Can protocol:

The **CAN (Controller Area Network)** protocol is a robust multi-master communication protocol standard widely used in automotive, industrial, and embedded systems. It is designed for real-time communication and can withstand harsh environmental conditions. It was originally developed by Bosch in the 1980s to allow reliable communication between multiple electronic control units (ECUs) in vehicles without the need for a host computer.

Has the following properties:

- Prioritization of messages
- Guarantee of latency time
- High reliability
- Error Detection and Handling
- Sleep and Wake-up Capability
- Broadcast Communication
- Multi-Master Communication
- Two-wire Bus

Connections:

Theoretically, There is no limit to the number of units that can be connected to the bus. Practically, it can be limited by the delay time.

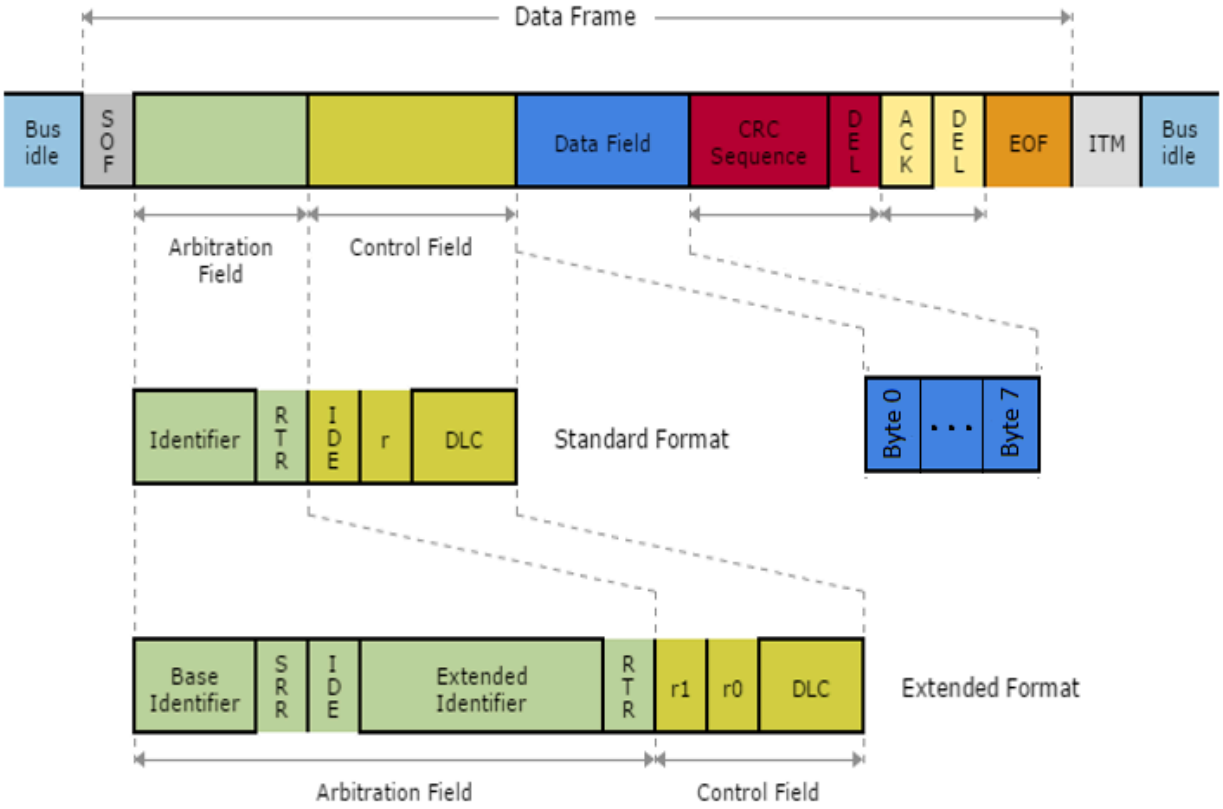
Purpose:

Commonly used in automotive and industrial systems. It is also used in Industrial Automation in Machine control, factory automation, and robotics.In addition to that CAN protocol can be applied in aircraft systems and communication between avionics components.

Wide Adoption in Automotive:

CAN is a standard protocol for vehicle systems, meaning it would be easy to integrate into existing automotive networks, where components like sensors, braking systems, and other ECUs are likely already CAN-compatible

CAN Frame Structure:



SOF (Start of Frame): Indicates the start of a new frame, signaling all nodes to synchronize.

Arbitration Field: Contains the **Identifier** and **RTR (Remote Transmission Request)** bit.

- In **Standard Format**, the identifier is 11 bits.
- In **Extended Format**, the identifier is 29 bits, combining the Base and Extended Identifier fields.

Control Field: Contains the **IDE (Identifier Extension)** bit, **r0** or **reserved bits**, and the **DLC (Data Length Code)**. The DLC specifies the number of bytes in the Data Field (up to 8 bytes).

Data Field: Carries the actual data, which can be up to 8 bytes. Each byte is used for specific messages, such as sensor data in automotive systems.

CRC (Cyclic Redundancy Check) Sequence: Provides error-checking for data integrity, detecting errors that may occur during transmission.

ACK (Acknowledgment) Field: Allows receiving nodes to acknowledge the successful reception of a frame. It includes two bits: an ACK slot and an ACK delimiter.

EOF (End of Frame): Signals the end of the CAN message, allowing other nodes to start transmitting if the bus is free.

ITM (Intermission): The minimum spacing between two consecutive messages, ensuring proper timing and signal processing.

Types of CAN Frames:

Data Frame: The most common frame, used to carry actual data.

Remote Frame: Used by a node to request specific data from another node.

Error Frame: Indicates detected errors in communication.

Overload Frame: Provides extra time for processing in cases of high network load.

Advantages of CAN Protocol:

1) Robust and Reliable: Error-checking mechanisms ensure data integrity, and differential signaling reduces noise interference.

2) Supports Long Distances: Reliable over distances up to 1 km.

3) Multi-Master and Scalable: Can have multiple nodes without requiring a central master control.

Lin protocol

The **LIN (Local Interconnect Network)** protocol is a serial network protocol widely used in the automotive industry, especially for connecting components that do not require high data rates or stringent real-time communication. LIN bus is a supplement to CAN bus. It offers lower performance and reliability - but also drastically lower costs.

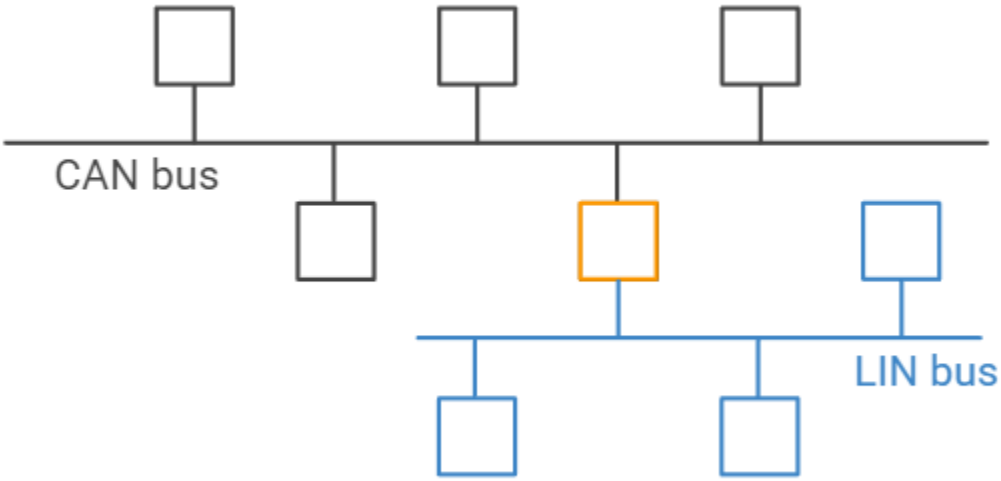
Features:

- Low-Speed communication
- Sigle-Wire Bus
- Single-Master, Multiple slave

Advantages:

- Low Cost: Simplified hardware and single-wire design make it very cost-effective.
- Low complexity: Easy to implement, making it ideal for basic applications.
- Ideal for Low-Speed tasks

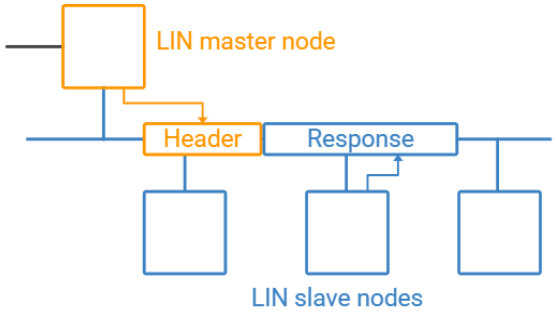
Difference between Can & Lin buses



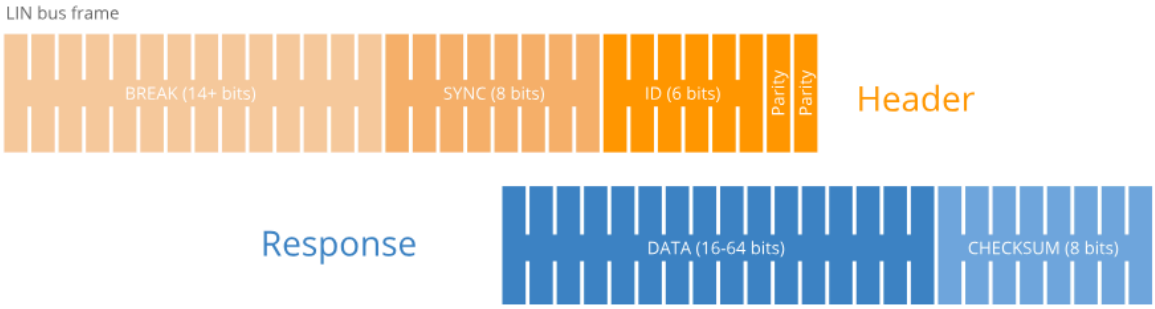
- A LIN master typically serves as gateway to the CAN bus
- LIN clusters have a single master - CAN can have multiple

How does LIN bus work?

A master node loops through each of the slave nodes, sending a request for information - and each slave responds with data when polled. The data bytes contain LIN bus signals (in raw form).



The LIN frame format:



The LIN bus message frame consists of a header and a response.

Typically, the LIN master transmits a header to the LIN bus. This triggers a slave, which sends up to 8 data bytes in response.

LIN Frame Fields:

Break: The Sync Break Field (SBF) Break is minimum 13 + 1 bits long (and in practice most often 18 + 2 bits). The Break field acts as a “start of frame" notice to all LIN nodes on the bus.

Sync: The 8 bit Sync field has a predefined value of 0x55 (in binary, 01010101). This structure allows the LIN nodes to determine the time between rising/falling edges and thus the baud rate used by the master node. This lets each of them stay in sync.

Identifier: The Identifier is 6 bits, followed by 2 parity bits. The ID acts as an identifier for each LIN message sent and which nodes react to the header. Slaves determine the validity of the ID field (based on the parity bits) and act via below:

1. Ignore the subsequent data transmission
2. Listen to the data transmitted from another node
3. Publish data in response to the header

Data: When a LIN slave is polled by the master, it can respond by transmitting 2, 4 or 8 bytes of data. The data length can be customized, but it is typically linked to the ID range (ID 0-31: 2 bytes, 32-47: 4 bytes, 48-63: 8 bytes). The data bytes contain the actual information being communicated in the form of LIN signals. The LIN signals are packed within the data bytes and may be just 1 bit long or multiple bytes.

Checksum: As in CAN, a checksum field ensures the validity of the LIN frame. The classic 8 bit checksum is based on summing the data bytes only (LIN 1.3), while the enhanced checksum algorithm also includes the identifier field (LIN 2.0).

AHB (Advanced High-performance Bus)

It is part of ARM's AMBA (Advanced Microcontroller Bus Architecture) family. It is widely used in System-on-Chip (SoC) designs to enable high-speed, high-bandwidth communication among components like CPU cores, memory, and high-speed peripherals. AHB is popular in embedded systems, especially where performance is essential.

Advantages

- **High Throughput and Low Latency**:
    - AHB is designed to handle high data rates with low latency, thanks to its pipelined architecture, which allows multiple data transfers to be processed simultaneously.
    - Its burst mode feature enables continuous data transfers without needing repeated handshakes.
- **Single Master-Multi Slave Architecture**:
    - AHB allows a single master or multiple masters to communicate with multiple slave components, such as memory controllers and peripherals, while maintaining efficient data flow.
    - The arbitration mechanism in AHB ensures efficient control of multiple masters, which is beneficial in complex SoCs.
- **Ease of Integration**:
    - AHB is part of the widely adopted AMBA standard, making it easier to integrate with other AMBA protocols like APB.
    - Many embedded systems rely on AMBA-compatible IP cores, making AHB a convenient choice for compatibility and ease of design.
- **Scalability**:
    - AHB is scalable to meet the demands of larger SoCs with higher data requirements.
- **Compatibility with Memory and Peripheral Interfaces**:
    - AHB supports both memory-mapped and register-mapped I/O, allowing for easy communication with both high-speed memory and lower-speed peripherals through other AMBA interfaces.

- This feature is especially useful in applications where different types of memory and devices require different communication speeds.

Disadvantages

1. **Limited Flexibility Compared to AXI**:
   - AHB has a simpler protocol with fewer features compared to the newer AXI protocol, which is also part of the AMBA family.
   - Unlike AXI, AHB does not support out-of-order transactions or separate read and write channels, which can be a disadvantage in highly parallelized designs requiring complex memory access patterns.
2. **Single-Clock Design**:
   - AHB operates on a single clock domain, which can limit its flexibility in systems where multiple clock domains are present.
   - For systems with components running at different clock speeds, additional clock domain crossing logic may be required, adding complexity.
3. **No Support for Advanced Error Checking**:
   - AHB lacks some of the advanced error-checking and response mechanisms found in other protocols like CAN or AXI.
   - In applications requiring critical error reporting, this can be a limitation, and additional error-handling mechanisms may be necessary.
4. **Complex Arbitration for Multi-Master Systems**:
   - Although AHB supports multiple masters, arbitration can become complex and may introduce overhead in designs with many competing masters.
   - Managing priorities and resolving contention between masters requires careful design and optimization to avoid bottlenecks.

Applications and Use Cases

- Primarily used for connecting high-speed components within an SoC, such as CPU, DMA, and memory subsystems.

- Typically paired with other AMBA protocols like APB for lower-speed peripherals to create a complete embedded system solution.

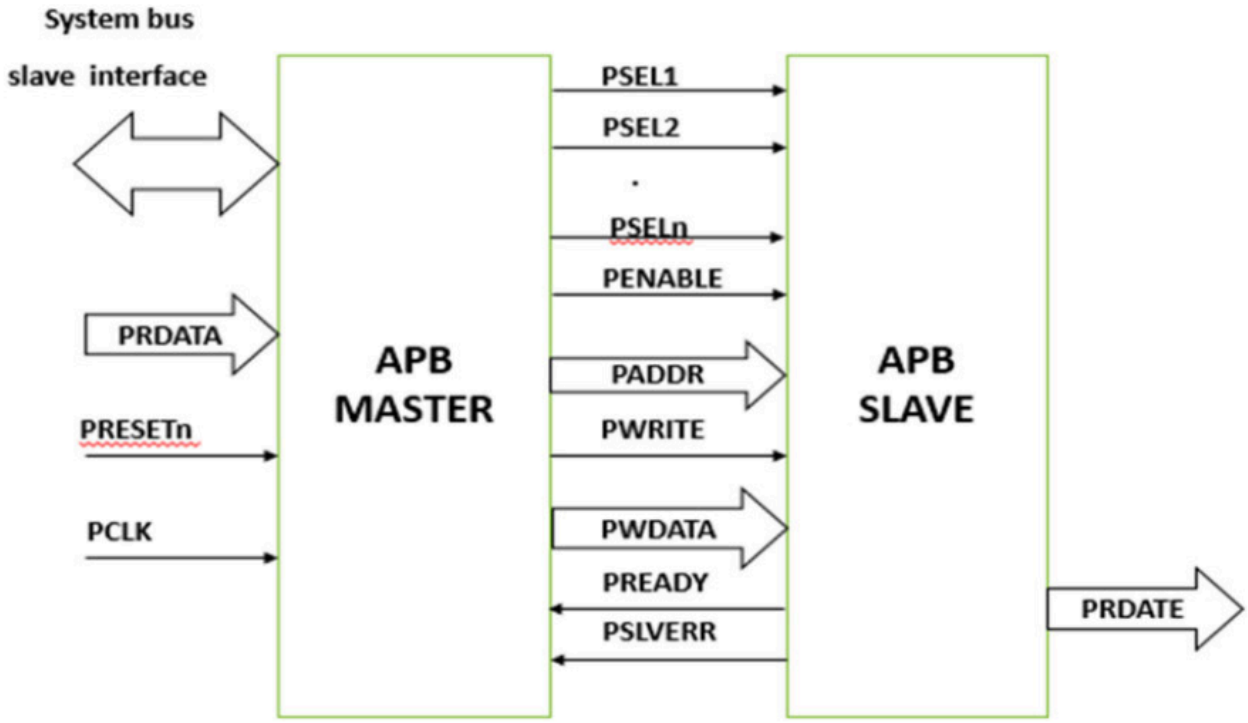
Summary

AHB is a high-performance, high-throughput bus protocol that is suitable for complex SoC designs where speed and bandwidth are priorities. However, it can lack the flexibility and advanced error-handling features of CAN

AMBA APB Protocol:

The AMBA APB (Advanced Microcontroller Bus Architecture Advanced Peripheral Bus) protocol is a low-power, simple communication protocol designed by ARM, typically used for connecting low-speed peripherals in

embedded systems. APB is specifically aimed at peripheral communications within SoCs (System-on-Chips) and embedded systems, where low-power consumption and simplicity are prioritized.



Key APB Signals and Their Functions:

- **PADDR:** Address bus for selecting a specific register within the peripheral.
- **PWDATA:** Data bus for write data.
- **PRDATA:** Data bus for read data.
- **PSEL:** Peripheral select signal, used to enable the specific peripheral on the bus.
- **PENABLE:** Enable signal to indicate the second phase of the transfer, where data can be transferred to/from the peripheral.
- **PWRITE:** Write signal, indicating a write operation when high and a read operation when low.

Basic APB Operation Phases:

APB transactions consist of a simple two-phase handshake:

1. Setup Phase: The address, control signals, and select signals are set up, with `PENABLE` set low.
2. Enable Phase: `PENABLE` is asserted high, indicating that the peripheral should complete the data transfer.

After the transfer, the APB returns to the setup phase for the next transaction. Each read or write cycle typically completes within two clock cycles.

Typical APB Transaction Types:

Read Transaction: During the setup phase, `PWRITE` is low, and the address is placed on `PADDR`. In the enable phase, data is transferred from the peripheral to `PRDATA`.

Write Transaction: During the setup phase, `PWRITE` is high, and data is placed on `PWDATA`. In the enable phase, data is transferred from `PWDATA` to the selected peripheral register.

Why did we choose CAN/LIN over the APB protocol:

Automotive Design: CAN/LIN are designed specifically for automotive environments to handle noise and interference, while APB is intended for internal, on-chip communication in embedded systems.

Distance Support: CAN/LIN can transmit data over long distances within a vehicle, whereas APB is limited to short, on-chip communication.

Multi-Node Support: CAN/LIN allows multiple devices to communicate on the same bus, ideal for distributed automotive systems; APB follows a simple master-slave model with limited device support.

Error Detection: CAN has advanced error detection and fault tolerance essential for automotive reliability, while APB has minimal error-handling capabilities.

Real-Time Communication: CAN supports prioritized, real-time communication, crucial for automotive functions, whereas APB does not prioritize urgent, time-sensitive messages.

Data Rate Flexibility: CAN/LIN provide flexible data rates suited for both critical and non-critical automotive functions, while APB operates at fixed, low-speed on-chip communication rates.

Industry Compliance: CAN/LIN are standardized and widely adopted in automotive industries, ensuring interoperability, while APB lacks compliance with automotive standards.

Single-Wire Operation: LIN can operate over a single wire for lightweight, cost-effective communication; APB requires more complex wiring and lacks single-wire support.

Robustness: CAN/LIN is built to withstand automotive environmental challenges, unlike APB, which is suited for controlled, on-chip environments.

Scalability: CAN/LIN scale effectively with complex vehicle systems, while APB is limited to simple internal communication within a chip.

UART:

Using a UART (Universal Asynchronous Receiver/Transmitter) module offers several advantages, as highlighted in the paper:

1. **Ease of Control for Embedded Devices:** UART is a straightforward solution to control devices with a serial interface. Since it is commonly used

in embedded systems, you can reliably communicate between modules on your FPGA-based project .

2. **Master-Slave Configuration Capability:** The UART module implemented on an FPGA can act as a master to control other modules configured as slaves, allowing the object detection unit to effectively manage signals sent to the braking system .

3. **Compatibility with Industrial Equipment:** Serial protocols like UART remain prevalent in industrial applications, which adds robustness and compatibility with other industrial modules, making it suitable for communication in safety-critical applications like braking systems .

4. **Lower System Complexity and Administrator Privileges:** UART does not require frequent reconfiguration or elevated access permissions, which simplifies setup and operation. This makes it more practical for systems that must run continuously without needing frequent intervention .

Disadvantages:

Using UART for communication between the object detection unit and the braking system also has some potential drawbacks:

1. **Limited Data Rate:** UART operates at relatively low data rates compared to other communication protocols, which might be a bottleneck if your object detection system requires rapid data transfer to the braking system for real-time responses. This could impact performance in time-sensitive applications .

2. **No Built-In Error Checking:** Basic UART does not include sophisticated error-checking mechanisms beyond simple parity checks, which may lead to undetected errors in critical safety systems. For a

braking system, robust error detection and correction might be crucial to prevent malfunction due to corrupted data .

3. **Lack of Synchronization:** UART is asynchronous, meaning it does not have a shared clock between sender and receiver. This can lead to issues if there is any clock drift, causing misinterpretation of data over time, especially over longer communication distances or with higher baud rates .

4. **Full-Duplex Limitation:** Traditional UART only supports half-duplex communication, where data can only flow in one direction at a time. While some implementations may support full-duplex, it is often limited, which could restrict simultaneous communication between your modules .

5. **Signal Interference Over Long Distances:** UART is generally intended for short-range communication. For longer distances, signal degradation or interference may occur, especially if noise is present, which could lead to data inaccuracies. This may be a concern if the modules are far apart within your design .

SPI Protocol (Serial Peripheral Interface)

SPI is a synchronous, full-duplex communication protocol used mainly for short-distance communication in embedded systems. It allows high-speed data transfer between a microcontroller and peripheral devices like sensors, displays, and memory chips.

- **Features:**

- Master-Slave Architecture: Consists of one master device and one or more slaves.

- Four-Wire Interface: Uses four lines — MOSI (Master Out Slave In), MISO (Master In Slave Out), SCLK (Serial Clock), and SS (Slave Select).

- Full-Duplex Communication: Simultaneous data transfer in both directions.

- High Speed: Suitable for high-speed data transmission, typically ranging from several MHz up to tens of MHz.

- **Advantages:**

- High Data Rate: SPI can achieve faster data rates than I2C or UART due to its synchronous nature.

- Simple Hardware: Requires minimal overhead with a straightforward four-wire connection.

- Full-Duplex Operation: Allows data to be sent and received simultaneously.

- **Disadvantages**

- No Standard Error Checking: SPI lacks built-in error-checking mechanisms.

- Limited Distance: Best for short-distance communication.

- Multiple Slave Support Complexity: Requires additional SS lines for each slave, making the wiring complex in large setups.

Proposal for Optimized Communication System in Automotive Safety Project

Objective:
This project aims to design and implement an optimized communication system for an automotive safety application, integrating **CAN (Controller Area Network)** and **LIN (Local Interconnect Network)** protocols. This system will support efficient communication between sensors, the main module, the object detection unit, and the braking system unit, ensuring robust performance while balancing cost and power efficiency.

Overview:
The communication network in this safety system plays a crucial role in real-time monitoring and response, especially for features like **collision detection** and **autonomous braking**. To achieve optimal data flow and reliable communication, this project will utilize CAN and LIN protocols in tandem, leveraging their unique strengths to fulfill different functional requirements across the system.

Solution Design

The network will comprise two primary communication pathways:

- **CAN Protocol** for high-priority, real-time data exchanges between the object detection unit and the braking system unit.
- **LIN Protocol** for low-speed, low-priority communications with secondary sensors to offload traffic from the CAN bus and reduce power consumption.

Advantages of the CAN-LIN Hybrid Approach

1. **Cost Efficiency**
   CAN will only be implemented for essential systems, while LIN is used with simpler sensors, minimizing the overall system cost without compromising performance.
2. **Optimized Bandwidth Allocation**
   LIN reduces bandwidth demands on CAN, allowing it to focus on safety-critical tasks like collision alerts and braking commands. This hybrid approach ensures efficient and targeted use of network resources.

3. **Increased System Reliability**
   CAN's high speed and robust error-handling capabilities ensure reliable communication for critical commands, while LIN's slower speed and simpler structure provide a stable network for non-critical components without interfering with CAN.
4. **Enhanced Flexibility and Modularity**
   The hybrid approach supports flexible expansion, with CAN and LIN sections independently scalable, facilitating the addition of sensors or modules as needed.
5. **Energy Savings**
   LIN's low-power capabilities conserve energy, beneficial for sensors that do not need continuous monitoring, thereby extending overall system efficiency.
6. **High Speed and Robustness for Critical Functions** can handle high-speed, time-sensitive communication with robust error detection, making it ideal for safety-critical functions like collision detection and ABS. It can ensure reliable and quick responses, which are crucial for preventing accidents.
7. **Reduced Wiring Complexity** LIN's single-wire design can reduce the overall wiring complexity for less critical components, which is beneficial in automotive applications where space and weight are considerations.
8. **Seamless Integration for Advanced Functionality** CAN can prioritize messages for real-time collision detection and ABS functions, while LIN can handle low-priority tasks, creating an efficient and integrated network. This division allows the system to focus processing power where it's most needed without wasting resources on non-critical communications.

References:

- https://www.researchgate.net/figure/Block-diagram-of-APB-Protocol_fig2_374432325

- https://www.csselectronics.com/pages/lin-bus-protocol-intro-basics
- https://ieeexplore.ieee.org/document/7342320
- https://www.typhoon-hil.com/documentation/typhoon-hil-software-manual/References/can_bus_protocol.html
- http://esd.cs.ucr.edu/webres/can20.pdf