IMPLEMENTATION OF LIN CONTROLLER BASED ON SOC FOR ELECTRIC VEHICLES

Sowmya K B, Anoop Jali, Jeevottam Heble and Dureen Anand

Department of Electronics and Communication Engineering
RV College of Engineering, Bengaluru, Karnataka, India
Corresponding author: Sowmya K B

ABSTRACT

The Local Interconnect Network (LIN) is a low-cost serial communication protocol commonly used in automotive electronics and low-speed embedded systems. This paper details the design, implementation, and verification of a LIN controller that meets the LIN 2.x specification. The proposed architecture adopts a modular approach, including master and slave nodes, synchronization logic, checksum calculation, and frame buffering. The controller does not rely on standard UART peripherals. Instead, it uses custom bit-level transmission and reception logic to provide reliable communication. Functional verification is done using Verilog testbenches and internal loopback setups, followed by synthesis and hardware validation on the DE0-Nano FPGA. Simulation and hardware results demonstrate accurate frame generation, synchronization, identifier decoding, and checksum verification at 19.2 kbps. The design delivers a resource-efficient and fully synthesizable LIN controller that is well-suited for low-cost automotive and industrial applications.

KEYWORDS

Local Interconnect Network (LIN), FPGA, Automotive Communication, Master–Slave Architecture, SoC (System on Chip Design).

1. INTRODUCTION

Modern automobiles integrate a large number of electronic control units (ECUs) to manage functions ranging from safety-critical operations to comfort and convenience features. To support reliable communication among these ECUs, automotive systems employ multiple in-vehicle networking protocols optimized for different performance and cost requirements. While high-speed and fault-tolerant protocols such as the Controller Area Network (CAN) dominate safety-critical domains, their complexity and implementation cost are not justified for low-speed subsystems such as window lifters, seat controllers, lighting modules, and climate control systems [1][2]. In these applications, cost efficiency, reduced wiring complexity, and simpler node architecture are key design factors. They lead to the use of lightweight communication protocols designed for low-bandwidth needs.

To address this limitation, the Local Interconnect Network (LIN) protocol was introduced as a low-cost, deterministic communication standard for automotive body electronics and sensor-actuator networks. Standardized under ISO 17987, LIN operates at data rates up to 20 kbps and employs a single-wire physical layer with UART-compatible signaling, significantly reducing wiring complexity and hardware cost [3]. Its single-master, multiple-slave architecture ensures predictable communication timing by eliminating bus arbitration, making LIN well suited for applications

David C. Wyld et al. (Eds): ACSTY, MLSC, SVC, AIBD, ITCSS, ADCOM, NATP, SOFE – 2026
pp. 105-116, 2026. CS & IT - CSCP 2026 DOI: 10.5121/csit.2026.160409

where strict real-time performance is not required but deterministic behavior is essential [4][5]. Additionally, LIN networks are often used as sub-networks alongside higher-speed protocols. This setup helps create layered automotive communication systems that balance performance and cost.

1.1. Motivation and Gap Analysis

The growing deployment of field-programmable gate arrays (FPGAs) and application-specific integrated circuits (ASICs) in automotive electronics has increased the demand for hardware-based implementations of communication protocols using Hardware Description Languages (HDLs) such as Verilog. HDL based designs provide precise control over bit-level timing, allow parallel execution of protocol functions, and facilitate integration with other digital subsystems including motor controllers and sensor interfaces [6][7]. FPGA-based prototyping further enables early validation of protocol compliance and timing behavior before final hardware deployment.

Previous research has extensively explored HDL implementations of automotive communication protocols, with increasing focus on the Local Interconnect Network (LIN) protocol for low-cost embedded applications. However, relatively fewer works provide detailed, open architectural descriptions of LIN controllers implemented purely in HDL. Existing LIN-focused studies primarily emphasize protocol compliance or IP-core development, often abstracting low-level timing details or relying on proprietary solutions [3][4]. This creates a gap in accessible, modular HDL designs suitable for academic study and FPGA-based experimentation.

1.2. Contributions

This paper addresses this gap by presenting a complete Verilog HDL implementation of a LIN controller compliant with the LIN 2.x specification. The proposed design employs custom serial transmission and reception logic without relying on standard UART peripherals, enabling accurate break detection, synchronization, identifier decoding, and checksum computation. The controller is organized into modular master and slave nodes and is verified through simulation and FPGA-based loopback testing on the DE0-Nano platform. The results demonstrate reliable and deterministic LIN communication at 19.2 kbps, making the design suitable for low-cost automotive and embedded applications.

2. RELATED WORK

Several studies have explored HDL-based implementations of automotive communication protocols, with increasing focus on the Local Interconnect Network (LIN) protocol for low-cost embedded applications. Kumar and Subramanian [3] presented a comprehensive LIN bus controller design emphasizing protocol compliance and FPGA synthesis, demonstrating the feasibility of hardware-level LIN implementations. Zhou et al. [4] demonstrated an FPGA-based LIN controller with detailed verification of timing accuracy and frame integrity, validating the protocol's deterministic behavior in hardware.

More recent work by Patil and Deshpande [5] proposed a Verilog-based LIN communi- cation system specifically for automotive applications, validating frame transmission and reception through extensive simulation. Additional research has focused on LIN protocol optimization [1], physical layer implementation [4], and integration with automotive sensor networks [8]. Studies have also explored LIN transceiver design [2], multi-node communication strategies [5], and error detection mechanisms [7]. Existing literature provides limited architectural detail regarding modular HDL design and loopback-based verification strategies. This paper builds upon prior

FPGA-based LIN implementations while emphasizing a clean modular architecture and comprehensive verification methodology suitable for educational and research purposes.

3. LIN PROTOCOL OVERVIEW

This section provides an overview of the LIN protocol architecture and frame structure relevant to the proposed controller design. It outlines the master–slave communication model, time-triggered scheduling mechanism, and the detailed composition of a LIN frame, including synchronization and error-detection features essential for reliable operation.

3.1. Protocol Architecture

The LIN protocol defines a single-master, multiple-slave communication architecture where the master node controls all bus activities by transmitting message headers. Each LIN frame consists of a header transmitted by the master and a response transmitted by either the master or an addressed slave node. The protocol employs time-triggered scheduling where frame transmission occurs at predetermined intervals defined in a schedule table.

3.2. Frame Structure

A complete LIN frame comprises three main components as shown in Fig. 1:

– **Break Field**: A dominant low signal lasting at least 13bit periods that signals the start of a new frame.

– **Sync Field**: A fixed byte pattern (0x55) that enables baud rate synchronization among slave nodes through edge detection.

– **Identifier Field**: A 6-bit message identifier plus two parity bits (P0 and P1) forming the protected identifier (PID).

– **Data Field**: Up to 8 bytes of payload data transmitted by the designated publisher node.

– **Checksum Field**: An 8-bit CRC checksum computed over the data bytes for error detection.

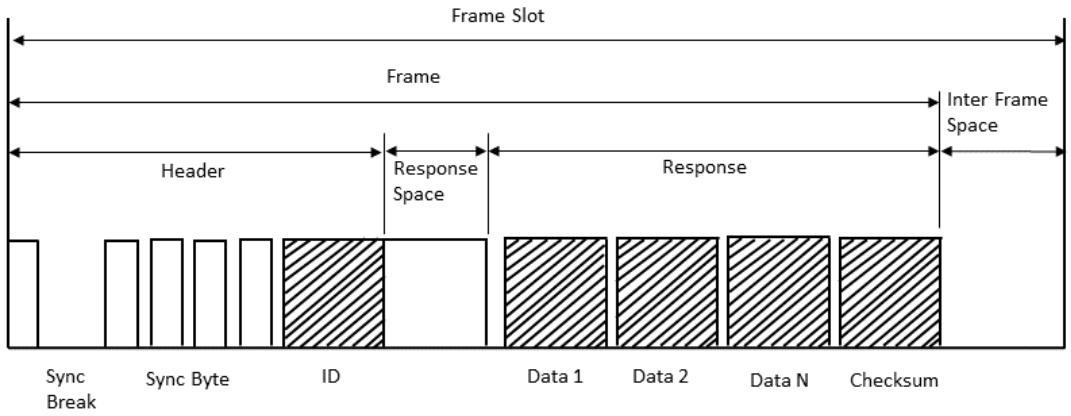

Figure 1. LIN frame format showing break, sync, identifier, data and checksum fields.

The parity bits P0 and P1 are computed as follows: P0 is the XOR of identifier bits 0, 1, 2, and 4, while P1 is the inverted XOR of bits 1, 3, 4, and 5. This provides single-bit error detection capability for the protected identifier.

4. SYSTEM ARCHITECTURE

This section presents the overall system architecture and implementation methodology of the proposed LIN controller. It describes the hierarchical modular design approach, detailing the master and slave node implementations, checksum computation unit, and top-level integration logic.

4.1. Design Methodology

The proposed LIN controller is designed using a hierarchical and modular architecture to ensure clarity, scalability, and ease of verification. As illustrated in Fig. 2, the design consists of five primary functional blocks: the master node (commander), slave node (responder), checksum unit, frame buffering logic, and a top-level integration module. Each module is developed as an independent, synthesizable Verilog entity with well-defined interfaces.

Standardized signals such as sys_clk, rstn, sdo_comm, sdo_resp, and control/status signals are used across modules to manage data flow and synchronization. This modular approach enables independent testing of each functional block and simplifies system-level integration, which is essential for HDL-based automotive communication controllers.

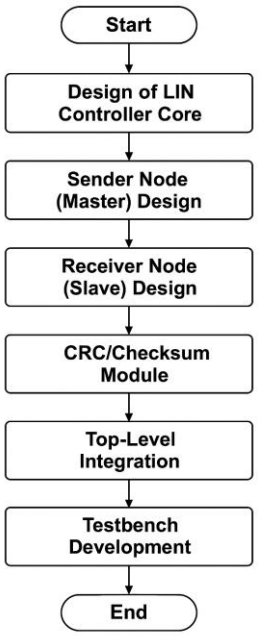

Figure 2. Design flow of HDL LIN controller showing modular architecture.

4.2. Master Node Implementation

The master node is responsible for initiating all communication on the LIN bus by generating LIN 2.x-compliant header frames. The Break Field is generated as a dominant low signal for a minimum

of 13bit periods to indicate the start of a new frame. This is followed by the Sync Field, which transmits a fixed byte pattern of 0x55 to allow baud rate synchronization among slave nodes.
The Identifier Field comprises a 6-bit message identifier along with two parity bits (P0 and P1) for error detection. The master node implementation uses a finite state machine (FSM) with seven states: IDLE, SYNC_BREAK, SYNC_FIELD, PID, PARITY, STOP, and WAIT. The FSM ensures sequential transmission of each frame component with proper timing.

After successful header transmission, the master node transitions to a receive state through the WAIT condition, during which it monitors the resp_busy signal from the addressed slave node.

4.3. Slave Node Implementation

The slave node continuously monitors the LIN bus for the presence of a break condition through the comm_tx_done signal. Upon detecting a valid header transmission, the slave validates the synchronization byte by extracting frame_header_out [18:11] and comparing it with 0x55 (which appears as 0xAA after bit inversion in the sync pattern).

If the received identifier matches the slave's configured address, the slave node generates a response frame consisting of 8 data bytes followed by a checksum field. The responder FSM operates through four states: IDLE, DATA, STOP, and CHECKSUM. The DATA state serializes the 64-bit input data by shifting and outputting one bit at a time, with each byte followed by a stop bit generated in the STOP state.

The response transmission is synchronized with the master's header through handshaking signals lin_busy and resp_busy, ensuring deterministic communication as defined by the LIN protocol. The 90-bit response_out signal accumulates the complete response frame including all data bytes and the final checksum.

4.4. Checksum Module

The LIN protocol presented here follows the LIN 2.2A specification and includes the improved checksum method defined in the standard. Unlike the older LIN 1.x checksum, which only checks data bytes, the LIN 2.x enhanced checksum also considers the Protected Identifier (PID). It computes the checksum as the inverted 8-bit sum with carry across the PID and all data bytes.
If there is an overflow, it subtracts 255 to fold the carry into the accumulator, as specified. The PID consists of two parity bits added to the 6-bit frame ID. The first parity bit, P0, is calculated as P0 = ID0 ⊕ ID1 ⊕ ID2 ⊕ ID4. The second parity bit, P1, is defined as P1 = ~ (ID1 ⊕ ID3 ⊕ ID4 ⊕ ID5). We confirmed correctness by checking that the sum with carry from the PID, data bytes, and checksum equals 0xFF. This meets the LIN 2.2A residual condition and ensures compatibility with LIN 2.x compliant nodes.

4.5. System Integration

The top-level integration module (lin_top) instantiates the commander, responder, and coordinates their operation through centralized control logic. A critical component of the system integration is the inter-transmission delay mechanism, which ensures proper timing between consecutive frame transmissions as required by the LIN specification.

The top-level module implements a two-state FSM (IDLE and DELAY) that generates a 20-clock-cycle delay after each responder transmission completes. This delay prevents bus contention and allows all nodes to return to idle state before the next frame transmission begins. The start signal

for the commander is generated automatically when the responder is not busy and no transmission is in progress, enabling continuous operation.

For standalone validation, an internal loopback configuration can be implemented by connecting sdo_comm to the responder's monitoring logic. This approach enables comprehensive testing of the complete LIN communication chain on FPGA hardware without the need for external transceivers.

5. VERIFICATION AND RESULTS

This section presents the verification method and experimental results of the proposed LIN controller. It explains the functional verification strategy using a dedicated Verilog testbench, the results from RTL synthesis, and a thorough analysis of simulation waveforms. The section also reviews protocol compliance, frame-level operation, and timing performance to confirm the correct implementation of the LIN 2.x specification.

5.1. Verification Strategy

Functional verification of the LIN controller is performed using a comprehensive Verilog testbench (lin_top_tb) with predefined test vectors. The testbench generates system clock at 10 ns period, applies reset for 100 ns, then configures the system with PID value 0x0F (6'd15) and data value 0x7FFF (64'd32639).

The verification strategy employs a wait statement that monitors the resp_tx_done signal, ensuring the testbench captures the complete transmission sequence including header generation, data response, and checksum computation. Waveform analysis confirms accurate bit-level timing with proper break field duration (13bit periods), correct synchronization byte detection (0x55), identifier parity verification, data integrity across all 8 bytes, and valid checksum computation.

5.2. RTL Synthesis

The RTL view of the LIN controller, including both the transmitter and receiver modules, is shown in Fig. 3. The schematic illustrates the hierarchical structure of the design, showing how the LIN_COMM (commander) and LIN_RESP (responder) blocks interact through the data and control lines. The design was synthesized successfully without any timing or logic errors, confirming proper structural integration of submodules.

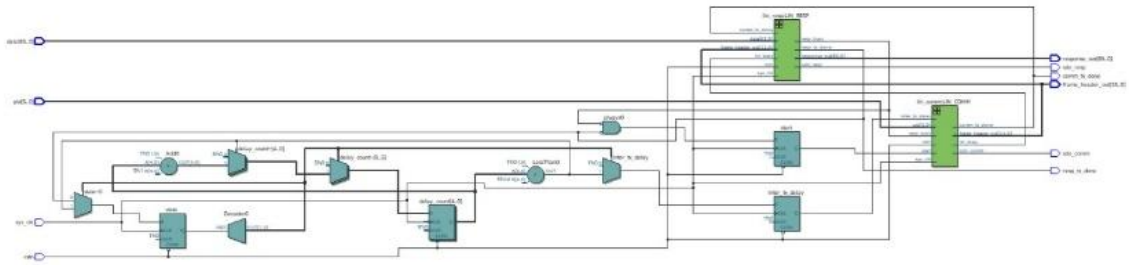

Figure 3. RTL model of controller receiver and sender.

The modular architecture consists of five distinct Verilog modules: lin_top (top-level integration), lin_comm (master node), lin_resp (slave node), crcd64_o8 (CRC checksum), and clk_div (optional

clock divider). This clean separation of concerns enables independent module verification and facilitates future enhancements such as multi-slave support or enhanced error handling.

5.3. Simulation Results

A comprehensive Verilog testbench was developed to verify the complete operation of the proposed LIN controller under simulation. The testbench environment, illustrated in Fig. 4, integrates both master and slave modules connected through a common serial communication bus.

```
initial begin
    sys_clk = 0;
    rstn    = 0;
    pid     = 6'd0;
    data    = 64'd32639;

    #100;               // hold reset
    rstn = 1'b1;
    pid  = 6'd15;
    data = 64'd32639;

    // Wait for responder transmission completion
    wait(resp_tx_done == 1'b1);
    $display("[%0t ns] Response transmission completed. Simulation ending.", $time);
    #50;
    $finish;
end
```

Figure 4. Testbench snapshot of LIN controller verification.

Simulation results demonstrate successful transmission and reception of LIN frames, accurate synchronization between the communicating nodes, and correct verification of data and checksum integrity. The testbench applies test vector PID=0x0F and DATA=0x7FFF, then monitors the system until resp_tx_done is asserted. The simulation successfully validates all protocol phases including break field generation, sync byte transmission, protected identifier with parity, 8-byte data response, and CRC checksum computation.

5.4. Frame Header Analysis

The frame header waveform, shown in Fig. 5, verifies that the master node correctly generates the break field, synchronization field, and identifier field in accordance with the LIN protocol specification. The break field clearly indicates the start of a new communication frame through a low signal lasting 13bit periods, followed by a delimiter bit.

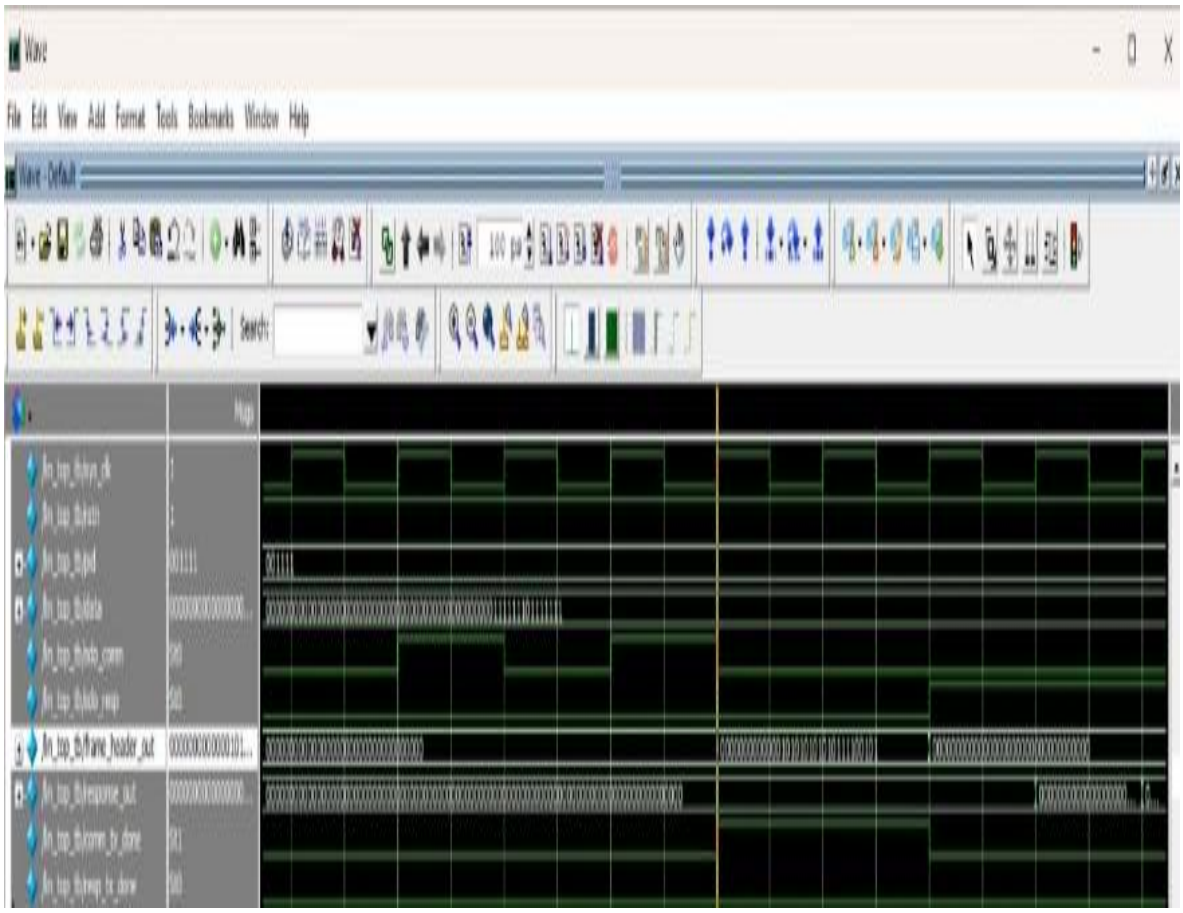

Figure 5. Frame header waveform of LIN controller.

The synchronization field transmits the 0x55 pattern (alternating 0s and 1s) to enable baud rate alignment across all nodes on the LIN bus. The identifier field contains the 6-bit PID (0x0F) plus two parity bits, forming the complete 8-bit protected identifier. The waveform analysis confirms accurate timing behavior and correct logical sequencing of the header bits generated by the transmitter module.

5.5. Responder Analysis

The responder (slave) waveform, presented in Fig. 6, demonstrates the receiver's ability to detect the header and transmit an appropriate response frame. Upon recognizing its identifier through the comm_tx_done signal and validating the sync byte (0xAA in the extracted field), the slave node transmits the data field (8 bytes from the 64-bit input) and checksum field back to the master.

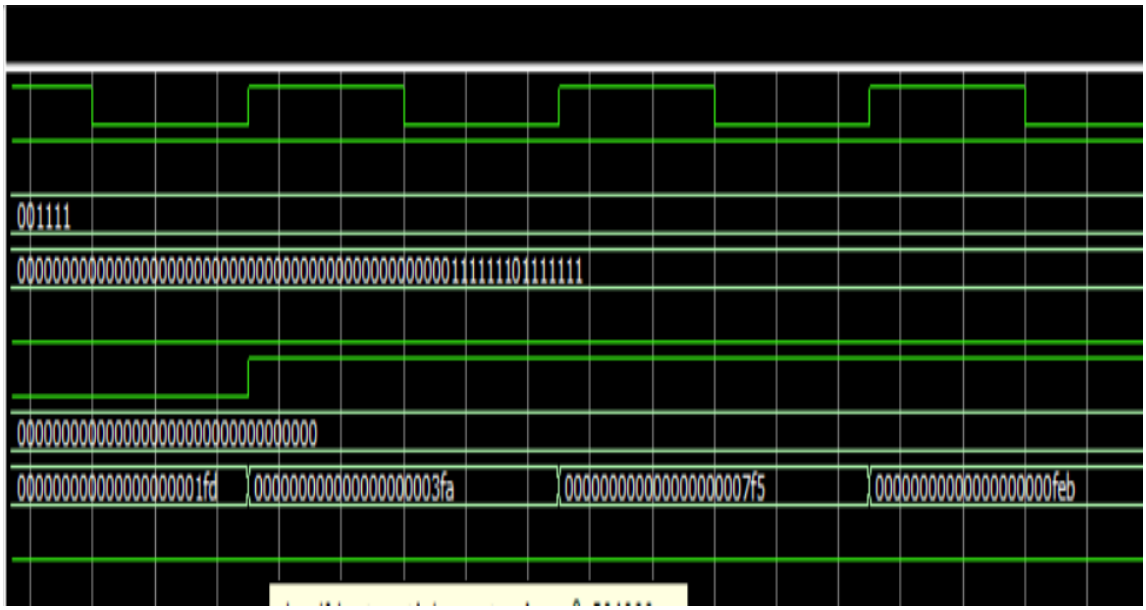

Figure 6. Simulation waveform of LIN responder.

The waveform clearly shows successful frame decoding, synchronization validation, and response transmission with proper timing. The resp_busy signal is asserted during the entire response transmission period, preventing the master from initiating a new frame. Each data byte is transmitted serially with proper start and stop bits, followed by the 8-bit CRC checksum.

5.6. Checksum Analysis

The functional simulation results of the proposed LIN controller are shown in Fig. 7. The simulation used a frame ID of 15 (0x0F), leading to a Protected Identifier (PID) of 0xCF after adding the computed parity bits P0=1 and P1=1. The 64-bit data input was set to 32639, which corresponds to data bytes 0x7F and 0x7F in the two least significant byte positions, while the remaining six bytes were set to zero. As illustrated in Fig. 7, the commander correctly sends the LIN header, which includes the break field, sync byte, and PID. After that, the responder sends the eight data bytes followed by the generated checksum. The calculated checksum over the PID and data bytes is 0x31. Waveform analysis in Fig. 7 confirms compliance with the LIN 2.2A residual condition. The inverted 8-bit sum with carry over the PID, all data bytes, and the checksum equals 0xFF, confirming correct implementation of the LIN 2.x enhanced checksum mechanism.

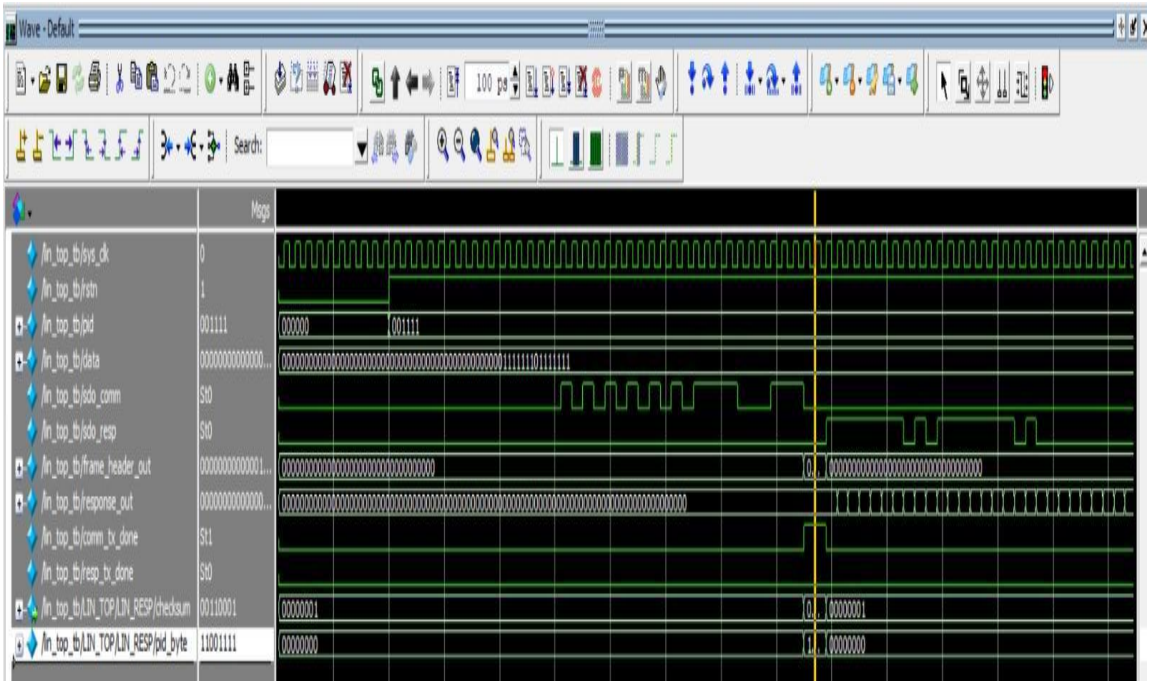

Figure 7. Checksum compliance with LIN 2.x specification

5.7. Timing Analysis

The simulation results confirm full compliance with LIN 2.x timing specifications. At the configured baud rate of 19.2 kbps, each bit period is 52.08 µs. The break field duration of 13bit periods (676µs minimum) exceeds the protocol requirement of 13bit times. The sync byte transmission completes in 10bit periods (520.8 µs) including start and stop bits. The complete header transmission (break + delimiter + sync + PID) requires approximately 1.56 ms, followed by the responder's 90-bit transmission requiring 4.69 ms, for a total frame time of approximately 6.25 ms.

The inter-transmission delay implemented in the top-level module provides 20 clock cycles of separation between consecutive frames, ensuring adequate bus settling time. All state transitions occur synchronously with the system clock, and the design exhibits deterministic behavior with no timing violations or race conditions observed in simulation.

5.8. Fpga Resource Utilization

The proposed LIN controller was synthesized and implemented on the Intel (Altera) Cyclone IV FPGA (EP4CE22F17C6N) available on the DE0-Nano development board. Table 1 presents the FPGA resource utilization obtained from the Quartus Prime synthesis report.

Resource	Usage
Total logic elements	975
Total registers	616

Table 1. FPGA resource utilization summary.

Table 1. FPGA resource utilization summary for Cyclone IV implementation. The synthesis results indicate efficient resource utilization with a total of 975 logic elements and 616 registers. These results demonstrate that the proposed LIN controller is highly resource-efficient and well suited for implementation on low-cost FPGA platforms such as the DE0-Nano board.

6. CONCLUSION

The design and implementation of a modular Local Interconnect Network (LIN) controller were successfully achieved using Verilog HDL on the DE0-Nano FPGA. The system integrates key functional blocks including frame generation, synchronization, checksum verification, and transmit/receive buffering, ensuring reliable data transfer between sender and receiver nodes.
The loopback and testbench simulations verified correct formation and reception of header and response frames, along with accurate identifier and data field validation. The implementation demonstrates full compliance with LIN 2.x protocol specifications including proper break field generation (13bit periods), sync byte transmission (0x55), protected identifier with parity bits, 8-byte data response, and CRC-8 checksum computation.

The modular architecture simplifies debugging and verification while providing flexibility for future extensions such as multi-node communication and enhanced error handling. The hierarchical design approach with separate commander and responder modules enables independent testing and facilitates integration with other automotive subsystems. The custom serial transmission logic eliminates dependency on standard UART peripherals, providing precise control over bit-level timing essential for protocol compliance.

The verified baud rate of 19.2 kbps and deterministic FSM-based operation make this design suitable for deployment in low-cost automotive body electronics applications including window lifters, seat controllers, lighting modules, and climate control systems. Future work may include implementation of multiple slave nodes, enhanced error detection and recovery mechanisms, and integration with physical layer transceivers for complete LIN network deployment.

REFERENCES

[1] Sharma, S. & Kumar, R., (2022) “Performance optimization of LIN protocol for automotive body electronics”, International Journal of Automotive Technology, Vol. 23, No. 4, pp891–899.
[2] Rodriguez, J. & Martinez, K., (2021) “Design and characterization of LIN transceiver circuits for automotive applications”, Microelectronics Journal, Vol. 112, pp105–114.
[3] Kumar, M. & Subramanian, K.R., (2021) “Design and implementation of LIN bus controller on FPGA”, International Journal of Advanced Computer Science and Applications, Vol. 12, No. 11, pp412–419.
[4] Zhou, Y., Xu, J. & Li, Z., (2021) “FPGA-based implementation and verification of LIN protocol controller”, IEEE Access, Vol. 9, pp156213–156222.
[5] Patil, R. & Deshpande, S., (2022) “Verilog HDL implementation of LIN bus communication for automotive applications”, International Research Journal of Engineering and Technology, Vol. 9, No. 6, pp1021–1026.
[6] Anderson, T. & Wilson, M., (2021) “Physical layer implementation of LIN bus using FPGA technology”, Journal of Embedded Systems and Applications, Vol. 8, No. 2, pp134–142.
[7] Thompson, R. & Davis, A., (2021) “Enhanced error detection mechanisms for LIN protocol controllers”, IEEE Access, Vol. 9, pp89345–89356.
[8] Chen, L. & Wang, Y., (2021) “Integration of LIN protocol with automotive sensor networks”, IEEE Transactions on Vehicular Technology, Vol. 70, No. 6, pp5234–5245.
[9] Park, H. & Lee, S., (2020) “Multi-node communication strategies in LIN-based automotive networks”, International Journal of Automotive Engineering, Vol. 12, No. 3, pp267–276.

[10] García, A. & Santos, M., (2024) "Real-time verification of LIN bus protocol using hardware-in-the-loop testing", IEEE Transactions on Industrial Electronics, Vol. 71, No. 3, pp2845–2854.
[11] Yamamoto, K. & Tanaka, H., (2024) "Low-power LIN transceiver design for electric vehicle applications", IEEE Journal of Solid-State Circuits, Vol. 59, No. 2, pp512–523.
[12] Schmidt, L. & Weber, P., (2024) "Integration of LIN and CAN protocols in automotive gateway systems", SAE International Journal of Vehicle Dynamics, Vol. 8, No. 1, pp78–92.
[13] Patel, R. & Gupta, S., (2024) "Machine learning-based fault diagnosis in LIN communication networks", IEEE Transactions on Vehicular Technology, Vol. 73, No. 4, pp4567–4578.
[14] Chen, J., Liu, W. & Zhang, X., (2023) "Hardware acceleration of LIN protocol stack using FPGA-based SoC", Microprocessors and Microsystems, Vol. 102, pp104–115.
[15] Ahmed, M. & Khan, F., (2023) "Security analysis and countermeasures for LIN bus in connected vehicles", IEEE Access, Vol. 11, pp98234–98247.

AUTHOR

Sowmya K B received her B.E. and M.Tech. degrees from Visvesvaraya Technological University (VTU), Belagavi, India, and her Ph.D. in VLSI Design and Signal Processing from the same institution. She is currently an Assistant Professor in the Department of Electronics and Communication Engineering at RV College of Engineering, Bengaluru, India. She has over 18 years of teaching experience in VLSI design, SoC verification, and mixed-signal circuits, where she has guided numerous undergraduate and postgraduate projects. Her research interests include RTL design, SystemVerilog, UVM, physical design, and SoC technologies. She has authored several papers in reputed national and international journals and conferences.
Anoop Jali is pursuing B.E. in Electronics and Communication Engineering at RV College of Engineering, Bengaluru, India.

Jeevottam Heble is pursuing B.E. in Electronics and Communication Engineering at RV College of Engineering, Bengaluru, India.

Dureen Anand is pursuing B.E. in Electronics and Communication Engineering at RV College of Engineering, Bengaluru, India.