Automotive serial buses – a detailed comparison

White Paper

1 Introduction and history of automotive electronics

Since the 1970s there has been an exponential growth in the number and complexity of electronic systems being used in vehicles of all kinds. However, from their practical beginnings in the first decade of the 20th century, vehicles powered by the internal combustion engine invariably contained electrical components such as points, magnetos and spark plugs. Over time the introduction of additional electrical components such as electric lighting, self-starters, heaters and screen wipers led to an increase in the complexity of electrical systems, including the need for a battery within vehicles.

The first use of electronics in vehicles was during the 1930s, when thermionic valve-based car radios were offered as factory installations in high-end cars.

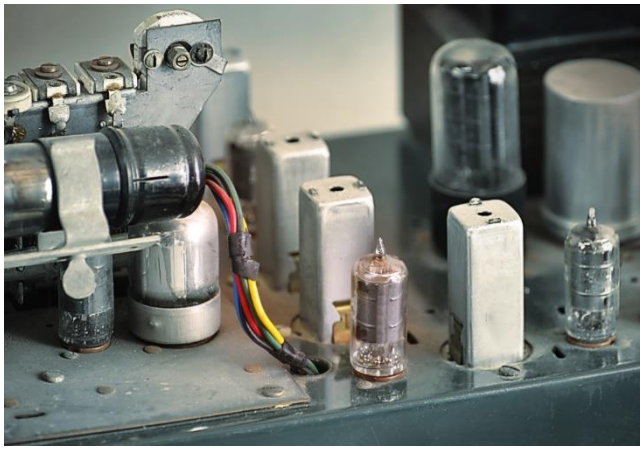

Figure 1 – Thermionic valve circuit

The introduction of compact solid-state electronics in the 1950s made the use of electronics in vehicle applications far more practical than using valves. The 1960s saw the widespread offering of in-vehicle entertainment systems such as solid-state radios, record players and, later in the 60s, eight-track players. This led to the broad introduction of electronic components into vehicle electrical systems. In the same decade, solid-state diode rectification made the alternator the default vehicle electrical generator and the first transistorized electronic ignition systems were introduced. In the 1970s additional electronic components such as early trip computers, LED dashboard instruments, cassette players and in-car stereos were introduced to meet the desire to make vehicles easier to drive and to increase vehicle functionality. The 1980s saw the focus begin to shift towards vehicle safety, reliability and comfort, with the addition of new systems such as anti-lock braking systems, airbags, CD players, electronic fuel injection and engine management systems. In the 90s and beyond the level of complexity of electronics within vehicles has continued to grow exponentially, with additional safety and environmental systems as well as more recently advanced driver-assistance systems (ADAS) including head-up displays, parking assistance, collision avoidance, lane keeping, and most recently fully autonomous driving systems.

Throughout the same period there has also been a gradual but continuous trend to replace existing mechanical and hydraulic vehicle systems with electronic systems. The sheer quantity of electronics deployed in today's vehicles has led to the cost of the electronic systems in a car being be more than 40% percent of the total manufacturing cost of the entire vehicle, double that of the previous decade.

At first, electronic systems with any computational function were implemented as a stand-alone Electronic Control Unit (ECU) consisting of a microcontroller (initially simple 8-bit

devices) as well as memory (RAM and ROM). Connected to the ECU were the necessary sensors and actuators needed to perform the system's designated function. It was soon found that this approach was insufficient for many systems, which required subsystem functions to be distributed over multiple ECUs with communication paths between them. For example, adaptive electric power steering must have access to vehicle speed information to enable it to adjust the level of steering effort applied. This information can be retrieved from the engine management system or from wheel rotation sensors. Vehicle speed information is just one example of data that is required by several different ECUs within the vehicle powertrain.

Until the 1990s each signal was sent across a dedicated cable, making up a point-to-point (P2P) network linking the ECUs. However, it is easy to see from the illustration of a simple point-to-point network below that, if an ECU at a node on the network needs to communicate with multiple ECUs at other nodes, the cabling requirements rapidly escalate. In this simple network, although there are only 5 ECUs, 25 dedicated cables are required to link them.

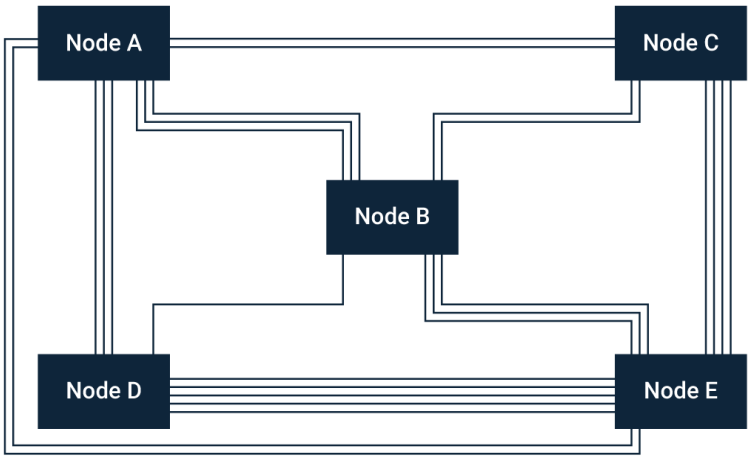

Figure 2 – Point-to-point communication network

With the increasing use of multiple ECUs throughout the 1980s, point-to-point networks became increasingly unacceptable from a cost, weight, complexity and reliability perspective.

The need to reduce the amount of wiring in vehicles led manufacturers to develop in-vehicle "buses", where multiple cables are replaced by a single cable (containing one or more individual wires). Data from different nodes on the network is serially multiplexed (combined) into a single signal, which can then be transmitted on this one cable. The word *bus* is a contraction of the Latin *omnibus*, meaning *for all*.

Many of the early automotive bus-based communication networks were both proprietary and centralized, with little or no interest in developing common standards. As time passed, the advantages of bus-based networking became clear and the vehicle market became increasingly globalized, the demand rose for more sophisticated and standardized distributed networks, which would ultimately lead to several industry-wide standards, each specialized for specific applications.

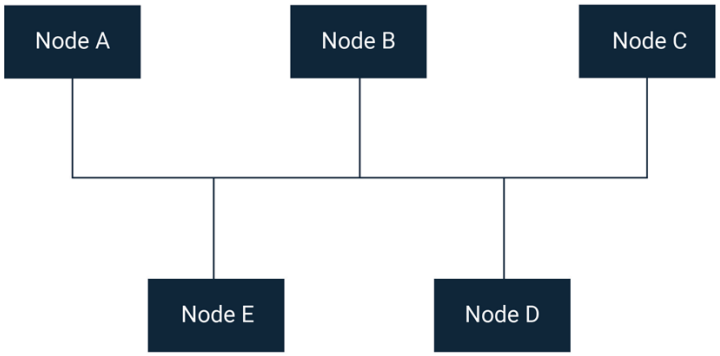

Figure 3 – Bus-based communication network

In the US, the Society of Automotive Engineers (SAE) classified these vehicle networks based on their data rate, as measured in kilobits per second (kb/s), as well as the intended application[i]:

- Class A, low speed (< 10 kb/s) for convenience features, such as electric mirror adjustment.
- Class B, medium speed (10 to 125 kb/s) for general information transfer, such as instrumentation.
- Class C, high speed (> 125 kb/s) for real-time control, such as powertrain control.
- Class D, very high speed (> 1 Mb/s) for high-bandwidth and time-critical applications, such as video and X-by-wire applications.

Note that the SAE has not yet defined a specific standard for Class D, but networks offering data rates greater than 1 Mb/s are often referred to a Class D networks.

These different bus-based networks have very different intended applications, which in turn govern their maximum data rates. As a result, different serial buses have very different associated sensor, interface and cabling costs. Typically, a vehicle has several networks using different serial bus types appropriate to each application.

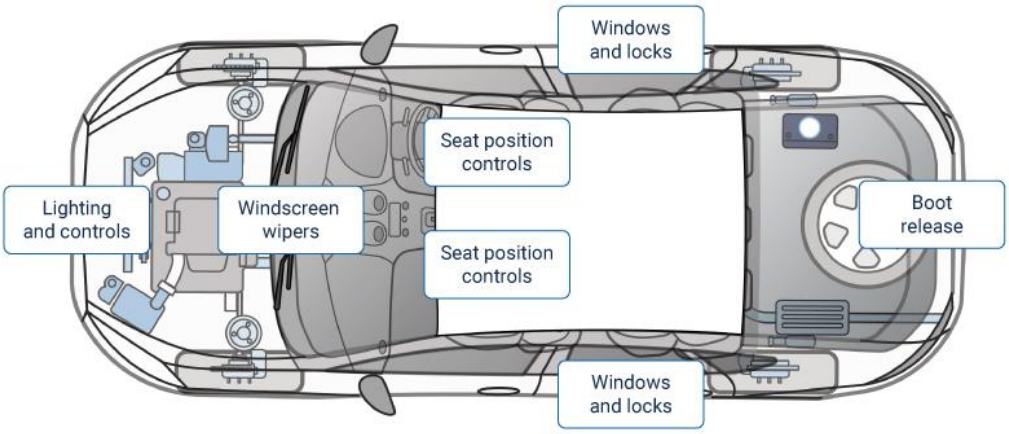

Figure 4 – Typical SAE Class A serial bus network applications

Class A serial bus networks offer the lowest data rate, with a maximum of 10 kb/s. Such networks use message-based, event-triggered data communication. The applications are for general purpose applications such as convenience and comfort actuators and sensors, such as lighting controls, wipers, windows, locks and seat position. Implementation of the cabling may be as simple as a single wire in a Class A serial bus, and deployment of these lowest-cost networks significantly reduces the weight of automotive wiring. As long ago as 1998, it was [1]announced[ii] that replacing the wiring harness in the four doors of a BMW with local area networks saved 15 kg in weight.

Class B serial bus networks support data rates between 10 kb/s and 125 kb/s. These networks support many of the non-critical vehicle applications such as instrumentation. Again, these networks support message-based, event-triggered communication and often have the ability for devices to be put into a sleep mode and then woken when required.

Class C serial bus networks support date rates greater than 125 kb/s and support the critical and real-time control vehicle applications, such as powertrain control and X-by-wire applications, where control/response is required within the millisecond range. Functionally (but not from an implementation perspective), Class C networks are a superset of Class B networks, which in turn are a superset of Class A networks.

Class D serial bus networks supporting data rates > 1 Mb/s supporting high-bandwidth and time critical applications may be event triggered, but for safety critical applications which require high degree of predictability (determinism) and fault tolerance, time-triggered data communication may be used. When communication is time-triggered, frames are transmitted at pre-determined time slots (as required by control loops) rather than simply when events have occurred (such as door closure). The form of data multiplexing offered by time-triggered communication is known as TDMA (time-division multiple access).

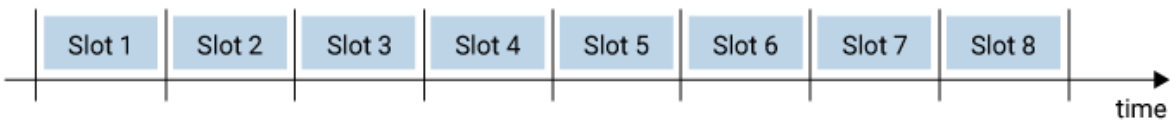

Figure 5 – Time-division multiple access (TDMA) time slots

From the implementation perspective, most automotive serial buses now deployed are analogous to Ethernet (IEEE 802.3, which is also a serial bus), in that the standards' primary definitions refer to the physical layer (cabling) and data link layers (message data format).

In addition to data rates, developers of automotive serial buses also must take in account many other factors including the automotive environment (heat, vibration, temperature, liquid spills etc.), electromagnetic radiation and susceptibility compatibility (EMC), message integrity, message latency, fault tolerance and recovery, physical space occupied and cost.

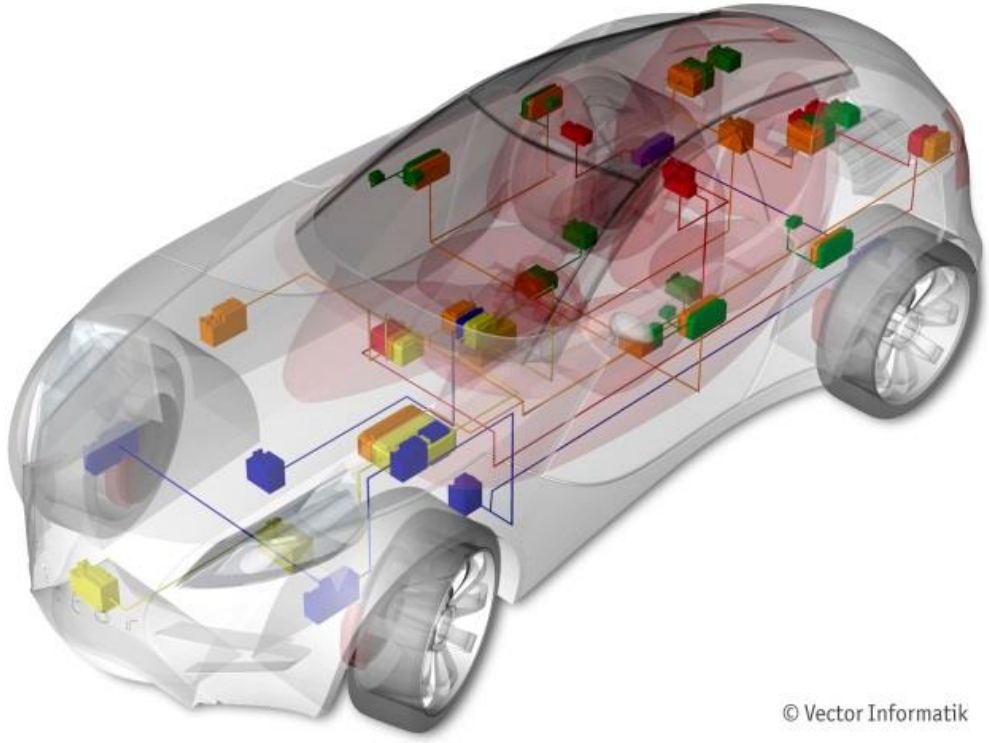

Figure 6 – Multiple vehicle electrical subsystems

Although developed initially for automotive applications and often associated with cars, these serial bus networks have subsequently found application in a wide range of vehicular and non-vehicular applications as diverse as agricultural machinery, military vehicles, trains, fork lifts, marine control and navigation, process control and industrial automation.

2 Serial bus standards overview

2.1 CAN

One of the earliest introduced and most widely deployed automotive serial bus networks is the CAN bus or Controller Area Network, which was developed in the mid-1980s by Robert Bosch GmbH. The intention was to develop a high speed and reliable communication system between multiple ECUs, without any need for a host controller. CAN 2.0 was introduced in 1991 and was first implemented in production by Mercedes in the early 1990s and was ultimately standardized as ISO 11898 (International Organization for Standardization) in 1993, later amended to three parts defining the physical layer, the data link layer and the physical layer for a lower speed, fault tolerant implementation. At the physical layer, CAN uses a differential two-wire cable (CAN High and CAN Low) terminated at each end by 120 Ω resistors. The termination resistors provide correct bus loading and

prevent data reflections. The differential topology offers improved noise immunity over a single-ended implementation, as in-phase voltages present on both wires are naturally suppressed. This is called common-mode rejection.

CAN offers data rates of up to 1 Mb/s, with common data rates of 125 kb/s and 500 kb/s also often being deployed, although all nodes on a CAN network must use the same data rate. An event-triggered message-based protocol is used between nodes, with all nodes receiving every message and each then acknowledging error free reception. Each receiving device must decide if a message is relevant to them. With the CAN system, only one node is permitted to transmit at a time, so nodes must wait for the bus to become idle before transmitting. An arbitrated priority system is also used to ensure that critical messages get through, as lower priority nodes stop transmitting when they receive higher priority messages being transmitted by another node.

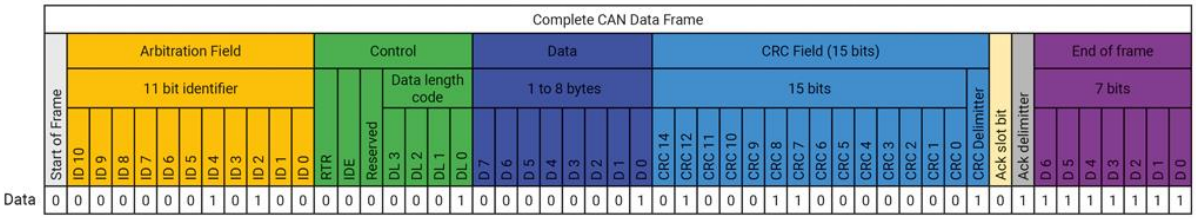

Figure 7 – CAN data frame

Although CAN is primarily used in automotive applications, it has also found widespread application in industrial process control, aerospace and medical applications.

2.2 CAN FD

In 2012, Bosch released an extension to the CAN bus, known as CAN FD (CAN with Flexible Data rate) to provide improved performance and high-bandwidth in applications such as end-of-line and garage-based software downloads, as well as to provide a means of sending longer single messages without the need to split up the messages into multiple CAN frames. This is achieved through data rates greater than 1 Mb/s and a 64-bit payload (data field within a CAN FD data frame) versus 8 bits for CAN 2.0. Although the arbitration and acknowledge phases are limited to 1 Mb/s to provide backward compatibility with CAN 2.0, the CAN FD data phase bitrate is only limited by the characteristics of the transmitter and receiver.

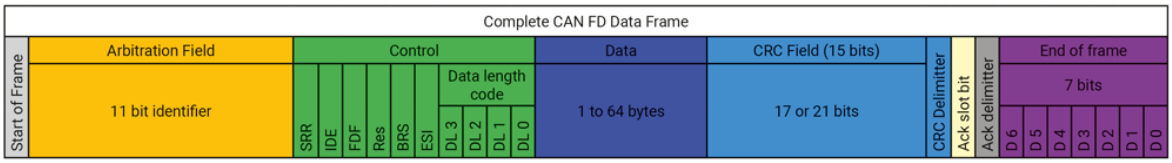

Figure 8 – CAN FD data frame

CAN FD interoperability has been tested at data rates of up to 10 Mb/s over a cable distance of 10 metres. As mentioned, CAN FD is backward-compatible with existing CAN 2.0 networks, so newer CAN FD devices can coexist on the same network with existing CAN devices.

2.3 FlexRay

The onset of increasingly demanding safety and driver assistance vehicle systems led to the

implementation of FlexRay, which was initially developed by a consortium of manufacturers to provide a deterministic, fault-tolerant and high-speed alternative to CAN, which it achieves at the expense of increased cost. Now standardized as ISO 17458, FlexRay uses a combination of event-triggered and time-triggered communication. These new safety and driver assistance vehicle systems require close integration between multiple ECUs, as well as the need to deterministically transmit data at very high rates. New X-by-wire features also require that data transmission is fault-tolerant, as these systems must be able to operate at an acceptable level even when a failure occurs. CAN and CAN FD are unable to support these requirements, primarily due to their reliance on non-deterministic event-triggered and priority-driven data communication. FlexRay addresses these requirements by ensuring that network nodes are not permitted to gain uncontrolled network access to signal events and instead must access the network using a predetermined time slot to transmit each message using TDMA (as described previously) in what is known as the Communication Cycle.

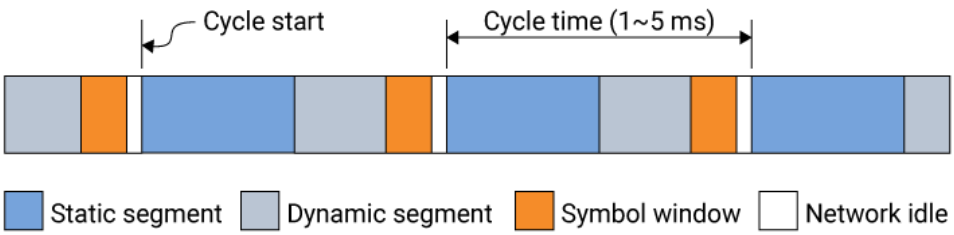

Figure 9 – FlexRay Communication Cycles

These reserved slots for time-triggered deterministic data communication are known as Static Segments. FlexRay also supports Dynamic Segments, which behave in a similar fashion to CAN and are used for event-triggered data communication that does not require determinism. At the physical layer, in addition to electrical communication, FlexRay also supports optical communication (using electrical to optical converters and vice versa) which offers much improved EMC performance. With a maximum data rate of 10 Mb/s, as its name implies, FlexRay is a very flexible standard, supporting both passive bus and active star-type network topologies or a combination of both. Star topologies offer the possibility to disconnect faulty branches or individual nodes as well better electrical performance than bus-based electrical networks.

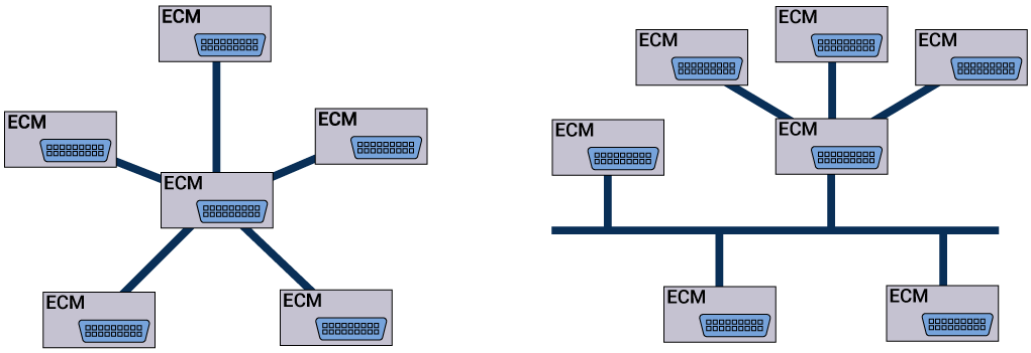

Figure 10 – Star network - Combined bus and star network

The ability to offer dual-redundant communication channels offers either fault tolerance or an increased maximum data rate of 20 Mb/s depending on the specific application. To provide additional protection from faulty networks nodes gaining bus access uncontrollably (commonly known as “babbling idiots”), FlexRay offers an optional independent control mechanism known as a Bus Guardian. This ensures that a node can only gain bus access when it is that node’s time slot in the Communication Cycle.

2.4 LIN

The growing number of convenience and comfort vehicle features, coupled with the desire to eliminate point-to-point wiring harnesses associated with non-critical features such as wipers, mirrors, climate control and rain sensors, led to the demand for lower-cost alternatives to CAN for sensor and actuator applications. In the 1990s many vehicle manufacturers and suppliers developed proprietary sensor/actuator bus implementations, leading to incompatibility issues between different vendors' products and systems. Later in the 1990s, the LIN (Local Interconnect Network) Consortium was formed to develop a common sensor/actuator bus standard. Initially introduced in 2000 as an inexpensive serial communication bus, the latest version (2.2A) is now standardized as ISO 17987. LIN uses a simple and low-cost single wire physical layer implementation based on ISO 9141 (as used by the K-line onboard diagnostic standard). This single-wire implementation does make LIN more susceptible to EMC than two-wire buses, which in turn limits the data rate to 20 kb/s and the recommended number of nodes to 16.

LIN uses a Master-Slave architecture with a single Master node and multiple Slave nodes connected to a common bus, making up a LIN Cluster.

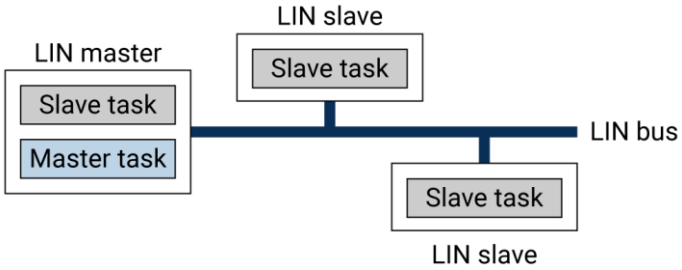

Figure 11 – Master-slave LIN cluster

Although LIN is a clock-scheduled TDMA bus, the only node required to possess an accurate clock is the LIN Master node, which coordinates all communication on the bus. LIN Slave nodes are only permitted to communicate when they are polled by the Master. To minimize cost, every node implements communication control in software, known as a Slave Task, along with a Master Task that is used to coordinate cluster communication in the Master node. Communication is through the repeated execution by the LIN Master of a predetermined LIN Schedule, which contains a list of Frame Slots to be transmitted ensuring deterministic communication. Each Frame Slot consists of a Header sent by the LIN Master inviting a specific LIN Slave to send its data as a Response.

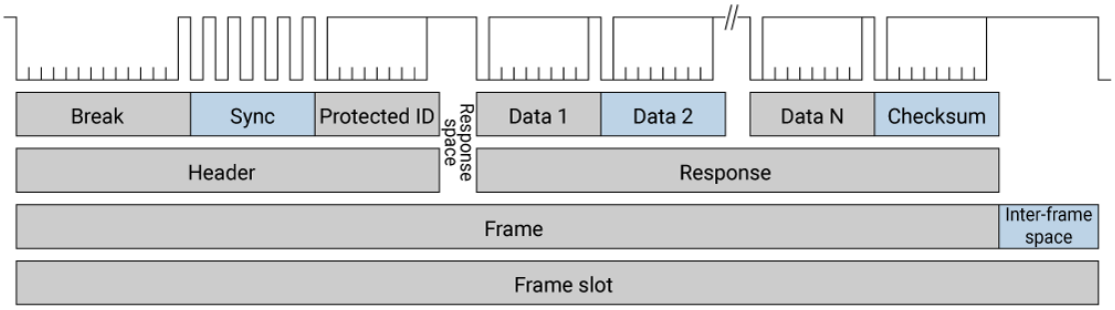

Figure 12 – LIN frame slot

Although the LIN schedule makes the communication cycle rigid, the standard accommodates flexibility by allowing five different frame types: unconditional, event-triggered, sporadic, diagnostic and user-defined. The last two types are application-specific

and the first three are used for regular communication, with unconditional frames being the normal method of communication. To minimize energy consumption, LIN offers capabilities to send Slave nodes into a sleep mode and wake them up as required. LIN has rapidly become a very successful standard for simple low-cost data communications within non-critical body and convenience/comfort applications and is now very widely deployed in production vehicles.

2.5 SENT

An even lower-cost serial bus is SENT (single-edge nibble transmission), a point-to-point network for transmitting data between a sensor and an ECU, now standardized as SAE J2716. Appropriate sensor applications include, amongst others, throttle position, pressure, mass airflow and high temperature. SENT is a lower-cost and even simpler alternative to LIN, but is limited to unidirectional (output only) P2P applications. It was designed as a replacement for analog to digital (A/D) converter and ratiometric (proportional) pulse width modulation (PWM) analog communications and offers the advantage to ECU and sensor OEMs of achieving lower cost through the industry-wide adoption of a common basic design. At the physical layer, SENT is implemented using a single wire for data communications in addition to 5 V and ground lines.

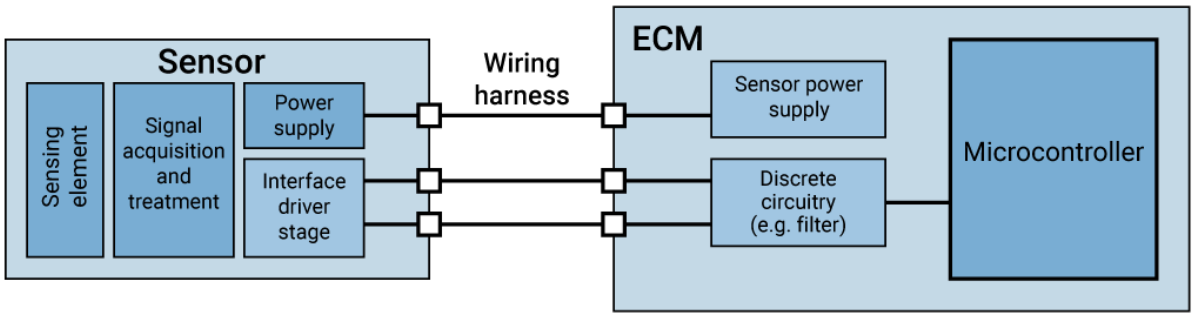

Figure 13 – SENT implementation

Although it is a single-wire implementation, the standard is designed to give good EMC performance as well as good sensor resolution (typically 12 to 16 bits) and reliability, albeit at a low data rate of 25 kb/s. It is intended to be used with “smart sensors” that contain the necessary microprocessors or ASICs (application-specific integrated circuits) to create the signal. Signals are sent from the sensor to the ECU as a series of pulses with data encoded on successive falling edges. It should be noted that there are no coordinating controlling signals transmitted by the ECU, which simply acts as a receiver. For these typically critical sensor applications, data must be transmitted constantly with no return path communications that could cause an interruption. This does mean that, to calibrate the sensor, a secondary interface is required to communicate with the device. In normal operation, once the sensor is powered up, it will start transmitting data in the same way that an analog sensor would do, although a SENT smart sensor is able to communicate additional information such as diagnostics and other secondary data. Primary data is transmitted using what is known as the Fast Channel and any secondary data is transmitted using the Slow Channel. The unit of time is a Tick (between 3 µs and 90 µs), with data being sent in units of four bits (nibbles).

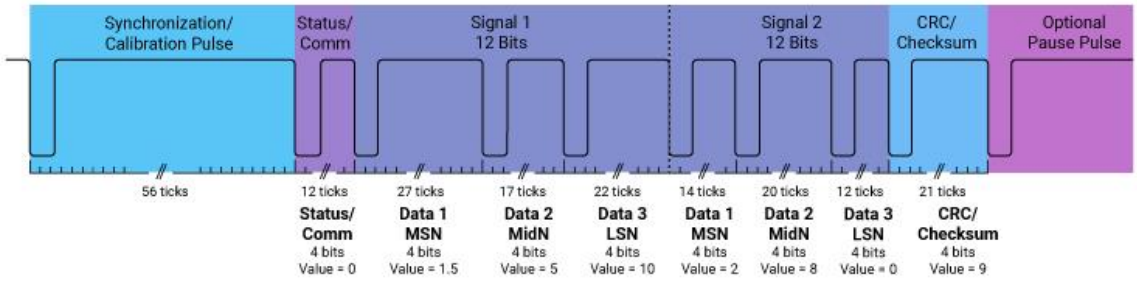

Figure 14 – SENT Fast Channel message

An initial Synchronization/Calibration pulse is used by the ECU's receiver to calculate the tick time. The next Status/Communication nibble either communicates sensor status or Slow Channel data depending on the mode. The signal data is then transmitted, usually followed by a CRC/Checksum nibble for error checking and an optional variable-length pause pulse, which can be used to maintain constant bit rate. Details of the signal data encoding vary for different sensor applications; these being described in various appendices of the SAE J2716 specification. The SENT bus was first defined in 2007, was most recently updated in 2016 and is seeing greatly increased interest as it offers a low-cost alternative to A/D and analog sensor output.

2.6 BroadR-Reach

It was inevitable that Ethernet networks would become a standard in automotive applications (as they already have in aerospace applications). The major benefit of implementing Ethernet is the economy of scale provided by a ubiquitous technology that is in widespread use in non-automotive applications. That said, there are several automotive requirements that must be considered with any serial bus implementation, including the hostile environmental conditions discussed previously. BroadR-Reach is an automotive-specific implementation of the Ethernet physical layer offering a 100 Mb/s low-cost network using an unshielded single twisted-pair copper cable for lower cabling cost and weight than standard Ethernet Cat 5 cable. The only difference between BroadR-Reach and standard Ethernet is at the physical layer, all other layers being identical. BroadR-Reach is being promoted by the OPEN Alliance SIG (Special Interest Group), a consortium of more than 140 members, as the standard for the automotive Ethernet physical layer, through its ability to meet the stringent automotive requirements for environmental conditions, power saving and cost.

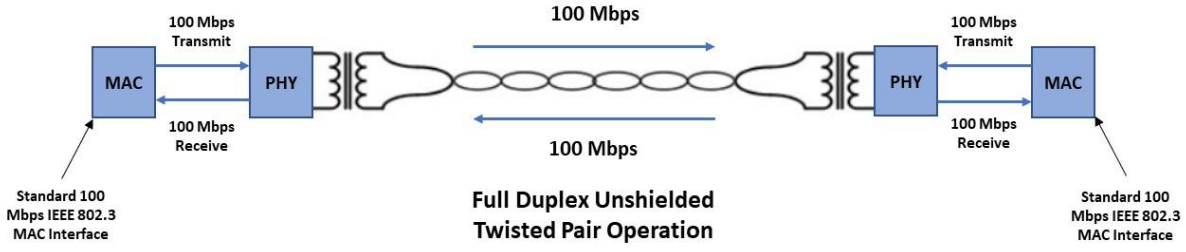

Figure 15 – BroadR-Reach Implementation

BroadR-Reach is also now standardized as 100Base-T1 (or IEEE 802.3bw), but there are some small differences between the two standards that will be discussed later in this white paper.

2.7 Other serial buses

PSI5 (Peripheral Sensor Interface) is another low-cost interface originally developed for airbag-to-ECU communications although it can be used for any kind of sensor application. It offers a higher data rate of 125 kb/s and uses just two wires for both data communication and power. It is possible to implement PSI5 both in P2P and bus topologies, where it operates in a time-triggered TDMA manner with each sensor transmitting data in a predefined time slot.

Several serial buses have been developed to transmit the high data rates associated with multimedia and infotainment applications and will become more important as vehicles become part of the Internet of Things (IOT). One such standard is MOST (multimedia-oriented system transport), which was first proposed in 1998 by a consortium of vehicle and component manufacturers. MOST uses optical fibre at the physical layer, giving good EMC performance as well as very high data rates. More recently, MOST150 provides the ability to transmit and receive Ethernet frames, thus providing in-vehicle internet access via IP (internet protocol).

3 Serial bus technical details

3.1 CAN

The CAN 2.0 specification details two parts:

- CAN 2.0A describes the standard CAN format with an 11-bit identifier
- CAN 2.0B describes the extended CAN format with a 29-bit identifier

The subsequent ISO 11898 specification details three parts:

- ISO 11898-1 describes the data link layer
- ISO 11898-2 describes the CAN physical layer for high-speed CAN (an SAE Class C network, up to 1 Mb/s).
- ISO 11898-3 describes the CAN physical layer for low-speed, fault-tolerant CAN (an SAE Class B network, up to 125 kb/s)

3.1.1 CAN Bus physical layer

Each high-speed CAN (ISO 11898-2) node uses differential transmission using a twisted pair of copper bus wires named CAN High and CAN Low, the linear bus being terminated at each end with a 120 Ω resistor.

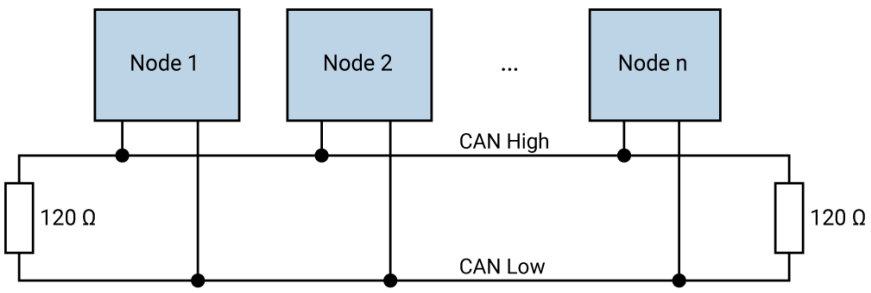

Figure 16 – High-speed CAN bus network

As noted previously, the use of differential voltages provides a high degree of common mode rejection, resulting in good noise immunity.

Logic level 0 is known as the dominant state and logic level 1 is known as the recessive state. With 5 V CAN systems, during a recessive (1) transmission, the bus is not actively driven and rests at around 2.5 V and during a dominant (0), CAN High is driven towards 5 V and CAN Low is driven towards 0 V.

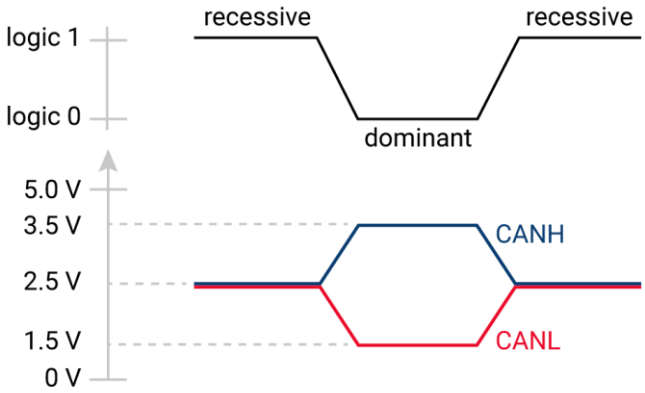

Figure 17 – High-speed CAN bit coding

A recessive (1) transmission occurs if the differential voltage is less than or equal to 0.5 V. If the differential voltage is greater than 0.9 V, a dominant (0) transmission occurs. The nominal dominant voltage for CAN High is around 3.5 V and for CAN Low is around 1.5 V.

Low-speed or fault-tolerant CAN (ISO 11898-3) is designed to withstand open circuits, short circuits and incorrect loads on one of the CAN High or Low bus wires, while still able to operate with a single bus wire when a fault occurs. To preserve network functionality if a bus wire becomes open-circuited, termination resistors are required at each node. The value of these resistors needs to be calculated so that the overall termination resistance is not less than 100 Ω.

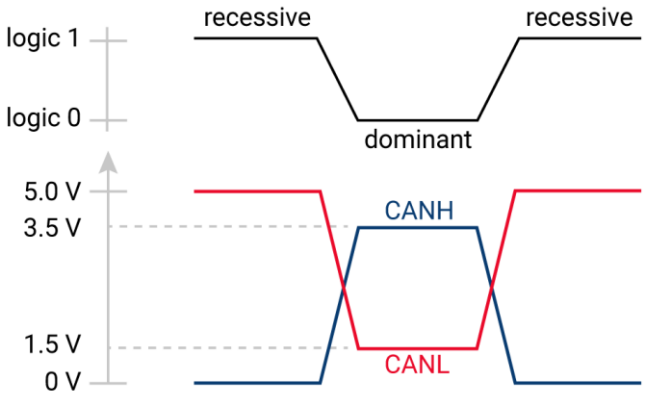

Figure 18 – Low-speed CAN bit coding

With low-speed CAN, neither CAN High (around 0 V) nor CAN Low (around 5 V) is driven when transmitting a recessive (1). During a dominant (0) transmission, CAN High is driven towards 5 V and CAN Low is driven towards 0 V. Therefore, a recessive transmission occurs if the differential voltage is around 5 V. If the differential voltage falls to around 2.5 V, then a dominant transmission occurs. The nominal dominant voltage for CAN High is again around 3.5 V and for CAN Low is around 1.5 V.

CAN uses non-return to zero (NRZ) bit coding, whereby the voltage level only changes when there is a change in the corresponding logic level. In other words, there can only be two states on the bus (0 or 1) unlike return to zero (RZ) coded systems, which also have a rest state between logic levels 0 and 1. This results in a higher average energy level in the signal, further improving noise immunity.

3.1.2 CAN data link layer

CAN data is sent in Frames starting with a dominant (0) Start of Frame (SOF) bit followed by a message Identifier (known as the CAN identifier or CAN ID), which forms the basis of arbitration (Priority) when two or more nodes attempt to transmit at the same time. Each message is assigned a unique Identifier (ID) which can be 11 bits (standard CAN 2.0A) or 29 bits (extended CAN 2.0B) in length. CAN network communications are based on message-based addressing rather than node-based addressing. Identifiers do not refer to CAN nodes, but to the data frames that the nodes transmit or request. The CAN ID specifies the content of the transmitted data frame and not the destination of the data frame, which means that all nodes must evaluate the CAN ID of each message, to determine whether the message is of use to them.

The table below shows three nodes attempting to transmit at the same time, each starting with a dominant (0). As the bus is only actively driven when a dominant is being transmitted, any dominant transmission will override a recessive transmission. This means that when a node transmits a recessive (1) it will see that the bus remains at dominant (0), which indicates that there is a bus contention, the node ceases to transmit and waits for the next opportunity to transmit.

	Start bit	ID bits											Rest of frame				
		10	9	8	7	6	5	4	3	2	1	0					
Node 2	0	0	0	1	Stops transmitting												
Node 5	0	0	0	0	0	0	1	0	0	1	1	0	x	x	x	x	x
Node 14	0	0	0	0	0	0	1	1	Stops transmitting								
CAN data	0	0	0	0	0	0	1	0	0	1	1	0	x	x	x	x	x

Figure 19 – CAN bus arbitration

The means that priority is always given to the node transmitting the lowest ID value as it has the largest number of leading zero bits (dominants) before the first recessive (1) bit. In this case Node 5 is transmitting the lowest ID value, wins arbitration and is given priority to transmit the rest of the frame. The allocation of message priority and associated ID is the responsibility of the system designer, but industry groups mutually agree on the significance of certain messages.

CAN has four frame types:

- **Data Frame** – contains node data for transmission
- **Error Frame** – transmitted by any node detecting an error
- **Remote Frame** – a frame requesting the transmission of a specific data frame
- **Overload Frame** – a frame to inject a delay between data and/or remote frame

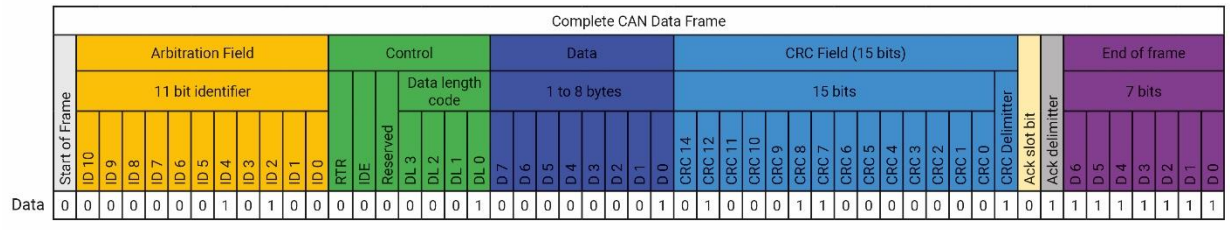

Figure 20 – CAN data frame

Following the Arbitration Field, there is a series of Control Bits within a Control Field.

The RTR (remote transmission request) bit distinguishes the frame as either a data frame (RTR=0) or a remote frame (RTR=1). In the above example, RTR=0, making this a data frame, with the Data Length Code (DLC) bits indicating the number of bytes (8 bits) contained within the data field. In this case the Data Field contains one byte of data.

The Identifier Extension (IDE) bit determines if the 18-bit ID extension is being used (IDE=1), giving a total of 29 bits. The following bit is reserved for future use and is transmitted dominant (0).

The ID in an extended data frame consists of two elements, the base ID (11 bits) and the extended ID (18 bits). Between these elements are the IDE and SRR (Substitute Remote Request) bits, which are both always transmitted recessive (1). In the control field, the RTR bit is transmitted dominant (0) for an extended data frame and recessive (1) for an extended remote frame. The next two bits (r1 and r0) are reserved and must be transmitted dominant (0). As with the CAN base frame format, the Data Length Code indicates the number of bytes (up to 8) contained within the data field.

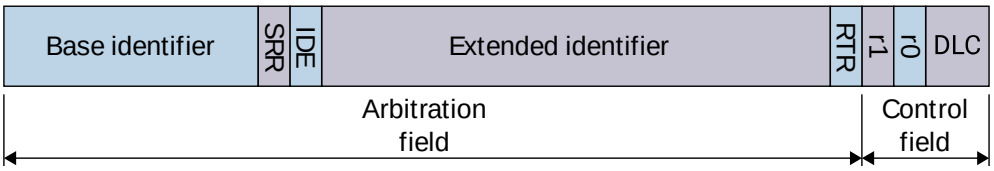

Figure 21 – CAN extended format arbitration and control fields

The 15-bit Cyclic Redundancy Check (CRC) field is used to ensure data integrity.

The Acknowledgement Slot (ACK) bit indicates that at least one node has received the frame without error. All nodes that receive a frame without detecting any errors transmit a dominant (0), which overrides the recessive (1) sent by the transmitter. If the transmitter detects a recessive, it knows that the frame was not received correctly. The End of Frame (EOF) field is confirmed by the transmission of 7 recessives.

Interframe space consists of at least three consecutive recessive bits. After that, if a dominant bit is detected, it will be regarded as the Start of Frame (SOF) bit of the next frame.

If any node detects an error, it aborts message transmission by sending an Error Flag of six dominant (0) bits. As this violates the bit stuffing (see “system timing” for a description of bit stuffing), which consists of a series of five dominant bits, it disrupts transmission to such an extent that other nodes will detect errors, which then send a second Error Flag. Each node will then wait for a series of eight recessive (1) bits known as the Error Delimiter. The two

Error Flags and Error Delimiter together make up an Error Frame.

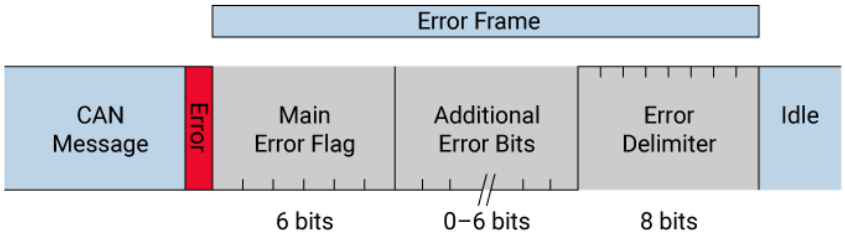

Figure 22 – CAN Error Frame

Remote Frames are used to request the transmission a specific data frame. They are structured in a similar fashion to data frames with the exception that the RTR bit is recessive (1) and there is no data field.

A Remote Frame requests the transmission of the corresponding data frame, by using the message ID of the requested data frame in the arbitration field. The most common implementation is that a node either responds to the request immediately with a stored message or it in turn requests its microcontroller to generate the requested message.

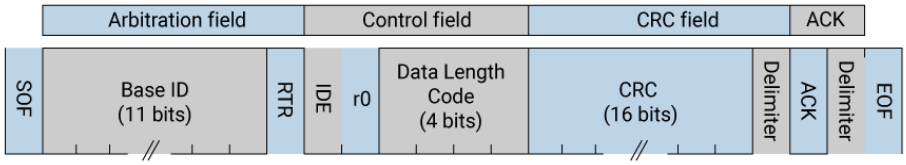

Figure 23 – CAN Remote Frame

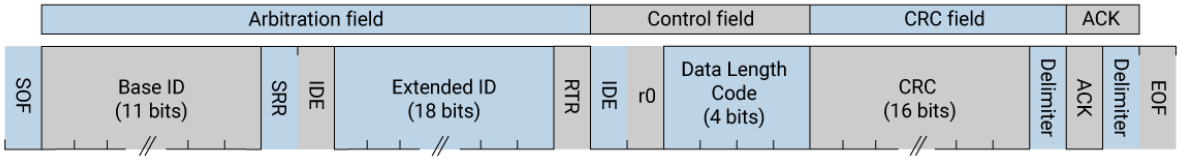

Figure 24 – CAN Remote Frame with Extended ID

The final frame type is the Overload Frame, which is rarely used. It was intended to allow a node to signal that it was too busy and that it required a delay before the next data or remote frame was transmitted. Modern CAN controllers are fast enough not to require this capability, so the Overload Frame has largely fallen out of use.

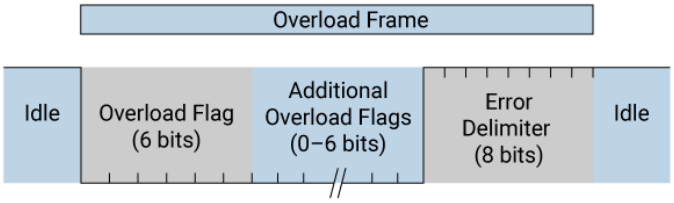

Figure 25 – CAN Overload Frame

From an implementation perspective, the Overload Frame format is similar to that of an Error Frame.

3.1.3 System timing

All nodes within the system must operate at the same nominal bit rate, so there is a need to ensure that the bit time (the time between the transmission of successive bits within the same frame) is preserved.

Initial synchronization is provided by the recessive (1) to dominant (0) transition of SOF bit and resynchronization is achieved on every successive recessive to dominant transition. However, as the data is NRZ coded, there are no rest state transitions that could be used for resynchronization when consecutive recessive or dominant bits are transmitted. Another mechanism is needed to ensure that there are sufficient bit transitions to ensure continuous resynchronization.

To achieve this, ISO 11898-1 specifies the use of “bit stuffing” with both Data and Remote frames to enable periodic resynchronization through the introduction of additional bit transitions. A bit of opposite polarity is inserted after five consecutive bits at the same logic level. A stuffing bit is inserted even if a bit of opposite polarity bit already followed the five consecutive bits.

In the example shown below, a recessive (1) stuffing bit is inserted after five consecutive dominant (0) bits, even though the next bit was in any case recessive. This means that, according to the specifications, the recessive stuffing bit itself is now counted along with the next four consecutive recessive bits, forcing the system to add a dominant bit after the fourth data bit. As the next data bit was dominant, again the next recessive stuffing bit is added after the fourth dominant data bit.

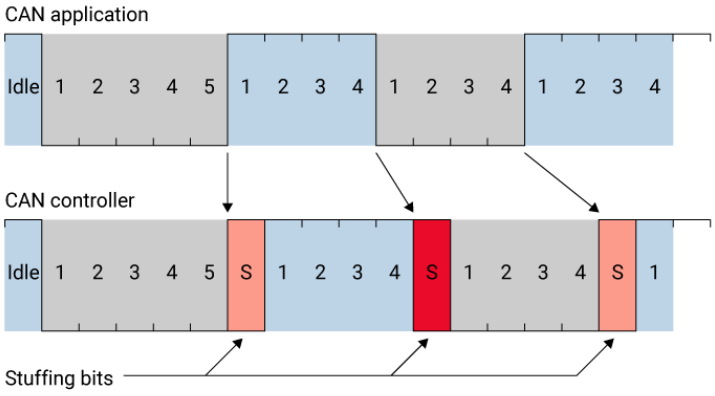

Figure 26 – Bit stuffing

The stuffing bits are inserted by the node’s CAN controller, which is the interface between the node’s CAN application and the CAN bus. Likewise, the CAN controller de-stuffs the bitstream on reception, before passing to the CAN application for processing.

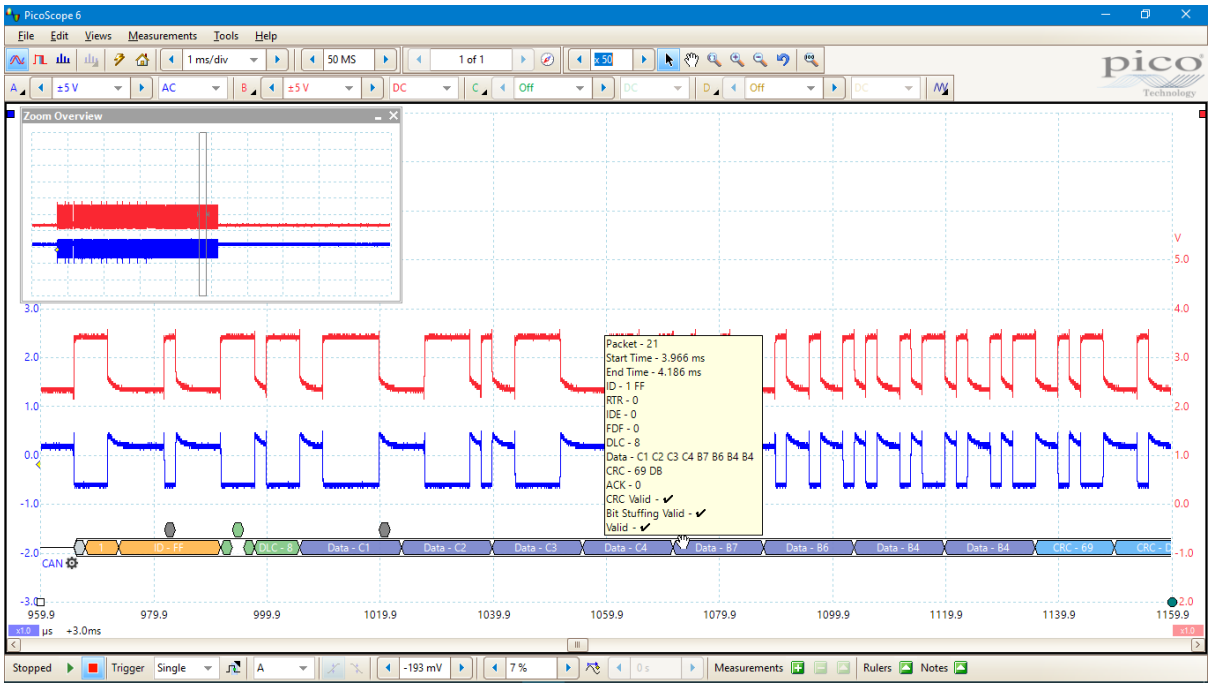

Figure 27 – PicoScope decoded CAN signal

3.2 CAN FD

- CAN FD is specified as part of the updated ISO 11898-1:2015 standard

3.2.1 CAN FD physical layer

Although the fundamental operation of the CAN FD physical layer is the same as high-speed CAN, there are several factors that need to be considered if data rates above 1 Mb/s are to be used reliably.

Although classic CAN transceivers have been tested with CAN FD, it is usually recommended that specialized CAN FD transceivers are used. It is important that the bus should avoid long stubs and care should also be taken with cabling capacitance. High ECU to CAN FD transceiver interface capacitive loading should also be avoided. High capacitive loading can cause both propagation delays and reflections. Reflections occur when a portion of a signal is reflected back to its source rather than being propagated to its intended destination. These reflections are caused by impedance mismatches and cable non-linearities. High capacitive loading can cause a reactive impedance mismatch, leading to reflections.

3.2.2 CAN FD data link layer

In CAN FD communications, the transmission consists of two phases, the arbitration phase and the data phase. To provide backwards compatibility with classic CAN, the arbitration phase is transmitted at a fixed data rate of up to 1 Mb/s. The data phase is transmitted at a flexible data rate which can greater than 1 Mb/s.

The arbitration phase and its use of Base and Extended frame formats are identical in both classic CAN and CAN FD.

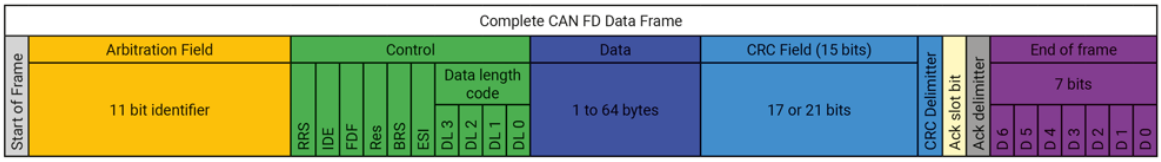

Figure 28 – CAN FD data frame

CAN FD does not use Remote Frames. As they have no data field, there is no requirement for an increased data rate with classic CAN Remote Frames being used to request CAN FD data frames.

As CAN FD does not use remote frames, the RTR bit used in CAN is redundant and is replaced with the Remote Request Substitution bit (RRS), which is always transmitted dominant (0).

The reserved bit of the classic CAN control field becomes the Flexible Data Format (FDF) bit, which signals that the data frame can carry a much larger payload than classic CAN. FDF is transmitted dominant (0) to indicate that the frame is in classic CAN format and transmitted recessive (1) if the frame is in CAN FD format. Although CAN FD controllers can receive and transmit classic CAN frames, if a classic CAN controller receives a CAN FD frame, the recessive (1) in the place of the normally dominant (0) of what it regards as a reserved bit will be regarded as incorrect and an error frame will be generated. Classic CAN and CAN FD nodes can only communicate with each other using the classic CAN frame format.

The next bit is reserved for future use.

The Bit Rate Switch bit (BRS) signals if a higher data rate will be transmitted. If BRS is transmitted dominant (0), a constant bit rate will be used for the entire frame. If BRS is recessive (1), a higher bit rate will be transmitted after the BRS bit up to and including the CRC Delimiter bit. Every CAN FD controller on the same bus must be configured to use the same data rates.

The Error State Indicator (ESI) bit is transmitted dominant (0) if the node is in an Error Active state, and recessive (1) if the node is in an Error Passive state. This bit is used for improved error tracking of CAN FD nodes within the network.

The length of the data field payload in bytes is indicated by the Data Length Code (DLC), but its coding is different in classic CAN and CAN FD.

	No of bytes	Data length code			
		DL3	DL2	DL1	DL0
Codes in CAN and CAN FD format	0	0	0	0	0
	1	0	0	0	1
	2	0	0	1	0
	3	0	0	1	1
	4	0	1	0	0
	5	0	1	0	1
	6	0	1	1	0
	7	0	1	1	1
CAN format	8	1	0/1	0/1	0/1
Codes in CAN FD format	8	1	0	0	0
	12	1	0	0	1
	16	1	0	1	0
	20	1	0	1	1
	14	1	1	0	0
	32	1	1	0	1
	48	1	1	1	0
	64	1	1	1	1

Figure 29 – CAN and CAN FD data length coding

The first eight codes are identical, but with classic CAN if DL3=1, then the data length is 8 bytes irrespective of the transmitted bit value of DL0, DL1 or DL2. It is these bit values that are used to specify the larger data fields (up to 64 bytes) used in CAN FD.

As CAN FD can carry more bits at a higher data rate than classic CAN, there is an increased probability of bit errors, which results in more bits being required for the CRC check. If the data frame contains up to 16 bytes, a 17-bit CRC is used (compared with 15 bits for classic CAN) and if the data frame contains more than 16 bytes, a 21-bit a CRC is used.

The CAN FD Ack and End of Frame structures are the same as classic CAN and are performed at the lower of the two bit rates.

3.2.3 Bit stuffing

With classic CAN, stuffing bits are inserted between the SOF bit and the end of the CRC field. After every five successive bits of the same state, dominant (0) or recessive (1), a complementary (opposite state) bit is inserted. These stuffing bits are not used in the CRC calculation.

With CAN FD, the same rule is used to stuff bits, but only up to the end of the data field, and any stuffed bits up to that point are then included in the CRC calculation. A 3-bit stuff count of the inserted stuffing bits, followed by a parity bit, are then transmitted as the first 4 non-stuffing bits of the CRC field. Stuffing bits are also inserted into the CRC field, including the 4-bits of stuff count and parity, but at fixed points. The CRC field begins with a stuffing bit which is inserted complementary to the state of previous bit. Stuffing bits are then inserted after every four bits irrespective of state, with each stuffing bit being complementary to state of the previous bit.

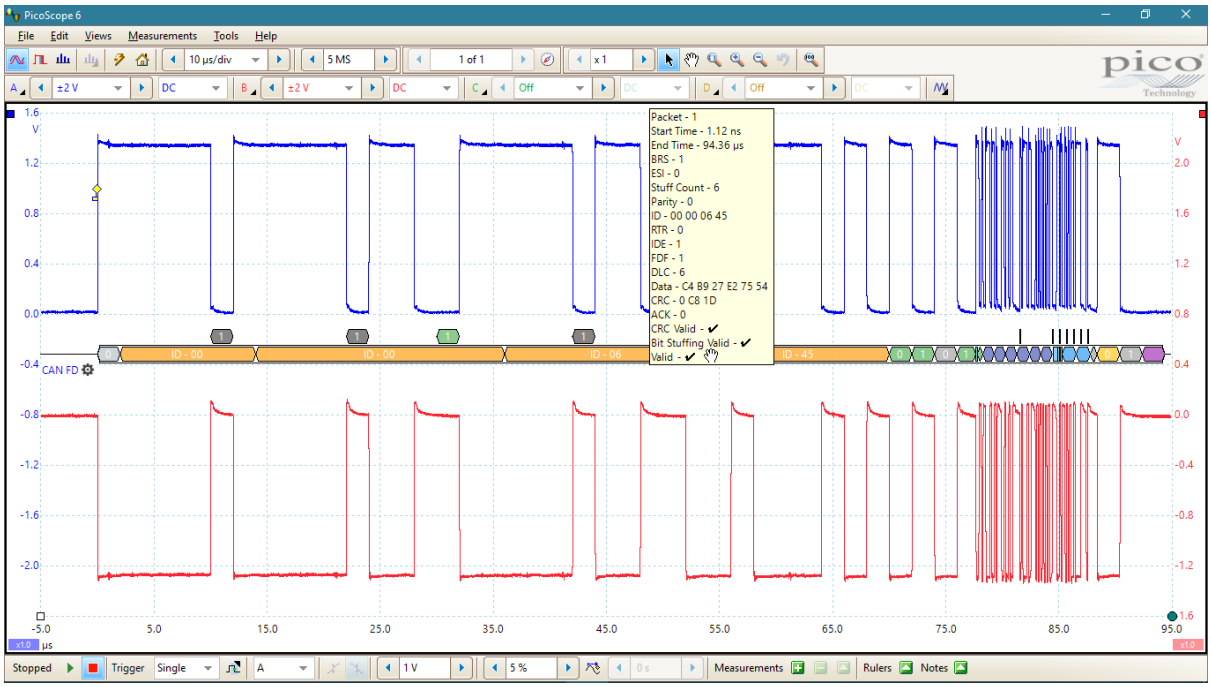

Figure 30 – PicoScope decoded CAN FD signal

3.3 FlexRay

- FlexRay is specified as the ISO 17458-1 to 17458-5 set of standards

3.3.1 FlexRay physical layer

FlexRay is implemented using differential twisted-pairs of shielded or unshielded cable to create a bus connecting a transmitting node and one or more receiving nodes. The specific implementation is intended to be flexible, but each differential pair must be terminated at each node with a resistance of between 80 and 110 Ω. The bus is bidirectional, so each node requires both a transmitter and receiver combined in what is known as a bus driver, with additional bus drivers being connected either in a linear or star bus topology. The two signal wires are denoted Bus Plus (BP) and Bus Minus (BM) and physical signal transmission is based on the voltage difference between these two wires, where V_{DIFF} = BP–BM.

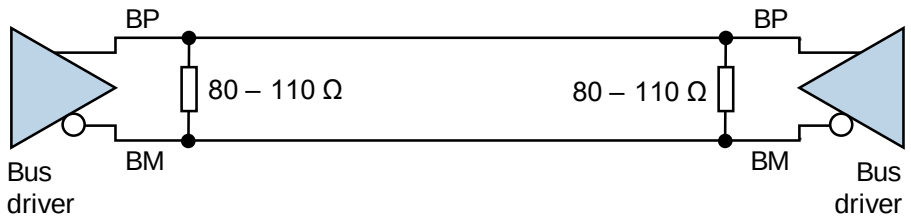

Figure 31 – FlexRay differential voltage bus

The FlexRay Electrical Physical Layer Specification defines four bus states, two of which are recessive and two are dominant. The recessive bus state has a differential voltage of 0 volts and the dominant state has a differential voltage not equal to 0 volts. The four bus states are Idle, Idle Low Power, Data_0 and Data_1.

In the Idle bus state, both BP and BM are driven to a nominal 2.5 V, giving a 0 V differential voltage. If a node is in its low power state (Standby, Sleep, Go-To-Sleep), then the idle low

power bus state is used, which also has a 0 V differential voltage, but the two bus wires are in this case at a nominal 0 V level. The Idle state is a defined length of time used by each node to maintain clock synchronization. The smallest unit of FlexRay time is a “macrotick” and the FlexRay controllers use the idle time to synchronize themselves by adjusting their local clock so that the macrotick occurs at the same point in time on every node in the network.

In the dominant Data_0 bus state (which represents logical state 0 or LOW), BP is driven to 1.5 V and BM is driven to 3.5 V giving a differential voltage of –2.0 V. In the Data_1 bus state (which represents logical state 1 or HIGH), BP is 3.5 V and BM is 1.5 V, giving a differential voltage of +2.0 V.

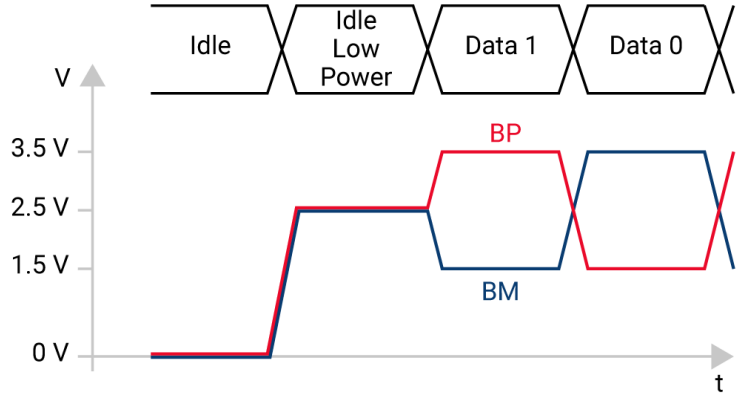

Figure 32 – FlexRay bus states

As with CAN, FlexRay uses non-return to zero (NRZ) bit coding, whereby the voltage level only changes when there is a change in the corresponding logic level, so there can only be two data states on the bus, LOW (0) or HIGH (1).

3.3.2 FlexRay network topologies

A group of FlexRay nodes connected via the FlexRay bus is known as a FlexRay Cluster. The standard allows for a very flexible topology within the cluster, using Linear, Star, active Star or a combination. To provide fault tolerance it is allowable to use two communication channels (a and b) each with maximum data rate of 10 Mb/s. The second channel can also be used as a way of increasing the maximum data rate to 20 Mb/s. The decision whether to use the second channel for fault tolerance or increased throughput can be made on a message-by-message basis.

In a point-to-point two-node bus, the standard specifies a maximum bus length of 24 meters to maintain a maximum data rate of 10 Mb/s. If additional nodes are passively spliced into this bus in a star configuration, the maximum bus length between any two nodes must also not exceed 24 meters and no more than a total of 22 nodes can be connected using this configuration. To increase the maximum cable length, as well as provide improved electrical performance, it is possible to use an active star coupler that receives each signal, buffers it and then distributes it to all the other branches of the star. In this case the cable length between each node and the active star coupler must not exceed 24 meters, which means that the maximum node to node cable length in a single active star is 48 meters.

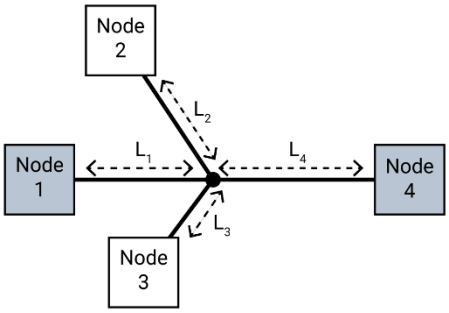

Figure 33 – FlexRay passive star network topology

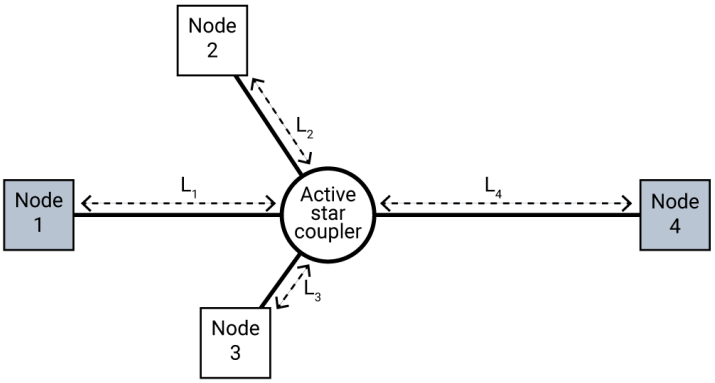

Figure 34 – FlexRay active star network topology

At each node, the ECU's host processor is connected to the FlexRay bus using a FlexRay Interface through a Controller Host Interface (CHI). The FlexRay interface consists of a FlexRay Controller as well as one or two (for two channel implementations) Bus Drivers in the FlexRay Bus Interface. The FlexRay controller implements the FlexRay communication protocol and is electrically coupled to the bus by the bus interface. This converts the logical data received from the controller into electrical signals when transmitting and converts the electrical signals into logical data when receiving. Additionally, the bus interface is also connected to the host ECU, which allows the host to control the state of the bus driver. The four allowable states are Normal, Standby, Sleep and Receive only (the last two being optional).

3.3.3 FlexRay bus access and timing

FlexRay nodes are granted bus access in two ways:

1) The first is using TDMA to send time-triggered messages using allocated Time Slots of equal length in a Static Segment time window, with each time slot being allocated to each message at every node using a Communication Schedule. During these time slots, the node is given exclusive bus access.

Slot	Start	Node	Message
1	t_1	P	P_{29}
2	t_2	L	L_{15}
3	t_3	B	B_6
4	t_4	W	W_{14}
5	t_5	C	C_{12}

Figure 35 – FlexRay static slot communication schedule

The length of the static time slots is driven by the length of the longest message in the static slot communication schedule. Each slot is made up of four time-segments: the Action Point Offset, which can be regarded as idle time before the start of message transmission; the FlexRay Frame which contains the message itself, beginning with the Action Point which signals the start of the message; the Channel Idle Delimiter signaling the end of the message; followed by a period of Channel Idle, whose length corresponds to the action point offset (11 recessive bits). The purpose of the Action Point Offset and Channel Idle is to allow

non-transmission time to accommodate signal delays and the largest possible advanced and delayed clock deviations permissible in the FlexRay cluster.

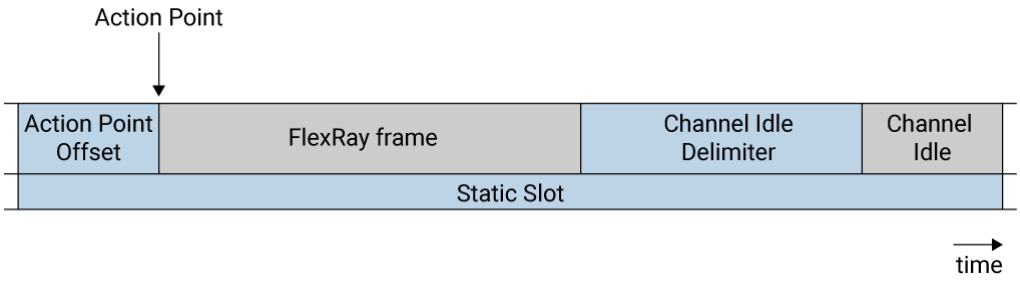

Figure 36 – FlexRay static slot

All nodes periodically execute the communication schedule, and this defines the FlexRay Communication Cycle. This ensures that all static messages are deterministically transmitted with a specific period. The length of the communication cycle is set during the network design phase, but typically may be in the region of 1 to 5 milliseconds in duration.

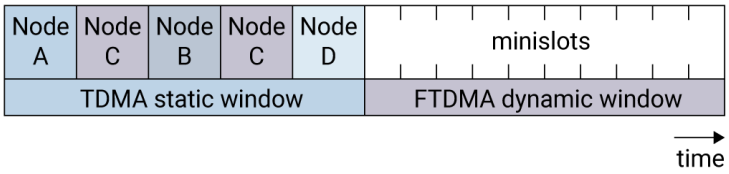

Figure 37 – FlexRay communication cycle

2) The second method is to send event-triggered messages asynchronously using Flexible TDMA (FTDMA), by extending the communication cycle using an optional Dynamic Segment. This directly follows the static segment and, to avoid affecting the deterministic data transmission of the static segment, the dynamic segment is of fixed length.

	Slot	Start	Node	Message
Static segment	1	t_1	P	P_{29}
	2	t_2	L	L_{15}
	3	t_3	B	B_6
	4	t_4	W	W_{14}
	5	t_5	C	C_{12}
Dynamic segment				

Figure 38 – FlexRay communication schedule (including dynamic segment)

The length of the dynamic segment is set statically in the system design phase and the time is divided up into “minislots”, with each message being allocated Dynamic Slots which use one or more minislots. Dynamic Slots are allocated on a message priority basis, with highest priority messages being allocated dynamic slots earliest in the dynamic segment time window using a Dynamic Segment Communication Schedule. On the occurrence of an event, the node requests to transmit the message using its allocated Dynamic Slot. During

transmission, other controllers in the cluster suspend counting the minislots and only resume counting at the end of the slot. Following the end of the slot is another minislot. If a send request is not transmitted by the node allocated a specific Dynamic Slot, the minislot count is incremented by one and that bandwidth is wasted. This process is continued until the segment is not long enough to transmit a new lower priority message and any unsent messages must wait for the dynamic segment in the next communication cycle. This means that the system designer must ensure that the dynamic segment length is long enough for the longest dynamic message and that the schedule ensures that low priority dynamic messages can be transmitted.

Slot	Node	Message	Event
6	A	A4	
7	C	C4	ϟ
8	D	D3	
9	B	B4	ϟ
10	E	E3	

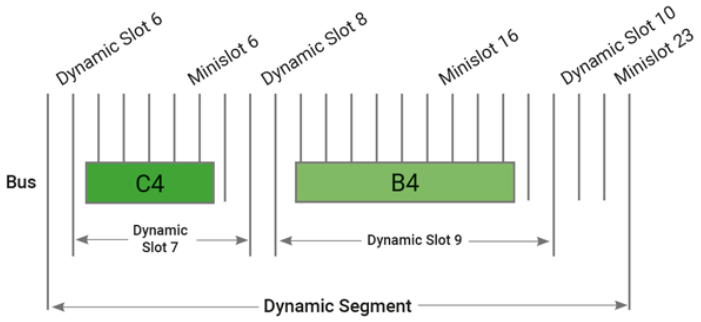

Figure 39 – FlexRay dynamic communication schedule and dynamic segment

A Dynamic Slot is structured in a similar fashion to a static slot, with the exception that the standard specifies that the message must end on the next minislot action point. To ensure this the message is lengthened with a Dynamic Trailing Sequence (DTS), which can be up to one minislot in length.

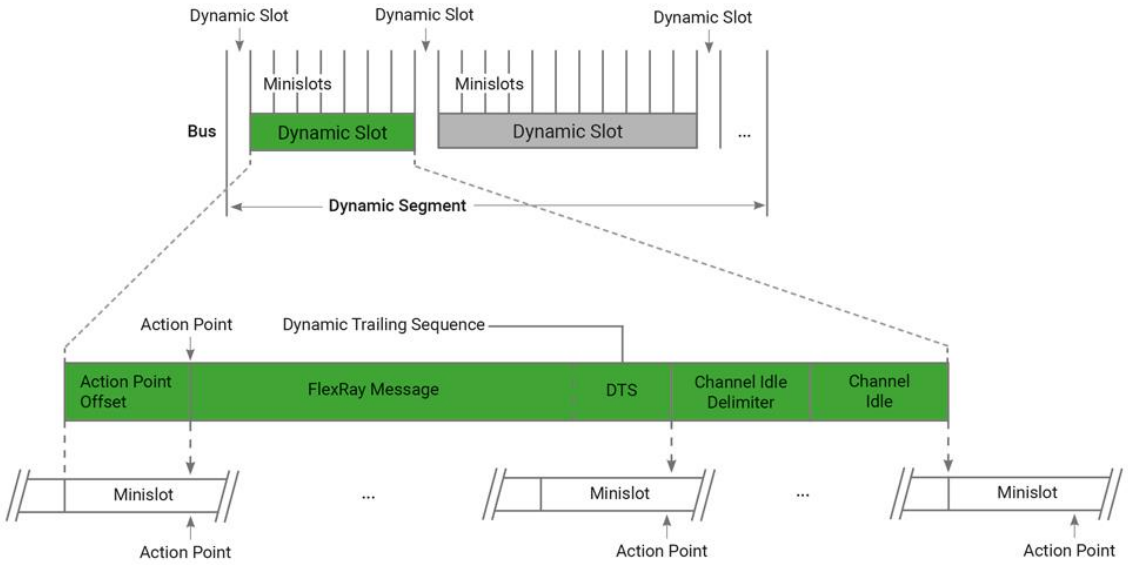

Figure 40 – FlexRay dynamic slot

The communication cycle consists of a minimum of two elements, the static segment and the Network Idle Time (NIT) segment, which is a transmission-free time used to synchronize local clocks. The dynamic segment is optionally transmitted, as is the Symbol Window, which is used for network maintenance as well as for waking up a FlexRay cluster. Each element within the communication cycle is a fixed number of macroticks in duration.

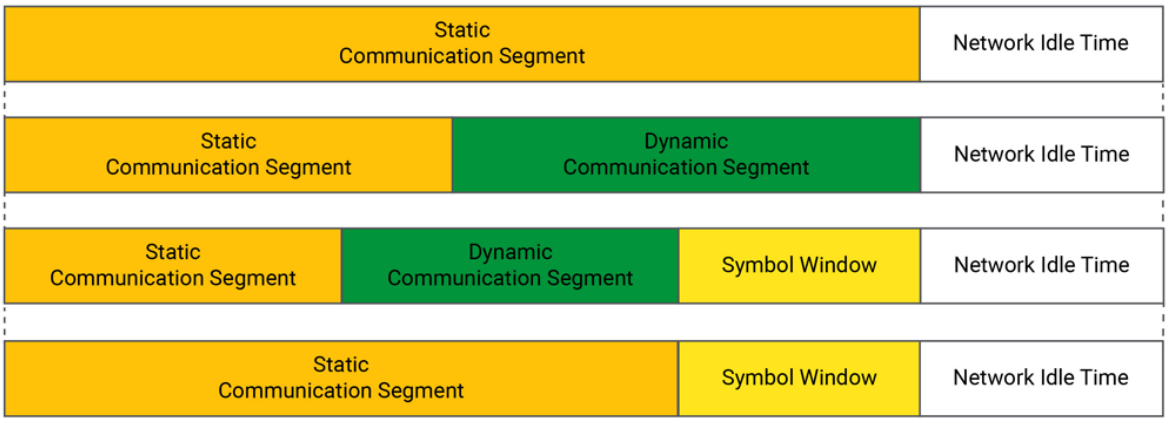

Figure 41 – FlexRay permitted communication cycle variants

As FlexRay is a time-sensitive synchronous network, accurate timing is essential and each FlexRay controller is responsible for ensuring that its own timebase is accurate. In a FlexRay cluster to define a virtual global timebase, between 2 and 15 nodes are assigned as Sync Nodes (Synchronization Nodes), which transmit a Sync Message in a defined static slot in each communication cycle. As each controller knows the start time of every static slot in the network, it builds a list of the timing of sync messages and using the Fault Tolerant Midpoint (FTM) algorithm calculates its own Offset Correction Value. Extreme values are discarded from the list to ensure that seriously inaccurate clocks do not disrupt system timing.

Each macrotick is typically 1 µs in duration. Every FlexRay controller in the cluster divides this into an integral number of microticks, which represent the timebase of the node's local oscillator. As the local oscillator's crystal frequency cannot be changed, changing the division ratio lengthens or shortens the microticks, effectively speeding up or slowing down the clock relative to other clocks in the network. Adjusting the frequency only ensures that the data rate is correct. To ensure that all communications start at the same moment in time, it is also necessary to correct for any oscillator phase offset. To allow for the phase and frequency offsets of its local oscillator, microticks are added or subtracted by the controller during the NIT at the end of each communication cycle. This ensures that each node can adjust the length of the communications cycle so that the first macrotick at the start of the next communication occurs at precisely the same time at every node on the network.

When used in time or safety critical applications, it is important that FlexRay nodes only attempt to gain bus access in their allocated time slots, to avoid disrupting communications through unauthorized transmissions. These can be avoided using an optional Bus Guardian (BG) which is allocated to each FlexRay Bus Interface to disable transmission outside its allocated time slots.

Physical message transmission on a FlexRay bus does not commence with the first bit of the frame, but with a Transmission Start Sequence (TSS), which consists of a continuous LOW level for a preconfigured period. This is provided to allow active star couplers sufficient time to reach their operating state (Star Truncation) allowing them to transfer the frame from the receiving branch to the other branches. Following the TSS is a Frame Start Sequence (FSS), consisting of one HIGH bit, after which the frame transmission can begin. Each byte of the frame is preceded with a Byte Start Sequence (BSS) which is used by the receiver for resynchronization and consists of one HIGH bit followed by one LOW bit. The end of the frame is terminated with a Frame End Sequence (FES), which consists of one LOW bit followed by one HIGH bit.

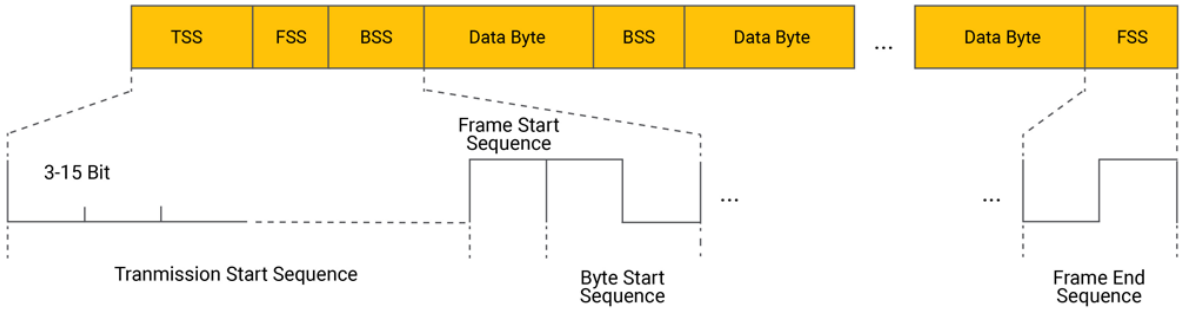

Figure 42 – Static message transmission

3.3.4 FlexRay data link layer

Data transmission is accomplished using FlexRay Frames, which consist of three segments: a 40-bit Header; the Payload Segment containing up to 254 bits of data; and a Trailer which is a 24-bit cyclic redundancy check (CRC).

The header starts with a Reserved bit (transmitted as 0) and then four indicator bits:

1. The Payload Preamble Indicator, when set to 1, is used in two ways. In a static frame, it indicates the presence of a Network Management Vector (NMVector) in the payload segment. In a dynamic frame, the payload preamble indicator signals that the Message Identifier is contained within the payload segment.
2. The Null Frame Indicator, when set to 0, indicates that the payload contains only bytes with a value of zero. An example of when the use of this indicator would be valid is when the communication schedule mandates that a controller should transmit a static message, but the host is not able to provide the message data to the controller. In this case the controller automatically sets the null frame indicator and transmits a Null Frame. When set to 1, the null frame indicator signals that the payload segment contains valid data.
3. The Sync Frame Indicator, when set to 1, signals that the frame contains a sync message. When set to 0, it signals that the message data should not be used for clock synchronization purposes.
4. The Startup Frame Indicator, when set to 1, signals that the frame is a Startup Frame, which is used during initialization of a FlexRay cluster. Only certain nodes, known as Coldstart Nodes (which are also always sync nodes) are permitted to transmit startup frames, which are in practice a special form of sync frame. This means that when the startup frame indicator is set to 1, the sync frame indicator is also set to 1.

The header also contains an 11-bit Frame ID which corresponds to a slot number. This can be of any value from 1 to 2047, with 0 being used to identify invalid frames.

The 7-bit Payload Length indicates the size of the payload segment in Words and is encoded as the number of payload bytes divided by 2. For static frames, this value is always the same, whereas the payload length for dynamic frames may vary.

The final segment within the header is a 6-bit Cycle Count, which represents the number (up to 63) of the communication cycle in which the message is sent.

The header CRC is calculated across the following elements: sync frame indicator, startup frame indicator, frame ID and payload length. The CRC for the header of each message is

not calculated by the transmitting controller but is calculated off-line and is then preconfigured in the controller. The CRC is however calculated by the receiving controller and cross-checked against the transmitted CRC value.

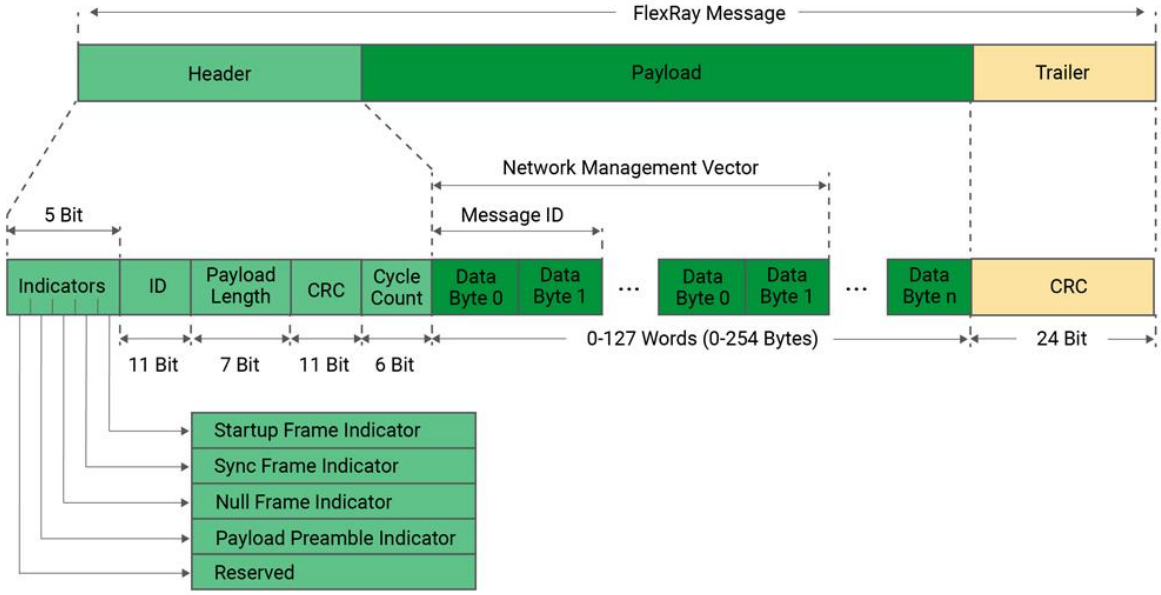

Figure 43 – FlexRay frame

The payload segment may carry up to 254 bytes. In a static frame payload segment, if the payload preamble indicator is set, the first 12 bytes carry the network management vector. This vector can be used by high-level host-based network management software to provide cluster-wide coordination of node behavior based on the state of higher-level applications. In a dynamic frame payload segment, if the payload preamble indicator is set, the first 2 bytes carry the message identifier. This identifier could be used, for example, as a way of filtering dynamic messages across the system.

The frame CRC is calculated across both the header and the payload and is performed both by the transmitting controller and the receiving controller. The polynomial sequence used to calculate the CRC is defined by the standard and once calculated is appended to the frame as a trailer.

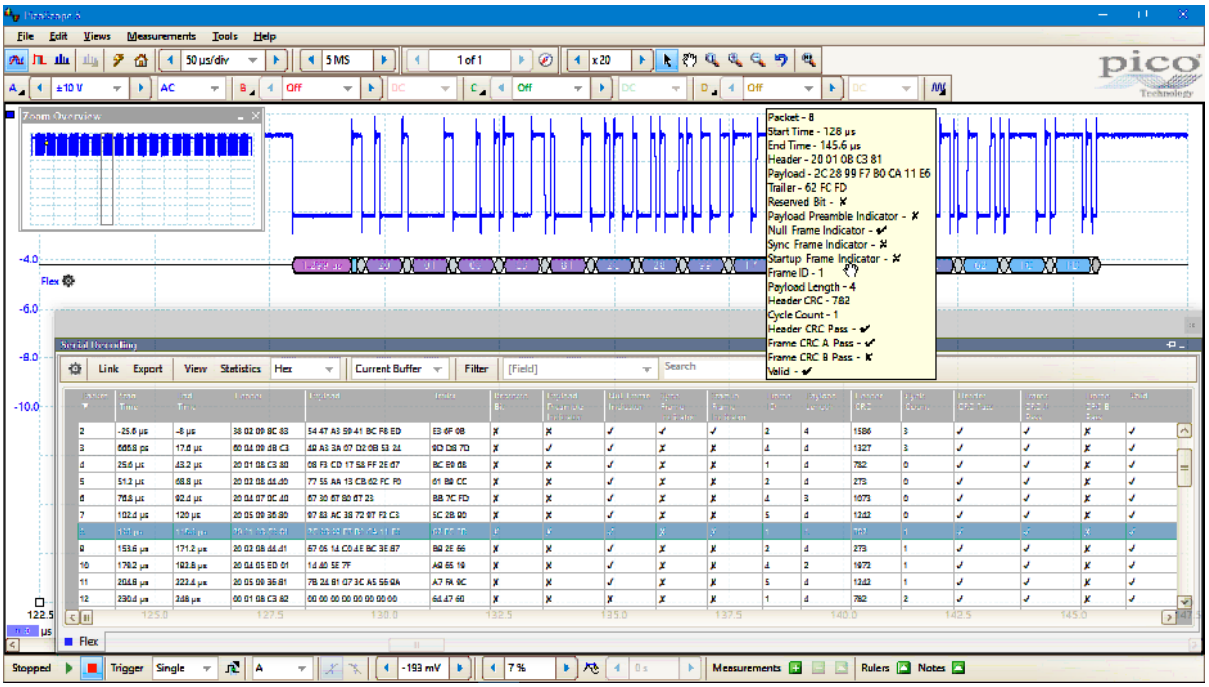

Figure 44 – PicoScope decoded FlexRay signal

3.4 LIN

- LIN is specified as the ISO 17987-1:2016 to 17987-7:2016 set of standards

3.4.1 LIN physical layer

The LIN physical layer is based on ISO 9141 (K-Line) and is a bidirectional single-wire configuration, with defined voltage ranges to specify recessive (logical 1) and dominant (logical 0) levels. The internal supply voltage to the ECU electronics (V_{sup}) and vehicle ground are used as the reference voltages for the recessive and dominant bus levels. As the system is non-differential, in order to ensure adequate noise immunity, the voltage ranges are specified differently for senders and receivers. Senders transmit voltages less than 20% of the supply voltage to represent dominant (0) and greater than 80% of the supply voltage to represent recessive (1). To accommodate induced noise on the signal wire, receivers interpret any voltage of less than 40% as dominant and any voltage greater than 60% as recessive.

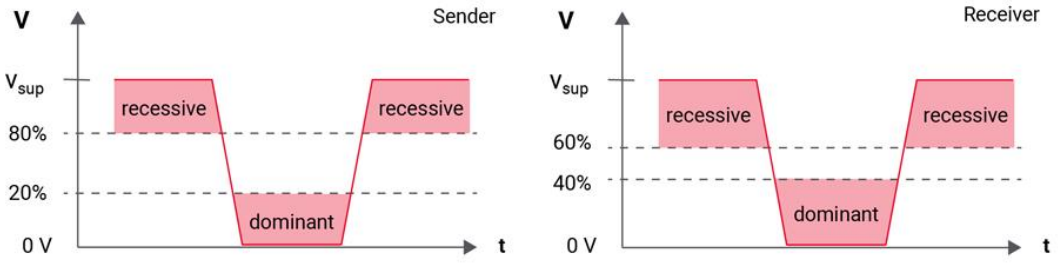

Figure 45 – LIN sender and receiver voltage levels

Each LIN node transceiver is connected to the LIN bus via an open collector circuit and the bus is connected to the battery voltage (V_{bat}) through a pull-up resistor and a protection diode. This diode prevents the bus from powering the ECU in the case of loss of battery. The pull-up resistors have a nominal value of 1 kΩ in the case of bus masters (this resistor

terminates the bus) and 30 kΩ in the case of bus slaves. V_{sup} can be in the range 7 V to 18 V and may be lower than the battery voltage (V_{bat}), due to ECU protection filters and bus loading.

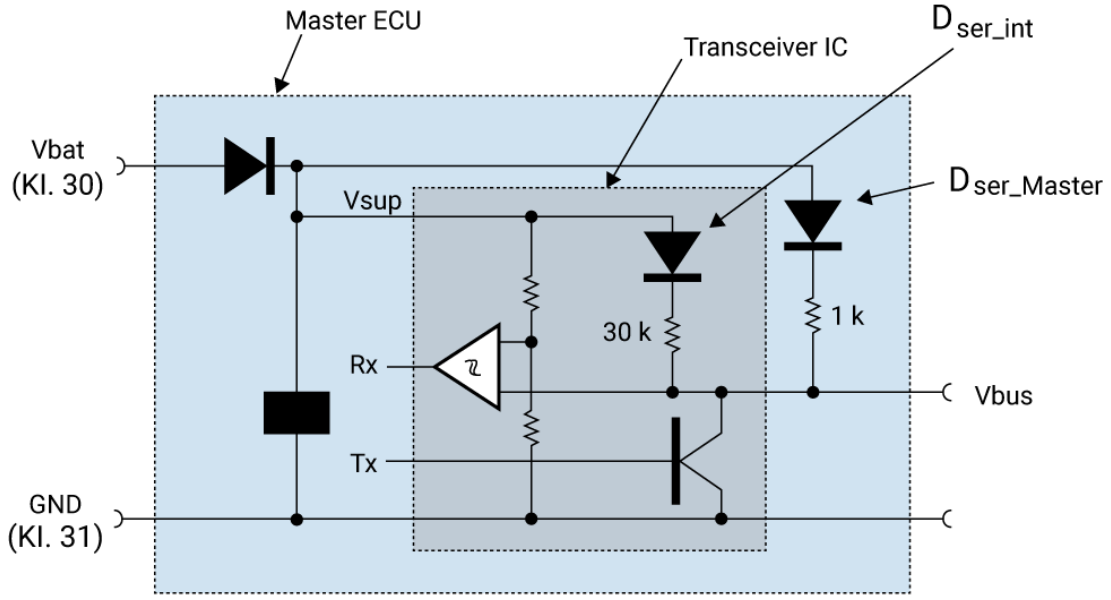

Figure 46 – ECU and LIN bus master transceiver voltages (from the LIN 2.0 specification)

3.4.2 LIN data link layer

A LIN Cluster consists of a single master and up to 15 slaves. The master contains a schedule table which it uses to determine both which and when the next frame is to be transmitted during its Frame Slot, which includes the duration of the frame and an Inter-frame Space. The total duration of the schedule is known as the Communication Cycle, which is repeated on completion. The communication in a LIN cluster is deterministic (predictable), as there is a fixed time sequence. However, as the system is entirely reliant on a single master, the system will fail if the master fails, making LIN unsuitable for safety-critical applications.

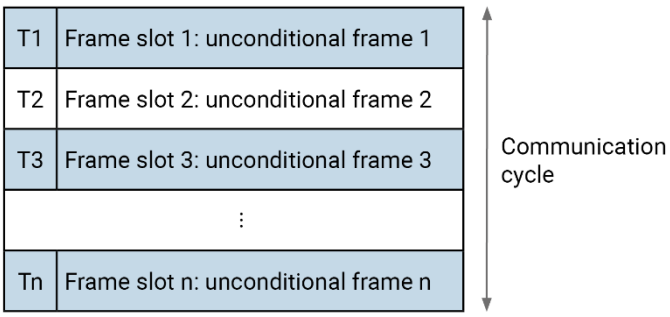

Figure 47 – LIN schedule

As there are no dedicated hardware communication controllers, the LIN communications protocol is implemented in software within the microcontroller of each LIN node. The microcontroller is connected to the transceiver through a Serial Communication Interface (SCI) which transmits SCI Frames. Each SCI frame consists of 8 data bits (one byte) preceded by a dominant start bit (used for receiver resynchronization) and terminated with a stop bit. A LIN frame is comprised of multiple SCI frames. The node can delay the sending of

SCI frames, which leads to so-called Interbyte Spaces or pauses in the LIN frame transmission.

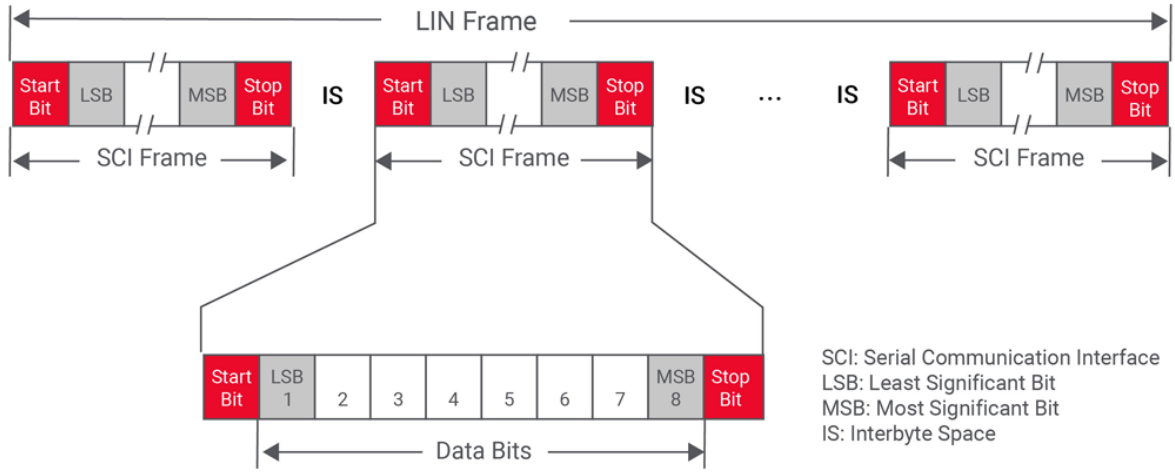

Figure 48 – LIN frame constructed of multiple SCI frames

The overall network is configured by the LIN Definition File (LDF), which defines the bus speed, node list, signals (sensor and other data), frame information as well as the schedule. The LDF is used to configure the software tasks in both the master and slaves. The master device runs both the Master Task, which processes the schedule, and a Slave Task, which undertakes communication. Slaves run just the slave task.

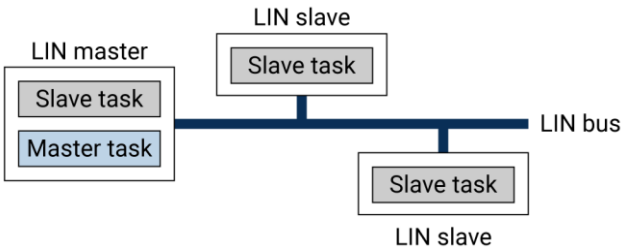

Figure 49 – LIN communications software tasks

All frames are initiated by the master, which transmits a header for the desired frame. The header consists of a unique sequence of at least 13 bits and 1 recessive delimiter bit, called a Sync Break Field, which all receivers recognize as the beginning of a header. This is followed by the master's cycle frequency defined by the Sync Field. This contains the hexadecimal value 0x55, which is a sequence of five falling and rising edges. The receivers use this for synchronization by measuring the time between the first and last falling edge and dividing that value by 8. The result is one bit-time, as the sync field is a single SCI frame of 8 bits in length.

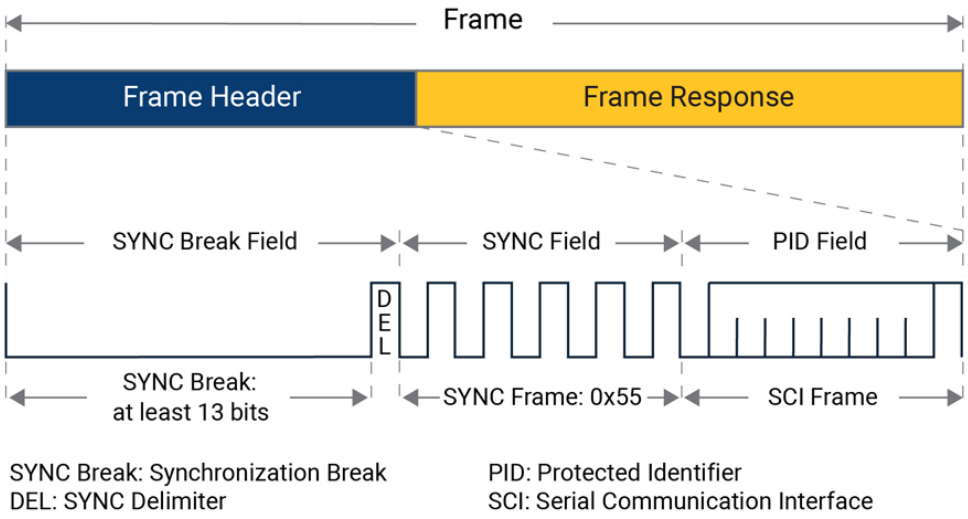

Figure 50 – LIN bus synchronization

This synchronization is required for each frame, as the nodes are implemented using low-cost resistor/capacitor (RC) resonant oscillators rather than precision crystal oscillators, these RC oscillators being prone to high levels of frequency drift.

The remaining part of the header is the Protected Identifier (PID), which is unique to each frame. Nodes determine from the PID whether they need to send a response (signal data). As the PID is 6 bits in length, there are 64 possible values, four of which are predefined for diagnostic and future reserved purposes. The PID is protected by a 2-bit parity check.

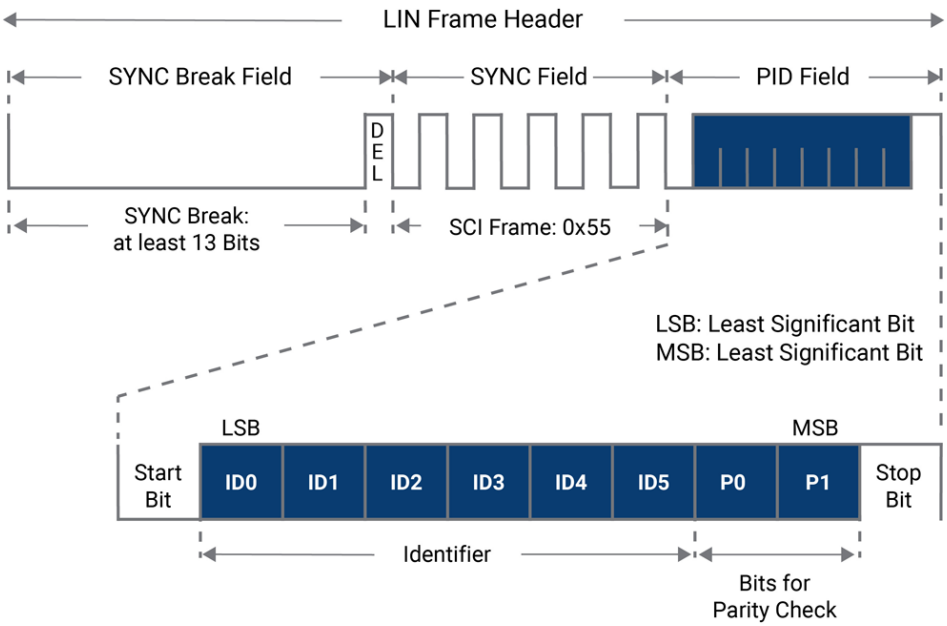

Figure 51 – Frame header sent by the LIN Master with protected identifier

After the header is completed there is a pause in transmission known as the Response Space, which allows the sending node responsible for the frame referred to by the PID (known as the Publisher) time to switch from receive to transmit. This node generates a Frame Response containing the signal data. This can be up to 8 bytes in length and is

transmitted in ascending order from the least significant bit to the most significant bit of the response data. The data is protected by a checksum.

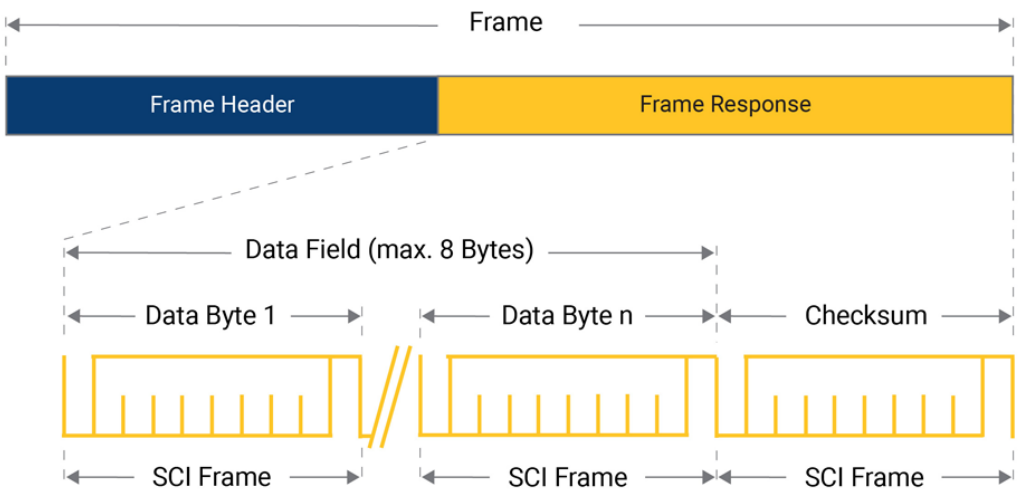

Figure 52 – LIN frame header and frame response

Nodes intended to use the data sent in the frame response (known as Subscribers) are defined in the LIN Description File (LDF).

There are three LIN frame types used to transmit signal data: Unconditional, Event triggered and Sporadic.

Unconditional frames are the regular form of LIN communication where the master sends a header in the scheduled frame slot and the publisher node designated by the PID responds with signal data. As only one node can respond to the header, no collisions can occur with unconditional frames.

Event-triggered frames are used to send data only when necessary, thereby saving bus bandwidth. The event triggered frame is associated with multiple unconditional frames, so any of the designated publishers can respond to the frame header, but only if at least one signal referenced in the frame has changed. If no signal data needs to be updated, no response will be sent and the header is ignored. If a node has updated data, it will send a frame response and to determine which node responded, the first byte of the frame response contains an additional PID that identifies the sender. As multiple nodes can respond to the header, it is possible for collisions to occur. When a collision occurs, the master immediately switches to a Collision Resolving Schedule, where the master sends a series of unconditional frame headers to get responses from each of the designated node in turn. On completion of the collision resolving schedule, the master returns to executing the original schedule.

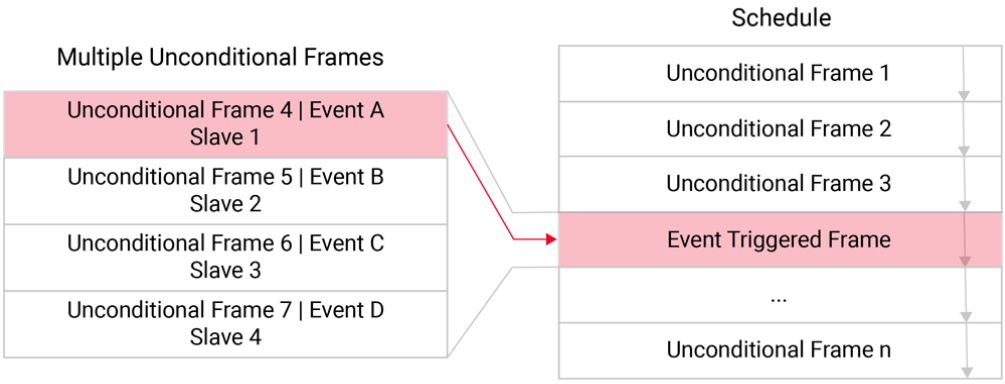

Figure 53 – Event triggered frame

Sporadic frames are used to send data that occurs infrequently and provides some ability to for LIN to provide dynamic behavior, as well as reducing the communication cycle time, by removing frame slots that will seldom contain updated response data. A sporadic frame could be regarded as an event triggered frame where only the master can be the publisher. The sporadic frame is, in practice, a group of unconditional frames that all share the same frame slot, but only one is sent by the master if necessary. The master only sends a frame header when it knows that signal data in the related frame has changed. The master's slave task sends the relevant unconditional frame response to the header and the designated subscribed slaves then use the data.

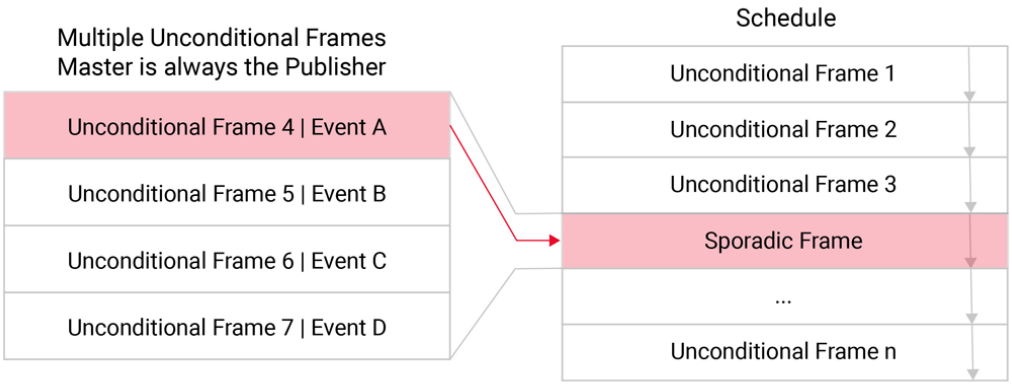

Figure 54 – Sporadic frame

If no signals have changed in any of the related unconditional frames, the frame slot is left empty. If multiple frames contain updated signal data, the master sends the related frame headers in a priority sequence defined by the LDF on multiple executions of the communication cycle. As only the master can publish a sporadic frame, no collisions can occur.

LIN also offers two kinds of Diagnostic Frames for the transmission of diagnostic messages, the Master Request Frame (diagnostic request) and the Slave Response Frame (diagnostic response) and both always contain 8 data bytes. In the case of the Master Request Frame (frame identifier = 60), the master sends both the Frame Header and Frame Response. The master then sends out a Frame Header for the Slave Response Frame (frame identifier – 61) and the target slave provides the frame response, which contains the diagnostic message.

Diagnostic frame responses always start with a 1-byte Node Address (NAD) followed by 1 byte of Protocol Control Information (PCI), which indicates the transfer mode for the diagnostic data. This can either be Unsegmented (single-frame) or Segmented (multi-frame), which is used when a single frame is insufficient to transfer the diagnostic message data. The third byte is the Service Identifier (SID), which is used by the master in the frame response of the Master Request Frame to indicate which diagnostic service should be executed. In the slave response frame, the third byte contains the Response Service Identifier (RSID) which specifies the contents of the response. Frame identifiers 62 and 63 are reserved frames and should not be used.

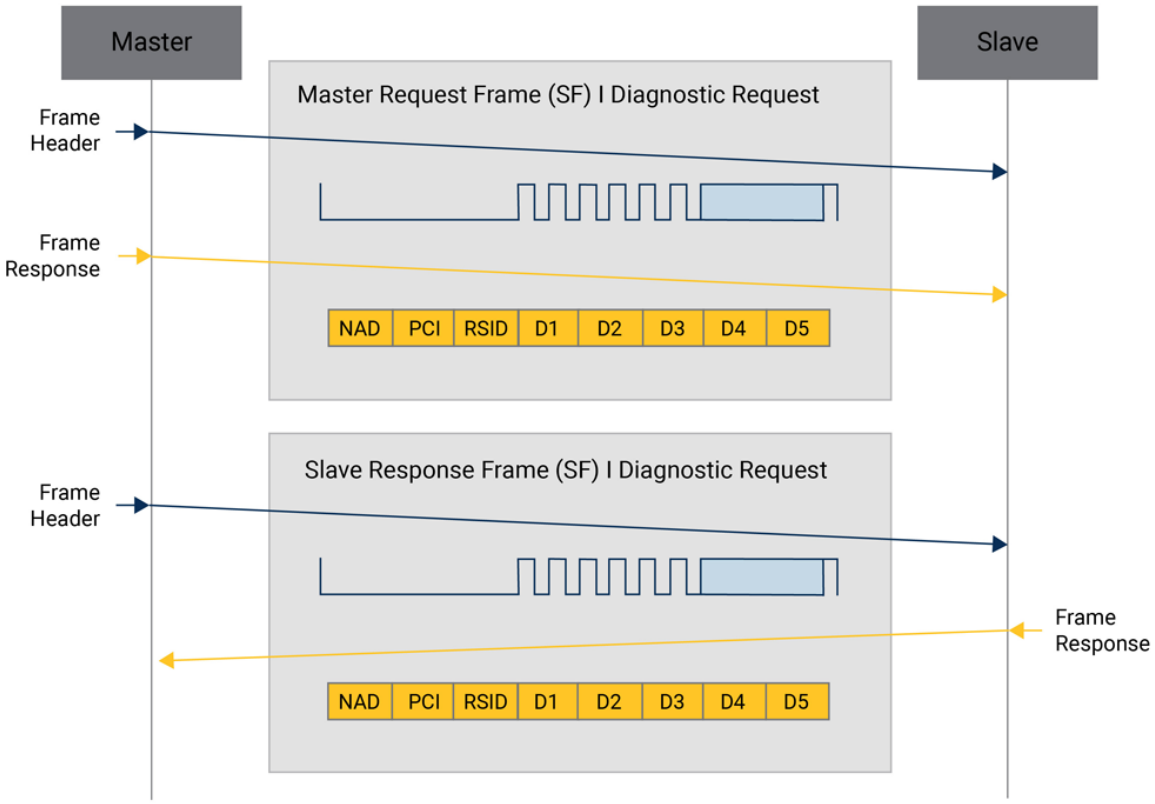

Figure 55 – Unsegmented (single frame) diagnostic request and response

LIN offers in-built network management facilities, which allow control over the state of the slaves in the cluster. There are three slave states: Initializing; Operational and Bus Sleep Mode. When powered up, a slave switches to the initializing state and within 100 milliseconds to the operational state. The master can set the cluster's slaves to the bus sleep mode by transmitting the Go-To-Sleep command, which is a master request frame where the first byte has a value of 0 (corresponding to a NAD of 0). All the remaining 7 bytes of data are set to logical 1. Slave nodes must also autonomously go into the sleep state after a minimum of 4 seconds and a maximum of 10 seconds of bus inactivity.

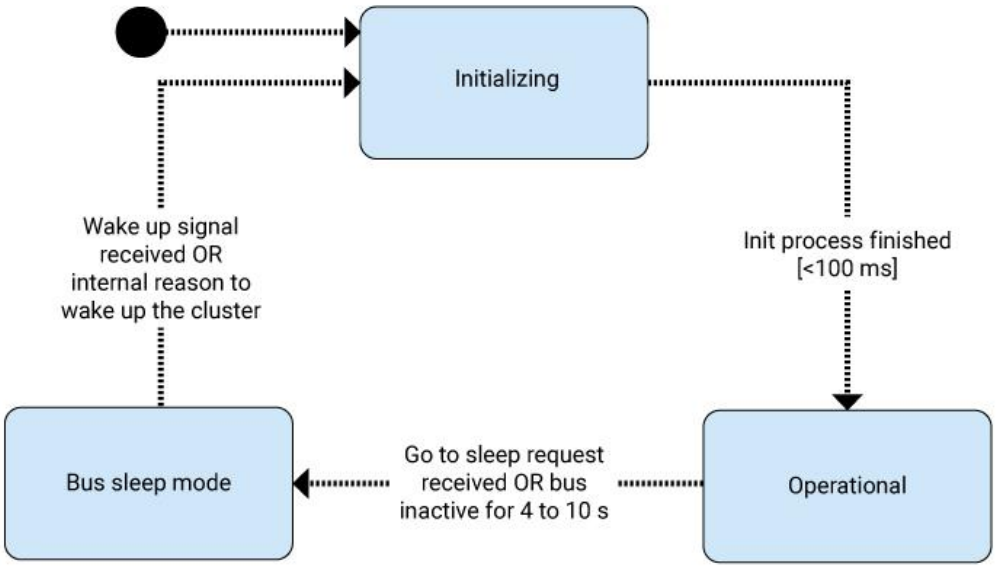

Figure 56 – Slave node state diagram (from the LIN 2.2A specification)

Any node can wake up the cluster by sending a wake-up signal, which is a dominant pulse of between 250 microseconds and 5 milliseconds in duration. All nodes should wake up and switch back to the operational state within 100 milliseconds.

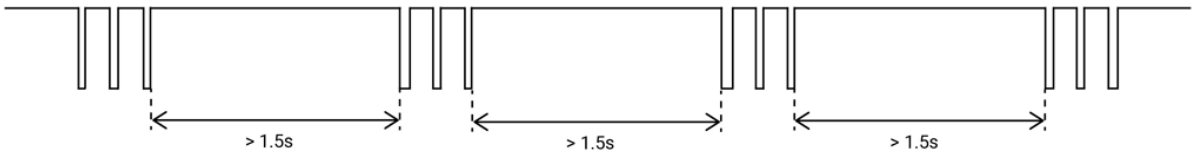

Figure 57 – Wake-up signal retry sequence (from the LIN 2.2A specification)

If no header is detected within 150 to 250 milliseconds, another wake-up signal is sent by the node. If a header is still not detected, a wake-up signal is sent for a third time. After three failed attempts, the node waits 1.5 seconds before repeating the process.

3.5 SENT

- SENT is specified as the SAE J2716 standard

3.5.1 SENT physical layer

The SENT physical layer is a unidirectional (output only) single-wire bus, requiring a 5 volt supply and a ground reference. The signal is nominally a 5 V square wave although the waveform is filtered to minimize radiated EMC emissions. These heavily damped rise and fall transitions do however limit the maximum data rate that can be achieved.

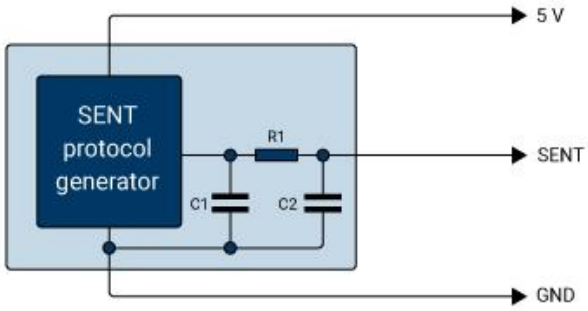

Figure 58 – SENT transmitter

The maximum low state voltage is specified as 0.5 V and the minimum high state voltage is specified as 4.1 V. For a nominal 3 microsecond clock tick, the rise time (1.1 V to 3.8 V) is specified as 18 microseconds and the fall time (3.8 V to 1.1 V) is specified as 6.5 microseconds. Although the nominal clock tick is 3 microseconds, the maximum allowable clock tick is 90 microseconds, with time being measured between successive falling clock edges. With higher values of clock tick, the rise and fall times should be increased proportionally. With a 3 microsecond clock tick, the net data rate is approximately 30 kb/s.

Figure 59 – SENT waveform

3.5.2 SENT data link layer

As soon as a SENT transmitter is powered up, it will begin transmitting data, making operation similar to analog sensor devices, although a SENT transmitter is not limited to transmitting a single parameter. The main data is sent on a fast channel, with optional secondary data sent on a slow channel. Data is sent as 24 bits (six 4-bit nibbles) as signal 1 and signal 2, allowing two sensor values to be sent in a single message. The number of nibbles in signal 1 and signal 2 is fixed for each application (such as throttle position, mass air flow) and can vary between applications (12+12, 16+8 and so on).

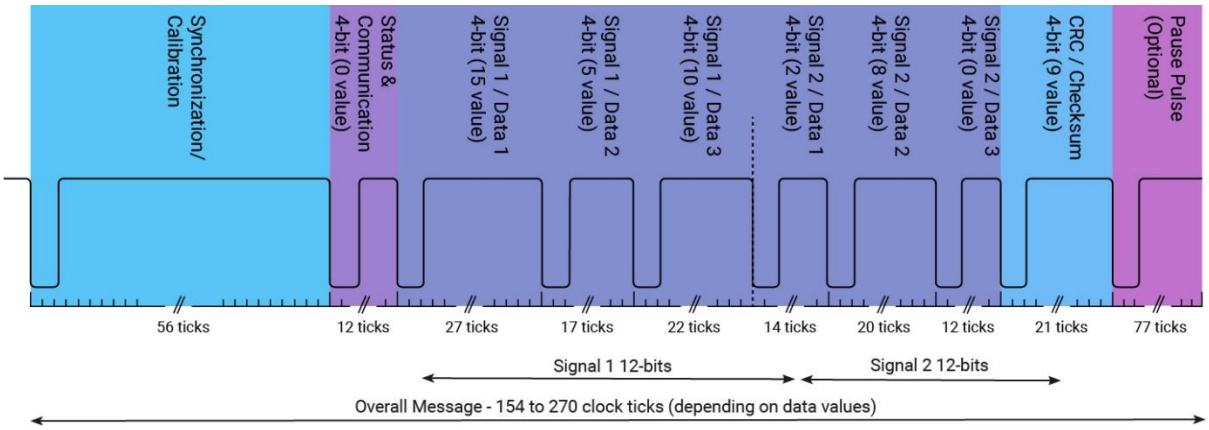

Figure 60 – SENT fast channel transmission (SAE J2716)

A SENT message starts with a synchronization/calibration pulse where the time between successive falling edges is equivalent to 56 clock ticks. This initial sequence is used by the receiver (typically an ECU) to measure the clock tick timing of the sensor transmission, by dividing the pulse duration by 56 to determine the clock tick time.

The first nibble after synchronization is used for Status and Communication and is used for status and/or to communicate slow channel data, depending on the SENT format.

Data is transmitted in units of 4 bits (nibbles). Within a nibble, the initial logic 0 time is a fixed width of 5 ticks or more, which is followed by logic 1 with a variable duration. The total nibble time encodes 4 bits of data in the measured number of tick units. 12 ticks' duration = binary 0000 (hex 0x0), 13 ticks = binary 0001 (hex 0x1), 14 ticks = binary 0010 (hex 0x2) and so on up to 27 ticks = 1111 (hex 0xF). An additional fast channel format is single-secured, where one 12-bit data message is transmitted, along with an 8-bit incremental counter, which ensures data integrity.

The CRC/Checksum nibble is used for error checking.

A pause pulse may be inserted at the end of each message to fill up the message to a fixed clock tick count not exceeding 1 millisecond.

Serial message bits, otherwise known as Slow Channel data, are sent two bits at a time within the standard SENT message frame. For each fast channel message frame, the transmitter can also include two bits of slow channel data using bit 3 and bit 2 of the status nibble.

Bit Number	Bit Function
0 (least significant)	Reserved for specific application
1	Reserved for specific application
2	Serial Data message bits
3 (most significant)	Message start = 1, otherwise = 0 or Serial data message bits

Figure 61 – Status and communication nibble (SAE J2716)

The purpose of the slow channel feature is to allow up to 32 slow channel secondary data messages per serial message cycle to be transmitted with minimal impact on the primary

sensor data. Slow channel messages can be used to continually monitor information such as error codes and part numbers, which either do not change or change at a slower rate than the primary sensor data.

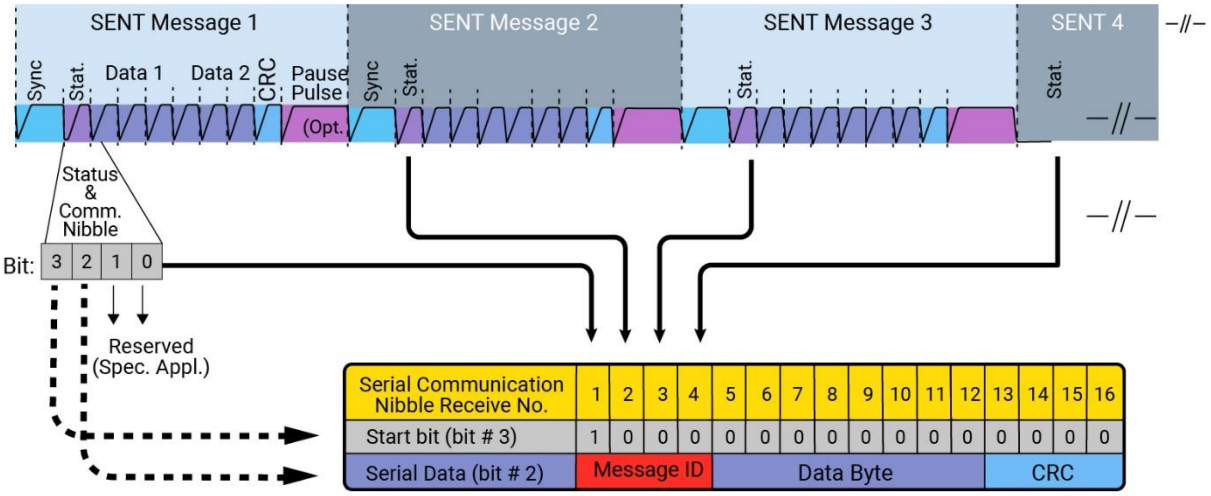

Figure 62 – SENT Slow serial data message constructed from 16 SENT messages (SAE J2716)

Note that the Status and Communication Nibble is not included in the frame CRC calculation and therefore can contain undetected errors.

The Status and Communication Nibble can also be used to transmit optional enhanced serial messages. These can be used by sensors that need to transmit longer messages or require a larger set of message IDs. Serial data is also transmitted in bits 3 and 2 of the status and communication nibble and the message frame stretches over 18 consecutive error-free SENT data messages.

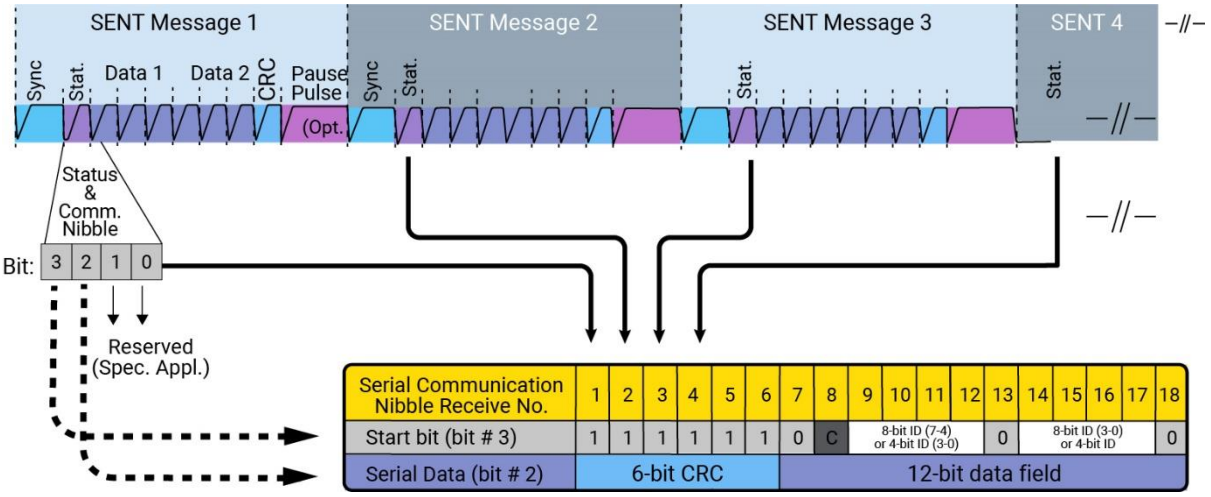

Figure 63 – Enhanced Serial Data Message constructed from 18 SENT messages (SAE J2716)

With SENT, it is important to choose which of the several formats is best suited to the application.

3.6 BroadR-Reach

BroadR-Reach is specified as:

- The Open Alliance BroadR-Reach® (OABR) Physical Layer Transceiver Specification for Automotive Applications – Version 3.2 - 2014
- IEEE P802.3bw-2015 Clause 96 Physical Layer Specifications and Management Parameters for 100 Mb/s Operation over a Single Balanced Twisted Pair Cable (100BASE-T1)

3.6.1 BroadR-Reach physical layer

The physical layer (Layer 1) implementation of BroadR-Reach differs from standard Ethernet (100BASE-TX) in several ways. Standard Ethernet cabling uses two twisted pairs to provide dual simplex transmission and reception, whereas in BroadR-Reach, a single differential voltage twisted pair is used for full duplex (bidirectional) transmission and reception. Pulse Amplitude Modulation (PAM3) is used to reduce the bandwidth used by over 2x compared with standard Ethernet. This allows BroadR-Reach to operate with lower quality cabling as well as allowing high levels of filtering to provide improved emissions and noise immunity. With Pulse Amplitude Modulation, the signal information is encoded as a series of pulses. To decode the signal information, the receiver must sample the amplitude value (voltage) at each signal period.

The MAC (Media Access Control) device transmits 4-bit words to the PHY (Physical layer interface) at 100 Mb/s clocked at 25 MHz. The PHY converts these 4-bit words to 3-bit words and increases the clock rate to 33.33 MHz to maintain the data rate at 100 Mb/s (66.66 megasymbols per second). Each 3-bit word (a symbol) is transmitted by the PHY as two ternary levels encoded using the PAM3 encoding described below (a ternary signal is one with three distinct levels).

3-bit Value	000	001	010	011	100	101	110	111
Ternary A (Volts)	-1	-1	-1	0	0	+1	+1	+1
Ternary B (Volts)	-1	0	+1	-1	+1	-1	0	+1

Figure 64 – PAM3 encoding (data symbols)

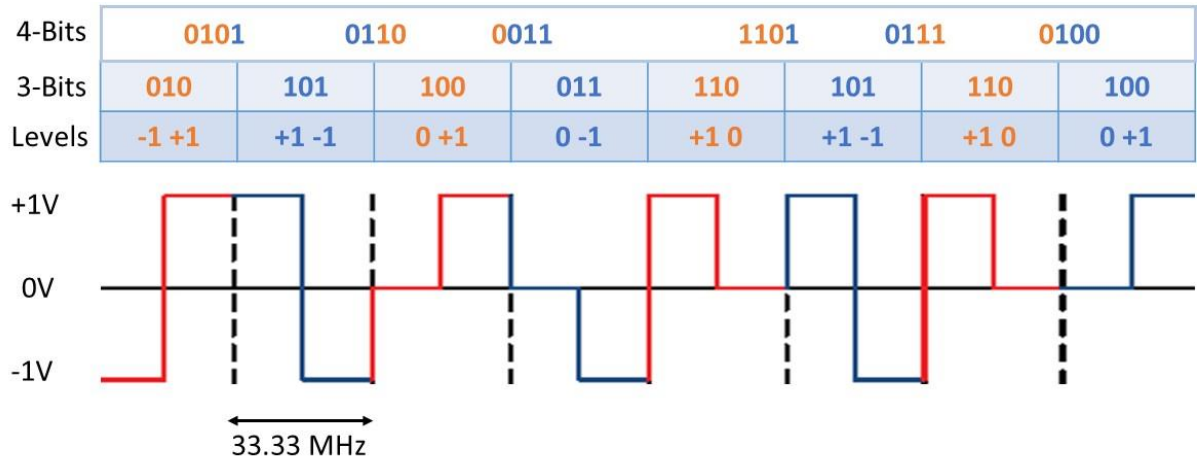

Figure 65 – PHY PAM3 4-bit to 3-bit conversion and data transmission

Each 33.33 MHz period represents one transition, which in turn represents 3 bits of data. In the case where an Ethernet frame generated by the MAC is not divisible by 3, stuffing bits are inserted by the transmitter's PHY and are then removed by the receiver's PHY.

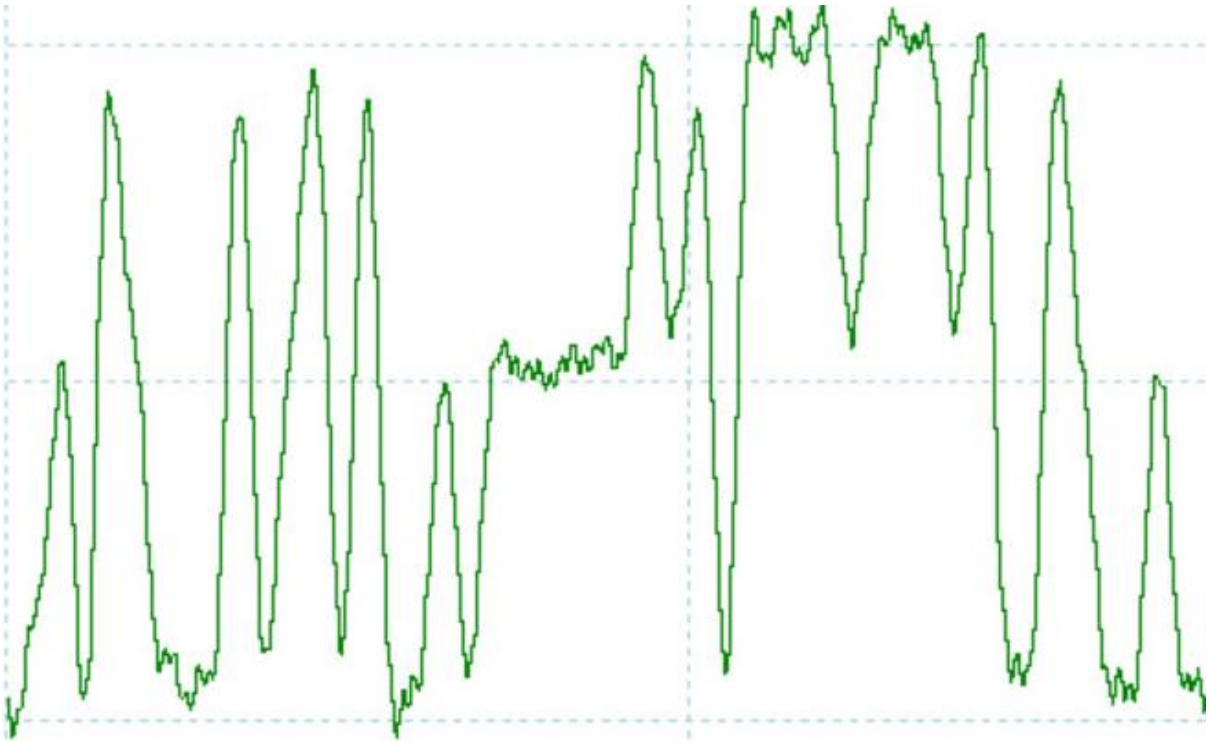

Figure 66 – PAM3 encoded waveform detail

As the system is full duplex, transmitted data from both PHYs is present at each PHY. BroadR-Reach PHYs use echo cancellation to remove their own transmitted signal and extract only the received data. Only two nodes can be connected, so in any link pairing, one PHY is dedicated by the controller as the master and the other as the slave. Additional nodes can be connected by using a coupling element, normally an Ethernet (Layer 2) switch.

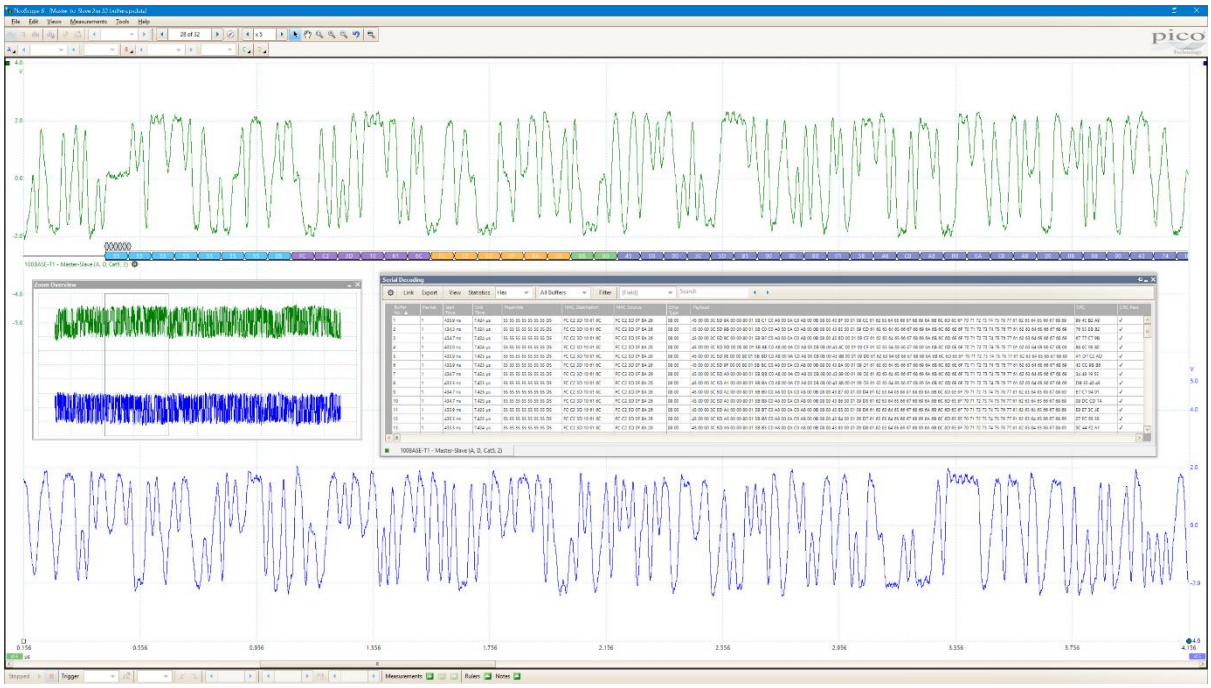

Figure 67 – BroadR-Reach Master and Slave waveforms and decoded data

With a BroadR-Reach PHY, data scrambling is used to randomize the sequence of transmitted symbols. This avoids a concentration of specific frequencies within the transmitted data spectrum. The master and slave scramblers and descramblers must be synchronized prior to operation. When the PHYs connect, they undertake a training process where the master transmits a continuous symbol stream to which the slave synchronizes. This leads to both the master and slave transmitting information at the same frequency and in phase.

In normal operation the link is in what is known as “idle mode”, where a specific sequence of symbols is generated as defined in the encoding rules for idle mode. When data transmission is enabled, a group of 3 ternary pairs all at 0 V are transmitted, known as Start-of-Stream Delimiter (SSD). Data is then transmitted as defined by the PAM3 data encoding. After transmission ends, another group of 3 ternary pairs all at 0 V are transmitted, known as End-of-Stream Delimiter (ESD). If an error is detected, then ERR_ESD is transmitted. The link then re-enters idle mode.

It was mentioned earlier that, although the terms BroadR-Reach and 100BASE-T1 are used interchangeably, there are variances. Although 100BASE-T1 is interoperable with BroadR-Reach, there are two small differences. In the physical-layer electrical test suite, the 100BASE-T1 specification defines a test for the maximum Transmit Peak Differential Output, which is not explicitly defined in the BroadR-Reach specification. The second difference is that 100BASE-T1 the specification shortens the protocol timing period for PHY-initialization (to ensure fast link establishment, BroadR-Reach does not use the IEEE 802.3 link auto negotiation process).

3.6.2 BroadR-Reach data link layer

At the data link layer (Layer 2) BroadR-Reach and 100BASE-T1 are identical to standard Ethernet. Each Ethernet controller implements a bus access and collision detection method known as Carrier Sense Multiple Access/Collision Detection (CSMA/CD). As the BroadR-Reach physical layer implements full duplex operation, collisions are unlikely to occur.

Data is transmitted as frames of 64 to 1522 bytes in length.

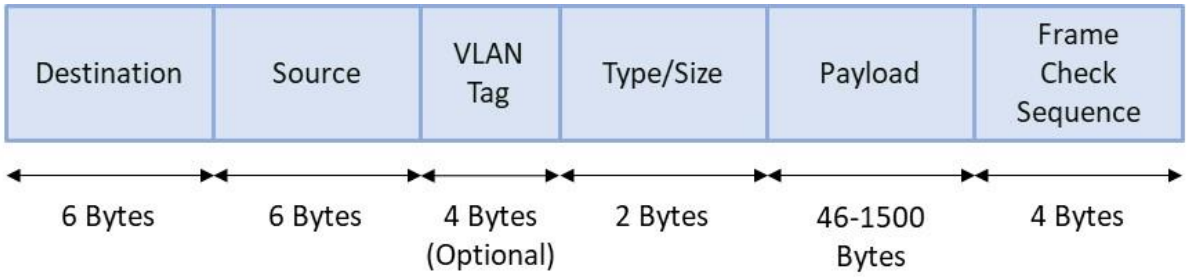

Figure 68 – Layer 2 Ethernet Frame

The header features the destination and source MAC addresses for the frame, an optional VLAN tag and the Ethernet Type/Size field. The VLAN tag, if present, indicates virtual LAN (VLAN) membership and message priority. A VLAN (Virtual Local Area Network) is a group of nodes on one or more links that communicate with each other privately as if they were attached to a dedicated single link. The Ethernet Type/Size field can be used in two ways. Values of 1500 and below mean indicate the size of the payload in bytes. Values of 1536 and above indicate that it is used as an “EtherType”, to indicate which protocol is encapsulated in the payload of the frame (for example, a value of 2048 decimal / 0x800 hexadecimal would indicate that the frame carries a payload of Internet Protocol version 4 (IPv4) data).

The minimum payload 46 bytes when a VLAN tag is absent and 42 bytes when the 4-byte VLAN tag is present. If the actual payload data is less than this value, padding bytes are added. The maximum payload is 1500 bytes.

Ending the frame is a 4-byte Frame Check Sequence (FCS) which is a Cyclic Redundancy Check (CRC) allowing detection of corrupted data within the frame as received. The FCS value is computed from all fields except the FCS itself.

MAC Destination	MAC Source	Ether Type	Payload	CRC
FC C2 3D 10 61 6C	FC C2 3D 0F 8A 26	08 00	45 00 00 3C 5D 9A 00 00 80 01 5B C1 C0 A8 00 0A C0 A8 00 0B 08 00 43 8F 00 01 09 CC 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69	99 4E B2 A9
FC C2 3D 10 61 6C	FC C2 3D 0F 8A 26	08 00	45 00 00 3C 5D 9B 00 00 80 01 5B C0 C0 A8 00 0A C0 A8 00 0B 08 00 43 8E 00 01 09 CD 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69	76 55 EB B2
FC C2 3D 10 61 6C	FC C2 3D 0F 8A 26	08 00	45 00 00 3C 5D 9C 00 00 80 01 5B BF C0 A8 00 0A C0 A8 00 0B 08 00 43 8D 00 01 09 CE 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69	67 77 C7 9B
FC C2 3D 10 61 6C	FC C2 3D 0F 8A 26	08 00	45 00 00 3C 5D 9D 00 00 80 01 5B BE C0 A8 00 0A C0 A8 00 0B 08 00 43 8C 00 01 09 CF 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69	88 6C 9E 80
FC C2 3D 10 61 6C	FC C2 3D 0F 8A 26	08 00	45 00 00 3C 5D 9E 00 00 80 01 5B BD C0 A8 00 0A C0 A8 00 0B 08 00 43 8B 00 01 09 D0 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69	A1 D7 C2 AD
FC C2 3D 10 61 6C	FC C2 3D 0F 8A 26	08 00	45 00 00 3C 5D 9F 00 00 80 01 5B BC C0 A8 00 0A C0 A8 00 0B 08 00 43 8A 00 01 09 D1 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69	4E CC 9B B6
FC C2 3D 10 61 6C	FC C2 3D 0F 8A 26	08 00	45 00 00 3C 5D A0 00 00 80 01 5B BB C0 A8 00 0A C0 A8 00 0B 08 00 43 89 00 01 09 D2 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69	34 48 19 5E
FC C2 3D 10 61 6C	FC C2 3D 0F 8A 26	08 00	45 00 00 3C 5D A1 00 00 80 01 5B BA C0 A8 00 0A C0 A8 00 0B 08 00 43 88 00 01 09 D3 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69	DB 53 40 45
FC C2 3D 10 61 6C	FC C2 3D 0F 8A 26	08 00	45 00 00 3C 5D A2 00 00 80 01 5B B9 C0 A8 00 0A C0 A8 00 0B 08 00 43 87 00 01 09 D4 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69	E7 C7 94 01
FC C2 3D 10 61 6C	FC C2 3D 0F 8A 26	08 00	45 00 00 3C 5D A3 00 00 80 01 5B B8 C0 A8 00 0A C0 A8 00 0B 08 00 43 86 00 01 09 D5 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69	08 DC CD 1A
FC C2 3D 10 61 6C	FC C2 3D 0F 8A 26	08 00	45 00 00 3C 5D A4 00 00 80 01 5B B7 C0 A8 00 0A C0 A8 00 0B 08 00 43 85 00 01 09 D6 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69	EB E7 3C 4E
FC C2 3D 10 61 6C	FC C2 3D 0F 8A 26	08 00	45 00 00 3C 5D A5 00 00 80 01 5B B6 C0 A8 00 0A C0 A8 00 0B 08 00 43 84 00 01 09 D7 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69	07 FC 65 55
FC C2 3D 10 61 6C	FC C2 3D 0F 8A 26	08 00	45 00 00 3C 5D A6 00 00 80 01 5B B5 C0 A8 00 0A C0 A8 00 0B 08 00 43 83 00 01 09 D8 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69	9C 44 F2 A1

Figure 69 – PicoScope decoded Ethernet frames

For transmission of the Ethernet frame, the Ethernet controller inserts a 7-byte preamble and a 1-byte Start Frame Delimiter (SFD) before the frame to signal the start of transmission. Often, the preamble and SFD are together referred to as an 8-byte preamble. The combination of 8-byte preamble and Ethernet frame is known as an Ethernet packet.

Preamble	MAC Destination	MAC Source	Ether Type	Payload	CRC
55 55 55 55 55 55 55 D5	FC C2 3D 10 61 6C	FC C2 3D 0F BA 26	08 00	45 00 00 3C 5D 9A 00 00 80 01 5B C1 C0 A8 00 0A C0 A8 00 0B 08 00 43 BF 00 01 09 CC 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69	99 4E B2 A9
55 55 55 55 55 55 55 D5	FC C2 3D 10 61 6C	FC C2 3D 0F BA 26	08 00	45 00 00 3C 5D 9B 00 00 80 01 5B C0 C0 A8 00 0A C0 A8 00 0B 08 00 43 8E 00 01 09 CD 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69	76 55 EB B2
55 55 55 55 55 55 55 D5	FC C2 3D 10 61 6C	FC C2 3D 0F BA 26	08 00	45 00 00 3C 5D 9C 00 00 80 01 5B BF C0 A8 00 0A C0 A8 00 0B 08 00 43 8D 00 01 09 CE 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69	67 77 C7 9B
55 55 55 55 55 55 55 D5	FC C2 3D 10 61 6C	FC C2 3D 0F BA 26	08 00	45 00 00 3C 5D 9D 00 00 80 01 5B BE C0 A8 00 0A C0 A8 00 0B 08 00 43 8C 00 01 09 CF 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69	88 6C 9E 80
55 55 55 55 55 55 55 D5	FC C2 3D 10 61 6C	FC C2 3D 0F BA 26	08 00	45 00 00 3C 5D 9E 00 00 80 01 5B BD C0 A8 00 0A C0 A8 00 0B 08 00 43 8B 00 01 09 D0 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69	A1 D7 C2 AD
55 55 55 55 55 55 55 D5	FC C2 3D 10 61 6C	FC C2 3D 0F BA 26	08 00	45 00 00 3C 5D 9F 00 00 80 01 5B BC C0 A8 00 0A C0 A8 00 0B 08 00 43 8A 00 01 09 D1 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69	4E CC 9B B6
55 55 55 55 55 55 55 D5	FC C2 3D 10 61 6C	FC C2 3D 0F BA 26	08 00	45 00 00 3C 5D A0 00 00 80 01 5B BB C0 A8 00 0A C0 A8 00 0B 08 00 43 89 00 01 09 D2 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69	34 48 19 5E
55 55 55 55 55 55 55 D5	FC C2 3D 10 61 6C	FC C2 3D 0F BA 26	08 00	45 00 00 3C 5D A1 00 00 80 01 5B BA C0 A8 00 0A C0 A8 00 0B 08 00 43 88 00 01 09 D3 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69	DB 53 40 45
55 55 55 55 55 55 55 D5	FC C2 3D 10 61 6C	FC C2 3D 0F BA 26	08 00	45 00 00 3C 5D A2 00 00 80 01 5B B9 C0 A8 00 0A C0 A8 00 0B 08 00 43 87 00 01 09 D4 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69	E7 C7 94 01
55 55 55 55 55 55 55 D5	FC C2 3D 10 61 6C	FC C2 3D 0F BA 26	08 00	45 00 00 3C 5D A3 00 00 80 01 5B B8 C0 A8 00 0A C0 A8 00 0B 08 00 43 86 00 01 09 D5 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69	08 DC CD 1A
55 55 55 55 55 55 55 D5	FC C2 3D 10 61 6C	FC C2 3D 0F BA 26	08 00	45 00 00 3C 5D A4 00 00 80 01 5B B7 C0 A8 00 0A C0 A8 00 0B 08 00 43 85 00 01 09 D6 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69	EB E7 3C 4E
55 55 55 55 55 55 55 D5	FC C2 3D 10 61 6C	FC C2 3D 0F BA 26	08 00	45 00 00 3C 5D A5 00 00 80 01 5B B6 C0 A8 00 0A C0 A8 00 0B 08 00 43 84 00 01 09 D7 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69	07 FC 65 55
55 55 55 55 55 55 55 D5	FC C2 3D 10 61 6C	FC C2 3D 0F BA 26	08 00	45 00 00 3C 5D A6 00 00 80 01 5B B5 C0 A8 00 0A C0 A8 00 0B 08 00 43 83 00 01 09 D8 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69	9C 44 F2 A1

Figure 70 – PicoScope decoded Ethernet packet including preamble

In a point-to-point BroadR-Reach link, there are dedicated MAC addresses from the Master and Slave, and all communication is between the two. This is known as unicast addressing, where Ethernet packets are sent from one to another node.

With Ethernet networks, if messages need to be sent to multiple destinations (known as multicast addressing), a Layer 2 Ethernet switch is used to direct the traffic to the correct destinations. The OPEN Alliance SIG define a set of requirements for such devices for use in automotive applications (Open Alliance Requirements for Ethernet Switch Semiconductors).

4 About Pico Technology

Pico Technology was established in 1991 and soon became a leader in the field of PC oscilloscopes and data loggers. Pico has always been recognized for providing innovative, cost-effective PC-based alternatives to traditional test equipment and data acquisition products, and is now the leader in automotive electronics test and diagnostics. We make high-quality instrumentation affordable. Whether you are an automotive electronics design engineer or a master technician seeking the most powerful test and diagnostic tools available, Pico has the ideal solution for your requirements.

Acknowledgement

Figures 2, 3, 6, 17, 18, 22, 32, 35, 36, 39–48, 50, 55 reproduced with kind permission of Vector Informatik GmbH.

References

[i] *Nicolas Navet, Françoise Simonot-Lion. In-vehicle communication networks - a historical perspective and review*. Richard Zurawski. Industrial Communication Technology Handbook, Second Edition, CRC Press Taylor&Francis, 2013.

[ii] Quoted in G. Leen and D. Hefferman – *Expanding automotive electronic systems* – IEEE Computer, 35(1), January 2002.