CAN XL error detection capabilities

With its higher data-rates and payload sizes, CAN XL is the next step in the evolution of CAN. Besides this, CAN XL also provides improved error detection capabilities.

CAN XL offers data-rates and payload sizes that are many times higher than in Classical CAN and CAN FD [1], [2]. Error detection is a crucial functionality provided by communication protocols. A receiving node has to be able to judge if a frame was received with or without errors. Autonomous driving and other safety relevant applications require that frame errors are detected with a very high probability. The acceptance of an erroneous frame should be practically impossible. This article first introduces the three CAN error types known in literature that might occur in a frame in harsh environments: (1) bit error, (2) bit drop and bit insertion, (3) burst errors. The two main pillars of the CAN error detection mechanism are: (A) the cyclic redundancy code (CRC) check and (B) the format checks. Both pillars are strengthened during the currently ongoing specification of CAN XL, to fit to tomorrow's applications.

We explain how these pillars were improved. Therefor we show the reasons for the chosen CRC concept of having both a header CRC and a frame CRC in a CAN XL frame. Further, we introduce the available format checks in CAN XL. Finally, we show systematically how the CAN XL error detection mechanisms master to detect the three error types. A deep dive into the properties and strengths of the used CRC polynomials is given in [9].

Introduction

CAN XL is currently being specified inside the CiA's (CAN in Automation) CAN XL Special Interest Group. The first specification meeting took place in Nuremberg (Germany) on December 17th 2018. The CiA 610-1 specification document, which focuses on OSI layer 2 (known as CAN XL protocol), was not yet finished at the time of writing this article. Consequently, the final CiA 601-1 specification may show differences compared to the content presented in here. [3] gives an overview about the current CAN XL status. Some of the main features of CAN XL are:

- data field size up to 2 048 byte
- gross bit-rate of 10 Mbit/s and more
- strong error detection capabilities

With this set of features CAN enables the usage of higher layer protocols like IP (Internet Protocol). At the same time, it eases the implementation of safety critical applications with its excellent error detection capabilities and its well-known robustness. Two very essential functions in a communication protocol are the error detection and the error handling. They have a large impact on the reliability of the communication system. The focus of this article is the error detection mechanisms in CAN XL.

This article consist of three parts. Part 1 introduces the CAN XL error detection mechanisms and explains how these were improved compared to CAN FD. In this part, the reasons are given for the chosen CRC concept of having a header CRC and a frame CRC in a CAN XL frame. Part 2 introduces the error types known in literature, along with their properties. Part 3 performs a systematic evaluation to show how the CAN XL error detection mechanisms master to detect all known error types up to a given extent.

CAN XL error detection mechanisms

In CAN communication, all nodes in a network check the validity of each frame, including the transmitter of the current frame. The checks are based on a combination of several protocol mechanisms for error detection. They are described in the following. Figure 1 shows the current version of the CAN XL frame format. The bits used to implement additional or updated error detection mechanisms (compared to CAN FD) are shaded.

Bit monitoring

Bit monitoring means that a node that transmits a bit also monitors the bit values on the CAN network. If the transmitted and received (monitored) bit values differ, the reaction of the node depends on the bit position in the frame. As example, if the transmitting node transmitted a 1 and received a 0 in the data field, it regards this as a bit error. However, if the same happens in the arbitration field, it regards this as arbitration lost.

▷

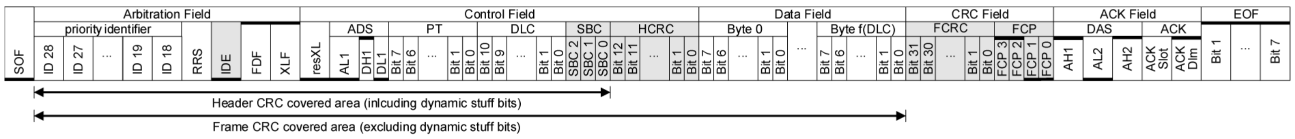


Figure 1: CAN XL frame format (Source: Bosch)

A detailed explanation of the bit monitoring in CAN FD can be found in [10]. If error signaling (via error frames) is enabled in CAN XL, bit monitoring is nearly equal to that in CAN FD. For the case that error signaling is disabled, bit monitoring is not yet fully specified in the current CiA 610-1 draft.

Frame format check

Most parts of a CAN frame (identifier, control, or data bits) are variable or are calculated from the variable bits (CRC sequence), but some bits (delimiters, end of frame) have a fixed format (see figure 1). The bit values of these bits are marked in the figure with a bold line. A receiver detects a form error when it samples a fixed format bit with the wrong value.

A special case is the reserved bit following the XLF bit in CAN XL frames. The reserved bit is expected to be dominant. In current applications, a form error is detected when this bit is sampled as recessive. For future applications, this bit may be used to distinguish between the CAN XL frame format and another – not yet defined – new frame format. When this alternative is selected (by software configuration) and if then this bit is sampled as recessive, the receiver enters a protocol exception state until the network is idle again. This allows the introduction of future new frame formats that are tolerated by existing CAN XL implementations.

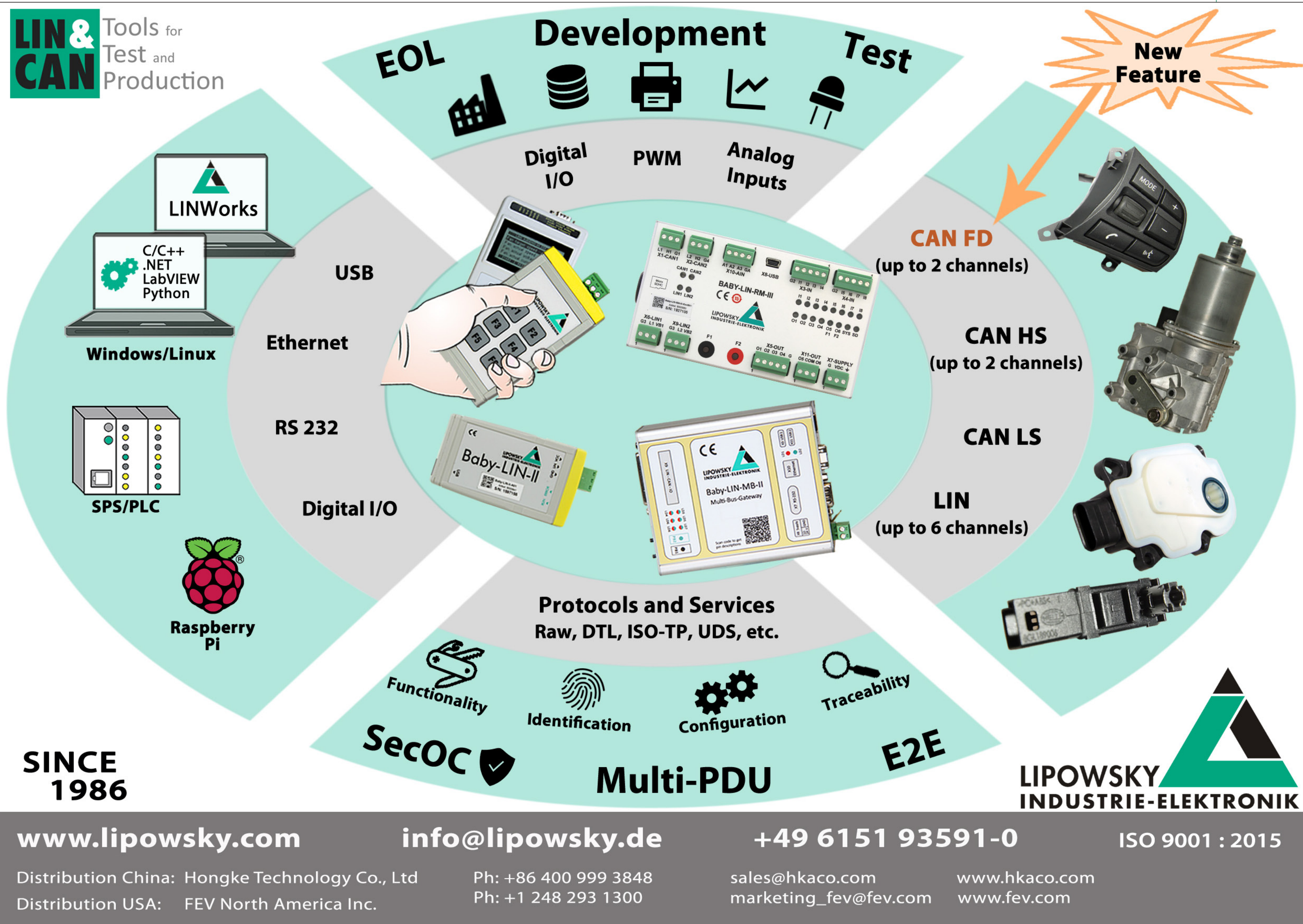
A node transmitting a CAN XL frame sends the FDF and XLF bits as recessive (logical '1'). These bits are part of the arbitration field, which is different compared to CAN FD. This means, if the transmitting node samples one of these bits as dominant, it loses arbitration and becomes a receiver.

In CAN XL, beside the bit-rate, also the mode of the transceiver can be switched. In the error free case, the CAN XL protocol controller signals the mode switch to the transceiver during the bits AL1 and AH1. The signaling of the mode switch to the transceiver, as well as the mode switch of the transceiver may have side effects on the RXD input signal of the protocol controller. Due to this, a CAN XL node does not perform a format check at the fixed format bits (bold lines mark bit value) AL1 and AH1.

Format check pattern (FCP)

The FCP field contains only fixed format bits and is used by a receiver for two purposes. The first purpose is that it provides a synchronization edge before the receiver switches from the data phase to the arbitration phase.

The second purpose is that a receiver can check with help of the FPC field if its frame decoding is aligned with the actual transmitted bit position. Disturbed synchronization edges may lead to so called bit insertions and bit drops in the receiver. A receiver ▷



can detect, with help of the FPC field, a misalignment of 3 bit in both directions.

CRC concept

In general, the transmitter and the receivers of a frame calculate the CRC (cyclic redundancy check) sequence. After reception of the CRC sequence, each receiver performs a CRC check, to judge if it received the frame correctly or not.

For the CRC's error detection capability to succeed with a very high probability the following two requirements have to be fulfilled:

- RQ1: Transmitter and receiver of the frame calculate the CRC sequence based on the equal number bits.
- RQ2: The receiver checks the CRC sequence at the right position inside the transmitted frame.

To fulfill RQ1, the CAN XL frame format uses fixed stuff bits in nearly the whole frame. Dynamic stuff bits are only used in the first bits of the header, to be compatible to CAN FD. A bit insertion or drop error at a dynamic stuff condition changes the number of bits fed into the CRC. As the error just adds or removes a dynamic stuff bit, the format checks described up to now cannot detect that error. With fixed stuff bits, the frame has a defined length in bits and the receiver can feed the exact number of bits into the CRC calculation.

To fulfill RQ2, we need to make sure that a transmission error cannot change easily the position, where the receiver expects the CRC. For example, if the DLC (data length code) is falsified, the receiver checks the CRC at a wrong position. To solve this, CAN XL uses, like Flexray, a header CRC, and a frame CRC. The header CRC safeguards a header of well-known length. If a receiver saw a valid header CRC, it is very likely that the DLC is correct. With the correct DLC, the data field length is also well known.

Scope of the frame CRC

The frame CRC is calculated over the header and the data field (see figure 1), which is similarly done in Flexray. The author in [9] describes in detail, which bits are included and which are excluded from CRC calculation. This “double checking” of the header is done, because on the one side the frame CRC performance is practically not weakened by safeguarding these few additional header bits. On the other side, “double checking” increases the probability to detect transmission errors in the header, which were not detected by the header CRC.

Dynamic stuff bits

If the dynamic stuff bits are not included into a CRC calculation (like in Classical CAN), an undetectable error can be caused by two bit flips, if one bit flip adds and the other removes a dynamic stuff condition. This case is described in [4]. If the dynamic stuff bits are included into the CRC calculation (like in CAN FD), the CRC calculation may be vulnerable to bit insertions and bit drops at dynamic stuff conditions [10]. CAN XL includes the dynamic stuff bits into the header CRC calculation, but excludes them from the frame CRC calculation. This enables detection of both aforementioned error cases.

In [9] the author assesses the performance of the CAN XL CRC polynomials and compares the results with the CRC polynomials used in Flexray and Ethernet. Both CAN XL CRC polynomials guarantee at least a Hamming distance (HD) of 6, up to the largest CAN XL frame length. This means that at least 5 bit errors can be detected. Beside this, both CRCs are able to detect any odd number of bit errors. Regarding burst errors, the header CRC can detect one burst error of up to 13 bit length, and the frame CRC of up to 32 bit length.

Acknowledgement

Transmitters expect to get an active acknowledgement for their frames, which is a dominant bit in the ACK (acknowledgement) slot. When a transmitter does not sample a dominant bit during ACK slot, it regards this as an ACK error. The transmitter considers a frame that does not get an acknowledgement as invalid and retransmits it (if retransmission is not intentionally disabled).

Stuff rule check

The bits of a CAN frame are coded by the method of bit stuffing. CAN uses as line coding Non-Return-to-Zero (NZR) which has no guaranteed edges. The purpose of stuff bits is to ensure that there are enough edges in the bit stream for resynchro-nization of the receivers. Receivers check the stuff rule and detect a stuff error if the stuff bit has not the expected value.

Before the FDF bit, a dynamic stuffing rule is applied. That means, the transmitter inserts, after each sequence of five consecutive equal bits, one bit of inverse value, called a dynamic stuff bit.

In the data phase, starting at DL1 bit up to the last bit of FCRC, a fixed stuffing rule is applied. That means, the transmitter inserts, after S 1 consecutive bits a fixed stuff bit. The fixed stuff bit has the inverse value of its preceding bit. This means every S^{th} bit is a fixed stuff bit. Currently S=15, but this value may be decreased in the final specification, depending on the results of the phase margin calculations.

Dynamic stuff count check

For compatibility reasons with CAN FD, the CAN XL frame header uses dynamic bit stuffing in the header before the FDF bit. To satisfy requirement RQ1 from chapter 2.4, we need to make sure that transmitter and receiver of a frame see the same amount of dynamic stuff bits. CAN FD solved this requirement by adding the field SBC (stuff bit count) which contains the number of dynamic stuff bits in the frame modulo 8.

CAN XL also uses this this solution and has therefore an SBC field in the header of the frame. It is located before the header CRC, because it is used to check the validity of the header. The number of dynamic stuff bits in a CAN XL frame is in the range 0 to 3. Therefore, the SBC field in the CAN XL frame has 3 bits, the first 2 bits contain information on the number of dynamic stuff bits in the arbitration ▷

field and the 3rd bit is a parity bit. The receiver detects a header CRC error if the SBC does not match to the number of received dynamic stuff bits, or if the SBC parity does not match.

Error signaling

CAN XL allows to enable or to disable error signaling. The software can enable and disable error signaling with a configuration bit in the CAN XL implementation. In case the user disables error signaling, the respective CAN XL node does not transmit error frames. In case the user enables error signaling, the error signaling is done with help of error frames, which is identical to the error signaling in CAN FD, which is described in [10]. Error signaling with error frames disturbs the current frame and thereby converts local errors into global errors in order to ensure data consistency in the network.

Improved error detection in CAN XL

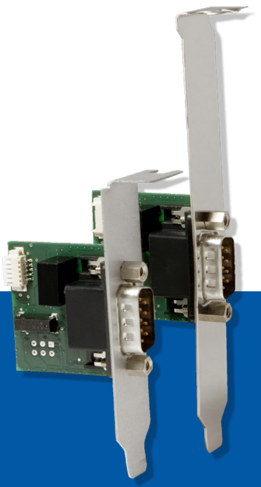
This chapter highlights the five improvements in the CAN XL error detection compared to CAN FD.

1. Header CRC: The newly introduced header CRC allows checking the validity of the header, which includes the DLC value. This allows fulfilling RQ2 and by this strengthens the CRC check.
2. Frame CRC: CAN XL uses a 32-bit frame CRC with a respective CRC generator polynomial to keep the Hamming distance at 6 (HD6) despite the long data field. The frame CRC polynomial was chosen carefully and it outperforms the polynomials of Ethernet and Flexray according to [9].
3. Fixed stuff bits: CAN XL uses fixed stuff bits in the data phase of the frame (short bits). This allows fulfilling RQ1 and by this strengthens the CRC check.
4. Frame CRC safeguards the header: The frame CRC also safeguards the header, which means a “double checking” for the header. To do this effectively, it excludes the dynamic stuff bits. The reason for that is given further in the article and can be summarized as follows: If the CRC calculation does not include dynamic stuff bits, it is vulnerable to a special error case known from Classical CAN [4]. If it includes dynamic stuff bits, it is vulnerable to another error case [10]. The header CRC safeguards the header including dynamic stuff bits and the frame CRC safeguards the header excluding dynamic stuff bits. This enables detection of both special error cases.
5. FCP (format check pattern): The format check pattern is a new field (see chapter 2.3). The receiver checks via FCP if it is aligned to the ▷

CAN and CAN-FD Products
for your requirements

CAN-FD Gateway

Ethernet/CAN Gateway



Embedded USB/CAN Interface

- Economical solutions for series applications
- Optimized for industrial applications
- Solutions for stationary and mobile use
- Software support for bus-analysis, measurement and control

EMS Thomas Wünsche

Sonnenhang 3
D-85304 Ilmmünster
Tel.: +49-8441-49 02 60
Fax: +49-8441-8 18 60
www.ems-wuensche.com

transmitted bit position. A receiver can detect, with help of the FPC, a misalignment of 3 bit in both directions.

Error types

This chapter gives an overview of the existing error types. Details to these error types are described in [10].

Bit error or bit flip means that a CAN node samples a bit with the inverse (flipped) value compared to the transmitted bit value. Figure 2 shows an example for such a bit error at bit 3.

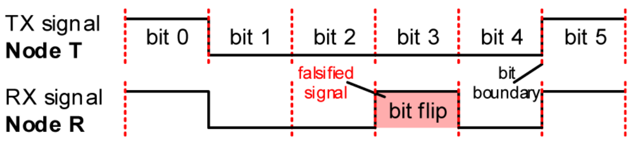


Figure 2: Bit error example (Source: Bosch)

Bit drop or bit insertion means that a receiving node drops a bit from or inserts a bit into the bit sequence. This is caused by a disturbed RXD signal and can occur only in receiving nodes.

In order to cause a bit drop or insertion, the following needs to happen: A disturbance (e.g. EM radiation) influences the CAN physical layer. As consequence, additional or shifted falling edges appear in the RXD signal. The receiving node resynchronizes, based on these faulty edges. This resynchronization may increase the phase error ([6], [2]) between transmitting and receiving node. When the absolute value of the phase error is above a critical level, the receiving node drops a bit from or inserts a bit into the bit sequence.

Case: $f_{RX} < f_{TX}$, i.e. $BitTime_{RX} > BitTime_{TX}$

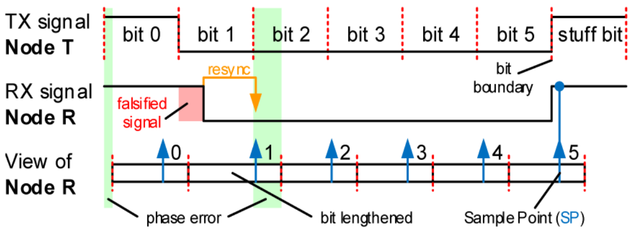


Figure 3: Bit drop example (Source: Bosch)

Figure 3 shows an example for a bit drop. Here a resynchronization on a falsified edge causes the receiver to drop one bit. The receiver samples the transmitted bit sequence "100000i" as "100001" ('i' stands for a dynamic stuff bit).

Important properties of bit drops and bit insertions are [10]:

- They can theoretically happen at any position in the frame. It is not limited to dynamic stuff conditions.
- This error type requires many pre-requirements: e.g. large clock tolerance between sender and receiver, disturbance needs to hit one or more dedicated edges, etc.
- Drop and insertion can practically not happen in the same frame. However, several bit drops or several bit insertions may happen in the same frame.
- Since many factors have to come together, a bit drop or insertion is much more difficult to cause, compared to a bit error. Therefore, one bit drop or insertion should be considered from the likelihood point of view as a "multi bit error".

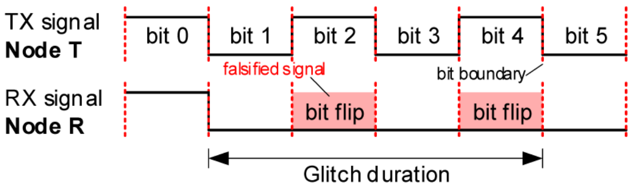


Figure 4: Burst error – all bits forced to one value (Source: Bosch)

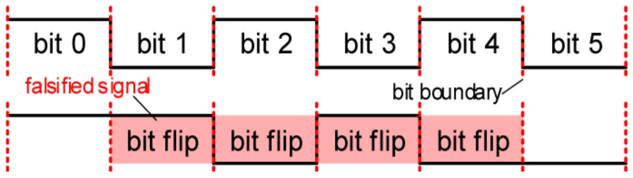


Figure 5: Burst error – due to several bit errors (Source: Bosch)

Several bit errors that are locally close to each other are called a burst error. The burst length (in bit) is the distance from the first to the last bit error. We distinguish here two types of burst errors. Type 1 is where all bits in the burst are forced to the same value, e.g. by a glitch. Figure 4 shows an example. We consider this a realistic type of burst error on the CAN physical layer. The second type of burst error is type 2, where several bits are flipped, but not necessarily all. Figure 5 shows an example.

We assume this type of burst error is very unlikely to be caused by glitches.

However, this type of burst error can be caused by two errors, where the first error leads to a misalignment of the receiver and the second error reverts the misalignment. As long as the receiver is misaligned, it sees all transmitted bits shifted by e.g. 1 bit. This can be achieved by two bit errors, where the one adds a dynamic stuff condition and the other bit error removes a dynamic stuff condition [4]. Consider that the CAN XL frame uses dynamic bit stuffing only at the beginning of the frame. The header CRC can detect this error easily, as it does include dynamic stuff into the CRC calculation – this means from header CRC point of view, there is no misalignment and consequently the two bit errors cause no burst error.

Another way to cause such a temporal misalignment of the receiver is a bit drop and a bit insertion in the same frame [10], which could theoretically occur also in the CAN XL data field [10]. However, one bit insertion and one bit drop, both in the same frame, are assumed practically impossible to occur [10].

Table 1 gives an overview to the error types known in CAN. The table also shows how an external cause (like a glitch on the bus lines) or an internal cause (like wrong system design) can create these errors. Further, it shows which error detection mechanism can detect the error.

▷

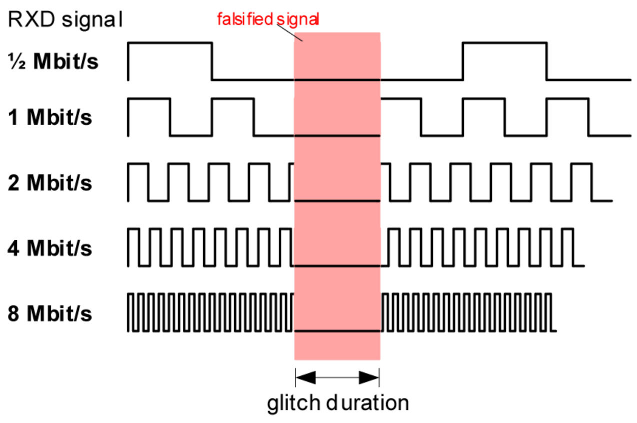


Figure 6: Errors caused by a 2 us glitch at different bit rates (Source: Bosch)

Evaluation: burst error detection

We introduced two burst error types. As described, burst errors of type 2 (several bits are flipped, but no necessarily all) can be caused by several circumstances. Based on the arguments we mentioned, we conclude that this type 2 burst error is practically extremely unlikely to occur and therefore can be neglected.

Burst error type 1 (all bits in the burst are forced to the same value, e.g. by a glitch) is considered as very realistic. The remainder of this chapter evaluates if and how the error detection mechanisms can detect such a burst error.

Since CAN XL can be used at different bit-rates, the same glitch on the bus lines can cause very different error scenarios for a receiver. Figure 6 visualizes the impact of a 2 us glitch. At 500 kbit/s this leads to one bit error, while at 2 Mbit/s it leads to a burst error of 4 bit and at 8 Mbit/s it leads already to a burst error of 16 bit.

The diagram in figure 7 shows the relation between glitch length and burst length in bits. Two glitch lengths are shown: 2 us and 5 us. These glitches translate to a burst duration of the same value. The actual glitch length that may occur on a specific CAN network depends on the environment around the CAN network. The authors in [7] observed in a very aggressive environment an average burst duration of 5 us. For example, at 5 Mbit/s a 5 us glitch causes a burst length of 25 bit.

▷

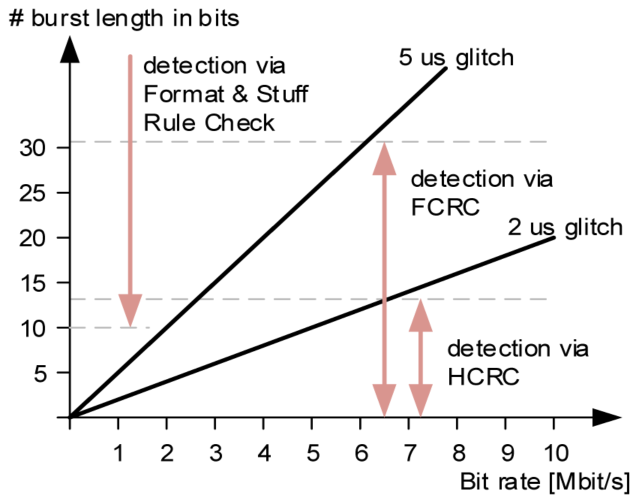


Figure 7 : Error detection mechanisms versus burst errors (Source: Bosch)

Table 1: Overview of error types in CAN

Error type	External cause: EMI	Internal cause	How to detect the error?	Literature
Bit error (bit flip)	Glitch length: ≈ one bit length	Bit asymmetry is too large	o CRC check o Format checks (limited)	Different bit error rates (BER) mentioned in: [4], [5], [7], [8]
Bit insertion or bit drop	Glitch length: < one bit length	CAN clock tolerance is too large	o Format checks, FCP o Dynamic stuff bit count (SBC)	First described in [10]
Burst error	Glitch length: > one bit length	temporary misalignment of receiver to transmitted bit stream	o Format checks o CRC check (up to a limited burst length	[7] mentions an average burst error length of 5 us; [8] is less explicit

Figure 7 also shows the main CAN XL error detection mechanisms that are capable detecting burst errors.

- *Stuff rule check:* The focus is here on the data phase where each S^th bit is a fixed stuff bit. Figure 7 assumes S=10. The arrow in the figure shows, that this check can detect any burst error with a length larger than S bit. Shorter burst errors may also be detected, but only if they hit a stuff bit. With S=15 the effectiveness decreases slightly for short burst lengths.
- *Frame CRC:* The frame can detect one burst error with a length of up to 32 bit.
- *Header CRC:* The header can detect one burst error with a length of up to 13 bit.

We conclude that both, the header and frame CRC, can detect one short burst error and the stuff rule check can detect long burst errors. In sum, these mechanisms can detect all burst errors.

Evaluation: detection of bit errors and bit drops/insertions

This chapter focuses on the two remaining error types: “bit errors” and “bit drops/ insertions”. It evaluates systematically whether they can be detected by CAN XL. The evaluation is limited to 5 bit errors (corresponds HD6) and 2 bit insertions/drops (corresponds to an equivalent of roughly >4 bit errors).

To simplify the description, the CAN XL frame is virtually partitioned into four parts. The evaluation in table 2 is partitioned accordingly. In each part, both error types are listed. For each error type, the relevant number of occurrences of this error type are listed. Additionally, special error cases generated by these two error types at dynamic stuff bits are also listed. Consequently, each row of the table corresponds to one error case. For each error case, the table contains information about the misalignment of the receiver and the way in which the receiver detects the error. ▷

References

[1] F. Hartwich, „CAN with Flexible Data-Rate,“ in Proceedings of the 13th international CAN Conference, Hambach Castle, Germany, 2012.
[2] ISO 011898-1:2015, Road vehicles - Controller area network (CAN) - Part 1: Data link layer and physical signaling, 2015.
[3] F. Hartwich, “Introducing CAN XL into CAN Networks” in Proceedings of the 17th international CAN Conference, Baden Baden, Germany, 2020.
[4] J. Unruh, H. J. Mathony und K. H. Kaiser, „Error Detection Analysis of Automotive Communication Protocols,“ in SAE Int. Congress, No. 900699, Detroit, 1990.
[5] J. Charzinski, “Performance of the Error Detection Mechanisms in CAN,” in Proceedings of the 1st International CAN Conference, 1994.
[6] A. Mutter, „Robustness of a CAN FD Bus System - About Oscillator Tolerance and Edge Deviations,“ in Proceedings of the 14th international CAN Conference, Paris, France, 2013.
[7] J. Ferreira, et al., “An Experiment to Assess Bit Error Rate in CAN”, in Proceedings of 3rd International Workshop of Real-Time Networks, 2004
[8] N. Navet and Y.-Q. Song, “Performance and Fault tolerance of Real-Time Applications Distributed over CAN”, in Proceedings of the International CAN Conference, 1997.
[9] C. Senger, “CRC Error Detection for CAN XL” in Proceedings of the 17th international CAN Conference, Baden Baden, Germany, 2020.
[10] A. Mutter and F. Hartwich, “Advantages of CAN FD error detection mechanisms compared to classical CAN,” in Proceed-ings of The international CAN Conference, Vienna, Austria, 2015.

Table 2: Systematic overview of error cases

Case	**Receiver misalignment**	**Error detected mainly by**
Frame part 1 [SOF to IDE]		
Bit drop or insertion @ dynamic stuff condition		
1 bit drop or insertion	no	SBC (dynamic stuff bit count changes)
2 bit drops or insertions	no	SBC (dynamic stuff bit count changes)
1 bit drop + 1 bit insertion	1 bit temporary for CRC	Practically not possible; header CRC
Bit drop or insertion @ no dynamic stuff condition		
1 bit drops or insertion	1 bit	Format check: IDE = '1' or FDF = '0'
2 bit drops or insertions	2 bit	Format check: IDE = '1' or XLF = '0'
1 bit drop + 1 bit insertion	1 bit temporary	Practically not possible; header CRC
Bit error @ dynamic stuff condition ➔ adds or removes stuff condition		
1 bit error (add/remove)	1 bit	Format check: FDF = '0' or IDE = '1'
2 bit errors (add/remove)	2 bit	Format check: XLF = '0' or IDE = '1'
3 bit errors (add/remove)	3 bit	AL1 = '1' ➔ transceiver will be not switched; or format check: FDF = '0'
1 bit error (add) + 1 bit error (remove)	not for CRC	Header CRC
Bit error @ no dynamic stuff condition		
1 to 5 bit errors	no	Header CRC
Frame part 2 [FDF to AL1]		
Bit drop or insertion	yes	Format Check
Bit error	no	Format Check
Frame part 3 [DH1 to HCRC]		
Bit drop or Insertion		
1 bit drop or insertion	1 bit	If DLC wrong ➔ all together, else FCP
2 bit drops or insertions	2 bit	If DLC wrong ➔ all together, else FCP
1 bit drop + 1 bit insertion	1 bit temporary	Practically not possible; header CRC
Bit error		
1 to 5 bit errors	no	Header CRC
Frame part 4 [Data field and CRC field]		
Bit drop or insertion		
1 bit drop or insertion	1 bit	FCP
2 bit drops or insertions	2 bit	FCP
3 bit drops or insertions	3 bit	FCP
1 bit drop + 1 bit insertion	1 bit temporary	Practically not possible
Bit error		
1 to 5 bit errors	no	Frame CRC

Summary and conclusion

CAN XL has five major improvements regarding error detection, compared to CAN FD. These are (1) header CRC, (2) 32 bit frame CRC, (3) fixed stuff bits in the data phase, (4) frame CRC additionally safeguards header, (5) format check pattern.

Three major error types are known in CAN: (1) Bit error, (2) bit drops/insertions, and (3) burst errors. These error types are introduced in detail.

The article shows how the error detection mechanisms can detect a burst error, where all bits in the burst are forced to the same value, independent of its length. Further, it shows systematically how bit errors and bit drops/insertions can be detected up to a given extent. We conclude that the error detection mechanisms in CAN XL can detect all known error types to a sufficient extent. This work can serve as basis for a review of the CAN XL error detection capabilities, which is planned by the SIG CAN XL. ◀

Author

Dr. Arthur Mutter
Robert Bosch
info@de.bosch.com
www.bosch.com

CRC error detection for CAN XL

CRC generator polynomials for detection of transmission errors in headers and frames of the upcoming CAN XL standard are proposed. Properties, which are chosen to provide error detection performance (compared to competing standards) in the CAN XL scenario, are described.

These properties include achieving Hamming distance 6 for the full range of possible message lengths. At the beginning of the article, a self-contained recap of CRC codes is given.

A new version of the CAN protocol is currently under development: CAN XL. With net data rates up to 10 Mbit/s and beyond, it is designed to bridge the gap between CAN FD and 100Base-T1 Ethernet [1]. Among the design goals for CAN XL are full interoperability with CAN FD as well as large payload length (up to 2 048 byte) in order to enable the use of higher layer protocols such as IP (Internet Protocol) and even encapsulation of complete Ethernet frames [2].

As in any communications system, data transmission in CAN XL is not perfect and transmission errors are inevitable. That is, a transmitted logical zero is detected at the receiver as a logical one or vice versa — a so-called bit error or bit flip. Due to certain physical perturbances in an actual system, bit errors tend to occur in temporally confined groups: so-called burst errors.

Based on elaborate mechanisms that exploit the CAN FD/CAN XL frame structure, certain transmission errors can be detected [3], [4] and corresponding measures can be taken. Frame structure-based error detection alone is not able to provide the required state of the art error detection performance for today's applications, namely probability of undetected bit error below 10^{-20} and guarantee to detect burst errors of a certain length. Thus, in order to provide the required error detection performance, CRC (cyclic redundancy check) codes are employed (Note: that the term "cyclic" at this point is misleading, as many CRC codes used these days do not actually fulfill the definition of a cyclic code (cf. textbooks on error control coding such as [5]). Today, this naming is mainly used for historical reasons).

Competing standards such as Flexray and Ethernet also use CRC codes for error detection and it is our goal to provide at least the same or better error detection performance for CAN XL. This can be accomplished by choosing particular CRC codes, which is the main contribution of this article.

Choosing a particular CRC code is based on certain performance criteria such as the probability of undetected error and the maximal length of a burst error that can be detected with certainty. These in turn depend on the messages that need to be protected and thus on the CAN XL frame structure. The choice is particularly challenging in cases where the messages have variable lengths. For that reason, it was decided early on in the design process of CAN XL to protect the comparatively short and fixed-length header by a so-called header CRC and the whole frame (whose length may vary from several to more than 2 000 byte) by a separate CRC, the so-called frame CRC.

CRC codes

We restrict ourselves to codes over the binary field $\mathbb{F}_2$, i.e., codes over the set {0,1} with operators + (XOR) and · (AND). We denote the set of polynomials of indeterminate x over as $\mathbb{F}_2[x]$. For some $p(x) = \sum_{i=0}^{\infty} p_i x^i$ from $\mathbb{F}_2[x]$ we denote the largest i where $p_i \neq 0$ as $\deg[p(x)]$, the degree of $p(x)$.

In general, the purpose of codes is to cope with transmission errors. The main idea is to add redundancy to a message and transmit the resulting codeword. At the receiver, the redundancy can then be used to recover the transmitted codeword, even if it got corrupted during transmission. This is called error correction. A much simpler task is to use the redundancy in order to determine whether the transmission was error-free or not. This is called error detection.

The message could, for example, be a polynomial $m(x)$ of degree at most $k-1$ (having at most k nonzero coefficients) from $\mathbb{F}_2[x]$. Such a message of message length k can be augmented by M redundant coefficients that are calculated as a function of the message. The process of augmenting message by redundancy is called encoding, M is called the CRC length, $n = k + M$ the code length. The result of encoding is referred to as a codeword. Encoding is called systematic if in any codeword, message and redundancy can be clearly separated (such as in the codeword

$$c(x) = \sum_{i=0}^{k+M-1} c_i x^i = \underbrace{\sum_{i=0}^{M-1} r_i x^i}_{=r(x)} + x^M \underbrace{\sum_{i=0}^{k-1} m_i x^i}_{=m(x)}$$

consisting of message $m(x)$ in the most significant coefficients and redundancy $r(x)$ in the least significant coefficients). Systematic encoders are preferred in practice due to their obvious implementation advantages.

One way of encoding messages $m(x)$, $\deg[m(x)] < k$, into codewords $c(x)$, $\deg[c(x)] < k + M - 1$, is to multiply them with a fixed generator polynomial

$$g(x) = \sum_{i=0}^{M} g_i x^i$$

of degree M from $\mathbb{F}_2[x]$.

$$\mathcal{C}_k = \{m(x)g(x) \ : \ m(x) \in \mathbb{F}_2[x], \deg[m(x)] < k\} \qquad (1)$$

▷

The set of all possible codewords obtainable in this way is called the CRC code $\mathcal{C}_k$, where we maintain the message length k as an index for purposes. This canonical way of encoding (multiplication of messages with the generator polynomial) is not systematic, message and redundancy are intertwined in the resulting codewords and cannot be clearly separated. Due to its definition in (1) one could also refer to $\mathcal{C}_k$ as a polynomial code.

Systematic encoding can be achieved as follows. Instead of multiplying messages with the generator polynomial, the mapping

$$c(x) = \left(\underbrace{m(x)\mathrm{mod}(g(x))}_{=r(x)}\right) + x^M m(x)$$

is performed. Using this form of encoding, the redundancy is the polynomial remain-der of the division $m(x)/g(x)$, i.e., the remainder of polynomial long division (over $\mathbb{F}_2$) applied to message and generator polynomial. It is easy to see that the codewords obtained this way can be written as $c(x) = m'(x)g(x)$, $\deg[m'(x)] < k$, and thus $c(x) \in \mathcal{C}_k$. That is, systematic encoding leads to the same code $\mathcal{C}_k$ as canonical encoding, only the mapping from messages to codewords is different.

The effect of systematic encoding as presented before can be described in words: codewords are polynomials of degree at most $k+M-1$, where the message is shifted into the k most significant coefficients $c_M, \ldots, c_{k+M-1}$ and the redundancy is written into the M least significant coefficients $c_0, \ldots, c_{M-1}$.
The main reason for the popularity of polynomial codes as described above is the fact that the polynomial remainder of $m(x)/g(x)$ can be calculated using a simple linear feedback shift register. In general, the register in Figure 1 calculates the polynomial remainder of $a(x)/(x^\mu + b(x))$, $\deg[b(x] < \mu$, and stores it (after k clock cycles) in the memory elements $\rho_0, \rho_1, \rho_2, \ldots, \rho_{\mu-1}$.

It is clear that the register can be used to calculate $r(x)$ as in (2) by setting $a(x) = m(x)$ $(\kappa = k)$ and $x^\mu + b(x) = g(x)$. Note that $m(x)$ is fed into the register starting with its most significant coefficient m_{k-1} and that its memory elements must be reset to some fixed binary vector (called the initialization vector) beforehand. After m_0 is fed into the register it holds $r(x) = \sum_{i=0}^{M-1} \rho_i x^i$.

Besides calculating $r(x)$ as required for systematic encoding, the same register can also be used to determine whether a given polynomial $v(x)$, $\deg[v(x)] \leq k+M-1$, is a codeword. In case it is a codeword, it has to be a polynomial multiple of the generator polynomial $g(x)$ as stated in (1). But this implies that $g(x)$ divides $v(x)$ and thus $\rho_0 = \rho_1 = \rho_2 = \cdots \rho_{\mu-1} = 0$ has to hold if the register is fed with $a(x) = v(x)$ $(\kappa = k+M)$ and $x^\mu + b(x) = g(x)$ $(\mu = M)$. Otherwise (if at least one out of the p_i is nonzero after $k+M$ clock cycles), $v(x)$ cannot be a codeword. It is important to note that the memory elements must be reset to the same initialization vector as used for encoding in the previous paragraph before the $v_{M+k-1}, \ldots, v_0$ are fed into the register. We stress that in case $v(x)$ is indeed a codeword, we have $v_{M+k-1} = m_{k-1}$, …, $v_M = m_0$, $v_{M-1} = r_{M-1}$, …, $v_0 = r_0$, where m_i ▷

Ixxat®
BY HMS NETWORKS

NEW!
MQTT support for cloud connection and easy-&-fast programming using „Action Rules"

CAN@net NT
CAN-to-Ethernet Gateway/Bridge
with 4 x CAN and 2 x CAN FD

CAN and CAN FD

Repeater, Bridges and Gateways

- Save costs due to simple wiring
- Increase your system reliability and protect devices by galvanic isolation (up to 4 kV)
- Filter/conversion functionality as well as coupling of CAN and CAN FD
- Bridging of large distances and easy system access via Bluetooth or Ethernet
- **NEW:** Cloud connection via MQTT and easy execution of tasks using “Action Rules“ – no programming!

All you need for CAN

Discover more:
www.all4CAN.com

CANblue II
Bluetooth PC Interface, Bridge, Gateway

CANbridge NT
(up to 4 x CAN / 2 x CAN-FD)

CAN-CR120/HV
CAN / CAN FD Repeater
(3 kV galv. iso.)

CAN-CR300
CAN / CAN FD Repeater
(4 channels)

CAN-CR 110/FO
CAN / CAN FD
to fiber optic

HMS Industrial Networks GmbH
Emmy-Noether-Str. 17 · 76131 Karlsruhe
+49 721 989777-000 · info@hms-networks.de
www.anybus.com · www.ewon.biz · www.intesis.com · www.ixxat.com

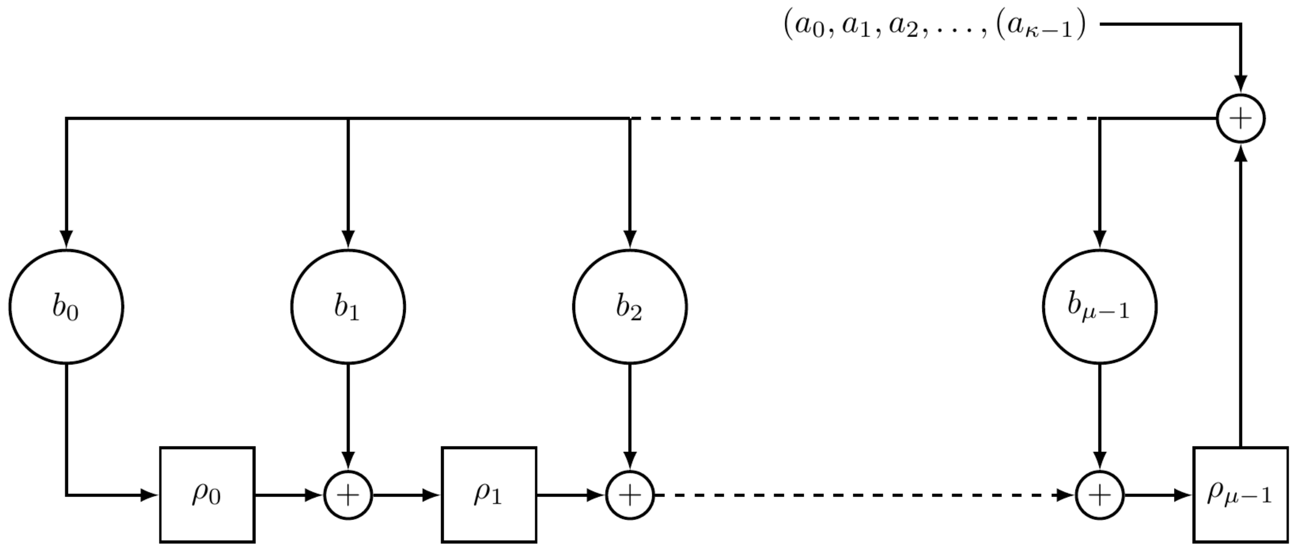


Figure 1: Linear feedback shift register for use with polynomial codes. All operators are from F_2, i.e., + denotes an XOR operation, b_i surrounded by a circle denotes an AND operation with b_i as one of the operands (Source: Dr. Christian Senger)

and r_i are the coefficients of message $m(x)$ and redundancy r(x), respectively.

In practice, CRC codes are used as follows: First, generator polynomial $g(x)$ and initialization vector are chosen as system parameters and made known to both transmitter and receiver. Each message $m(x)$ is encoded into a codeword $c(x)$ at the transmitter (systematically as in (2) using the linear feedback shift register from figure 1 in order to calculate the redundancy $r(x)$.

The codeword is transmitted over a communications channel where it may be exposed to bit and burst errors. As a result, the received word at the receiver $v(x)$ may not be identical to $c(x)$. The receiver now uses the register (configured with $g(x)$ and the initialization vector) in order to check whether $v(x)$ is a codeword or not. If it is not a codeword then a transmission error is detected and appropriate measures are taken.

If it actually is a codeword then two cases are possible: Either $v(x)$ coincides with $c(x)$, which means errorfree transmission. Otherwise, if it does not coincide with $c(x)$, the channel transformed $c(x)$ into another codeword from $\mathcal{C}_k$. The receiver has no way of distinguishing between the two cases and thus the latter case corresponds to an undetected error. Since the probability of having undetected errors depends on the actual generator polynomial, choosing generator polynomials that result in low undetected error rate is of utmost importance.

Properties of CRC codes

As we will see in the following, the undetected error rate is mainly determined by a code parameter referred to as minimum Hamming distance or, in the context of CRC codes, simply Hamming distance HD_k. It states the minimum number of coefficients, in which any two codewords $c(x), c'(x) \in \mathcal{C}_k$, $c(x) \neq c'(x)$ differ.

In our setting (since the considered polynomial codes are linear), HD_k is defined by the minimum Hamming weight of the codewords from $\mathcal{C}_k$, i.e.,

$$\mathrm{HD}_k = \min_{c(x)\in\mathcal{C}_k} \{\mathrm{wt}[c(x)]\}.$$

The Hamming weight $\mathrm{wt}[\cdot]$ of a polynomial $p(x) \in \mathbb{F}_2[x]$ is in turn defined as the number of its nonzero coefficients, i.e., $\mathrm{wt}[p(x)] = \left|\{i \in \{0,\ldots,k+M-1\} \,:\, p_i \neq 0\}\right|$.

Since the CRC length M is fixed (by the choice of generator polynomial) the code rate $R = {}^{k}/_{(k+M)}$ approaches one as the message length k grows. Consequently, larger k results in a denser packing of the linear code space and thus (in general) in smaller Hamming distance. Since CAN XL (both header and frame) generates a range of message lengths we have to carry k along as an index for both $\mathcal{C}_k$ and HD_k. Transmission errors can be represented by nontrivial error polynomials

$$e(x) = \sum_{i=0}^{k+M-1} e_i x^i$$

with $\deg[e(x)] \leq \deg[c(x)]$ that distort transmitted codewords $c(x) \in \mathcal{C}_k$ into received polynomials

$$v(x) = c(x) + e(x) = \sum_{i=0}^{k+M-1} v_i x^i.$$

In order to cause an undetected error, the channel has to cause at least HD_k nonzero coefficients in $e(x)$, i.e., it has to cause $\mathrm{wt}[e(x)] \geq \mathrm{HD}_k$ bit errors. It is not possible to take the transmitted $c(x)$ to a different codeword with a smaller number of bit errors and thus transmission errors with $\mathrm{wt}[e(x)] < \mathrm{HD}_k$ bit errors can always be detected. Consequently, larger Hamming distances result in smaller undetected error rates, which is why we always aim for large Hamming distance in the rest of the paper.

Undetected error rate

The undetected error rate states the probability that transmission of a codeword $c(x) \in \mathcal{C}_k$ results in received word $v(x) \in \mathcal{C}_k$ and $v(x) \neq c(x)$. It can be calculated explicitly under the assumption (suggested in [6]) that the transmission channel is a binary symmetric channel (BSC) that flips each transmitted bit with cross-over probability p. Besides this assumption, the weight distribution $(A_k[0], A_k[\mathrm{HD}_k], \ldots, A_k[n])$ of $\mathcal{C}_k$ is required. Its ▷

components $A_k[w], w \in \{0, \mathsf{HD}_k, \dots, n\}$, give the number of codewords in $\mathcal{C}_k$ having Hamming weight w. Despite being computationally not trivial, it is still possible to calculate weight distributions for moderately sized polynomial codes.

Under the given assumptions, the undetected error rate of a code can be calculated as

$$P_{\mathrm{ue},k} = \sum_{w=\mathsf{HD}_k}^{k+M} A_k[w] p^w (1-p)^{k+M-w}.$$

Since we assume a BSC it is $(1-p)/p$ times less likely to have $\mathrm{wt}[e(x)] = t+1$ compared to having $\mathrm{wt}[e(x)] = t$. This fraction goes to infinity as $p \longrightarrow 0$ and thus $P_{\mathrm{ue},k}$ is dominated by its first term, that is, $A_k[\mathsf{HD}_k] p^{\mathsf{HD}_k} (1-p)^{k+M-\mathsf{HD}_k}$.

As a consequence, our criterion for picking generator polynomials for the header CRC in Section V from multiple candidate polynomials with the same HD_k is going to be small $A_k[\mathsf{HD}_k]$ for the full range of relevant message lengths k.

Guaranteed-detectable errors

Some transmission errors can be detected with guarantee. Take for example a code $\mathcal{C}_k$ with $\mathsf{HD}_k = 6$. Any two distinct codewords $c(x), c'(x) \in \mathcal{C}_k$ differ in at least 6 coefficients. That is, taking $c(x)$ and flipping at most 5 arbitrary coefficients cannot result in some $c'(x) \in \mathcal{C}_k$. Or, in other words:

$$\begin{aligned} &\mathrm{wt}[e(x)] < \mathsf{HD}_k \wedge c(x) \in \mathcal{C}_k \\ &\Longrightarrow c(x) + e(x) = v(x) \notin \mathcal{C}_k. \end{aligned}$$

This shows that, in any case, up to $\mathsf{HD}_k - 1$ bit errors can be detected with guarantee. Many transmission errors with much larger Hamming weight can be detected as well but this can in general not be guaranteed. An exception (where there actually are guarantees) are burst errors of a certain maximal length as we will see in the following.

For any transmission error $e(x) \neq 0$, $\deg[e(x)] < k+M$, we define the following two notions: The trailing coefficient $e(x) \neq 0$, $\deg[e(x)] < k+M$ and the leading coefficient $\ell[e(x)] = \deg[e(x)] < k+M$.

The value $\ell[e(x)] - t[e(x)] + 1 \in \{1, \dots, k+M\}$ is referred to as the burst-length of the error. In general, detecting errors is easier if their Hamming weight and their burst-length are small.

If any $e(x) \neq 0$ is a codeword then (by definition) it has to be a polynomial multiple of $g(x)$. That is, $e(x) = m(x)g(x)$ for some $m(x) \in \mathbb{F}_2[x]$. But this implies

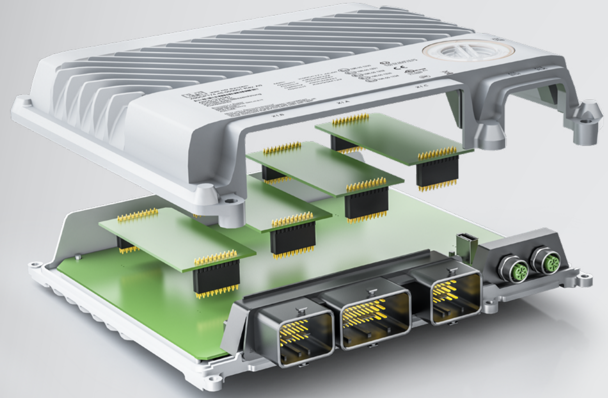
$$\begin{aligned} \ell[e(x)] &= \overbrace{\ell[g(x)]}^{\deg[g(x)]} + \ell[m(x)] \\ t[e(x)] &= t[g(x)] + t[m(x)] \end{aligned}$$

and thus

$$\begin{aligned} &\ell[e(x] - t[e(x)] + 1 \\ &= \deg[g(x)] + \overbrace{\ell[m(x)] - t[m(x)]}^{\geq 0} - t[g(x)] + 1 \\ &\qquad > \deg[g(x)] - t[g(x)] = M - t[g(x)]. \end{aligned}$$ ▷

Completely integrated automation for mobile machinery - **X90**

Complete portfolio:
www.br-automation.com/mobile-automation



- Easy handling
- Integrated safety
- Faster development

OPC UA open SAFETY mapp TECHNOLOGY

PERFECTION IN AUTOMATION
A MEMBER OF THE ABB GROUP

B&R

As a result, $e(x)$ cannot be a codeword if $\ell[e(x)-t[e(x)]+1 \leq M - t[g(x)]$ and consequently any transmission error can be detected as long as its burst-length is at most $M - t[g(x)]$.

In order to guarantee detection of preferably long burst errors it is instrumental to choose $g(x)$ with $g_0 = 1$ resulting in $t[g(x)] = 0$, which (with the above) guarantees detection of error bursts up to burst-length M.

Generator polynomials from $\mathbb{F}_2[x]$ having the special form $g(x) = (x+1)a(x)$, where $\deg[a(x)] = \deg[g(x)] - 1$, impose the factor $x+1$ on any codeword $c(x) = m(x)g(x)$, i.e., any codeword can be written as $c(x) = (x+1)b(x)$ with $\deg[b(x)] = \deg[c(x)] - 1$. For any such codeword holds $c(1) = 0$ because in $\mathbb{F}_2$ we have 1 + 1 = 0 (XOR operation). But the evaluation at 1 of any polynomial from $\mathbb{F}_2[x]$ results in 1 if its Hamming weight is odd and in 0 if the Hamming weight is even. This lets us conclude that all codewords from the resulting CRC code have even Hamming weight and consequently a received word of odd Hamming can never be a codeword. In other words: if the generator polynomial $g(x)$ has x + 1 as a factor then all transmission errors $e(x)$ affected by an odd number of bit flips are detected with guarantee.

In summary we can list types of non-trivial transmission errors $e(x) \neq 0$, $\deg[e(x)] < k + M$, that are guaranteed-detectable by CRC codes with certain generator polynomials g(x):

1. In any case: $e(x)$ is guaranteed-detectable as long as $\mathrm{wt}[e(x)] \leq \mathrm{HD}_k - 1$.
2. If $g_0 = 1$: $e(x)$ is guaranteed-detectable as long as it contains a single burst error of burst-length at most M.
3. If $g(x)$ has $x+1$ as a factor: $e(x)$ is guaranteed-detectable as long as $\mathrm{wt}[e(x)]$ is odd.

CAN XL frame structure

CAN XL frames consist of a multitude of fields, out of which some are protected by the header CRC (HCRC), some by the frame CRC (FCRC), and some by both. Table I provides an overview. Here, being protected by a CRC means being included in its message polynomials.

It can be seen in the table that besides the obvious data field also the header fields ID, RRS, PT, DLC, SBC as well as the HCRC redundancy are part of the FCRC messages. This approach, which provides extra protection to the header fields at negligible cost, was decided as a result of discussions with Dr. Arthur Mutter and Florian Hartwich, Robert Bosch. The same approach is taken in the Flexray standard.

Some of the fields are affected by dynamic bit stuffing after each run of five identical bits, namely SOF, ID, RRS, and IDE. The number of dynamic stuff bits is stored in the SBC field. Note that the last dynamic stuff bit may be added after the IDE field. Fixed stuff bits as well as any fixed-value fields are not included in any CRC calculation.

We emphasize that the dynamic stuff bits are protected by the HCRC but not by the FCRC. The explanation is given in the following.

Excluding dynamic stuff bits from CRC messages (as in Classical CAN) can result in an undetectable error caused by two bit flips if one bit flip adds and the other removes a dynamic stuff condition. This case is described in [7]. However, including dynamic stuff bits (as in CAN FD) makes the CRC code vulnerable to bit insertions and bit drops at dynamic stuff conditions as described in [3].

Therefore, it was decided to include the dynamic stuff bits in the HCRC calculation but to exclude them from the FCRC calculation. This enables detection of both aforementioned types of errors.

Header CRC (HCRC)

As mentioned before, a dedicated header CRC is proposed for CAN XL. The same approach is followed by the Flexray standard, where fixed-length headers are protected by an M = 11 bit CRC code that achieves Hamming distance 6. The Ethernet standard does not stipulate a dedicated header CRC.

The achievable undetected error rates of codes with Hamming distance 6 are well below 10^{-20}. For relevant CAN XL scenarios with data rates around 10 Mbit/s this means that less than one undetected header error per year per billion devices can be expected. Thus, going to larger Hamming distance seems to be over the top. Consequently, the proposed generator polynomial for protecting the CAN XL header provides Hamming distance 6.

Due to dynamic bit stuffing, HCRC message polynomials consist of at least 34 and at most 37 coefficients (Table I). Thus, any HCRC candidate has to fulfill $\mathrm{HD}_{34}, \mathrm{HD}_{35}, \mathrm{HD}_{36}, \mathrm{HD}_{37} \geq 6$.

It can be verified by exhaustive search that the smallest CRC length M for which candidates fulfilling the HD requirement can be found is M. Out of all the candidates, we propose the generator polynomial

$$\begin{aligned} g_{\mathrm{HCRC}}(x) &= x^{13}+x^{12}+x^{11}+x^{8}+x^{7}+x^{6}+x^{5}+x^{2}+x+1 \\ &= (x^{12}+x^{10}+x^{9}+x^{8}+x^{6}+x^{4}+x^{3}+x^{2}+1) \\ &\quad \cdot (x+1) \end{aligned}$$

for use in the HCRC. Our arguments are described in the following. Note that when we talk about header in this context we mean HCRC message as given by table 1 plus HCRC parity. First, we have (as for any CRC code with Hamming distance 6):

1. Any erroneous header that is affected by no more than 5 bit errors can be detected with guarantee.

Additionally, due to our special choice of $g_{\mathrm{HCRC}}(x)$ (least significant coefficient $g_0 = 1$ and factor $x + 1$) we have:

2. Any erroneous header that is affected a single burst error of burst-length no more than 13 can be detected with guarantee. In other words, any received header where the bit flips are constrained to a set of 13 consecutive bits is guaranteed-detectable.
3. Any erroneous header that is affected by an odd number of bit errors can be detected with guarantee.
4. The undetected error rates $P_{\mathrm{ue},34}$, $P_{\mathrm{ue},35}$, $P_{\mathrm{ue},36}$,$P_{\mathrm{ue},37}$ are minimal among all possible candidate generator polynomials with properties (1) to (3).

We stress that many error patterns that do not fall into cases (1) to (3) can also be detected, but without guarantee.

For the convenience of the reader we state $g_{\mathrm{HCRC}}(x)$ (x) in three commonly used notations: ▷

M	ISO	Normal	Koopman
13	0x39E7	0x19E7	0x1CF3

In order to cope with the aforementioned vulnerability to bit insertions and bit drops related to dynamic bit stuffing the linear feedback shift register must never assume the all-zero state in the first $12 + s$ clock cycles when it is fed with the message (cf. [3]). Here, $s \in \{0, \ldots, 3\}$ denotes the number of dynamic stuff bits that occur in, in between or after the SOF, ID, RRS, and IDE fields. Note that 12 bits protected by the HCRC (ID and RRS fields) are affected by dynamic bit stuffing. The all-zero state can be avoided by choosing a particular initialization vector such as the proposed initialization vector $(\rho_0, \rho_1, \ldots, \rho_{12}) = (1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)$.

Frame CRC (FCRC)

Standards competing with CAN XL such as Flexray and Ethernet utilize CRC codes that achieve Hamming distance HD = 4 for maximum-length frames. For minimum-length frames, Hamming distance HD = 8 (Flexray) and HD = 6 (Ethernet) is achieved. The M = 24 generator polynomial 0xAEB6E5 (Koopman notation) used in Flexray achieves HD = 8 only for ultra short payload sizes (up to 8 byte) and goes down to HD = 6 already at a payload size of 9 byte. On the other hand, it maintains HD = 6 almost up to the maximal payload size of 259 byte. It is thus fair to say that the Flexray FCRC provides HD = 6 for almost all practical payload sizes.

The M = 32 generator polynomial 0x82608EDB (Koopman notation) used in Ethernet performs comparatively bad (despite having 8 bit more redundancy): it achieves only HD =5 for very small payload sizes and this deteriorates to HD = 4 already at a payload size of 372 byte.

In order to achieve comparable CRC error detection performance as the Flexray and Ethernet polynomials, we propose to use a generator polynomial that achieves HD = 6 throughout the full range of possible CAN XL payload sizes, i.e., from 1 byte to 2 048 byte. This is not possible with the M = 24 Flexray polynomial and, in fact, it is not possible with any generator polynomial with M < 31. Thus, in order to provide some safety margin, we propose to use a generator polynomial with M = 32 for the CAN XL FCRC (same CRC length as Ethernet).

It follows that the FCRC message length varies between $34 + 13 + 1 \cdot 8 = 56$ and $34 + 13 + 2048 \cdot 8 = 16431$ bit. Thus, the task at hand is to find an M = 32 generator polynomial that achieves $\text{HD}_{56}, \ldots, \text{HD}_{16431} \geq 6$. Ideally (in order to lower the undetected error rate by guaranteed detection of long burst errors and any odd number of bit errors), the polynomial should have $g_0 = 1$ and it should be divisible by $x + 1$.

Finding such polynomials is a computationally very demanding task. For the case M = 32, it has already been tackled in literature. Reference [8] lists the M = 32 generator polynomial 0xFA567D89 (Koopman notation). The same polynomial was already found in [9], but wrongly listed as 0x1F6ACFB13 (normal notation), while it should have been 0x1F4ACFB13 as pointed out by [8].

The Hamming distance profile of the code generated by 0xFA567D89 is shown (among the Flexray and Ethernet polynomials) in Figure 2, solid curve. It can be clearly seen that the polynomial achieves HD = 8 for small payload sizes.

The Hamming distance goes down to HD = 6 at message length k = 275 bit, which is maintained until the maximal payload size. As stated, this includes most of the header fields as well as the HCRC redundancy and thus double-protecting these fields by the FCRC causes no degradation in terms of Hamming distance.

We stress that 0xFA567D89 never falls below one of the Flexray and Ethernet polynomials in the full range of possible CAN XL payload sizes (it actually also outperforms the Ethernet polynomial over the full range of possible Ethernet payload sizes and also the Flexray polynomial over almost the full range of Flexray payload sizes).

Using the code generated by 0xFA567D89 results in the following properties of the FCRC:

1. Any erroneous frame (including all fields marked as "part of FCRC message" in table 1) that is affected by no more than 5 bit errors can be detected with guarantee.

Table 1: Fields of the CAN XL frame that are protected by either of the two CRCs

field name	field size [bit]	dynamic bit stuffing	part of HCRC message	comment	
SOF	1	y	n	n	start of frame, fixed to 0
ID	11	y	y	y	unique identifier
RRS	1	y	y	y	remote request substitution
IDE	1	y	n	n	identifier extension, fixed to 0
dynamic stuff bits	0-3	y	y	n	positions according to dynamic bit stuffing rule
FDF	1	n	n	n	flexible data rate format, fixed to 1
XLF	1	n	n	n	fixed to 1
resXL	1	n	n	n	fixed to 0
ADS	3	n	n	y	fixed pattern
PT	8	n	y	y	payload type
DLC	11	n	y	y	payload length (in byte)
SBC	3	n	y	y	dynamic stuff bit count
HCRC	13	n	n	y	HCRC redundancy (M = 13)
payload	8 to 16384	n	n	y	payload data
FCRC	32	n	n	n	FCRC redundancy (M = 32)

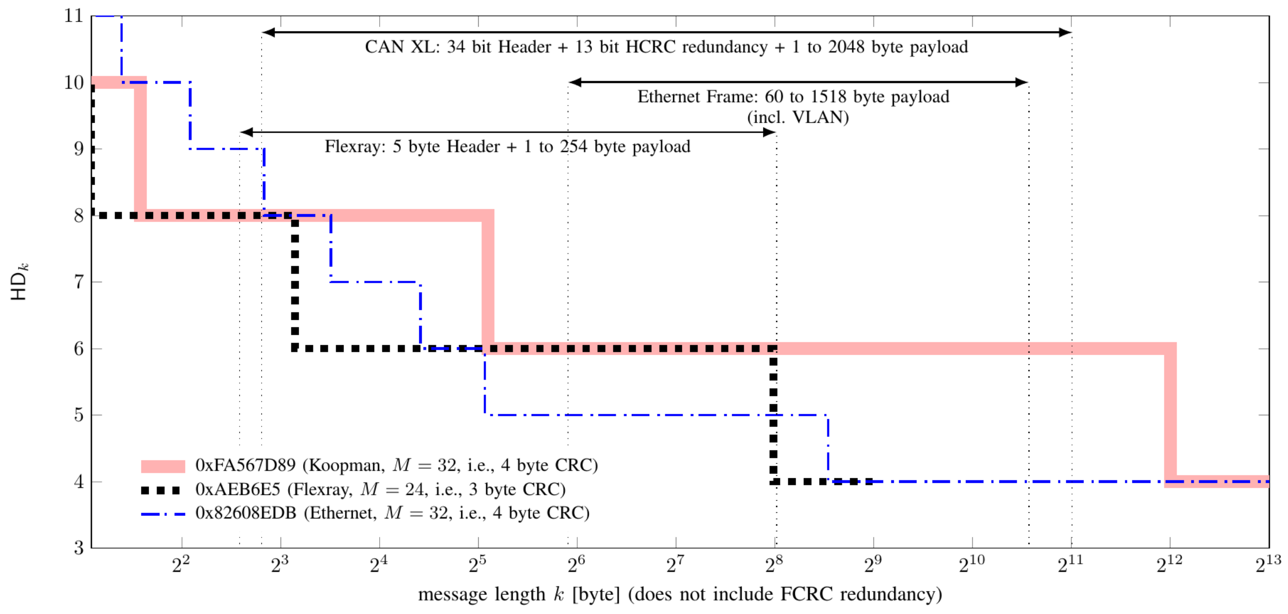


Figure 2: Hamming distance profiles for the Flexray, Ethernet, and proposed CAN XL FCRC generator polynomials. Note that the x-axis is logarithmic and given in byte and thus message length is k = 8 · x (since k is given in bit). All polynomials are given in Koopman notation. (Source: Dr. Christian Senger)

Additionally, due to the fact that 0xFA567D89 has least significant coefficient g_0 = 1 and factor x + 1 (see definition of g_HCRC (x) below) we have:

2. Any erroneous frame that is affected by a single burst error of burst-length no more than 32 can be detected with guarantee. In other words, any received header where the bit flips are constrained to a set of 32 consecutive bit is guaranteed-detectable.
3. Any erroneous header that is affected by an odd number of bit errors can be detected with guarantee.

We stress once again that many error patterns that do not fall into cases (1) to (3) can also be detected, but without guarantee.

Due to its aforementioned properties we propose to use 0xFA567D89 as the FCRC generator polynomial, that is, we propose

$$\begin{aligned} g_{\mathrm{FCRC}}(x) &= x^{32}+x^{31}+x^{30}+x^{29}+x^{28}+x^{26}+x^{23} \\ &\quad +x^{21}+x^{19}+x^{18}+x^{15}+x^{14}+x^{13} \\ &\quad +x^{12}+x^{11}+x^{9}+x^{8}+x^{4}+x+1 \\ &= (x^{15}+x^{14}+x^{11}+x^{6}+x^{4}+x^{3}+x^{2}+x+1) \\ &\quad \cdot (x^{15}+x^{4}+1) \\ &\quad \cdot (x+1)^2. \end{aligned}$$

For the convenience of the reader we state $g_{FCRC}(x)$ in the three commonly used notations:

M	ISO	Normal	Koopman
13	0x1F4ACFB13	0xF4ACFB13	0xF4ACFB13

The initialization vector plays only a minor role since dynamic stuff bits are excluded from the FCRC. However, defining an initialization vector is inevitable and we propose to use $(\rho_0, \rho_1, \ldots, \rho_{31}) = (1, 0, \ldots, 0)$.

Conclusion and outlook

We presented generator polynomials for use in the header and frame CRCs of the current CAN XL draft and showed that their error correction performance matches or outperforms the CRC codes in competing standards. Further improvements in the undetected error rate could be achieved by taking the actual error patterns that occur in CAN XL systems into consideration, which would require a detailed characterization of those patterns for different real-world scenarios. So far, our proposal is based on the simplifying assumption that the CAN XL bus behaves like a binary symmetric channel with occasional error bursts. In order to improve the detection capabilities for burst errors, CRC codes over larger alphabets could be taken into consideration.

Appendix

Generator polynomials are frequently represented as hexadecimal numbers in order to save space. One way to do that is used in ISO 11898 [10] and works as follows: write the coefficient vector of the polynomial with most significant bit (MSB) first, pad it on the left with zeros to length 4s, where $s = \lceil (M+1)/4 \rceil$), and then interpret each block of four bits by the corresponding hexadecimal number (again MSB left). This is called the ISO notation. in which, for example, the generator polynomial

$$\begin{aligned} g(x) &= x^{18}+x^{15}+x^{13}+x^{12}+x^{11}+x^{10}+x^{9} \\ &\quad +x^{7}+x^{6}+x^{4}+x^{2}+1, \end{aligned}$$

having coefficient vector $(1,0,0,1,0,1,1,1,1,1,0,1,1,0,1,0,1,0,1)$ and $s = 5$, is represented by

$$(\underbrace{0,1,0,0}_{4},\underbrace{1,0,1,1}_{\mathrm{B}},\underbrace{1,1,1,0}_{\mathrm{E}},\underbrace{1,1,0,1}_{\mathrm{D}},\underbrace{0,1,0,1}_{5}) \longleftrightarrow \text{0x4BED5}.$$

For the first alternative notation, write the coefficient vector with MSB first, pad it on the left with zeros to length 4s, replace the leftmost nonzero bit (i.e., g_M = 1) by a zero and then interpret each block of four by the corresponding hexadecimal number. This is called the normal notation in which, for example, g(x) as above is represented by

$$(\underbrace{0,1,0,0}_{4},\underbrace{1,0,1,1}_{\mathrm{B}},\underbrace{1,1,1,0}_{\mathrm{E}},\underbrace{1,1,0,1}_{\mathrm{D}},\underbrace{0,1,0,1}_{5}) \longleftrightarrow \text{0x4BED5}.$$

▷

Another alternative representation (popularized by Koopman [8]) can be obtained for $g(x)$ that fulfill the $g_0 = 1$ property (such as the polynomials proposed): write the coefficient vector with most significant bit (MSB) first, pad it on the left with zeros to length $4s + 1$, delete the rightmost bit (i.e., $g_0 = 1$) and then interpret each block of four by the corresponding hexadecimal number. This is called the Koopman notation. In Koopman notation, g(x) as above is represented by

$$(\underbrace{0,0,1,0}_{2},\underbrace{0,1,0,1}_{5},\underbrace{1,1,1,1}_{F},\underbrace{0,1,1,0}_{6},\underbrace{1,0,1,0}_{A},\cancel{1})$$
$$\longleftrightarrow 0\text{x}25\text{F}6\text{A}.$$

It is straightforward to recover g(x) from any of the hexadecimal notations by simply reversing the respective process. ◀

Author

Dr. Christian Senger
Institute of Telecommunications, University of Stuttgart
senger@inue.uni-stuttgart.de
www.inue.uni-stuttgart.de

References

[1] "Standard for Ethernet - Amendment 1: Physical Layer Specifications and Management Parameters for 100 Mb/s Operation over a Single Balanced Twisted Pair Cable (100BASE-T1)," ISO/IEC/IEEE 8802- 3:2017/Amd 1:2017(E), pp. 1–92, March 2018.
[2] Robert Bosch GmbH. (2019) CAN XL, Next step in CAN evolution. [Online]. Available: https://www.bosch-semiconductors.com/ news/ t-newsdetailpage-4.html
[3] A. Mutter and F. Hartwich, "Advantages of CAN FD error detection mechanisms compared to classical CAN," in In Proceedings of The international CAN Conference 2015 (iCC 2015), 2015.
[4] A. Mutter, "CAN XL error detection capabilities," in In Proceedings of The international CAN Conference 2020 (iCC 2020), 2020.
[5] F. MacWilliams and N. Sloane, The Theory of Error-Correcting Codes, 2nd ed. North-Holland Publishing Company, 1978.
[6] J. Charzinski, "Performance of the Error Detection Mechanisms in CAN," in Proceedings of the 1st International CAN Conference, September 1994, pp. 1/20–1/29.
[7] J. Unruh, H.-J. Mathony, and K.-H. Kaiser, "Error detection analysis of automotive communication protocols," SAE Transactions, vol. 99, pp. 976–985, 1990.
[8] P. Koopman, "32-bit cyclic redundancy codes for internet applications," in Proceedings International Conference on Dependable Systems and Networks, 6 2002, pp. 459–468, doi: 10.1109/DSN.2002.1028931.
[9] G. Castagnoli, S. Brauer, and M. Herrmann, "Optimization of cyclic redundancy-check codes with 24 and 32 parity bits," IEEE Transactions on Communications, vol. 41, no. 6, pp. 883–892, June 1993, doi: 10.1109/26.231911.
[10] "Data link layer and physical signalling," ISO 11898-1:2015, pp. 1–65, December 2015.

HIGH-END CONNECTIVITY AND DATA MANAGEMENT

TELEMATICS AND CLOUD SYSTEMS FOR IOT AND SERVICE 4.0

www.s-i-e.de

Sontheim

Continuous digitization for smart vehicles

Modular on-board units with Linux – ready for condition based monitoring. Including flash-over-the-air and embedded diagnostic functionality.

Sontheim IoT Device Manager and IoT Analytics Manager – for a highly secure, comfortable and individual visualization and management of your data.

Telematic ECU – COMhawk® xt

IoT Device Manager and IoT Analytics Manager

Integrated flash-over-the-air functionality

Modular on-board telematics series

Embedded diagnostics functionality

Multi-protocol support (J1939, J2534, UDS, KWP, …)

Ready for condition based monitoring