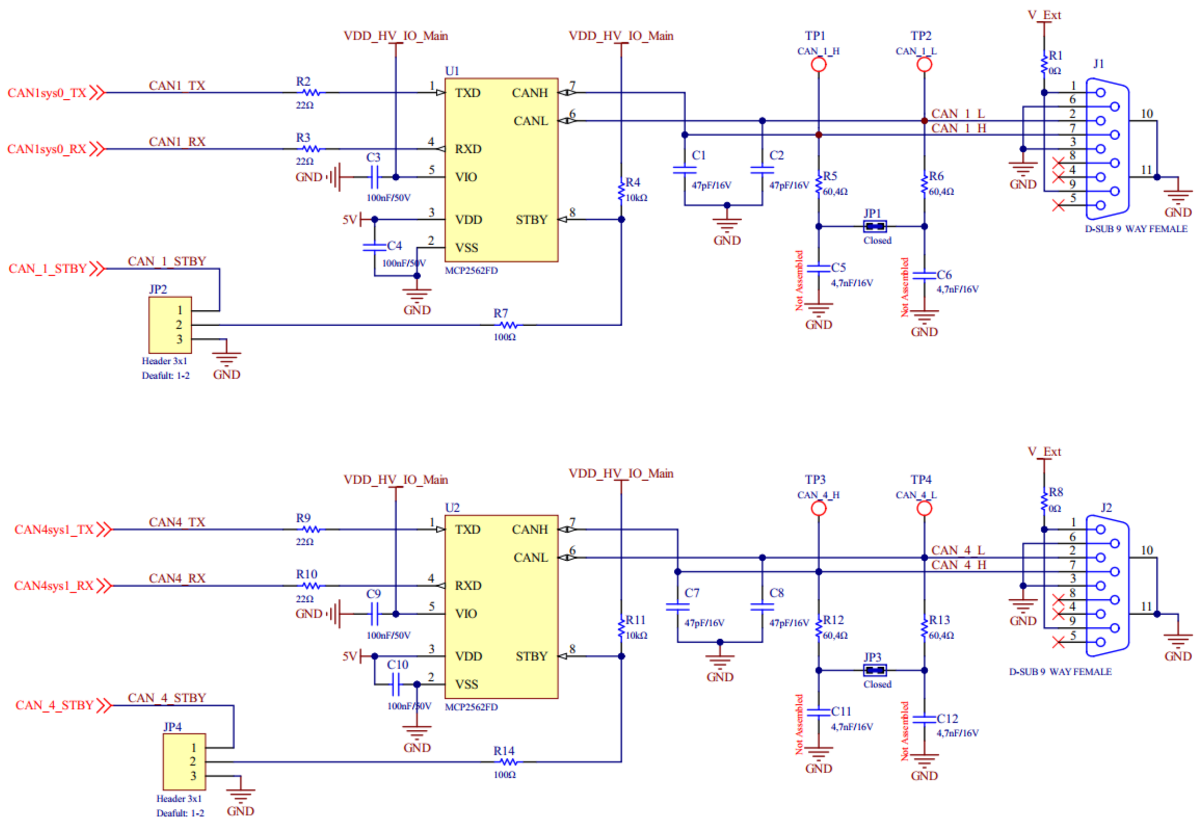


*Figure 4-5 CAN and ISO CAN-FD interface circuit of the board is exploiting MCP2562FD (High-Speed CAN Flexible Data Rate Transceiver)*

# 5 Software Overview

## 5.1 SPC5Studio Overview

SPC5Studio is an Eclipse-based development environment tailored for SPC5x Power Architecture 32-bit microcontrollers. It provides an extensible platform where users can create embedded applications by utilizing pre-built components and developing custom plug-ins. The generated code is ANSI C compliant, adhering to MISRA 2012 standards for high-quality, reliable software.

SPC5Studio offers an intuitive user interface, simplifying the creation of embedded applications by enabling easy configuration of MCU resources. Its application wizard automatically manages dependencies, helping developers generate syntax-error-free projects from the beginning.

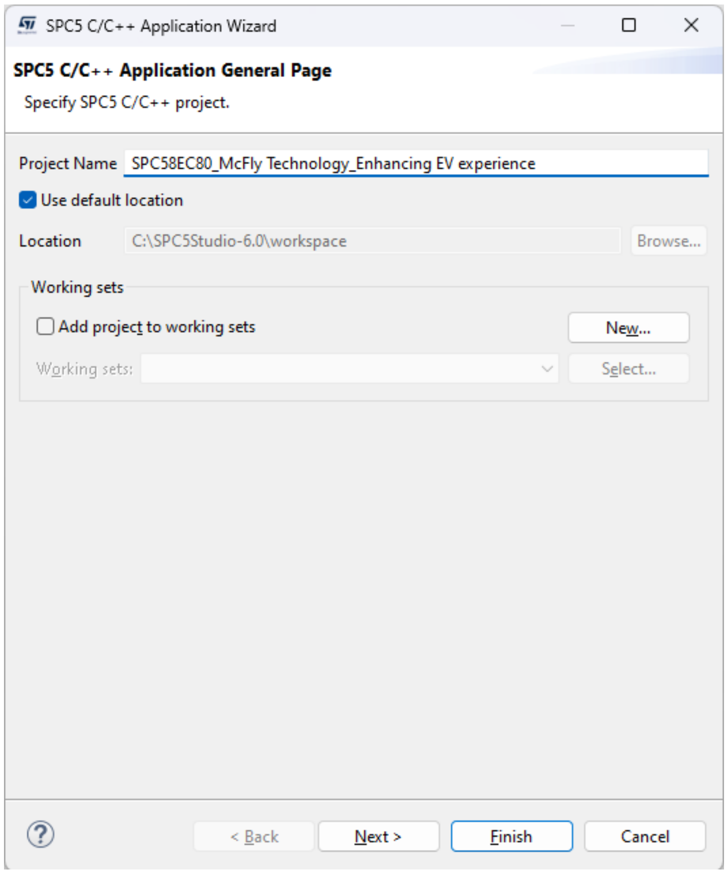
The tool is free to download from ST's website, with updates provided via an automatic notification system. Further customization and increased functionality are available through a marketplace filled with installable components.

### 5.1.1 Creating a New Application

- Project name

In this step, you begin by creating a new project within SPC5Studio, which involves assigning a unique name to the project. The project name serves as the primary identifier and helps differentiate it from other projects you may have in your workspace. Choosing a descriptive name is especially helpful when managing multiple projects, as it allows you to keep track of different versions or types of embedded applications you're developing. This initial naming step is fundamental to keeping your work organized and ensuring efficient project management throughout the development process.

Additionally, the project name will be used to create the directory structure where all the related files—such as configuration, source code, and build outputs—are stored. This means that having a clear and unique project name can also help avoid confusion and conflicts between different projects.


SPC5 C/C++ Application Wizard
SPC5 C/C++ Application General Page
Specify SPC5 C/C++ project.
Browse...
Working sets
New...
Select...
< Back
Next >
Finish
Cancel

Project Name SPC58EC80_McFly Technology_Enhancing EV experience

[✓] Use default location

Location C:\SPC5Studio-6.0\workspace

[ ] Add project to working sets

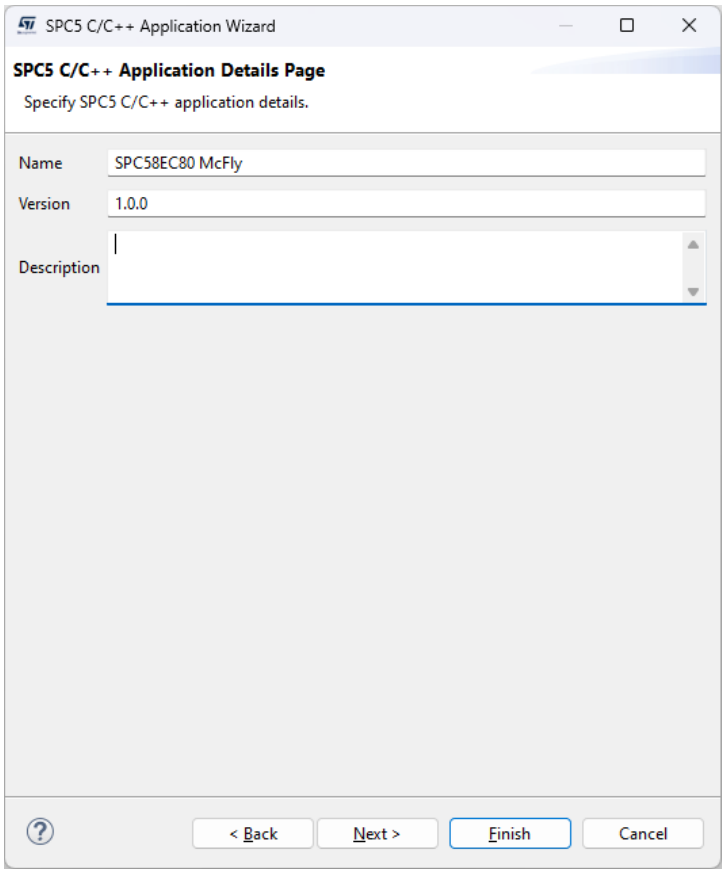
Working sets:

*Figure 5-1 Project name window in which should define a name for the project*

- Application name

After setting a project name, you need to define an application name within SPC5Studio. The application name is a crucial identifier that represents the specific embedded software or functionality you are building as part of your project. It serves as a secondary layer of identification within your project—while the project name might refer to the overarching task, the application name indicates a specific component or purpose of the software being developed.

The application name helps maintain clarity and focus as it points directly to what your code is intended to accomplish. It may also be reflected in the directory and file structures, making it easier to navigate and understand the different components of the project later on. Having an appropriate application name is essential when working in teams or revisiting the project after some time, as it provides quick insight into the purpose and scope of the specific application you are developing.


SPC5 C/C++ Application Wizard

**SPC5 C/C++ Application Details Page**

Specify SPC5 C/C++ application details.

Name: SPC58EC80 McFly

Version: 1.0.0

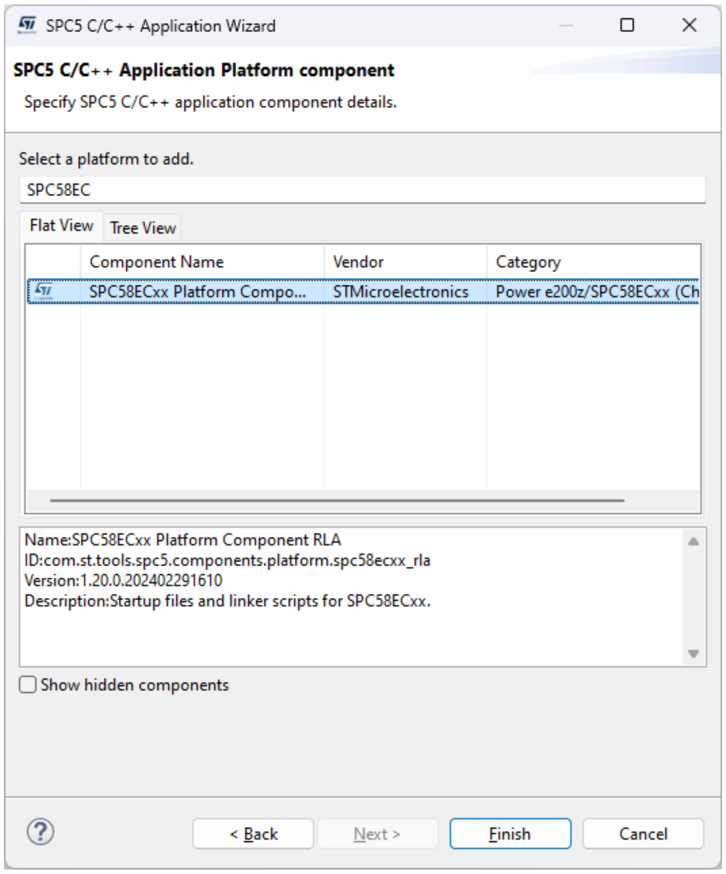
Description:

*Figure 5-2 Application name window where we can define an application name and a brief description*

- Select platform

In this step, you specify the target hardware platform for your embedded application. Essentially, you are telling SPC5Studio which specific SPC5 microcontroller (MCU) or family of microcontrollers you plan to use. The platform selection includes choosing a particular microcontroller that meets your project's requirements in terms of performance, available peripherals, and memory size.

By carefully selecting the platform and defining the specific details about your target MCU, you make sure that all subsequent steps, like component selection and code generation, are aligned with the hardware you are targeting. This alignment reduces the risk of incompatibility, saves development time, and ensures that the generated code will run optimally on your target microcontroller.


SPC5 C/C++ Application Wizard
SPC5 C/C++ Application Platform component
Specify SPC5 C/C++ application component details.
Select a platform to add.
Flat View
Tree View
Show hidden components
< Back
Next >
Finish
Cancel

SPC58EC

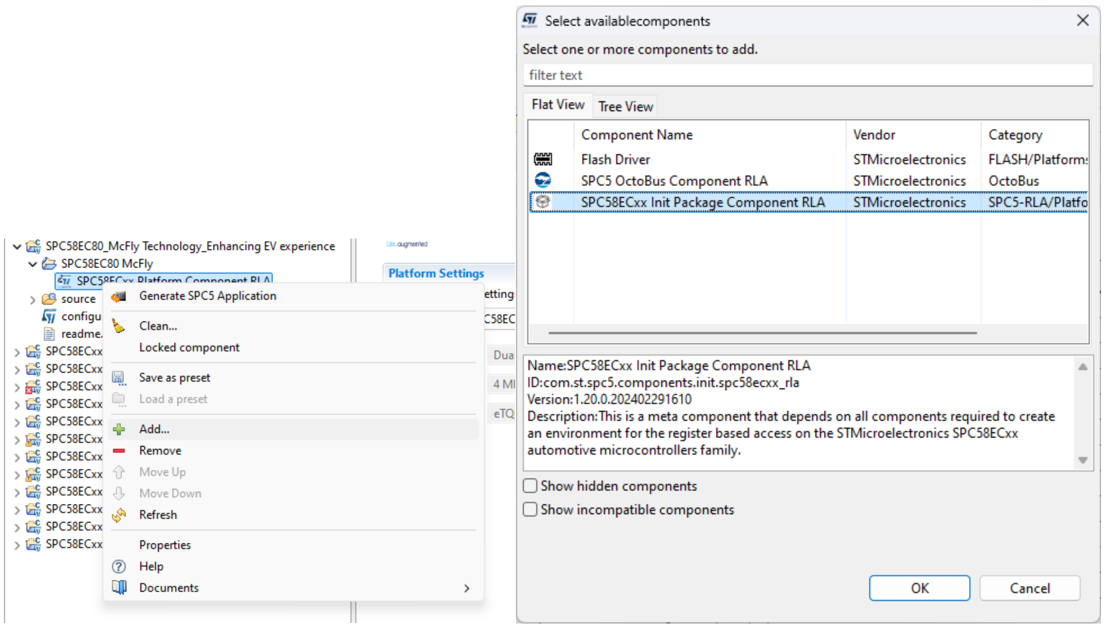
| | Component Name | Vendor | Category |
|---|---|---|---|
| | SPC58ECxx Platform Compo... | STMicroelectronics | Power e200z/SPC58ECxx (Ch |

Name:SPC58ECxx Platform Component RLA
ID:com.st.tools.spc5.components.platform.spc58ecxx_rla
Version:1.20.0.202402291610
Description:Startup files and linker scripts for SPC58ECxx.

*Figure 5-3 Select platform window is used to specify the target hardware platform e.g. SPC58ECxx in this project*

### 5.1.2 Add Components to the Project

This step is about selecting and configuring the building blocks of your embedded application. It allows you to add the drivers, libraries, middleware, and other software modules needed to make your project work. The ease of use provided by the graphical interface, combined with automated compatibility checks, makes this step efficient and minimizes the risk of integration errors. This approach gives you flexibility and ensures that your project is both scalable and maintainable.



| Component Name | Vendor | Category |
|---|---|---|
| Flash Driver | STMicroelectronics | FLASH/Platform: |
| SPC5 OctoBus Component RLA | STMicroelectronics | OctoBus |
| SPC58ECxx Init Package Component RLA | STMicroelectronics | SPC5-RLA/Platfo |

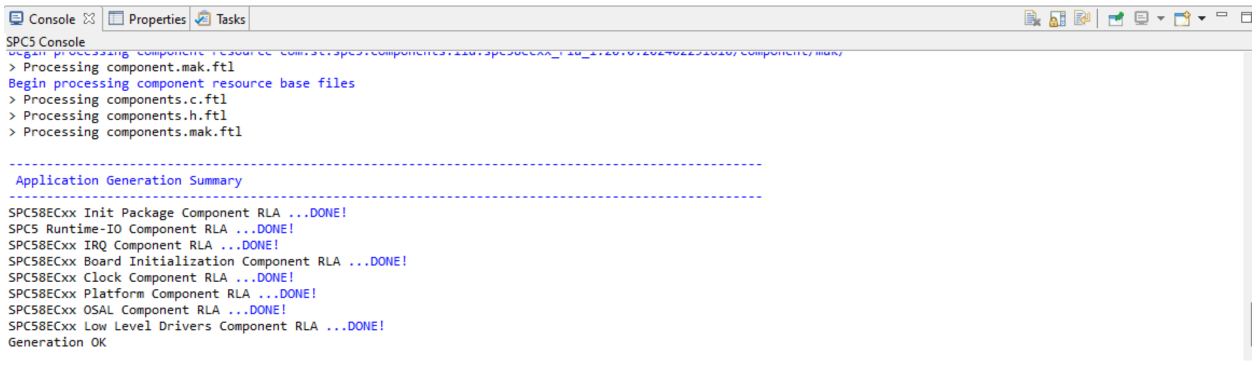
Name:SPC58ECxx Init Package Component RLA
ID:com.st.spc5.components.init.spc58ecxx_rla
Version:1.20.0.202402291610
Description:This is a meta component that depends on all components required to create an environment for the register based access on the STMicroelectronics SPC58ECxx automotive microcontrollers family.

*Figure 5-4 Add components to project*

### 5.1.3 Generate Application Code

In this step, SPC5Studio takes all of the configurations you have specified so far—including the platform selection, added components, and resource configurations—and automatically generates the necessary source code for your embedded application. This step is one of the key advantages of using an integrated development environment like SPC5Studio, as it simplifies what can otherwise be a labor-intensive and error-prone task.

The "Generate Application Code" step in SPC5Studio is a powerful feature that automatically creates a basic code framework for your embedded system, including peripheral drivers, middleware setup, and a structured main application file. By ensuring MISRA 2012 compliance, managing dependencies, and organizing code in a clear and modular way, SPC5Studio accelerates the development process and helps developers produce reliable, maintainable, and ready-to-deploy embedded software for SPC5 microcontrollers.



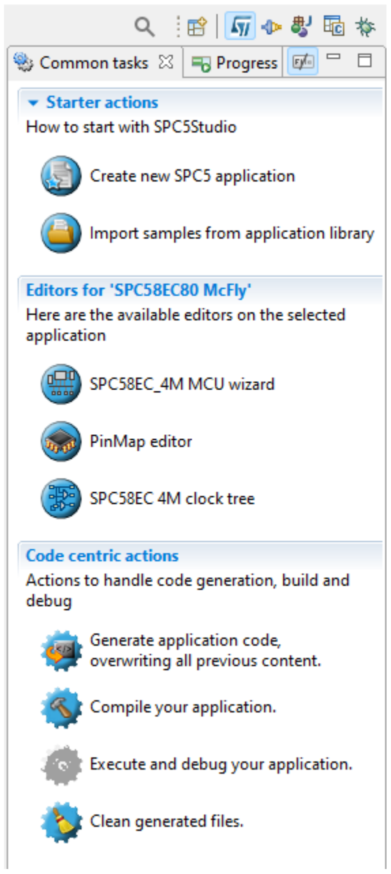
```
[illegible]
> Processing component.mak.ftl
Begin processing component resource base files
> Processing components.c.ftl
> Processing components.h.ftl
> Processing components.mak.ftl

----------------------------------------------------------------------------------------------------
 Application Generation Summary
----------------------------------------------------------------------------------------------------
SPC58ECxx Init Package Component RLA ...DONE!
SPC5 Runtime-IO Component RLA ...DONE!
SPC58ECxx IRQ Component RLA ...DONE!
SPC58ECxx Board Initialization Component RLA ...DONE!
SPC58ECxx Clock Component RLA ...DONE!
SPC58ECxx Platform Component RLA ...DONE!
SPC58ECxx OSAL Component RLA ...DONE!
SPC58ECxx Low Level Drivers Component RLA ...DONE!
Generation OK
```

*Figure 5-5 Application generation summary shows any error or warning if we did something wrong*

### 5.1.4 Compile Your Application

Once you've configured your project and generated the application code, the next step is to compile that code to produce an executable that can be uploaded and executed on the microcontroller. Compiling involves several stages that transform your source code into a machine-readable format, while also verifying that your code is error-free.

The "Compile Application" step is about transforming your human-readable source code into machine code that the SPC5 microcontroller can execute. It involves preprocessing, compiling into object files, linking them to create an executable, and generating a binary that can be flashed onto the microcontroller. This step also includes error-checking, optimizing the code for performance or memory usage, and preparing the final output for deployment. The compilation process ensures that your embedded system code is both syntactically correct and appropriately configured to run reliably on your target hardware.


Common tasks
Progress

Starter actions
How to start with SPC5Studio
Create new SPC5 application
Import samples from application library

Editors for 'SPC58EC80 McFly'
Here are the available editors on the selected application
SPC58EC_4M MCU wizard
PinMap editor
SPC58EC 4M clock tree

Code centric actions
Actions to handle code generation, build and debug
Generate application code, overwriting all previous content.
Compile your application.
Execute and debug your application.
Clean generated files.

*Figure 5-6 Compile application*

## 5.2 UDE STK Overview

UDE STK 5.2, or Universal Debug Engine Starter Kit version 5.2, is a comprehensive debugger software tool used primarily for debugging and testing embedded systems. Developed by PLS Development Tools, UDE STK is designed to interface with microcontrollers and System-on-Chip (SoC) devices, providing developers with in-depth access to internal resources of the microcontroller during the software development process. Below is an overview of its features, functionalities, and application areas:

### 5.2.1 Key Features and Overview of UDE STK 5.2

- **Integrated Development and Debug Environment**:
  - UDE STK 5.2 is an integrated debugger designed to work with multiple development environments and supports various microcontroller families. It provides an easy-to-use GUI with features that streamline the debugging process.
  - It is particularly useful for embedded developers working with complex systems, as it integrates seamlessly with compilers and RTOS.
- **Support for Multiple Architectures**:
  - UDE STK 5.2 supports a wide range of 32-bit microcontroller architectures. It is commonly used with automotive microcontrollers, such as those based on the Power Architecture (like SPC5 series), ARM Cortex cores, and other popular microcontroller families.
  - This makes it a suitable choice for automotive applications, industrial automation, and consumer electronics that use high-performance microcontrollers.
- **Real-Time Debugging Capabilities**:
  - UDE STK 5.2 allows developers to observe the behavior of an embedded system in real-time without halting the target microcontroller, which is critical for real-time applications.
  - It provides live access to registers, variables, and memory, enabling developers to monitor system status as it executes. This non-intrusive debugging capability is particularly useful in safety-critical systems where continuous monitoring is essential.
- **Graphical User Interface**:
  - The UDE software interface provides multiple windows that help visualize and analyze data in an embedded system. These windows include memory views, watch windows, variable views, call stacks, and peripheral register access views.
  - The GUI is customizable, allowing developers to arrange the windows to best suit their workflow and easily monitor critical variables and hardware status.

- **Powerful Debugging Features**:
  - **Breakpoint Management**: UDE STK 5.2 supports both software and hardware breakpoints. Breakpoints help developers pause the execution of the application at a specific line of code or address in memory, allowing for detailed inspection of the system state.
  - **Watchpoints**: Conditional breakpoints or watchpoints can be set to pause execution only when a specific condition is met (e.g., a variable changes to a certain value).
  - **Step-by-Step Execution**: The tool allows single-stepping through the code—line by line or function by function—making it easy to locate the exact point where issues occur.
- **Trace Functionality**:
  - **ETM and DWT Trace**: For advanced debugging, UDE STK 5.2 supports trace functions, such as ETM (Embedded Trace Macrocell) and DWT (Data Watchpoint and Trace). These functions allow developers to trace the execution history of the program, providing detailed insights into how the program reached its current state.
  - **Code Coverage and Profiling**: The trace features also assist in code coverage analysis, which is essential in determining how thoroughly a program has been tested. This is crucial for validating software, especially in automotive and other safety-critical applications.
- **Scripting and Automation**:
  - UDE STK 5.2 includes scripting capabilities, allowing developers to automate repetitive debugging tasks. Scripts can be written using Python or other supported languages to control the debugger programmatically.
  - This functionality helps save time and ensures consistency when debugging similar issues across multiple devices or projects. Automated scripts can also be used in production testing environments.
- **Multi-Core and Multi-Target Debugging**:
  - The tool is capable of debugging systems with multiple cores or multiple microcontrollers. This is particularly important for automotive systems where multiple processors work together in a distributed system architecture.
  - UDE STK 5.2 manages the debugging of multiple cores, allowing synchronized execution control, simultaneous variable monitoring, and coordinated breakpoint handling across all cores in the system.

- **RTOS Awareness**:
  - UDE STK 5.2 is designed to work effectively with real-time operating systems. It provides RTOS-aware debugging, meaning it can show information about tasks, threads, queues, and other RTOS objects, which helps in diagnosing issues in RTOS-based applications.
  - RTOS task monitoring allows developers to see which tasks are running, their states, priorities, and stack usage, which is valuable for optimizing performance and troubleshooting timing issues.
- **Flash Programming and Diagnostics**:
  - UDE STK 5.2 also supports integrated flash programming, allowing developers to program the non-volatile memory of the target microcontroller directly from the debugger environment. This functionality simplifies the process of uploading and testing new code versions.
  - Diagnostics tools are included to verify the integrity of the programmed data, which is especially important for ensuring reliability in production environments.
- **Hardware Support**:
  - UDE STK 5.2 is compatible with a variety of debugging interfaces and hardware probes, such as JTAG, SWD (Serial Wire Debug), and the high-performance PLS UAD (Universal Access Device) family.
  - The hardware support allows developers to interface with a range of microcontrollers and choose the connection type that best fits their requirements—whether it's rapid prototyping or more in-depth, performance-critical analysis.
- **Flexibility and Integration**:
  - **IDE Integration**: UDE STK can be used as a standalone tool or integrated into popular integrated development environments (IDEs), such as Eclipse or SPC5Studio. This helps developers use their preferred software tools without compromising on debugging capabilities.
  - **Third-Party Tool Compatibility**: It is also compatible with various third-party compilers, such as Green Hills, IAR, and GCC, providing flexibility in the development toolchain.
- **User-Friendly Licensing**:
  - UDE STK 5.2 offers a "Starter Kit" version, which is typically more cost-effective and suitable for developers and students who are just getting started with microcontroller development. This makes it an accessible option for smaller projects or educational purposes, providing essential debugging tools without the need for extensive licensing costs.

### 5.2.2 Application Areas and Benefits

- **Automotive Systems**: Given its robust support for SPC5 microcontrollers and Power Architecture, UDE STK 5.2 is well-suited for automotive applications, especially those involving safety-critical functions, such as ECUs for engine management, braking, and ADAS.
- **Industrial Automation**: Its support for multi-core debugging, RTOS awareness, and trace analysis makes UDE STK ideal for use in industrial control systems, where real-time performance and reliability are crucial.
- **Embedded System Development**: The powerful debugging, tracing, and multi-core support make it an excellent choice for embedded systems in consumer electronics and IoT devices, providing visibility into the inner workings of complex applications.

UDE STK 5.2 is a powerful and versatile debugger for embedded systems, providing a range of tools to help developers test, debug, and optimize their applications. Its support for multiple microcontroller architectures, advanced trace and breakpoint capabilities, RTOS-awareness, and integration with various IDEs make it a suitable choice for developing complex embedded systems, particularly in automotive and industrial environments. By offering real-time debugging, multi-core support, and flexible licensing options, UDE STK 5.2 is both accessible and highly functional, catering to both professional developers and those just starting in embedded development.

# 6 Development of the Embedded System for Electric Vehicles

In the previous chapters, we introduced the motivation, background, and system requirements for the development of an embedded system for EVs, with particular attention to hardware-software integration. With the foundational concepts established, we now proceed to the practical aspects of system implementation. This chapter will focus on the specific steps involved in developing and integrating the embedded system for our EV project.

We begin with the initial configuration of the embedded platform, detailing the setup of the microcontroller and other essential components that form the core of our system. This includes configuring clock settings, pin mapping, and initializing the basic modules required for further development. The subsequent section addresses the reading from SARADC (Successive Approximation Register Analog-to-Digital Converter) inputs, which play a crucial role in capturing analog signals such as those from the accelerator pedal, providing the digital data needed for system control.

Following this, we provide an overview of the CAN bus, a critical communication protocol in automotive systems, and describe its implementation within our project. This involves setting up the CAN controller, configuring message filters, and ensuring reliable data exchange between different modules of the EV. Lastly, we delve into the DSPI (Deserial/Serial Peripheral Interface) implementation, which is utilized for interfacing with various peripheral devices, such as digital-to-analog converters, that are crucial for real-time control and feedback.

This chapter will walk through each of these key stages, providing insights into the methodologies and tools used, as well as practical considerations and challenges encountered during the implementation. By breaking down these steps, we aim to provide a comprehensive understanding of how the embedded system components are configured and integrated to meet the specific needs of electric vehicles, with a focus on seamless hardware-software integration.

## 6.1 State Machine Design for Sound Synthesis

A Finite State Machine (FSM) is a model of computation used to design systems that can be in one of a finite number of states at any given time. In an FSM, the system transitions from one state to another in response to specific inputs or conditions, making it highly suitable for managing complex sequences of operations in embedded systems.

Figure 6-1 represents an FSM tailored for an embedded system that continuously collects data from vehicle sensors and communicates with a miniPC for processing. The FSM provides a structured approach to control flow, ensuring the system can handle data acquisition, communication, error handling, and response processing in a closed-loop fashion. Each state in the FSM has a defined purpose, with transitions based on events or conditions, creating a robust and organized control structure for real-time applications.

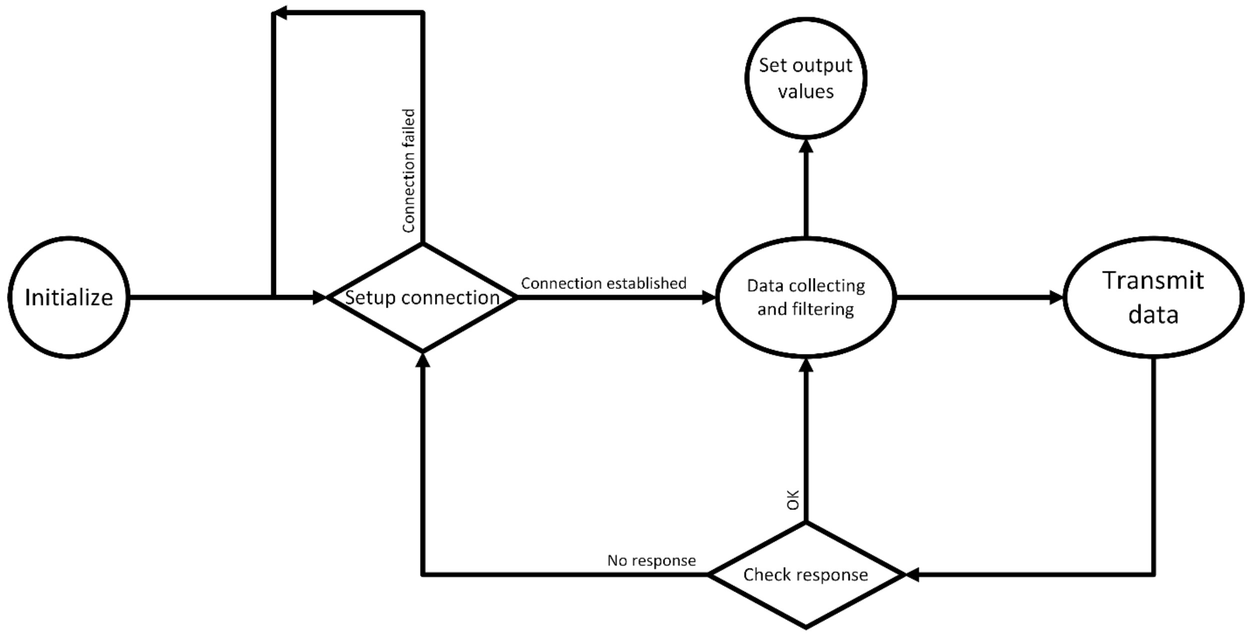


*Figure 6-1 Finite state machine flowchart for the electric vehicle sound synthesis system, showing the operating states and transitions based on vehicle speed, acceleration, and user input*

1. **Initialize**
   - **Purpose**: This is the starting point of the FSM where the system performs necessary setup operations. In this stage, the board initializes its components, such as UART communication, CAN interface, ADCs for analog inputs, and any other peripherals.
   - **Transition**: After initialization, the FSM proceeds to the next step, Setup Connection.
2. **Setup Connection**
   - **Purpose**: Here, the board attempts to establish communication with the miniPC. This involves setting up the UART connection or any protocol handshakes required for reliable data exchange.
   - **Transition**:
     - **Connection Established**: If the connection is successfully established, the FSM moves to Data Collecting and Filtering.
     - **Connection Failed**: If the connection fails, it loops back to Initialize to retry setup. This loop allows the system to reset and try again if any connection issues arise, enhancing robustness.
3. **Data Collecting and Filtering**
   - **Purpose**: In this state, the system gathers data from various sources on the board. This includes:
     - Collecting CAN messages from the vehicle network.
     - Reading analog inputs through the ADC (e.g., sensor data).
     - Filtering or processing the raw data as needed before transmission.
   - **Transition**: Once the data is collected and filtered, the FSM transitions to Transmit Data to send this information to the miniPC.
4. **Transmit Data**
   - **Purpose**: In this state, the collected and filtered data is transmitted to the miniPC via UART. This step is essential for communicating the current state of the board's environment (e.g., sensor readings, vehicle information).
   - **Transition**: After data transmission, the FSM moves to Check Response.

5. **Check Response**

   - **Purpose**: Here, the board checks for an acknowledgment or response from the miniPC. The response might contain new control values or commands for the board.
   - **Logic**:
     - Instead of waiting indefinitely, the board checks for response availability within a set time limit.
     - If the response is received in time, the FSM proceeds to Set Output Values.
     - If there is no response within the timeout, it raises an error and loops back to continuously check for response availability.
   - **Transition**: Based on response status:
     - **OK (Response Received)**: Moves to Set Output Values.
     - **No Response (Timeout)**: Enters an error-checking loop to wait for a response indefinitely, ensuring the system can detect and handle communication issues.

6. **Set Output Values**

   - **Purpose**: This state involves applying the received response (if any) from the miniPC. For example, the response might include updated analog output values or other settings for the board to adjust its outputs accordingly.
   - **Feedback Verification**: Since this is a closed-loop system, the correctness of these output values isn’t directly verified here. Instead, when the FSM cycles back to Data Collecting and Filtering, it will verify the effect of the applied output by observing changes in the collected data.
   - **Transition**: After setting the output values, the FSM loops back to Data Collecting and Filtering, continuing the cycle of data acquisition, transmission, and response processing.

**Summary**

- This flowchart represents a closed-loop FSM that continuously collects data, communicates with the miniPC, and adjusts its outputs based on feedback.
- If any part of the process (like connection or response) fails, the system can retry or wait for resolution, making it resilient.
- The closed-loop verification method leverages the natural feedback from data collection to ensure outputs are functioning as expected, without requiring an explicit output-checking mechanism.

This FSM is well-suited for embedded applications that require real-time data acquisition, communication, and response adjustments, such as your system for synthesizing engine sounds in an electric vehicle.

## 6.2 Initial Configuration

The initial configuration is a crucial step in the development of the embedded system for electric vehicles. This phase involves setting up the foundational hardware and software parameters that ensure reliable and efficient operation. The configuration of the microcontroller, clock distribution, pin mapping, and initialization of key peripherals are all essential tasks that lay the groundwork for subsequent stages of development. By establishing these core settings, we ensure that the system can operate in a stable manner and meet the specific requirements of the EV application.

### 6.2.1 Clock Distribution

The clock distribution network within the microcontroller is a key aspect of the initial configuration. It determines the timing and synchronization of all internal modules and peripherals. The clock tree must be configured to provide appropriate clock signals to each subsystem, ensuring optimal performance and energy efficiency. Figure 6-2 illustrating the clock tree of the microcontroller is provided to give a general overview without delving into deep technical details.

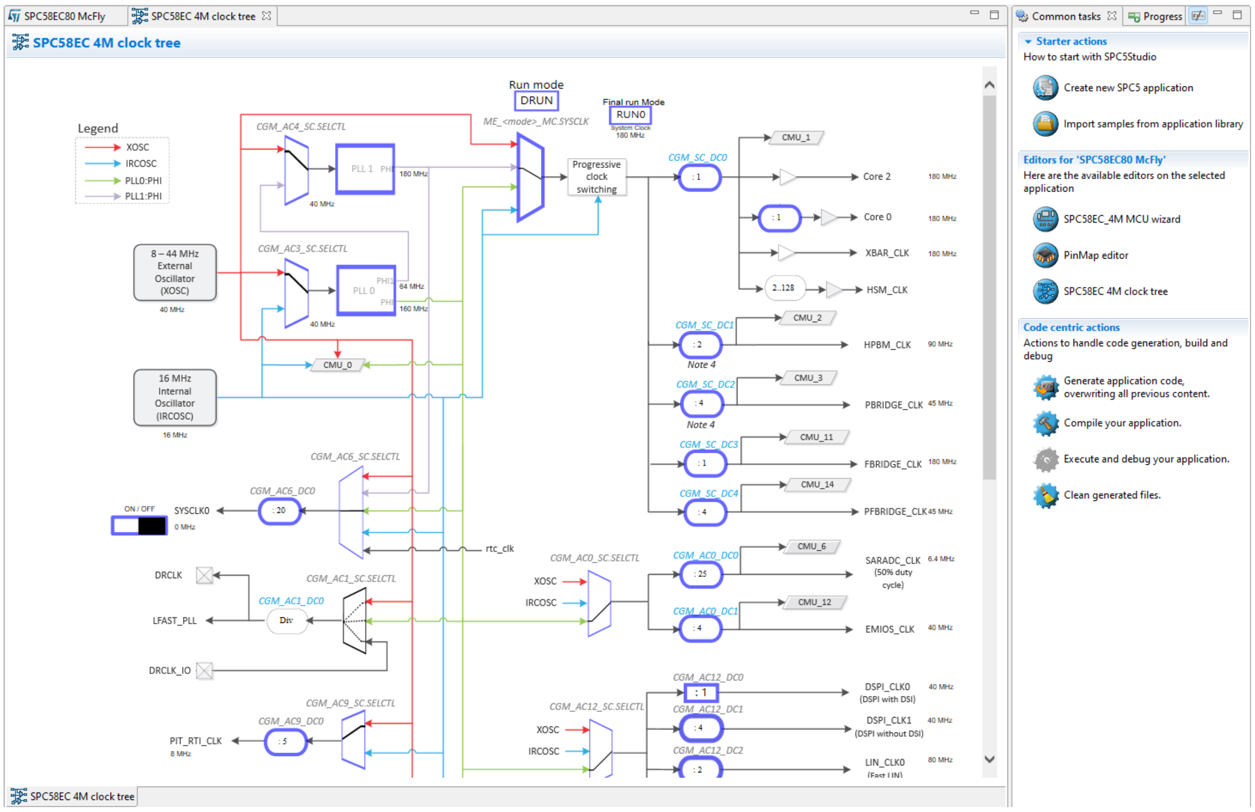
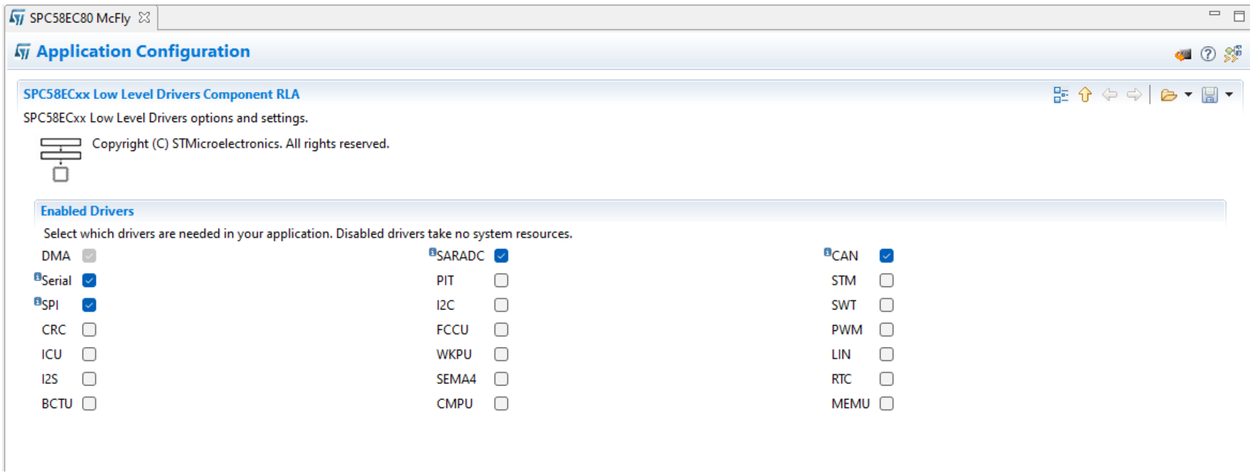

SPC58EC80 McFly
SPC58EC 4M clock tree
Legend
XOSC
IRCOSC
PLL0:PHI
PLL1:PHI
Run mode
DRUN
Final run Mode
RUN0
Progressive clock switching
8 – 44 MHz External Oscillator (XOSC)
16 MHz Internal Oscillator (IRCOSC)
PLL 1
PLL 0
CMU_0
Core 2
Core 0
XBAR_CLK
HSM_CLK
HPBM_CLK
PBRIDGE_CLK
FBRIDGE_CLK
PFBRIDGE_CLK
SARADC_CLK
EMIOS_CLK
DSPI_CLK0
DSPI_CLK1
LIN_CLK0
SYSCLK0
DRCLK
LFAST_PLL
DRCLK_IO
PIT_RTI_CLK
rtc_clk
Common tasks
Progress
Starter actions
How to start with SPC5Studio
Create new SPC5 application
Import samples from application library
Editors for 'SPC58EC80 McFly'
Here are the available editors on the selected application
SPC58EC_4M MCU wizard
PinMap editor
SPC58EC 4M clock tree
Code centric actions
Actions to handle code generation, build and debug
Generate application code, overwriting all previous content.
Compile your application.
Execute and debug your application.
Clean generated files.

*Figure 6-2 Clock tree is used to set clock signal for peripherals*

### 6.2.2 Low Level Driver Component Register-Level Abstraction (RLA)

The Low Level Driver Component RLA is an essential part of the microcontroller's initial configuration, providing direct access to hardware registers for efficient and precise control. This layer allows developers to interface directly with the hardware, optimizing performance for specific tasks within the EV system. Figure 6-3 is provided to show the driver selection page of the SPC5studio, where various drivers can be enabled.



SPC58ECxx Low Level Drivers Component RLA

SPC58ECxx Low Level Drivers options and settings.

Copyright (C) STMicroelectronics. All rights reserved.

Enabled Drivers

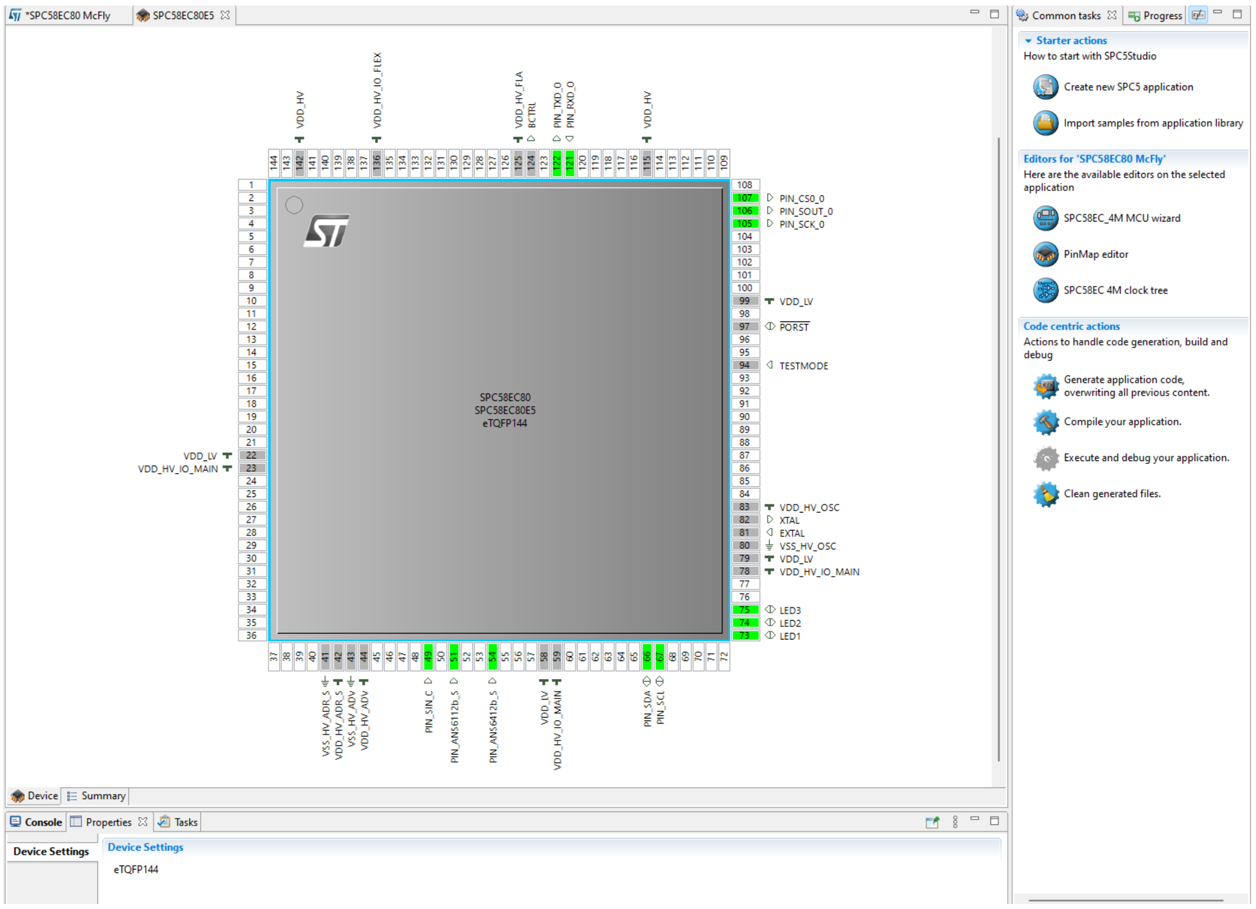
Select which drivers are needed in your application. Disabled drivers take no system resources.

| | | | | | |
|---|---|---|---|---|---|
| DMA | ✓ | SARADC | ✓ | CAN | ✓ |
| Serial | ✓ | PIT | ☐ | STM | ☐ |
| SPI | ✓ | I2C | ☐ | SWT | ☐ |
| CRC | ☐ | FCCU | ☐ | PWM | ☐ |
| ICU | ☐ | WKPU | ☐ | LIN | ☐ |
| I2S | ☐ | SEMA4 | ☐ | RTC | ☐ |
| BCTU | ☐ | CMPU | ☐ | MEMU | ☐ |

*Figure 6-3 Low Level Driver Component RLA , by checking the boxes we can add those drivers to our project*

### 6.2.3 Pin Mapping

The pin mapping process is a fundamental part of configuring the embedded system, as it ensures that the microcontroller's input/output pins are correctly assigned to their intended functions. In SPC5Studio, the pin mapping tool allows developers to easily assign peripherals to specific pins, simplifying the hardware-software integration process. This configuration step is critical for ensuring that signals are routed properly and that the system performs as intended. Figure 6-4 is provided to illustrate the pin mapping window in SPC5Studio, showing how pins can be assigned to different functions and how the interface assists in managing these assignments.



How to start with SPC5Studio

Here are the available editors on the selected application

Actions to handle code generation, build and debug

Generate application code, overwriting all previous content.

Device Settings

Device Settings

eTQFP144

*Figure 6-4 Pin mapping is used to set a name for each pin and define the functionality of that pin*

### 6.2.4 SARADC Configuration

The SARADC (Successive Approximation Register Analog-to-Digital Converter) is utilized in the SPC58EC80-DISP board to convert analog signals to digital form. This conversion plays a key role in acquiring real-time data from sensors, which is critical in various embedded systems and automotive applications. For my project, involving the generation of synthetic engine sounds for EVs, I have configured two channels of the SARADC module to read data from input sources such as the accelerator pedal sensor.

The SARADC configuration was performed using SPC5Studio, and the Figure 6-5 illustrates the configuration interface, providing an overview of the setup parameters. This includes specifying the Conversion Group, Channel Configuration, and various conversion timing details.

**Conversion Mode and Triggering**

The ADC in SPC5Studio was set to operate in 'Scan Mode' under the Conversion Group settings. This mode allows for multiple channels to be sampled in sequence. The trigger mode for the SARADC was set to 'Disabled', which means that conversion starts without any external or software trigger. Instead, the conversions are controlled internally within the scanning cycle.

**Clock Settings and Resolution**

The SARADC clock settings have been configured to achieve an appropriate balance between speed and accuracy for our application. The attached figure provides details on the registers used for conversion timing configuration:

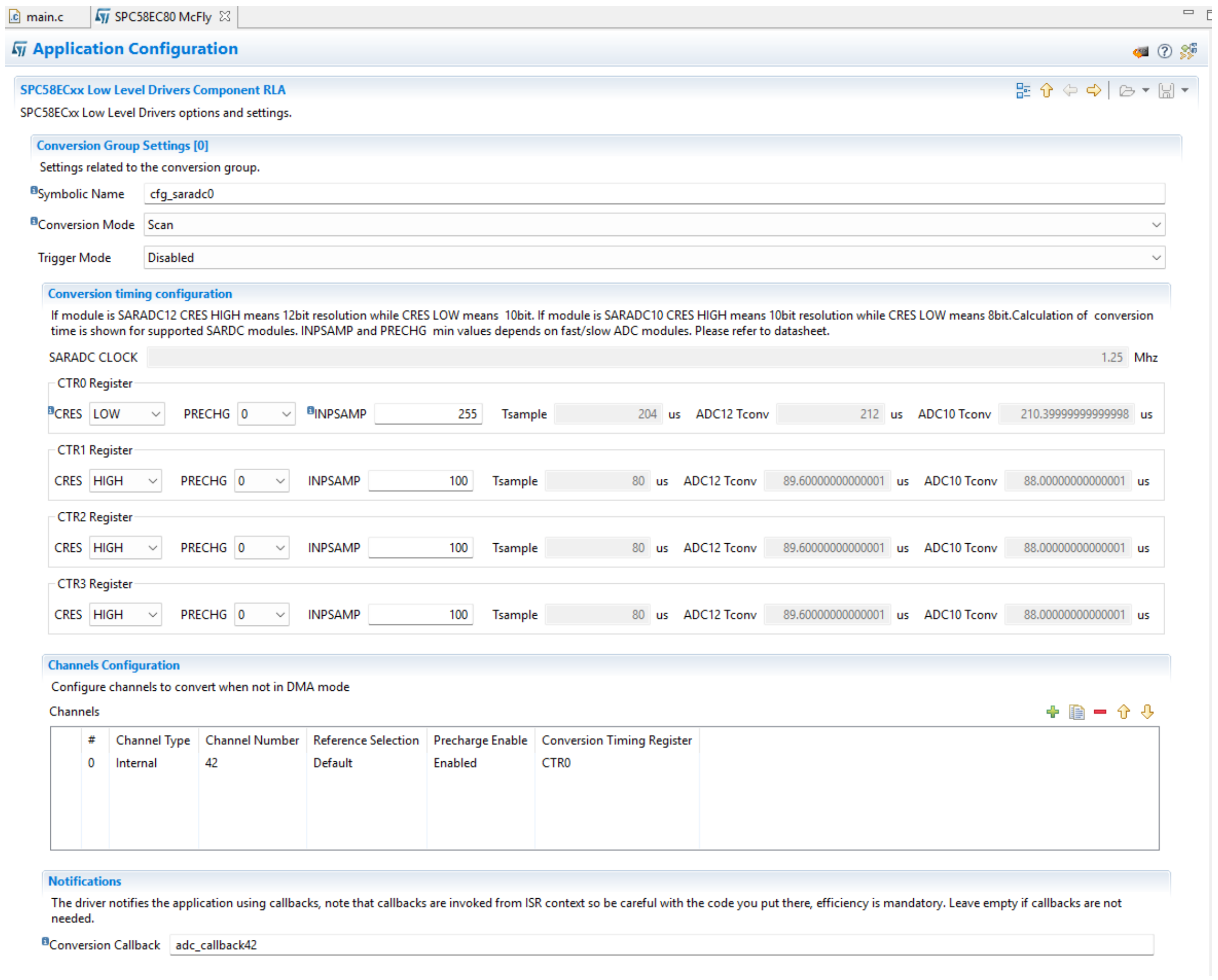
- **CTRx Registers**: The configuration screen displays settings for four control registers (CTR0, CTR1, CTR2, CTR3). Each CTR register manages parameters like conversion resolution (CRES), precharge time (PRECHG), and sample time (Tsample).
- The SARADC module can operate at different resolutions, where **CRES** can be set to LOW (10-bit) or HIGH (12-bit). For our application, we used CTR0 which is set to LOW and uses 10 bits resolution.
- The conversion time (Tconv) for the ADC channels is shown in the Figure 6-5, which is essential for estimating latency and managing the data acquisition rate.

**Channel-Specific Configuration**

In this project, two internal channels were configured to capture analog inputs. As shown in the Figure 6-5, the following parameters were defined:

- **Channel Type**: Internal, indicating that no external trigger or additional peripherals are used.
- **Channel Number**: Channel 42 was used for input acquisition, which corresponds to the accelerator pedal sensor.

- **Precharge and Reference Selection**: Precharge was enabled, and the default reference was used to stabilize the input signal before conversion.
- **Conversion Timing Register**: CTR0 was selected for configuring the timing parameters of channel conversion.



main.c | SPC58EC80 McFly

**Application Configuration**

**SPC58ECxx Low Level Drivers Component RLA**

SPC58ECxx Low Level Drivers options and settings.

**Conversion Group Settings [0]**

Settings related to the conversion group.

| | |
|---|---|
| Symbolic Name | cfg_saradc0 |
| Conversion Mode | Scan |
| Trigger Mode | Disabled |

**Conversion timing configuration**

If module is SARADC12 CRES HIGH means 12bit resolution while CRES LOW means 10bit. If module is SARADC10 CRES HIGH means 10bit resolution while CRES LOW means 8bit.Calculation of conversion time is shown for supported SARDC modules. INPSAMP and PRECHG min values depends on fast/slow ADC modules. Please refer to datasheet.

SARADC CLOCK 1.25 Mhz

| Register | CRES | PRECHG | INPSAMP | Tsample | | ADC12 Tconv | | ADC10 Tconv | |
|---|---|---|---|---|---|---|---|---|---|
| CTR0 Register | LOW | 0 | 255 | 204 | us | 212 | us | 210.39999999999998 | us |
| CTR1 Register | HIGH | 0 | 100 | 80 | us | 89.60000000000001 | us | 88.00000000000001 | us |
| CTR2 Register | HIGH | 0 | 100 | 80 | us | 89.60000000000001 | us | 88.00000000000001 | us |
| CTR3 Register | HIGH | 0 | 100 | 80 | us | 89.60000000000001 | us | 88.00000000000001 | us |

**Channels Configuration**

Configure channels to convert when not in DMA mode

Channels

| # | Channel Type | Channel Number | Reference Selection | Precharge Enable | Conversion Timing Register |
|---|---|---|---|---|---|
| 0 | Internal | 42 | Default | Enabled | CTR0 |

**Notifications**

The driver notifies the application using callbacks, note that callbacks are invoked from ISR context so be careful with the code you put there, efficiency is mandatory. Leave empty if callbacks are not needed.

Conversion Callback adc_callback42

*Figure 6-5 SARADC configuration parameters*

**Use of Callbacks**

In the configuration, callbacks were also set for the ADC driver to notify the application whenever a conversion is completed. The function **adc_callback42** was defined to handle the completion event, facilitating real-time processing of the converted digital data.

**Read from SARADC inputs**

The SARADC on the SPC58EC80-DISP board is an analog-to-digital converter that converts an analog input signal (like a varying voltage from a sensor) into a digital value that the microcontroller can process.

**Key Features of the SARADC on the SPC58EC80-DISP Board:**

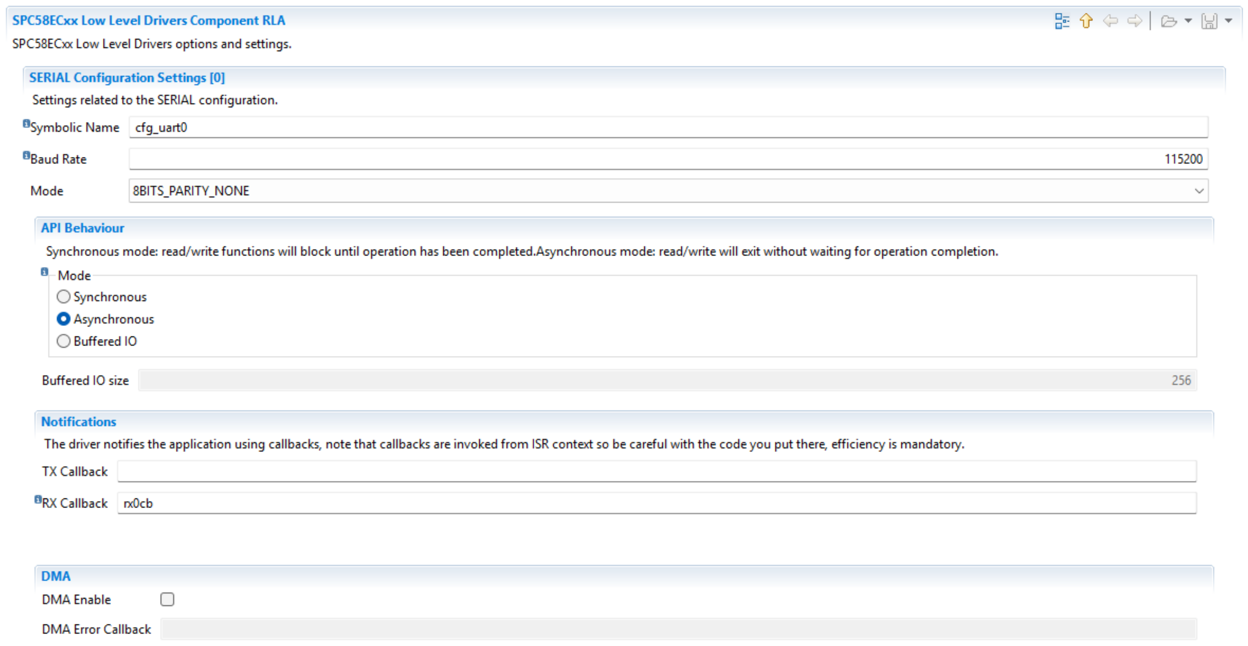
1. **Successive Approximation**: The SARADC uses the successive approximation method, which is efficient and fast. It approximates the analog input by comparing it to a series of reference voltages, narrowing down to a precise digital value in a set number of steps.
2. **Resolution and Accuracy**: SARADCs generally offer good accuracy with moderate resolution (typically 10-12 bits for automotive microcontrollers), allowing the SPC58EC80 to capture fine details in the input signal, such as the accelerator pedal's exact position.
3. **Speed**: The SAR approach is relatively fast compared to other ADC types, making it suitable for real-time applications, like reading rapid changes in sensor values.
4. **Multiplexing Capability**: The SARADC can handle multiple input channels, meaning you can connect several analog sensors to it and switch between them, enabling you to monitor various signals efficiently.
5. **Automotive-Grade Performance**: Designed for automotive applications, the SARADC on this board provides the robustness and reliability required for vehicle systems, with good noise immunity and precision.

The SARADC on the SPC58EC80-DISP is optimized for fast, accurate, and reliable conversion of analog signals, making it ideal for applications like monitoring an accelerator pedal, where real-time, precise data is essential.

In this thesis I used the SARADC to read the accelerator pedal position. Once the pedal position is digitized, MCU can send this value along with other sensor data to the miniPC, where it can be used as an input for sound synthesis, among other processing tasks. This data is critical for synthesizing engine sounds, where the pedal position can control the sound dynamics, reflecting acceleration or deceleration in real-time.

### 6.2.5 UART Configuration

The UART (Universal Asynchronous Receiver-Transmitter) module was configured for serial communication to facilitate the data exchange between the board and external devices, which is essential for debugging and real-time data monitoring. The UART configuration in SPC5Studio was performed with the following settings, as illustrated in Figure 6-6.



SPC58ECxx Low Level Drivers Component RLA

SPC58ECxx Low Level Drivers options and settings.

SERIAL Configuration Settings [0]

Settings related to the SERIAL configuration.

| | |
|---|---|
| Symbolic Name | cfg_uart0 |
| Baud Rate | 115200 |
| Mode | 8BITS_PARITY_NONE |

API Behaviour

Synchronous mode: read/write functions will block until operation has been completed.Asynchronous mode: read/write will exit without waiting for operation completion.

Mode: Synchronous [✓ Asynchronous] Buffered IO

| | |
|---|---|
| Buffered IO size | 256 |

Notifications

The driver notifies the application using callbacks, note that callbacks are invoked from ISR context so be careful with the code you put there, efficiency is mandatory.

| | |
|---|---|
| TX Callback | |
| RX Callback | rx0cb |

DMA

| | |
|---|---|
| DMA Enable | ☐ |
| DMA Error Callback | |

*Figure 6-6 UART configuration parameters*

#### General Configuration

- **Symbolic Name**: The UART configuration is represented symbolically as **cfg_uart0** for easy reference within the code.
- **Baud Rate**: Set to **115200** bps, which is a standard rate for serial communication, balancing speed and reliability for debugging and logging.
- **Mode**: Configured as **8BITS_PARITY_NONE**, which means 8 data bits, no parity, and 1 stop bit. This is a typical configuration for straightforward communication without error-checking overhead.

**API Behavior**

- **Mode**: The UART module was configured to operate in **Asynchronous** mode, allowing the read/write operations to continue without blocking until completion. This enables better responsiveness and non-blocking behavior for other tasks.
- **Buffered IO Size**: The buffered I/O size was left unconfigured (default), implying that standard buffers were used for data handling.

**Notifications**

- **RX Callback**: A receive callback function **rx0cb** was defined to handle received data, enabling the application to respond in real-time to incoming messages.
- **TX Callback**: No transmit callback was defined for this configuration, as the application does not require explicit notifications for completed transmissions.

**DMA Configuration**

- **DMA Enable**: Direct Memory Access (DMA) was not enabled for this UART configuration, meaning that data transfer is handled directly by the CPU. This decision was made to reduce the complexity of the implementation since the data rate requirements did not necessitate the use of DMA.
- **DMA Error Callback**: Not configured, as DMA was not utilized.

### 6.2.6 DSPI Configuration

The DSPI (Deserial/Serial Peripheral Interface) module was configured for SPI communication to interface with external peripherals, such as sensors or DACs, providing a synchronous data transfer mechanism. The DSPI configuration in SPC5Studio was set up with the following details.

**General Configuration**

- **Symbolic Name**: The DSPI configuration is represented as **low_speed_16** for easy reference in the code, indicating a specific SPI setup with a lower speed and a frame size of 16 bits.
- **Mode**: Configured as **Master**, which allows the board to initiate communication with connected slave devices.
- **Clock Polarity and Phase**: Set to **Low** clock polarity and **Leading Edge** clock phase. This combination ensures compatibility with the target SPI peripheral's timing requirements.
- **Frame Size**: Set to **16 bits**, which means that each SPI transaction involves 16 bits of data being transferred per frame.
- **Frame Ordering**: **MSB first**, which means that the most significant bit is transmitted first in each data frame.
- **Endianness**: Configured as **Little Endian** for consistency with the rest of the data operations in the system.

**Timing Configuration**

The timing settings for DSPI were carefully configured to match the requirements of the connected peripheral:

- **Baud Rate Prescaler**: Set to **PRE2**, which helps determine the clock rate for SPI communication.
- **Baud Rate Divider**: Set to **DIV64**, which further divides the clock to achieve the desired communication speed of **312500 bits/s**.
- **Chip Select Timing**: Specific prescalers and dividers were configured for the chip select assertion and de-assertion timings to ensure stable communication with the connected devices.

### Chip Select Configuration

- **Mode**: The **Chip Select** was configured in **Hardware (continuous)** mode, ensuring that the SPI line remains selected throughout the data transfer, preventing communication interruptions.
- **GPIO Port and Bit**: The Chip Select line is managed via **PORT_A, GPIO Bit 0**, which is linked to the hardware to provide precise control over which peripheral is being addressed.
- **PCS Line**: Set to **PCS0**, which represents the particular line used for selecting the SPI device.

### Notifications

- **Transfer Complete Callback**: A **Transfer Complete Callback** was left unconfigured, indicating that for this setup, the SPI transactions are handled without interrupt-driven completion notifications, which is suitable for simpler or non-time-critical transfers.

The SPI protocol allows for high-speed data transfer by shifting data on every clock pulse. The clock polarity and phase settings are configurable, allowing compatibility with various peripheral devices like the MCP4922. The **MCP4922** is a 12-bit digital-to-analog converter (DAC) that communicates using the **SPI** protocol. In this thesis project, we need to use the **DSPI** module because it provides a way to control and communicate with SPI-compatible devices like the MCP4922. As soon as the miniPC returns the values, using SPI and MCP4922 corresponding values will produce on the output. A detailed view of the DSPI signaling is captured from the real project and illustrated in Figure 6-7.

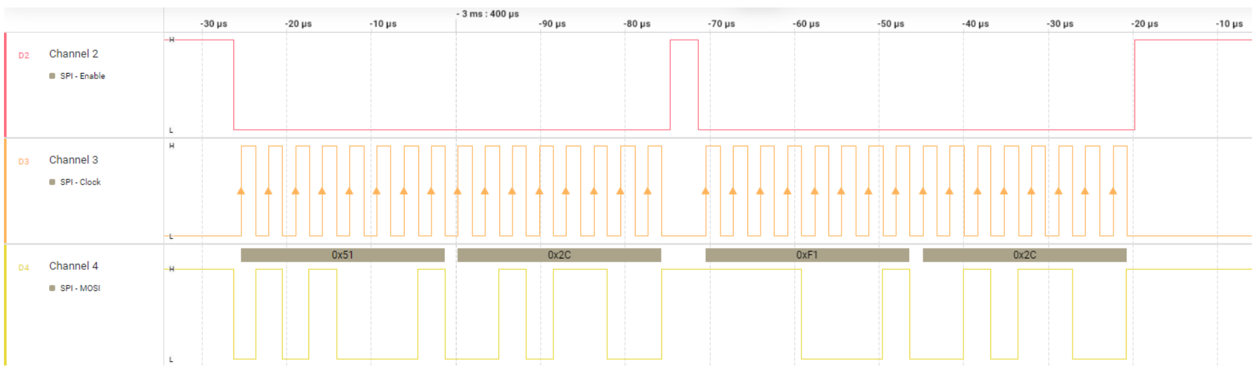


*Figure 6-7 SPI communication signals captured from the real project and shows sending data to the MCP4922*

**Summary of Configuration**

The SARADC, CAN, UART, and DSPI configurations for the SPC58EC80-DISP board are pivotal for ensuring accurate data acquisition, robust communication, and effective data logging within the system. The CAN, UART, and DSPI configurations, demonstrate the adjustable control over parameters such as timing, buffering, and filtering, which are crucial for reliable system performance. Together, these configurations enable the seamless integration of analog sensor data, communication protocols, and debugging capabilities into the broader vehicle control network, supporting the objectives of the project.

## 6.3 CAN Bus Overview and Implementation

The CAN (Controller Area Network) bus is a robust communication system used primarily in vehicles and industrial applications to enable different Electronic Control Units or “nodes” to communicate with each other without needing a central host computer. Here's a breakdown based on the images and the description provided:

### 6.3.1 CAN Bus as a Communication System

- Think of the CAN bus as the **nervous system** in a human body, allowing different parts (nodes or ECUs) to exchange information quickly and reliably.
- Each ECU in the network is responsible for controlling or monitoring specific functions, such as braking, engine control, or air conditioning. These functions often need to interact, and the CAN bus facilitates this communication.
- By allowing ECUs to communicate with each other directly, the CAN bus eliminates the need for point-to-point wiring, which makes the system simpler, more efficient, and easier to troubleshoot.

### 6.3.2 Physical Structure of the CAN Bus

- The CAN bus uses a **two-wire twisted pair** for transmission, often color-coded for easy identification:
    - **CAN High (CAN H):** Typically yellow in color.
    - **CAN Low (CAN L):** Typically green in color.

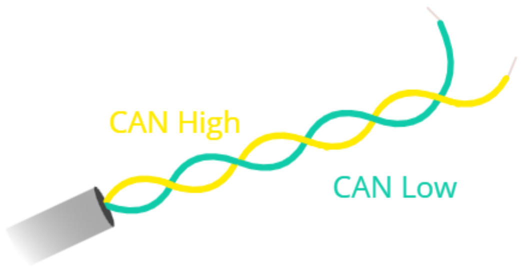


*Figure 6-8 CAN twisted pair wire typical color, yellow for CAN high and green for CAN low*

- Twisting the wires together helps **reduce electromagnetic interference (EMI)**, ensuring data integrity even in environments with electrical noise, like in a car or industrial setup.
- When data is transmitted, the CAN High and CAN Low wires carry opposite signals (differential signaling), which enhances noise immunity.

### 6.3.3 Working Mechanism

- The CAN bus operates as a **multi-master, message-oriented protocol**, meaning any node can start a transmission if the bus is free.
- Each message on the CAN bus has a unique identifier that determines its priority. If two nodes attempt to transmit at the same time, the message with the lower identifier value gets priority, ensuring the more critical data goes through first.
- Messages are transmitted across the network to all nodes, but each node only processes messages it's programmed to receive.

### 6.3.4 Advantages of the CAN Bus

- **Reliability and Error Handling:** The CAN bus includes built-in error detection and handling mechanisms, such as CRC, which minimizes the chance of erroneous data being accepted by any ECU.
- **Real-Time Communication:** The CAN bus allows real-time data sharing between ECUs, making it ideal for time-critical functions like braking or airbag deployment in automotive systems.
- **Efficiency:** The system is efficient because each message has a specific identifier, allowing nodes to filter and prioritize messages, reducing processing time and bandwidth usage.
- **Simplified Wiring:** Since all ECUs are connected to the same twisted pair of wires, there's less wiring complexity compared to point-to-point connections, which reduces the weight and cost of wiring in vehicles.

### 6.3.5 Use Cases in Vehicles

- In modern vehicles, CAN bus networks are extensively used to link various ECUs like the **engine control module (ECM)**, **transmission control module (TCM)**, **antilock braking system (ABS)**, **airbag control unit**, and **infotainment systems**.
- For instance, when you press the brake pedal, the **braking ECU** can send a message over the CAN bus to inform the **engine ECU** and **transmission ECU** to reduce power and prepare for stopping. This integrated communication results in smoother and safer operation.

In summary, the CAN bus is an essential communication protocol for modern automotive systems, providing a robust, reliable, and efficient way for multiple ECUs to communicate in real time, making vehicle systems responsive, integrated, and secure.

### 6.3.6 CAN Message Characteristics

The characteristics of a CAN message are essential for understanding how data is transmitted and received over this protocol. Here's an overview:

- **Message Frame Types**

CAN uses different types of message frames for varied purposes:

  - **Data Frame:** Carries data from transmitter to receiver(s).
  - **Remote Frame:** Requests data from another node.
  - **Error Frame:** Signals errors detected on the network.
  - **Overload Frame:** Adds extra delay between data or remote frames if needed.

- **Bit Rates and Synchronization**

CAN supports various bit rates, typically ranging from **125 Kbps to 1 Mbps**. The network synchronizes by resynchronizing each node to the bit time whenever a transition on the CAN bus occurs, allowing precise timing even across multiple nodes.

- **CAN Identifier**

The identifier field (11-bit or 29-bit in extended format) in each CAN frame represents message priority:

  - **Standard Frame:** 11-bit identifier, commonly used.
  - **Extended Frame:** 29-bit identifier, allowing for a more extensive range of message identifiers.

- **Data Length Code (DLC) and Data Field**

The **Data Length Code (DLC)** indicates the number of bytes (0-8) in the data field of a frame. The **Data Field** can contain up to 8 bytes of application data, allowing flexible data transmission in each frame.

- **Cyclic Redundancy Check (CRC)**

Each frame includes a CRC field that ensures data integrity by checking for transmission errors. Any inconsistency in the CRC check triggers an error frame.

- **Arbitration**

CAN uses a non-destructive bitwise arbitration mechanism based on the identifier. The message with the lowest identifier wins arbitration, allowing higher-priority messages to access the bus first.

- **Acknowledgment**

Each node checks for errors in every frame and sends an acknowledgment (ACK) if it correctly receives the message. This mechanism ensures reliable communication across the network.

- **Error Handling**

CAN features five types of errors (bit, stuff, CRC, form, and acknowledgment errors) and utilizes a robust error-handling system where nodes transition between error-active, error-passive, and bus-off states based on error frequency.

- **Message Length and Format**

CAN messages have a compact frame structure, optimizing transmission time. Key fields include:

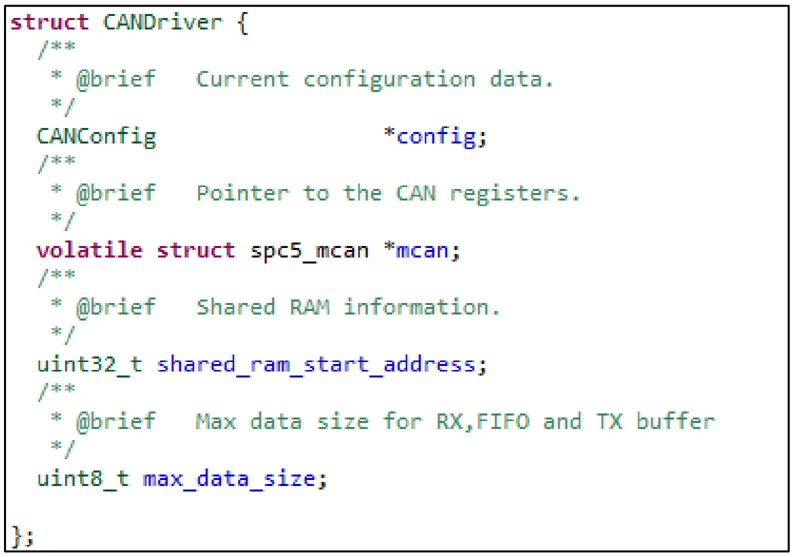
  - **SOF (Start of Frame):** Indicates the start of the frame.
  - **Control Field:** Contains DLC and message flags.
  - **EOF (End of Frame):** Marks the end of the frame.

Understanding these CAN message characteristics is vital for efficient communication design and effective troubleshooting within CAN networks, especially in automotive and industrial applications where reliability is critical.

### 6.3.7 CAN Implementation Codes

#### CAN Configuration

The CAN (Controller Area Network) module was configured to facilitate communication between various components of the electric vehicle system. This communication is crucial for transmitting control commands and sensor data efficiently in a reliable manner. The CAN configuration in SPC5Studio was done with the following parameters.



```
struct CANDriver {
  /**
   * @brief   Current configuration data.
   */
  CANConfig                 *config;
  /**
   * @brief   Pointer to the CAN registers.
   */
  volatile struct spc5_mcan *mcan;
  /**
   * @brief   Shared RAM information.
   */
  uint32_t shared_ram_start_address;
  /**
   * @brief   Max data size for RX,FIFO and TX buffer
   */
  uint8_t max_data_size;

};
```

*Figure 6-9 CAN driver structure*

#### General Configuration

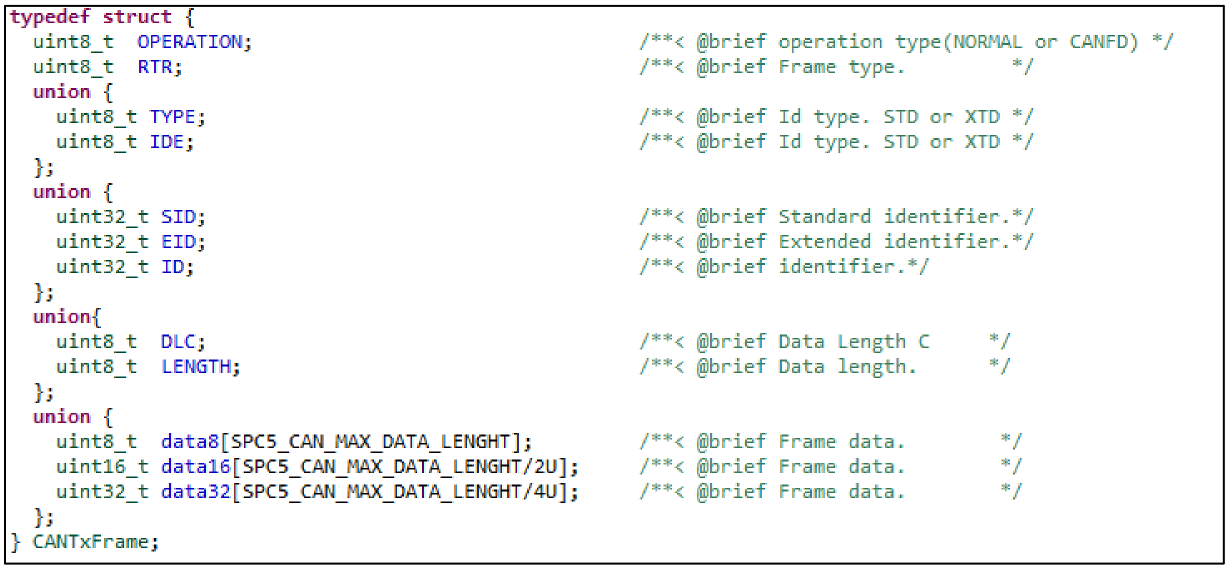
- **Loopback Mode**: The CAN configuration was set with **No_Loopback** mode, meaning that the transmitted messages are not fed back into the receive path. This is suitable for normal operation where external nodes communicate over the CAN bus.
- **Endianness**: The configuration was set to **Big Endian** to align with the data representation format used in other parts of the system.

#### Timing Configuration

The timing parameters of the CAN module were carefully selected to ensure reliable communication across the network:

- **Clock Prescaler**: The prescaler value was set to **1**, which determines the rate at which the CAN clock operates relative to the system clock.
- **Nominal Synchronization Jump Width (NSJW)**: Set to **3**, which allows for re-synchronization in the event of phase errors, improving robustness.

- **Nominal Time Segment 1 (NTSEG1)**: Set to **10**, and **Nominal Time Segment 2 (NTSEG2)** was set to **3**. These segments control the timing characteristics of bit sampling, which is crucial for the accuracy of CAN communication.
- The calculated **CAN Bit Rate** was **500 kbit/s**, which is suitable for most vehicle applications where timely and reliable data transfer is needed.



```
typedef struct {
  uint8_t  OPERATION;                          /**< @brief operation type(NORMAL or CANFD) */
  uint8_t  RTR;                                /**< @brief Frame type.         */
  union {
    uint8_t TYPE;                              /**< @brief Id type. STD or XTD */
    uint8_t IDE;                               /**< @brief Id type. STD or XTD */
  };
  union {
    uint32_t SID;                              /**< @brief Standard identifier.*/
    uint32_t EID;                              /**< @brief Extended identifier.*/
    uint32_t ID;                               /**< @brief identifier.*/
  };
  union{
    uint8_t  DLC;                              /**< @brief Data Length C     */
    uint8_t  LENGTH;                           /**< @brief Data length.      */
  };
  union {
    uint8_t  data8[SPC5_CAN_MAX_DATA_LENGHT];     /**< @brief Frame data.       */
    uint16_t data16[SPC5_CAN_MAX_DATA_LENGHT/2U]; /**< @brief Frame data.       */
    uint32_t data32[SPC5_CAN_MAX_DATA_LENGHT/4U]; /**< @brief Frame data.       */
  };
} CANTxFrame;
```

*Figure 6-10 CAN TX message frame structure, showing all fields in the TX frame*

### CANFD Operation

- **CANFD Enabled**: The CANFD (Flexible Data Rate) was not enabled for this configuration as the standard CAN communication sufficed for the project requirements.
- **Bit Rate Switching**: Disabled, meaning that a single bit rate was used throughout.

### RX Buffer Configuration

- **Interrupt Line**: Configured to **LINE0**, with the callback function **can1sub0rxcb** to handle received messages.
- **Number of RX Buffers**: **16** buffers were set up to handle incoming messages, providing adequate buffering for high-throughput scenarios.
- **RX Buffer Filters**: Two filters were defined to accept specific CAN IDs, **0x108** and **0x118**, which correspond to critical messages for the system.

### FIFO Configuration

Two FIFO buffers were also configured to manage incoming CAN messages:

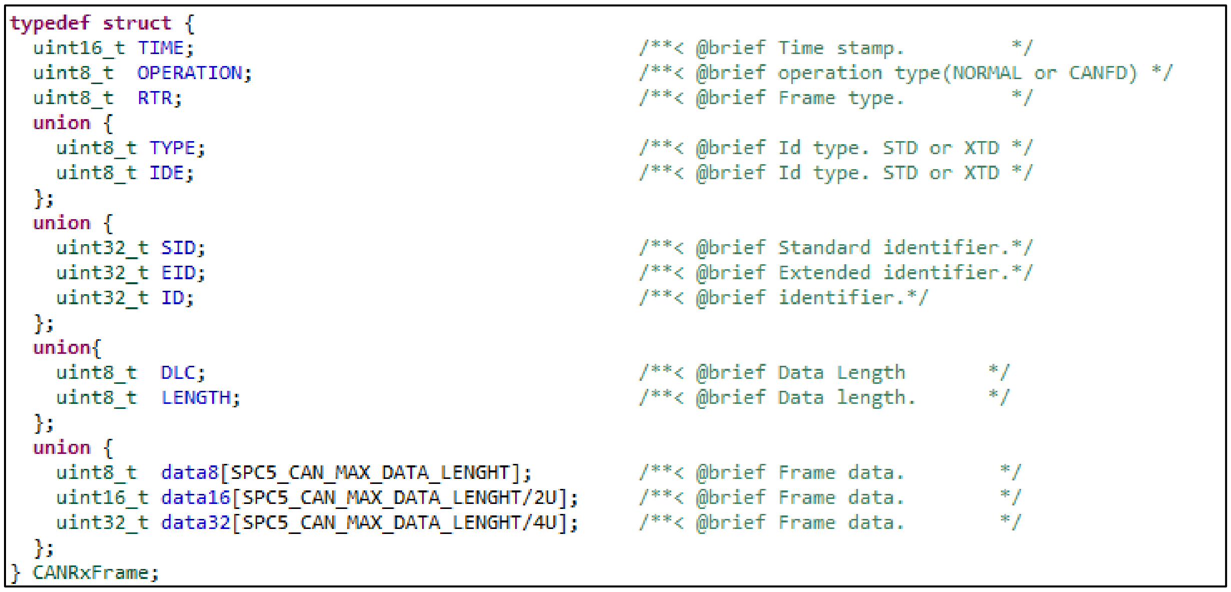
- **FIFO 0**: Configured with **64** buffers, enabling efficient handling of multiple messages. A **new message callback** named **can1sub0rxcb** was used to notify the application of new messages.
- **FIFO 1**: Not configured, as FIFO 0 was sufficient for the application.

### TX Configuration

- **TX Mode**: Set to **DEDICATED**, which allocates fixed buffers for each transmitted message, minimizing latency.
- **Number of TX Buffers**: **16** buffers were used to ensure that outgoing messages can be queued effectively without dropping.

### Error Management

- **Error Callback**: The callback function **errorcb** was defined to handle error conditions on **LINE0**. This callback is triggered whenever an error occurs, providing details on the type of error via the PSR register, which helps in diagnosing communication issues.



```
typedef struct {
  uint16_t TIME;                                     /**< @brief Time stamp.         */
  uint8_t  OPERATION;                                /**< @brief operation type(NORMAL or CANFD) */
  uint8_t  RTR;                                      /**< @brief Frame type.         */
  union {
    uint8_t TYPE;                                    /**< @brief Id type. STD or XTD */
    uint8_t IDE;                                     /**< @brief Id type. STD or XTD */
  };
  union {
    uint32_t SID;                                    /**< @brief Standard identifier.*/
    uint32_t EID;                                    /**< @brief Extended identifier.*/
    uint32_t ID;                                     /**< @brief identifier.*/
  };
  union{
    uint8_t  DLC;                                    /**< @brief Data Length       */
    uint8_t  LENGTH;                                 /**< @brief Data length.      */
  };
  union {
    uint8_t  data8[SPC5_CAN_MAX_DATA_LENGHT];        /**< @brief Frame data.        */
    uint16_t data16[SPC5_CAN_MAX_DATA_LENGHT/2U];    /**< @brief Frame data.        */
    uint32_t data32[SPC5_CAN_MAX_DATA_LENGHT/4U];    /**< @brief Frame data.        */
  };
} CANRxFrame;
```

*Figure 6-11 CAN RX message frame structure, showing all fields in RX frame*

# 7 Results and Discussion

This chapter presents an analysis of the results obtained from functional, user experience, and environmental testing, alongside a detailed discussion on system performance and comparison with existing technologies. The outcomes are interpreted in light of the objectives of this thesis, providing insight into the viability and effectiveness of the proposed electric vehicle (EV) sound synthesis system.

## 7.1 Functional, User Experience, and Environmental Testing

### 7.1.1 Functional Testing Outcomes

Functional testing demonstrated that the system could effectively collect data from multiple sensors, including accelerator pedal and speed, extract the necessary information from the CAN bus, and transmit all the data to the miniPC for sound synthesis. Notably, the integration of the CAN bus interface enabled real-time data transfer without any significant latency issues. Data processing within the SPC58EC80-DISP board showed robust performance, ensuring the generation of corresponding engine sounds that closely matched the vehicle's state, all while meeting strict delay limitations.

A key aspect of functional testing was evaluating the latency between sensor data acquisition, extraction from the CAN bus, data transmission to the miniPC, and the auditory output. The system achieved an average latency of approximately 20 ms, which was well within acceptable limits for real-time applications. This rapid response time ensured that synthesized engine sounds responded promptly to driver inputs, thereby enhancing the immersive driving experience.

### 7.1.2 User Experience Testing Outcomes

User experience testing focused on evaluating how well the synthesized engine sounds improved driver engagement and satisfaction. Participants were asked to drive a test vehicle equipped with the sound synthesis system, and feedback was collected through interviews. Most participants reported a notable enhancement in their driving experience. The added engine sound provided an increased sense of control and awareness, which was particularly evident during acceleration and deceleration phases.

The synthesized engine sounds were generally perceived as realistic and dynamic, contributing positively to the overall enjoyment of driving an EV. Several drivers highlighted that the auditory feedback helped them intuitively gauge vehicle speed and motor load, improving confidence while driving. However, there were also suggestions to fine-tune certain sound characteristics, such as reducing the intensity of the synthesized sounds at higher speeds to avoid auditory fatigue.

### 7.1.3 Environmental Testing Outcomes

The system was also subjected to various environmental conditions to assess its robustness and reliability. Tests included exposure to temperature variations, and electromagnetic interference (EMI), which are common challenges in automotive environments. The SPC58EC80-DISP board and sensors demonstrated stable operation across a temperature range of 0°C to 60°C. The system's EMI shielding was effective, as no significant data corruption or signal interference was observed during high-noise testing conditions.

## 7.2 System Performance Analysis

The overall performance of the proposed sound synthesis system was evaluated based on real-time responsiveness, sound quality, and the effectiveness of the bidirectional communication setup. The SPC58EC80-DISP board's processing power was sufficient for handling sensor data acquisition and preliminary filtering, while the miniPC successfully performed the computationally intensive task of sound synthesis.

### 7.2.1 Real-Time Responsiveness

One of the critical performance indicators was the system's ability to provide real-time auditory feedback. The latency tests revealed that the system maintained a consistent delay below 20 ms, which was sufficient to create a natural auditory response that matched the vehicle's dynamics. This level of responsiveness was found to be crucial in enhancing driver engagement, particularly during rapid throttle changes.

### 7.2.2 Sound Quality and Synchronization

The quality of the synthesized engine sounds was analyzed both subjectively and objectively. Subjective assessments by drivers indicated that the synthesized sounds closely resembled internal combustion engine vehicles in terms of tonal characteristics, such as pitch and modulation during acceleration. Objectively, frequency analysis showed that the sound spectrum aligned well with that of typical ICE vehicles, particularly during idle and low-speed conditions.

Synchronization between vehicle dynamics and auditory feedback was also evaluated. The bidirectional communication setup enabled the SPC58EC80-DISP board to provide timely adjustments to sound parameters based on the vehicle's state. This synchronization was critical for achieving a coherent driving experience, where the auditory output precisely matched the driver's actions.

## 7.3 Comparison With Relevant References

The system was benchmarked against existing commercial and academic solutions for electric vehicle sound synthesis. Compared to systems like the Nissan Leaf's pedestrian alert and the BMW i8's artificial engine sound, the proposed solution demonstrated significant advantages in terms of customization and real-time driver feedback. Unlike basic alert sounds meant primarily for pedestrian safety, this system provides a comprehensive auditory feedback mechanism specifically tailored for enhancing the driving experience, rather than simply meeting regulatory requirements.

Compared to the approach by (Pavlo Bazilinskyy, Roberto Merino-Martínez, Elif Özcan, Dimitra Dodou, & Joost de Winter, 2023), which focused on enhancing driver engagement through exterior synthesized sounds for electric vehicles, the proposed solution features a more advanced sensor integration scheme. This scheme not only uses vehicle speed but also integrates accelerator position and torque demand to produce a more context-aware sound output. Additionally, (Maunder & Munday, 2017) examined in-cabin sound augmentation to enhance driver engagement; however, their work lacked the bidirectional communication essential to the current system. The present system introduces an innovative methodology that surpasses previous efforts, especially in ensuring real-time synchronization between sound output and vehicle control units, thereby maintaining consistency even during rapid changes in driving conditions.

The integration of real-time data collection, bidirectional communication, and an adaptive feedback mechanism makes this system a robust contribution to the evolving domain of electric vehicle auditory feedback. By offering a solution that not only complies with regulatory demands for pedestrian safety but also enhances the driver’s sensory experience, this project sets itself apart from many current implementations in both industry and academia.

# 8 Conclusion and Future Improvements

The development of an embedded system for enhancing the driving experience in EVs through engine sound synthesis yielded promising outcomes. The proposed system effectively integrates the SPC58EC80-DISP board with various vehicle sensors and a miniPC for sound generation, creating a robust, real-time feedback loop between vehicle state and auditory output. The implementation was successful in synthesizing realistic engine sounds that adapted dynamically to vehicle conditions such as speed, acceleration, and braking, addressing the sensory gap inherent in EVs. The functional testing results demonstrated the accuracy of data acquisition, minimal latency in sound generation, and reliable communication between the hardware components. User experience testing indicated a significant enhancement in driver engagement and satisfaction, while environmental tests confirmed the system's reliability across a range of challenging operating conditions.

## 8.1 Summary of Main Results

In summary, the results of this study demonstrate that the proposed electric vehicle sound synthesis system effectively enhances the driving experience by reintroducing essential auditory feedback. The system successfully integrates sensor data acquisition, CAN bus communication, and real-time sound generation, achieving an average latency of approximately 20 ms, which meets the delay requirements for immersive auditory feedback. Functional, user experience, and environmental testing confirmed the reliability of the SPC58EC80-DISP board and the sound synthesis process in various conditions. User testing showed that the synthesized sounds significantly improved driver engagement, providing an intuitive sense of speed and vehicle behavior. Compared to existing solutions, the bidirectional communication setup and real-time adaptability of the system were found to offer a more responsive and immersive experience, bridging the sensory gap in modern EVs.

The proposed system's performance was compared with existing approaches documented in the literature. The system's use of the SPC58EC80-DISP board, with its dedicated automotive-grade features, provided an advantage over traditional microcontroller setups that rely on less robust communication interfaces. The bidirectional communication capability, in particular, offered an edge in maintaining synchronization between sound output and vehicle operation, a feature lacking in some commercially available EV sound modules.

Compared to previous studies that utilized granular synthesis techniques, the current system's use of real-time sensor data to modulate sound synthesis parameters resulted in a more adaptive and responsive auditory experience. The ability to fine-tune the sound output based on real-time torque and acceleration data was found to be a significant improvement over static sound playback methods used in earlier systems.

The proposed sound synthesis system successfully addressed several gaps identified in prior research, including latency reduction, realistic sound generation, and the effective use of bidirectional communication. These improvements contributed to an enhanced driving experience, making electric vehicles more engaging while preserving safety and control.

## 8.2 Limitations

Despite the promising results obtained in this study, several limitations need to be acknowledged. Firstly, the synthesized engine sound, although realistic, is limited in terms of personalization. Although the system offers different sound profiles, these predefined options may not fully cater to the varied preferences of all drivers. Future iterations of the system should explore further customization options that allow users to modify or create their preferred auditory experience.

Secondly, the system's performance heavily depends on the accuracy and reliability of sensor data. Any signal degradation or noise in the sensor data can adversely affect the quality and timing of the synthesized sound. Although environmental testing demonstrated robust performance across various conditions, extreme environments or faulty sensors could still lead to inconsistent auditory feedback.

Thirdly, the current setup relies on a miniPC for sound synthesis, which, while effective, adds to the system's overall cost and complexity. A more compact and integrated hardware solution would be beneficial for real-world deployment, reducing the physical space required and potentially improving system efficiency.

Another limitation is the latency in data transmission, which, although minimized to approximately 20 ms, may still be noticeable in highly dynamic driving scenarios or by particularly sensitive users. Further improvements in reducing latency, possibly through enhanced processing capabilities or optimized communication protocols, would enhance the seamless integration of auditory feedback with vehicle dynamics.

Finally, the system's scalability and compatibility with different vehicle models have not been extensively tested. The integration of the SPC58EC80-DISP board and sensor setup may require significant adaptation when applied to different electric vehicle platforms, limiting its broader applicability without further engineering and customization efforts. Future research should focus on making the system more adaptable to a range of vehicle models and conditions, including different sensor configurations and communication protocols.

## 8.3 Implications for the Industry

The introduction of synthetic engine sound systems for EVs presents significant implications for the automotive industry. This technology bridges the sensory gap between traditional ICE vehicles and modern EVs, thereby enhancing driver satisfaction and helping EVs appeal to a broader range of consumers. The successful implementation of such a system could lead to greater consumer acceptance of EVs, particularly among drivers who miss the auditory feedback of ICE vehicles. Moreover, it suggests a new direction for enhancing EV safety by providing auditory cues to drivers and pedestrians alike, potentially reducing accident risks in low-speed environments where EVs are otherwise silent. The use of the SPC58EC80-DISP board demonstrates how existing embedded microcontroller technology can be effectively leveraged in innovative applications, providing a model for future automotive electronics solutions.

## 8.4 Future Improvements

To address the identified limitations, several future improvements are proposed. Enhancing the fidelity of the synthesized engine sounds could be achieved by employing more advanced sound synthesis algorithms, such as granular synthesis or machine learning-based techniques, to better capture the nuances of ICE engine acoustics. Integration with additional vehicle sensors, such as vibration sensors, could also improve the richness and authenticity of the auditory experience. To mitigate the reliability concerns related to hardware, future versions of the system could integrate the sound synthesis function directly into the vehicle's main control unit, reducing the need for multiple processing devices and improving system robustness. Additionally, optimizing the firmware for better efficiency and exploring hardware acceleration options could help lower latency, ensuring even tighter synchronization between driver actions and auditory feedback. Lastly, the incorporation of driver-customizable sound profiles would add an element of personalization, allowing users to select sound types that align with their preferences, further enhancing driver engagement.

# 9 Bibliography

C., S. (1992). CAN Specification 2.0: Protocol and Implementations. *CAN Specification 2.0: Protocol and Implementations*. SAE International. Retrieved from https://doi.org/10.4271/921603

Cesbron, J., Bianchetti, S., Pallas, M.-A., Bellec, A. L., Gary, V., & Klein, P. (2021, 6 4). Retrieved from https://doi.org/10.1515/noise-2021-0017

Chang K., Cho G., Song W., & Kim M. (2022). Retrieved from https://doi.org/10.4271/2022-01-0972

Hella GmbH & Co. KGaA. (2021, 05 19). *HELLA*. Retrieved from HELLA: https://www.hella.com/soe/en/News/Acoustic-warning-system-AVAS-4159/

Lazaro, M. J., Kim, S., Choi, M., Kim, K., Park, D., Moon, S., & Yun, M. H. (2022). Retrieved from https://aes2.org/publications/elibrary-page/?id=21566

Maunder, & Munday. (2017). System for Augmenting the In-Cabin Sound of Electric Vehicles. *System for Augmenting the In-Cabin Sound of Electric Vehicles*. Institute of Noise Control Engineering.

Nikolaos Kournoutos, & Jordan Cheer. (2019, 6 16). An Environmentally Adaptive Warning Sound System For Electric Vehicles. *INTER-NOISE 2019*. Madrid, Spain: University of Southampton.

Pavlo Bazilinskyy, Roberto Merino-Martínez, Elif Özcan, Dimitra Dodou, & Joost de Winter. (2023). Exterior sounds for electric and automated vehicles. *Exterior sounds for electric and automated vehicles*. Retrieved from https://doi.org/10.1016/j.apacoust.2023.109673

Tsugi Studio. (2018, 5 1). *Real-time Synthesis of Engines*. Retrieved from Tsugi Studio: https://tsugi-studio.com/blog/2018/05/01/real-time-synthesis-of-engines/

Valter Prpic, Elena Gherri, & Luisa Lugli. (2024, 9 18). *A perspective review on the role of engine sound in speed perception and control*. Retrieved from Frontiers: https://www.frontiersin.org/journals/psychology/articles/10.3389/fpsyg.2024.1391271/full