Local Interconnect Network (LIN)

1st Syed Rafsan Ishtiaque
Electronic Engineering
Hochschule Hamm-Lippstadt
syed-rafsan.ishtiaque@stud.hshl.de

2nd Muhammad Rohail Usman
Electronic Engineering
Hochschule Hamm-Lippstadt
muhammad-rohail.usman@stud.hshl.de

3rd Yigitxcan Aydin
Electronic Engineering
Hochschule Hamm-Lippstadt
yigitcan.aydin@stud.hshl.de

***Abstract*—Since the knowledge of network design is increasingly popular especially in the cyber-physical systems, the use of networks in automotive industry to increase and modify vehicle features (wiper control, airbag, door lock, etc.) are realized over networked electronic components and software. These innovations are implemented by tapping in information from actuators, sensors, and ECUs via networks. Correctly applied network technologies can be used to increase comfort and safety while keeping the cost low.**

Taking the low cost, longevity, safety and security in prime consideration, LIN (Local Interconnect Network) deals with sensors and actuators with very low cost of installation. As a hands-on development project, an arduino ecosystem will be demonstrated with one master node three slave nodes will exchange data via the LIN Node. Additionally, with the help of push buttons integrated on the master-slave nodes we can display LIN frame transfers on the oscilloscope.

***Index Terms*—LIN, communication, wireless, car industry, node**

I. Introduction

LIN, Local Interconnect Network, that tries to provide everything that we need to build a simple, low-cost bus tailored to the needs of the automotive industry. It is quite unique to know that it goes beyond the standard of OSI Layer 1 and 2. Having an aim to build simple, cheap, and reliable networks with relatively low-cost components. LIN is widely popular and present in many vehicle platforms, with the usage of 20+ LIN nodes per vehicle is not an exception. Up until 2010, the specification was maintained and advanced by the LIN consortium. A consortium originated with 5 major OEMs (original equipment manufacturers) with its first release of specification, LIN 1.0, in 1999. The consortium was disbanded on 31st December 2010 with its release of last LIN official consortium of version 2.2.A. This specification is considered stable and was transcribed and made available to the International Organization for Standardization (ISO). Later, ISO updated the specification in 2016, partitioned it out into multiple documents and made the documents available under the ISO 17987 Part 1-7 [1].

While LIN is more than OSI layer 1 and 2, main interests surround around the characterization of LIN Network. Since it aims at cost efficient, small sensor, actuators, and control networks. It is widely used where bandwidth is not the main criteria rather low-cost and accessibility. LIN is a serial byte-oriented network protocol, the smallest data unit to be transferred is one byte. We can run LIN on very low-cost ECUs. Theoretically, even without a quartz clock but only with an RC element. Moreover, the single wire connectivity intends to cater up to 40m length and can transmit up to 20Kb/s with a required operation voltage of 12V. The OSI layer 2 specifies a shared multiplexed communication, where all the participants share the single wire. The architecture is a master-slave setup with up to 16 nodes, one master and up to 15 slave nodes. Furthermore, within the time-trigger shared multiplexed communication the master initiates all network flows. The master is in control so that collisions and bus arbitration can be avoided. LIN typically supports the data size of 1-8 bytes. There is a set of error detection measures and diagnostic support included as part of the network standard. In the picture a typical architecture setup, on master node controls and steers all small lean participants within its domain over the single wire. For sure, while cheap if the master fails our communication will fail, master end wire act as a single point of failure, this is accepted in respect to the cost. If fault tolerance is needed the idea is to cater for it from a design and architecture perspective. Plus, there is also a set of error detection measures and diagnostic support included as part of the network standard.

Summing up to application areas, LIN is ideal for sensor networks used in low-cost body electronics. Typical applications are roof, door, window, seat, dashboard, airbag, climate control, engine, and steering wheel [2].

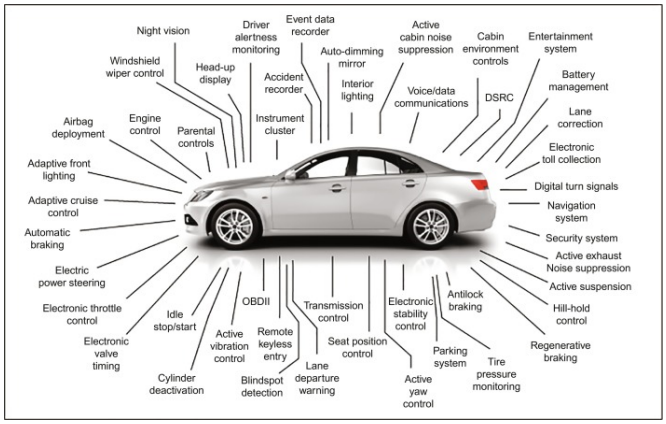

Fig. 1. Introduction of LIN [2]

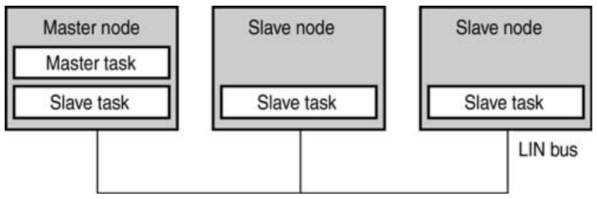

Fig. 2. Introduction of LIN [2]

II. LIN COMPARISON

Positioning LIN among the other currently popular protocols. Our graph is the bit rate speed on the Y-axis and the relative costs per node on the X-axis. The blocks stretch out across the X axis to highlight those costs can vary and are only a rough estimate just to make a comparison. LIN can be seen as roughly a third cheaper in implementation cost than the Can. Bandwidth of LIN is up to 20 Kilobits per second. Next is CAN, it exists in multiple versions, low speed CAN is operated between 40 kilobits per second and up to 125 kilobits per second, also known as CAN-B. The standard CAN-C reaches up to 1 megabit per second. There is one extension to CAN, not graphically demonstrated but worth mentioning, which is currently winning on popularity and that is CAN-FD (CAN Flexible Data rate). CAN-FD supports up to 12 megabits per second. The costs are slightly higher because it is a relatively new technology making it a little higher in installation operation costs, simply because experience in quantity is missing there. This will eventually change overtime with increased experience as well as increase in the demand of silicon. Last but not the least, which is also unfortunately not in the image, is FlexRay. It is a mixed time and event triggered protocol with safety in mind. It can be operated in redundant mode with 10 megabits mirrored over two wires or with a non-redundant setup, where each wire can run different data traffic speeds up to 10 megabits each reaching 20 megabits of a total gross rate [3] [**?**].

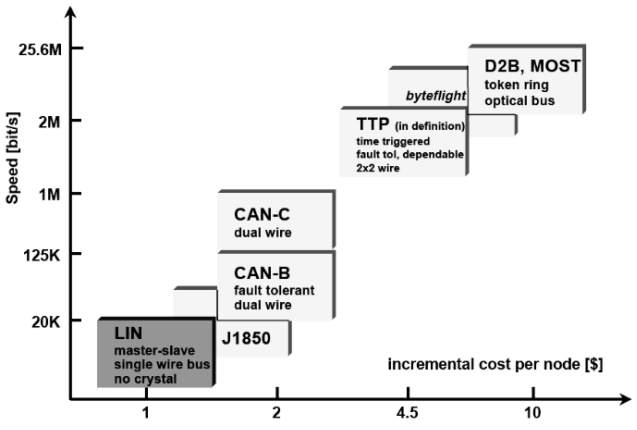

Fig. 3. LIN comparison [3]

III. ADVANTAGES OF USING LIN

In automotive we have different challenges to face, depending on the domain we are working on. Since the target is not build a single system but rather the aim is to go into mass production. So, a minor save is cost will result a lot when dealing in large quantities [1].

A. Automotive Domains Overview

The individual domains of automotive can be identified as, powertrain, chassis, driver assistance and safety, vehicle body control and human machine interface. The powertrain controls all components that generate the power which allows us to move the vehicle. The bandwidth requirements are actually quite low in powertrain. Though the latency matters and that is the reason why accuracy between component is very high, less 10 microseconds. Moreover, chassis covers the area close to the powertrain and deals with the maneuver of the car, for instance steering, breaking and suspension. The bandwidth requirements are between low to moderate. In addition, driver assistance and safety deals with the topic around personal protection, airbags, emergency brakes and hill decent control are some examples for it. Only lately ADAS (Advanced Driver Assistance System) have gained the focus of the industry because of the rapid the rapid growth and interest in autonomous driving. Especially the use of cameras and radars will increase the bandwidth needs in the near future significantly. Other than this, Vehicle body control or body is about driving support and comfort. Somehow indirectly body also contributes to safety without being safety critical. Also, body covers for example the car radio, its air conditioning, window functionalities, mirrors, doors, lights immobilizer and remote controlling of the car's radio and for infotainment system. Body has relatively low bandwidth needs as well as latency requirements to have a well working feature. It can be analysed that the body domain is actually less demanding, regarding network speeds. Last but not the least, HMI and infotainment control displays and interfaces to the human user. Using the infotainment content, for instance, the Internet or streaming music requires a higher and increased rate of bandwidth. Even though as human being, response rate requirements compared to the other domains are quite relaxed but for sure, still a good user interface should not feel too sluggish. As a human, anything below 250 milliseconds sometimes even 100 milliseconds are usually considered as instant reaction. So, in view of all this, LIN is targeted at vehicle body control body. A lot of distributed sensors and actuators are used within this area with relatively low bandwidth requirement. Costs does matter because there are many smaller features within the car that nowadays many customers also consider as standard or as granted [4] [5] [6].

B. RELATION OF LIN WITH UART

LIN networks are built on top of UART technology. LIN uses UARTs to communicate. Following is the comparison between LIN and UART which will make it easier to understand what drove the evolution of LIN, instead of simply using

UARTs with point-to-point connections. LIN is actually a set of specifications, while UARTs are specific devices. bandwidth of LIN is below 20 Kb per second while modern UARTs can easily sustain 3 Mb per second. LIN aims at multiple participants by default up to 16, UARTs connect point-to-point two participants and needs two wires each for transmission and reception. One major challenge with respect to costs, and costs really matter in automotive connecting 16 participants requires one wire plus power related cabling for LIN because it is a shared medium. While in UART, connecting point-to-point 16 devices over a UART could easily become its own management challenge and certainly will need 240 connections without counting for the power related cabling, 120 connections for transmission and same for reception. Next is, LIN uses UART plus specified transceivers, the transceivers are basic level shifters that are very low cost. Whereas UARTs are devices by itself, transceivers are recommended but not a must. Moreover, LIN transceivers work with 12 volts, it is an automotive standard designed to the battery voltage used across the car. UART's voltage is not standardized and is specified by the designer. Popular voltages are, 3 V, 5 V and 25 V, so mixing those voltages will probably the death of some of the devices. Furthermore, smallest transferable data unit is equal across both networks and that is 7 to 9 bits, a Byte certainly. The architecture of LIN is a time triggered master-slave setup. UARTs are by default bi-directional, full duplex devices. In addition, the fault tolerance is an issue with LIN. Having a single master node and one single wire makes this setup not fit by default for safety critical usage. Cost is for sure, the major win for LIN and this is why LIN was actually designed. While UARTs are super cheap, LIN makes things in a connected scenario even cheaper. Wiring and interface costs within the overall application makes the only difference.

Summing up, LIN seems to have an upper-hand, LIN utilizes the great universality of UARTs. LIN compasses our UART devices and applies them to its best use. UARTs are part of LIN and therefore plays an important role as well [4] [5].

IV. WORKFLOW OF LIN

The media access is closely associated with the physical layer, it specifies the agreed strategy to access the physical medium. The strategy defines how to manage access and deal with communication collisions between participants. For instance, if there are multiple people in group call and everyone speaks simultaneously then no one will clearly be able to listen each other. Therefore, in LIN nomination of one master like managing the group call. The master acts as a conductor. Based on a defined arrangement our schedule table, the master requests each individual communication participant to speak up. Now everyone can listen to the information and speak up if there is a turn agreed in advance, this is exactly how LIN tries to resolve any communication collisions. LIN's media access strategy implements a master-slave strategy, the master conducts the order of communication. Each slave has a unique identifier, each slave is only allowed to speak once, the master calls out the slaves. The call out order is done by a pre-configured schedule those are the essential principles of LIN's media access strategy.

On ECU level, we have two types of ECUs. One master and multiple slave nodes, up to 15 slaves. The LIN master is unique in the respect that it schedules the frames, it is oftenly referred as the gatekeeper to the backbone network. The job of the LIN slaves is to serve the communication requests from the LIN master node. Other than this, there are two software parts which are highly important for the LIN concept. Firstly, there is a piece of software called the master task and another piece is called the slave task. The master task is a routine that is only contained in the LIN master node. Its job is to read the schedule table entries and call out all the slave nodes in the defined order. This is technically done by the master task being in-charge of assembling the header parts of the overall frame. The slave task despite its name, resides in all nodes also the master node. Its job is to listen to the bus for a header message and react accordingly depending on the identifier. If it feels that the node it sits on is addressed, it will listen to the message or even reply to the message identifier. Reply happens by creating the so-called frame response. While the header contains the address node identifier, the frame response contains the actual payload data. The analysis of the technique to find out how master send data is where the master node slave task comes into play. The master node has its own identifier and instructs its own slave task to reply to the master task created frame header call out. A master node will invoke its connected slave nodes as well as itself according to the schedule table. The master task creates the header, the header contains information to first of all make all slaves aware that the request is coming in. This is called the break symbol, it activates all slaves to listen to the bus. The break symbol acts as a synchronization start, followed by a sync byte, where the nodes have the chance to collaborate its asynchronous timing to the message. After the sync byte, an identifier is transmitted. Considering if the identifier is relevant to the slave task, the slave task will listen or even create a response. The created response contains the payload data including a checksum.

V. STRUCTURE OF LIN FRAME

The full frame consists of a header and the payload data response. The master task creates the header, the slave task is in-charge of the response. But to actually raise awareness on the bus that the data exchange is upcoming, we need to start each frame with a break symbol. So that the break symbol indicates the start of a frame and unintended data transfer. A LIN frame can be thought of as a break symbol, followed by a chunk of byte-based data. So, the sync break indicates from where the LIN frame is going to start. Then we have one sync-field and the identifier. Sync break, sync field and identifier compose the header, following we have a small, accepted transmission gap then the response is composed out of one to eight data bytes ending with a checksum. The transmission gap can be shorter or longer depending on who replies, if the

master replies to its own request the cap will be much smaller than if for example another node has to reply the other node needs to basically filter first for the identifier craft internally response and then message response on the bus. LIN uses a byte-based frame setup, all the symbols except the break symbol have the smallest unit of a byte. All bytes in the LIN frame can be generated with a UART, liking this to our above explained LIN relation with UART.

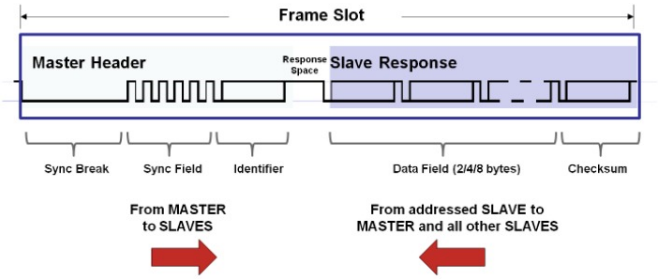

Fig. 4. Lin Frame structure [7]

A. SYNCHRONIZATION BREAK FIELD

The synchronization break field is there to make all ECU LIN nodes aware to pay attention. Communication is going to start on the LIN bus. The sync break is used to align nodes on the 'coarse' scale, and it makes all nodes aware to listen. Considering the dominant is high and recessive is low, the sync break brings the bus from the default dominant value, the highest state, to recessive value, to a low state for a certain while. It indicates an upcoming start of a frame and at the same time it marks the beginning of this data frame. In detail, the sync break starts with a dominant bus value of at least 13-bit times length. This can even be longer. It contains a sync delimiter, an excessive bus value and length of around one bit time or longer. The delimiter is required to bring back the bus into a valid dominant starting position, high position. So the value goes basically down, stays down for a while then all the nodes start to listen and then all the nodes start to become aligned to this down signal then again it goes up shortly for a while, to be ready to move on to the next byte
[4] [5].

B. SYNCHRONIZATION BYTE FIELD

While the brake signal shakes up all listening notes to pay attention, the sync field is there to start with the asynchronous synchronization mentioned during our UART introduction. Here it is all about getting the oscillator clocks temporarily in-sync to be able to read the transfer data. A 0x55 pattern is favorable in this case because the created pattern provides 5 unambiguous falling edges in 8-bit times [4].

C. IDENTIFIER

Finally, we have the identifier byte. This is the last element within the header, created by the master task. Up to here, all nodes are active and try to listen to the identifier byte. If the byte is not relevant to them, they will simply proceed with their other tasks. In case it is relevant they will continue to pay attention to pick up the following data or they might be even asked to provide the data by their slave task. The identifier field is a combination of 6-bits of ID identifiers and 2-bits reserved for parity. If parity is used, the LIN specification refers to the term 'Protected Identifier (PID)'. Within LIN, we can have 64 identifiers, 0 to 59 (0x00 – 0x3b) are reserved for standard data frame transmission. Those are the most common identifiers which we will see and which are used within the LIN. Identifier 60 and 61 (0x3c – 0x3d) are reserved for diagnostic data. Diagnostic support is optional. And identifiers 62 (0x3e) and 63 (0x3f) are reserved for future protocol enhancements [7].

Fig. 5. LIN frame identifier [7]

D. DATA FIELD

The response section is prepared by the slave tasks, this is a section which contains the relevant payload data. Early nodes will listen to the response or participate in creating the response if the previous header identifier was relevant to them. A data field consists of a byte-field, of up to 8 data byte fields are supported by the specification. The number of used data fields are basically part of the LDF file configuration.

E. Checksum Field

At the end of the response checksum is added. Depending on the LIN version, the checksum field protects the payload data from corruption, or it protects the payload data and the identifier. In LIN versions before LIN 2.0, the checksum field used to only protect the data fields. The checksum is called a classical checksum. LIN versions above LIN 2.0 introduced enhanced checksum, it caters for all the data fields including the identifier field. While most of the frame identifiers can be configured for a classical or enhanced checksum model, 60(0x3C) and 61(0x3D), the diagnostic frames, always use a classical checksum. The classical checksum contains the inverted 8-bit sum with carryover of all data bytes. The enhanced checksum contains the inverted 8 bit sum with carryover of all data bytes plus the protected identifier. The receiver detects a transmission error if the sum of the data bytes and arriving checksum does not equal to 0xff.

VI. FRAME TIMING

Frame timing impacts if the nodes can understand each other. It is the pace that is expected the communication should take place as well as allows delays, which can be tolerated. Frame timing has a direct impact on the sender and receiver tolerance to identify and decode messages. Due to its master slave UART based architecture, to get the configuration right for the frame timing matters, perhaps even stronger than other

automotive protocols that inherently cater for timing and jitter related aspects in hardware. Configuration plays a fundamental part in setting up the UART based point-to-point connections. The importance of frame timing stems from the fact that, first we need to have the right hardware in place which can be operated within the defined space. Secondly, LIN is very strongly dependent on the correct software configuration. Therefore, our design tools need to be strongly aware, not only of the physical capabilities but also of the overall timing behavior required to keep the whole network within tolerable boundaries.

A. NOMINAL FRAME TIME

The nominal value for transmission of a frame exactly matches the number of bits sent. This frame time is over optimistic because in reality our hardware will not be able to react instantly. To calculate the nominal frame time, we will look at the time required for the master tasks and at the time required for the slave tasks. Both values will be summed up for the resulting total nominal frame timing [4].

B. FRAME SPACES

Physical devices need a little bit of time when creating the header and the response. To keep the system participants aligned, we need to give everyone the chance to provide and pick up a message to and from the bus. First there is the inter-byte space. There is a little gap between the bit fields which is required for the euro in re-calibrate and start its a synchronous synchronization. The second space to consider is the response space, the response space is a gap of time it takes from a system to actually create and put a response on the bus. If the response is created by the master, the response space will be very small. It will be minimum because our response can follow very quickly after the header. But any response created by a slave node will come with a certain reaction time. The reaction time is the so-called response space time. It is the time it takes for the node to pick up the identifier from the bus and verify the identifier. It assures that it is the real addressed node [4].

VII. CLOCK AND INTERRUPT INITIALIZATION

The application can initialize resources not used by the LIN program as needed, but it is also responsible for initializing the OC defined for timeouts handling in the following way: Interrupt from the OC Control Register 1's OCIE flag must be set. To disable the related interrupt, a write access to the high byte of the declared Output Compare Register is required. When it is required, the LIN program will enable it. If the application is not using the other Output Compare, the functionality must be disabled by writing to the high byte of the relevant output compare register. Initialization of the Timer Clock To meet the LIN software requirements, the timer clock/prescaler must be defined. The reason behind this is that

VIII. HARDWARE SPECIFICATION

The LIN specification was created to enable for the use of very low-cost hardware nodes in a network. It's a single-wire, low-cost network based on ISO 9141. Microcontrollers with UART capability or specialised LIN hardware are employed in today's vehicle networking topologies. The microcontroller creates all LIN data protocol required by software and connects to the LIN network via a LIN transceiver, a level shifter, and some add-ons. Working as a LIN node is merely one of the features available. This transceiver may be included in the LIN hardware, and it functions as a pure LIN node with no additional functionality. Because LIN Slave nodes must be as inexpensive as possible, they may create their internal clocks using RC oscillators rather than crystal oscillators (quartz or a ceramic).

IX. SCHEDULES IN LIN PROTOCOL

X. VOLCANO LIN TOOL CHAIN

LNA is used to enter the LIN network requirements first. Second, LNA generates automatic frame compilation and schedule tables. LNA, in turn, generates LDF. The LDF and private file are then converted to target-dependent ".c" and ".h" code by the LIN configuration generating program. Fifth, application code is linked to the LIN target package (LTP) library and built with target-dependent configuration code. Sixth, utilizing the resulting LDF, LIN Spector is used to undertake analysis and emulation. LIN Network Architect (LNA) is a program that allows you to construct and administer LIN networks. LNA guides the user through all stages of network definition, starting with the entering of basic data such as signals, encoding types, and nodes.

LNA is in charge of two categories of information: 1. Objects that are global (signals, encoding types, and nodes) 2. Information about the project (network topology, frames, and schedule tables) Global objects should be created first and then reused in several projects. They can be manually defined or imported using a standard XML input file (based on Fibex rev. 1.0). The program will be able to import data straight from the standardized NCF in future versions. Version and variant management is supported in its entirety. The systems integrator creates networks by combining subsets of these elements. Throughout this process, a consistency check is performed on a regular basis. The signals are then automatically packed into frames. The final activity is to create the schedule table based on the time criteria that were gathered earlier in the process. Several aspects, such as bandwidth and memory utilization, are taken into account in the optimization. The program will automatically determine gateway requirements between subnetworks based on the allocation of signals to networks via node interfaces, whether LIN to LIN or LIN to CAN. The application of the automatically selected gateway node will make signal transfer from one subnetwork to another fully transparent. A publish/subscribe model is used by the tool. Only one node

can send out a signal, but it can be received by any number of other nodes. End-to-end time needs may differ between nodes. In the Volcano timing model, the maximum age is the most critical timing parameter. The maximum allowable duration between the generation and consumption of a signal in a distributed function is described by this parameter. Changes can be made quickly, with frame definitions and schedule tables being updated immediately to match the new requirements. LDF will be constructed automatically for each network once the timing analysis has been completed and the viability of the individual subnetworks has been determined. Textual reports can also be created to make information more readable for everyone involved in the design, verification, and maintenance process [4].

The LTP is the Volcano tool chain's embedded software component for LIN. Te LTP is offered as a precompiled and fully tested object library with accompanying documentation and a command line configuration utility called Lin Configuration generator tool (LCFG) that generates configuration-specific code and data structures. Using an application software to implement the LTP is a simple operation. The communication-related network information is contained in the LDF prepared by the offline tool. A target file, which is an ASCII (American Standard Code for Information Interchange)-based script, also provides low-level microcontroller information to the LTP, such as memory model, clock, SCI, and other node details. The command line utility LCFG is used to run these two files as input. It translates both files into target-specific code that the microcontroller can understand. All necessary configuration information is formatted into compiler-ready C source code in the output. Along with the precompiled object library, the target-dependent source code is introduced to the module build system. The LTP is linked to the application functionality after compilation to create the target image that is ready for download.

Through the standardized LIN API, application programmers can connect to the LTP and hence to the LIN subnetwork. Signal-oriented read and write calls, signal-related flag control, as well as node-related initialization, interrupt handling, and timed-task management are all covered by API calls. The LTP hides the low-level specifics of communication from the application programmer. Signal allocation within frames, frame IDs, and other details are stored in the LDF, allowing programs to be reused by simply linking to different configurations specified by different LDFs. As long as the signal formats aren't changed, a network reconfiguration is as simple as repeating the process outlined earlier, which will result in a new target image with no impact on the application. When the node allows for reflashing, the configuration can even be changed without the assistance of the supplier, allowing for end-of-line programming or after-sales service adjustments. LTPs are designed and built to work with a specific microcontroller and compiler. There are a variety of ports available to common targets, and new ports can be created at the customer's request [4].

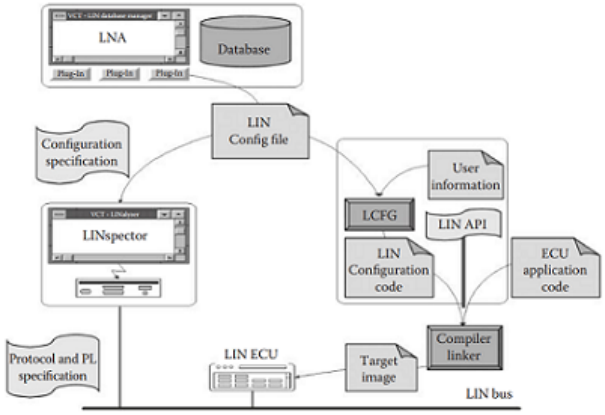

Fig. 6. Volcano LIN tool [4]

XI. Usability

LIN is a key component in the installation of a hierarchical vehicle network, which allows automakers to achieve improved quality and lower costs. This is made possible by introducing the industry to best practices in software development, such as abstraction and composability. LIN allows for the lowering of many existing low-end multiplex solutions, along with efficiency gains in car electronics design, manufacturing, servicing, and delivery. The expanding number of car lines equipped with LIN, as well as the ambitious ambitions for next-generation vehicles, are the best evidence of LIN's success. The LIN specification's simplicity and completeness, along with a holistic networking paradigm that allows for a high degree of automation, has made LIN the ideal complement to CAN as the backbone of industrial automation.

XII. Advantage over CAN

It's simple to use LIN. Components are available all around the world. It's also less expensive than the CAN and other communications buses. In the instance of LIN, the extension is simple to implement. Furthermore, there is no price for the protocol license. The CAN bus (Controller Area Network) is a common bus interface that most automobiles utilize to interact with independent control ECUs that handle the engine, transmission, climate, security alarm, and safety bags. Twisted pair signal wires are used to connect CAN devices because they are more noise resistant. Signals are normally operated at a voltage of 5 volts. For 40m cable lengths, the transfer speed can exceed 1Mb/s. Engineers have spent a lot of time thinking about the CAN protocol. It was created to be adaptable, dependable, and durable. There is a chance

XIII. Domain in Automotive Industry

Modern automobiles include more technology than we can imagine. Almost every important component has a specialized computer called an ECU that houses a slew of sensors

(Electrical Control Unit). On a typical car, there are anywhere from a few to hundreds of ECUs. Especially the high-end ones. All components must function as a cohesive entity. As a result, a dependable connection interface is required. In current vehicles, the LIN protocol plays an increasingly essential role in offering low-cost feature extension [4] [5].

As a result, the LIN bus has grown in popularity over the previous decade, with more than 700 million nodes predicted in automobiles by 2020, up from 200 million in 2010.

XIV. LIN API SOFTWARE

The utility enables for the monitoring and presentation of all network signal data, starting with LDF import. With logical names and scaled physical value views, advanced analysis is feasible. LDF information can be used to fully emulate one or more nodes, regardless of whether they are master or slave. Logging and replay of communication is possible, as well as the option to create a log using logic-based triggers. The user can mimic whole applications or perform test cycles while modifying emulated signal levels and switching schedule tables in real time using an optional emulation module. The user defines the routines using LIN Emulation Control (LEC) files written in a C-like programming language.

This can also be used to verify that a target module's LIN communication is complete. The stressors in the test cases are defined [4].

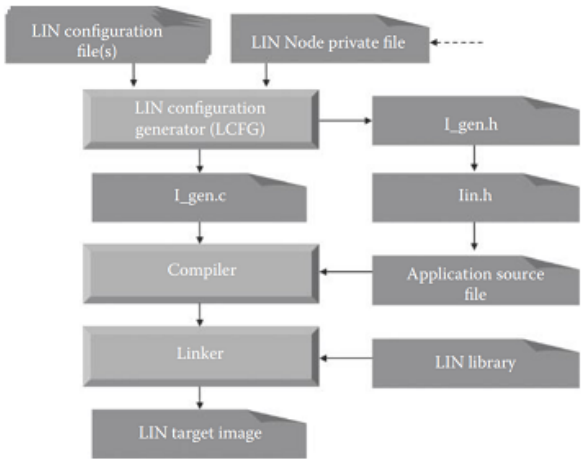

Fig. 7. LIN API Software [4]

XV. CONSTRAINTS

Some of the main drawbacks of LIN are lower bandwidth and less effective bus access scheme with the master- slave configuration.

XVI. CONCLUSION

LIN is a key component in the installation of a hierarchical vehicle network, which allows automakers to achieve improved quality and lower costs. This is made possible by introducing the industry to best practices in software development, such as abstraction and compos-ability. The expanding number of car lines equipped with LIN, as well as the ambitious ambitions for next-generation vehicles, are the best evidence of LIN's success. The LIN specification's simplicity and completeness, along with a holistic networking paradigm that allows for a high degree of automation, has made LIN the ideal complement to CAN as the backbone of industrial automation.

XVII. APPENDIX

A. Blueprint

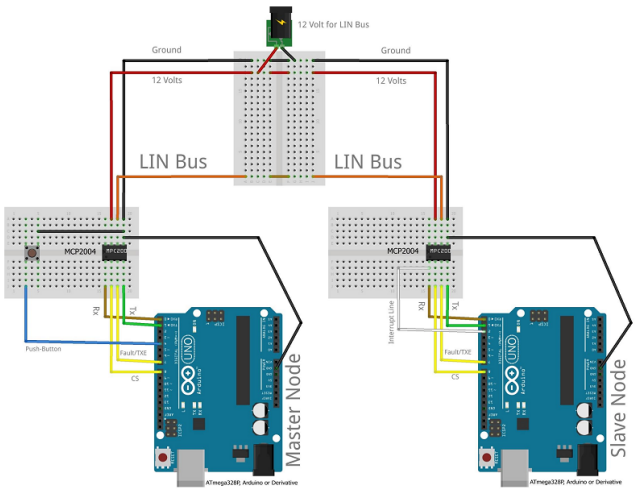

Fig. 8. Blueprint

REFERENCES

[1] F. Nouvel, W. Gouret, P. Mazério, and G. Zein, *Automotive Network Architecture for ECUs Communications*, 04 2009, pp. 69–90.

[2] P. S. Quanyan Zhu', Stefan Rass, "Community-based security for the internet of things," pp. 11–19, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128150320000020

[3] B. Fijalkowski, "Mechatronics: Operational and practical issues, intelligent systems, control and automation: Science and engineering," 2011.

[4] R. Zurawski, *Industrial Communication Technology Handbook*. CRC Press, 2015.

[5] D. P. Wiley. Sons Ltd, 2007.

[6] D. Ibrahim, *Project Inspiration- PIC Microcontroller Projects in C, Second Edition*, 2014.

[7] "Lin message frame." [Online]. Available: https://microchipdeveloper.com/lin:protocol-dll-lin-message-frame