Document Title	E2E Protocol Specification
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	849

Document Status	Final
Part of AUTOSAR Standard	Foundation
Part of Standard Release	1.4.0

Document Change History			
Date	**Release**	**Changed by**	**Description**
2018-03-29	1.4.0	AUTOSAR Release Management	• No content changes
2017-12-08	1.3.0	AUTOSAR Release Management	• No content changes
2017-10-27	1.2.0	AUTOSAR Release Management	• Initial Release

Disclaimer

Table of Contents

1 Introduction and functional overview

The concept of E2E communication protection assumes that safety-related data exchange shal lbe protected at runtime against the effects of faults on the communication link (see Figure 1.1). Faults detected between a sender and a receiver using E2E communication protection include systematic software faults, such as fauls that are introduced on the lower communication layers of sender or receiver, and random hardware faults introduced by the MCU hardware, communication peripherals, transceivers, communication lines or other communication infrastructure.

The concept of E2E communication protection assumes that safety-related data exchange shall be protected at runtime against the effects of faults within the communication link (see Figure 1.1). Examples for such faults are random HW faults (e.g. corrupt registers of a CAN transceiver), interference (e.g. due to EMC), and systematic faults of the lower communication layers (e.g. RTE, IOC, COM and network stacks).

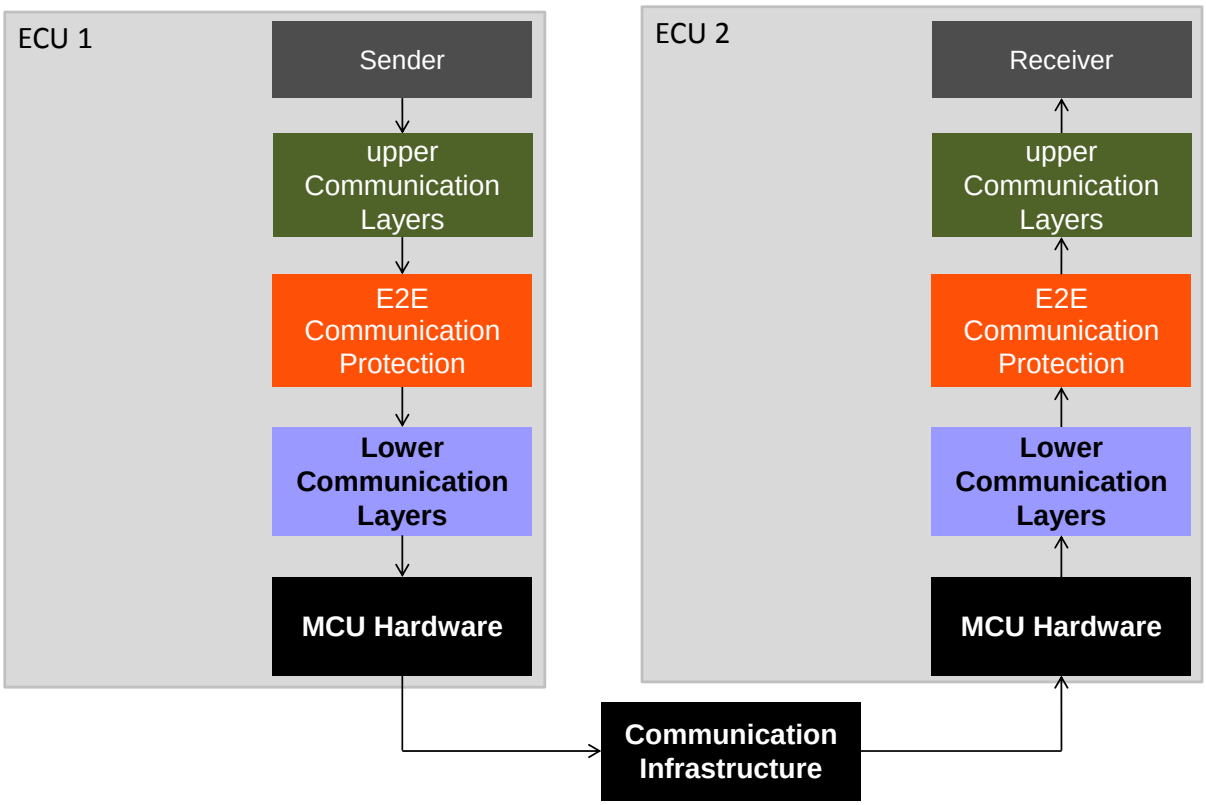

Figure 1.1: Overview of E2E communication protection between a sender and a receiver

By using E2E communication protection mechanisms, faults in lower software and hardware layers can be detected and handled at runtime. The E2E Supervision provides mechanisms for E2E communication protection, adequate for safety-related communication having requirements up to ASIL D.

The algorithms of protection mechanisms are implemented in the E2E Supervision. The callers of the E2E Supervision are responsible for the correct usage of the E2E

Supervision, in particular for providing correct parameters the E2E Supervision routines.

The E2E communication protection allows the following:

1. It protects the safety-related data to be sent by adding control data,
2. It verifies the safety-related data received using this control data, and
3. It provides the check result to the receiver, which then has to handle it sufficiently.

To provide the appropriate solution addressing flexibility and standardization, AUTOSAR specifies a set of flexible E2E profiles that implement an appropriate combination of E2E communication protection mechanisms. Each specified E2E profile has a fixed set of mechanisms, as well as configuration options to configure the protocol header layout and status evaluation on the receiver side.

The E2E Supervision can be invoked from communication middleware e.g. from Adaptive Platform's ARA, Classic Platform's RTE. It can be also invoked in a non-standardized way from other software, e.g. non-volatile memory managers, local IPCs, or intra-ECU bus stacks.

Appropriate usage of the E2E Supervision to fulfill the specific safety requirements for communication depends on several aspects. The specified profiles are capable, to a high probability, of detecting a large variety of communication faults. However, the use of a specific E2E profile requires the user to demonstrate that the selected profile provides sufficient error detection capabilities for the considered use case (taking into account various contributing factors, such as hardware failure rates, bit error rates, number of nodes in the network, repetition rate of messages, the usage of a gateway, potential software faults on the communication channel), as well as appropriate reaction on detected faults (e.g. by revoking repeated messages, determining timed-out communication or reacting on corrupt messages by initiating a safety reaction).

This specification specifies also the functionality, API and the configuration of the CRC routines.

The following routines for CRC calculation are specified:

- CRC8: SAEJ1850
- CRC8H2F: CRC8 0x2F polynomial
- CRC16
- CRC32
- CRC32P4: CRC32 0x1F4ACFB13 polynomial
- CRC64: CRC-64-ECMA

For all routines (CRC8, CRC8H2F, CRC16, CRC32, CRC32P4 and CRC64), the following calculation methods are possible:

- Table based calculation: Fast execution, but larger code size (ROM table)

- Runtime calculation: Slower execution, but small code size (no ROM table)
- Hardware supported CRC calculation (device specific): Fast execution, less CPU time

All routines are re-entrant and can be used by multiple applications at the same time. Hardware supported CRC calculation may be supported by some devices in the future.

2 Acronyms and Abbreviations

The glossary below includes acronyms and abbreviations relevant to the Communication Management that are not included in the [1, AUTOSAR glossary].

3 Related documentation

3.1 Input documents & related standards and norms

[1] Glossary
AUTOSAR_TR_Glossary

3.2 Related specification

1. SAE-J1850 8-bit CRC
2. CCITT-FALSE 16-bit CRC. Refer to:

   ITU-T Recommendation X.25 (1096) (Previously „CCITT Recommendation”)
   SERIES X: DATA NETWORKS AND OPEN SYSTEM COMMUNICATION
   Public data networks - Interfaces
   Interface between Data Terminal Equipment (DTE) and Data Circuit-terminating Equipment (DCE) for terminals operating in the packet mode and connected to public data networks by dedicated circuit

   Section 2.2.7.4 „Frame Check Sequence (FCS) field” and Appendix I „Examples of data link layer transmitted bit patterns by the DCE and the DTE”
   http://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-X.25-199610-I!!PDF-E&type=items
3. IEEE 802.3 Ethernet 32-bit CRC
4. ”32-Bit Cyclic Redundancy Codes for Internet Applications” [Koopman 2002]
5. Wikipedia.org -listing of CRCs, including CRC-64-ECMA
   https://en.wikipedia.org/wiki/Cyclic_redundancy_check

4 Constraints and assumptions

4.1 Limitations

E2E communication protection is limited to periodic or semi-periodic data communication paradigm, where the receiver (subscriber) has an expectancy on the regular reception of data and in case of communication loss/timeout or error, it performs an error handling.

Data communication is called sender/receiver in Classic Platform, and it is called event communication in Adaptive Platform. Note that the word event is a bit confusing as a periodic communication is required.

This means, a protection of client-server (methods) as well as non-periodic data communication (e.g. transmission only on occurence of a specific event) are not supported by E2E communication protection.

4.2 Applicability to car domains

The E2E supervision is applicable for the realization of safety-related automotive systems implemented by various SW-Cs distributed across different ECUs in a vehicle, interacting via communication links. The Supervision may also be used for intra-ECU communication (e.g. between memory partitions, processes, OSes/VMs in the same micorcontroller, between CPU cores or microcontrollers).

4.3 Background information concerning functional safety

This chapter provides some safety background information considered during the design of the E2E supervision, including the fault model for communication and definition of sources of faults.

4.3.1 Functional safety and communication

With respect to the exchange of information in safety-related systems, the mechanisms for the in-time detection of causes for faults, or effects of faults as listed below, can be used to design suitable safety concepts, e.g. to achieve freedom from interference between system elements sharing a common communication infrastructure (see ISO 26262-6:2011, annex D.2.4):

- repetition of information;
- loss of information;
- delay of information;

- insertion of information;
- masquerade or incorrect addressing of information;
- incorrect sequence of information;
- corruption of information;
- asymmetric information sent from a sender to multiple receivers;
- information from a sender received by only a subset of the receivers;
- blocking access to a communication channel.

4.3.2 Sources of faults in E2E communication

E2E communication protection aims to detect and mitigate the causes for or effects of communication faults arising from:

1. (systematic) software faults,
2. (random) hardware faults,
3. transient faults due to external influences.

These three sources are described in the sections below.

4.3.2.1 Software faults

Software like, communication stack modules and RTE, may contain faults, which are of a systematic nature.

Systematic faults may occur in any stage of the system's life cycle including specification, design, manufacturing, operation, and maintenance, and they will always appear when the circumstances (e.g. trigger conditions for the root-cause) are the same. The consequences of software faults can be failures of the communication, like interruption of sending of data, overrun of the receiver (e.g. buffer overflow), or underrun of the sender (e.g. buffer empty). To prevent (or to handle) resulting failures the appropriate technical measures to detect and handle such faults (e.g. program flow monitoring or E2E supervision) have to be considered.

4.3.2.2 Random hardware faults

A random hardware fault is typically the result of electrical overload, degradation, aging or exposure to external influences (e.g. environmental stress) of hardware parts. A random hardware fault cannot be avoided completely, but its probability can be evaluated and appropriate technical measures can be implemented (e.g. diagnostics).

4.3.2.3 External influences, environmental stress

This includes influences like EMI, ESD, humidity, corrosion, temperature or mechanical stress (e.g. vibration).

4.3.3 Communication faults

Relevant faults related to the exchange of information are listed in this section.

4.3.3.1 Repetition of information

A type of communication fault, were information is received more than once.

4.3.3.2 Loss of information

A type of communication fault, were information or parts of information are removed from a stream of transmitted information.

4.3.3.3 Delay of information

A type of communication fault, were information is received later than expected.

4.3.3.4 Insertion of information

A type of communication fault, were additional information is inserted into a stream of transmitted information.

4.3.3.5 Masquerading

A type of communication fault, were non-authentic information is accepted as authentic information by a receiver.

4.3.3.6 Incorrect addressing

A type of communication fault, were information is accepted from an incorrect sender or by an incorrect receiver.

4.3.3.7 Incorrect sequence of information

A type of communication fault, which modifies the sequence of the information in a stream of transmitted information.

4.3.3.8 Corruption of information

A type of communication fault, which changes information.

4.3.3.9 Asymmetric information sent from a sender to multiple receivers

A type of communication fault, were receivers do receive different information from the same sender.

4.3.3.10 Information from a sender received by only a subset of the receivers

A type of communication fault, were some receivers do not receive the information.

4.3.3.11 Blocking access to a communication channel

A type of communication fault, were the access to a communication channel is blocked.

5 Functional specification

This chapter contains the specification of the internal functional behavior of the E2E supervision. For general introduction of the E2E supervision, see first chapter 1.

5.1 Overview of communication protection

An important aspect of a communication protection mechanism is its standardization and its flexibility for different purposes. This is resolved by having a set of E2E Profiles, that define a combination of protection mechanismsm, a message format, and a set of configuration parameters.

Moreover, some E2E Profiles have standard E2E variants. An E2E variant is simply a set of configuration options to be used with a given E2E Profile. For example, in E2E Profile 1, the positions of CRC and counter are configurable. The E2E variant 1A requires that CRC starts at bit 0 and counter starts at bit 8.

Apart from E2E Profiles, the E2E Supervision provides also elementary functions (e.g. multibyte CRCs) to build additional (e.g. vendor-specific) safety protocols.

E2E communication protection works as follows:

- Sender: addition of control fields like CRC or counter to the transmitted data;
- Receiver: evaluation of the control fields from the received data, calculation of control fields (e.g. CRC calculation on the received data), comparison of calculated control fields with an expected/received content.

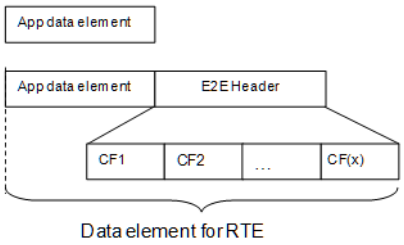

Figure 5.1: Safety protocol concept (with exemplary location of the E2E header)

Each E2E Profile has a specific set of control fields with a specific functional behavior and with specific properties for the detection of communication faults.

5.2 Overview of E2E Profiles

The E2E Profiles provide a consistent set of data protection mechanisms, designed to protecting against the faults considered in the fault model.

Each E2E Profile provides an alternative way to protect the communication, by means of different algorithms. However, E2E Profile have similar interfaces and behavior.

[PRS_E2EProtocol_00221] ⌈ Each E2E Profile shall use a subset of the following data protection mechanisms:

1. A CRC, provided by CRC Supervision;
2. A Sequence Counter incremented at every transmission request, the value is checked at receiver side for correct incrementation;
3. An Alive Counter incremented at every transmission request, the value checked at the receiver side if it changes at all, but correct incrementation is not checked;
4. A specific ID for every port data element sent over a port or a specific ID for every I-PDU group (global to system, where the system may contain potentially several ECUs);
5. Timeout detection:
   (a) Receiver communication timeout.

(b) Sender acknowledgement timeout.

Depending on the used communication and network stack, appropriate subsets of these mechanisms are defined as E2E communication profiles.

⌋(*RS_E2EProtocol_08531*)

Some of the above mechanisms are implemented in RTE, COM, and/or communication stacks. However, to reduce or avoid an allocation of safety requirements to these modules, they are not considered: E2E Supervision provides all mechanisms internally (only with usage of CRC Supervision).

The E2E Profiles can be used for both inter and intra ECU communication. The E2E Profiles were specified for specific communication infrastructure, such as CAN, CAN FD, FlexRay, LIN, Ethernet.

Depending on the system, the user selects which E2E Profile is to be used, from the E2E Profiles provided by E2E Supervision.

[PRS_E2EProtocol_00217] ⌈ The implementation of the E2E Supervision shall provide at least one of the E2E Profiles.

⌋(*RS_E2EProtocol_08528*)

However, it is possible that specific implementations of E2E Supervision do not provide all profiles, but only a one of them.

5.2.1 Error detection

[PRS_E2EProtocol_00012] ⌈The internal Supervision mechanisms error detection and reporting shall be implemented according to the pre-defined E2E Profiles specified in the following sections. ⌋(*RS_E2EProtocol_08528*)

5.3 Specification of E2E Profile 1 (Only for CP)

Profile 1 shall provide the following mechanisms:

[PRS_E2EProtocol_00218] ⌈

Mechanism	Description
Counter	4bit (explicitly sent) representing numbers from 0 to 14 incremented on every send request. Both Alive Counter and Sequence Counter mechanisms are provided by E2E Profile 1, evaluating the same 4 bits.
Timeout monitoring	Timeout is determined by E2E Supervision by means of evaluation of the Counter, by a nonblocking read at the receiver. Timeout is reported by E2E Supervision to the caller by means of the status flags in E2E_P01CheckStatusType.

Data ID	16 bit, unique number, included in the CRC calculation. For dataIdMode equal to 0, 1 or 2, the Data ID is not transmitted, but included in the CRC computation (implicit transmission). For dataIdMode equal to 3: • the high nibble of high byte of DataID is not used (it is 0x0), as the DataID is limited to 12 bits, • the low nibble of high byte of DataID is transmitted explicitly and covered by CRC calculation when computing the CRC over Data. • the low byte is not transmitted, but it is included in the CRC computation as start value (implicit transmission, like for dataIDMode equal to 0, 1 or 2) .
CRC	CRC-8-SAE J1850 - 0x1D (x8 + x4 + x3 + x2 + 1), but with different start and XOR values (both start value and XOR value are 0x00). This CRC is provided by CRC Supervision. Starting with AUTOSAR R4.0, the SAE8 CRC function of the CRC Supervision uses 0xFF as start value and XOR value. To compensate a different behavior of the CRC Supervision, the E2E Supervision applies additional XOR 0xFF operations starting with R4.0, to come up with 0x00 as start value and XOR value. Note: This CRC polynomial is different from the CRC-polynomials used by FlexRay, CAN and LIN.

⌋(*RS_E2EProtocol_08529*, *RS_E2EProtocol_08530*, *RS_E2EProtocol_08533*)

The E2E mechanisms can detect the following faults or effects of faults:

E2E Mechanism	Detected communication faults
Counter	Repetition, Loss, insertion, incorrect sequence, blocking
Transmission on a regular basis and timeout monitoring using E2E-Supervision [1]	Loss, delay, blocking
Data ID + CRC	Masquerade and incorrect addressing, insertion
CRC	Corruption, Asymmetric information [2]

[PRS_E2EProtocol_00070] ⌈

E2E Profile 1 shall use the polynomial of CRC-8-SAE J1850, i.e. the polynomial 0x1D (x8 + x4 + x3 + x2 + 1), but with start value and XOR value equal to 0x00.

⌋(*RS_E2EProtocol_08531*)

[1] Implementation by sender and receiver, which are using E2E-Supervision
[2] for a set of data protected by same CRC

For details of CRC calculation, the usage of start values and XOR values see CRC Supervision in section 5.13.

5.3.1 Data Layout

In the E2E Profile 1, the layout is in general free to be defined by the user - it is only constrained by the byte alignment user requirements E2E0062 and E2E0063 (i.e. bytes of data elements signals must be aligned to byte limits). However, the E2E Profile 1 variants constrain the layout, see subsection 5.3.6.

5.3.2 Counter

In E2E Profile 1, the counter is initialized, incremented, reset and checked by E2E profile.

[PRS_E2EProtocol_00075] ⌈In E2E Profile 1, on the sender side, for the first transmission request of a data element the counter shall be initialized with 0 and shall be incremented by 1 for every subsequent send request (from sender SW-C). When the counter reaches the value 14 (0xE), then it shall restart with 0 for the next send request (i.e. value 0xF shall be skipped). All these actions shall be executed by E2E Supervision.

⌋*(RS_E2EProtocol_08528)*

[PRS_E2EProtocol_00076] ⌈In E2E Profile 1, on the receiver side, by evaluating the counter of received data against the counter of previously received data, the following shall be detected by the E2E Supervision: (1) no new data has arrived since last invocation of E2E Supervision check function, (2) no new data has arrived since receiver start, (3) the data is repeated (4) counter is incremented by one (i.e. no data lost), (5) counter is incremented more than by one, but still within allowed limits (i.e. some data lost), (6) counter is incremented more than allowed (i.e. too many data lost).

⌋*(RS_E2EProtocol_08528)*

Case 3 corresponds to the failed alive counter check, and case 6 correspond to failed sequence counter check.

The above requirements are specified in more details by the UML diagrams in the following document sections.

5.3.3 Data ID

The unique Data IDs are to verify the identity of each transmitted safety-related data element.

[PRS_E2EProtocol_00163] ⌈There shall be following four inclusion modes for the two-byte Data ID into the calculation of the one-byte CRC:

1. E2E_P01_DATAID_BOTH: both two bytes (double ID configuration) are included in the CRC, first low byte and then high byte (see variant 1A - PRS_E2EProtocol_00227) or
2. E2E_P01_DATAID_ALT: depending on parity of the counter (alternating ID configuration) the high and the low byte is included (see variant 1B - PRS_E2EProtocol_00228). For even counter values the low byte is included and for odd counter values the high byte is included.
3. E2E_P01_DATAID_LOW: only the low byte is included and high byte is never used. This equals to the situation if the Data IDs (in a given application) are only 8 bits.
4. E2E_P01_DATAID_NIBBLE:
   - the high nibble of high byte of DataID is not used (it is 0x0), as the DataID is limited to 12 bits,
   - the low nibble of high byte of DataID is transmitted explicitly and covered by CRC calculation when computing the CRC over Data.
   - the low byte is not transmitted, but it is included in the CRC computation as start value (implicit transmission, like for the inclusion modes _BOTH, _ALT and _LOW)

⌋*(RS_E2EProtocol_08528)*

[PRS_E2EProtocol_00085] ⌈In E2E Profile 1, with E2E_P01DataIDMode equal to E2E_P01_DATAID_BOTH or E2E_P01_DATAID_ALT the length of the Data ID shall be 16 bits (i.e. 2 byte). ⌋*(RS_E2EProtocol_08528)*

[PRS_E2EProtocol_00169] ⌈In E2E Profile 1, with E2E_P01DataIDMode equal to E2E_P01_DATAID_LOW, the high byte of Data ID shall be set to 0x00. ⌋ *(RS_E2EProtocol_08528)*

The above requirement means that when high byte of Data ID is unused, it is set to 0x00.

[PRS_E2EProtocol_00306] ⌈ In E2E Profile 1, with E2E_P01DataIDMode equal to E2E_P01_DATAID_NIBBLE, the high nibble of the high byte shall be 0x0. ⌋ *(RS_E2EProtocol_08528)*

The above requirement means that the address space with E2E_P01_DATAID_NIBBLE is limited to 12 bits.

In case of usage of E2E Supervision for protecting data elements, due to multiplicity of communication (1:1 or 1:N), a receiver of a data element receives it only from one sender. In case of usage of E2E Supervision for protecting I-PDUs, because each I-PDU has a unique Data ID, the receiver COM of an I-PDU receives it from only from

one sender COM. As a result (regardless if the protection is at data element level or at I-PDUs), the receiver expects data with only one Data ID. The receiver uses the expected Data ID to calculate the CRC. If CRC matches, it means that the Data ID used by the sender and expected Data ID used by the receiver are the same.

5.3.4 CRC calculation

E2E Profile 1 uses CRC-8-SAE J1850, but using different start and XOR values. This checksum is already provided by AUTOSAR CRC Supervision, which typically is quite efficient and may use hardware support.

[PRS_E2EProtocol_00083] ⌈E2E Profile 1 shall use CRC-8-SAE J1850 for CRC calculation. It shall use 0x00 as the start value and XOR value. ⌋ *(RS_E2EProtocol_08529, RS_E2EProtocol_08533)*

[PRS_E2EProtocol_00190] ⌈E2E Profile 1 shall use the Crc_CalculateCRC8 () function of the SWS CRC Supervision for calculating CRC checksums. ⌋ *(RS_E2EProtocol_08528, RS_E2EProtocol_08531)*

Note: The CRC used by E2E Profile 1 is different than the CRCs used by FlexRay and CAN and is provided by different software modules (FlexRay and CAN CRCs are provided by hardware support in Communication Controllers, not by CRC Supervision).

The CRC calculation is illustrated by the following two examples.

For standard variant 1A:

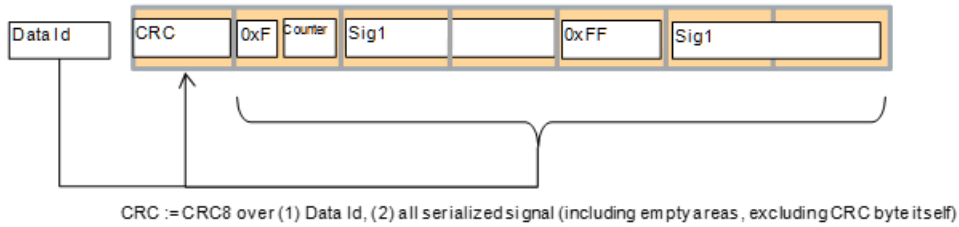

Figure 5.2: E2E Profile 1 variant 1A CRC calculation example

For standard variant 1C:

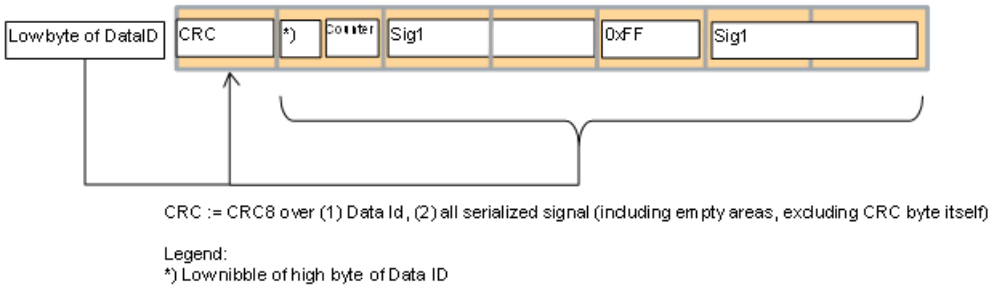

Figure 5.3: E2E Profile 1 variant 1C CRC calculation example

The Data ID can be encoded in CRC in different ways, see [PRS_E2EProtocol_00163].

[PRS_E2EProtocol_00082] ⌈In E2E Profile 1, the CRC is calculated over:

1. First over the one or two bytes of the Data ID (depending on Data ID configuration), and
2. then over all transmitted bytes of a safety-related complex data element/signal group (except the CRC byte).

⌋*(RS_E2EProtocol_08536)*

5.3.5 Timeout detection

The previously mentioned mechanisms (CRC, counter, Data ID) enable to check the validity of received data element, when the receiver is executed independently from the data transmission, i.e. when receiver is not blocked waiting for Data Elements or respectively signal groups, but instead if the receiver reads the currently available data (i.e. checks if new data is available). Then, by means of the counter, the receiver can detect loss of communication and timeouts.

The attribute State->Status = E2E_P01STATUS_REPEATED means that there is a repetition (caused either by communication loss, delay or duplication of the previous message). The receiver uses State->Status for detecting communication timeouts.

5.3.6 E2E Profile 1 variants

The E2E Profile 1 has variants. The variants are specific configurations of E2E Profile.

[PRS_E2EProtocol_00227] ⌈ The E2E Profile variant 1A is defined as follows:

1. CRC is the 0th byte in the signal group (i.e. starts with bit offset 0)
2. Alive counter is located in lowest 4 bits of 1st byte (i.e. starts with bit offset 8)
3. E2E_P01DataIDMode = E2E_P01_DATAID_BOTH
4. SignalIPdu.unusedBitPattern = 0xFF.

⌋*(RS_E2EProtocol_08528)*

[PRS_E2EProtocol_00228] ⌈ The E2E Profile variant 1B is defined as follows:

1. CRC is the 0th byte in the signal group (i.e. starts with bit offset 0)
2. Alive counter is located in lowest 4 bits of 1st byte (i.e. starts with bit offset 8)
3. E2E_P01DataIDMode = E2E_P01_DATAID_ALTERNATING
4. SignalIPdu.unusedBitPattern = 0xFF.

⌋(*RS_E2EProtocol_08528*)

Below is an example compliant to 1A/1B:

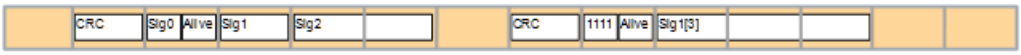

Figure 5.4: E2E Profile 1 example layout (two signal groups protected by E2E in one I-PDU)

[PRS_E2EProtocol_00307] ⌈ The E2E Profile variant 1C is defined as follows:

1. CRC is the 0th byte in the signal group (i.e. starts with bit offset 0)
2. Alive counter is located in lowest 4 bits of 1st byte (i.e. starts with bit offset 8)
3. The Data ID nibble is located in the highest 4 bits of 1st byte (i.e. starts with bit offset 12)
4. E2E_P01DataIDMode = E2E_P01_DATAID_NIBBLE
5. SignalIPdu.unusedBitPattern = 0xFF.

⌋(*RS_E2EProtocol_08528*)

5.3.7 E2E_P01Protect

[PRS_E2EProtocol_00195] ⌈ The function E2E_P01Protect() shall:

1. write the Counter in Data,
2. write DataID nibble in Data (E2E_P01_DATAID_NIBBLE) in Data
3. compute the CRC over DataID and Data
4. write CRC in Data
5. increment the Counter (which will be used in the next invocation of E2E_P01Protect()),as specified by Figure 5.5 and Figure 5.6

⌋(*RS_E2EProtocol_08528*)

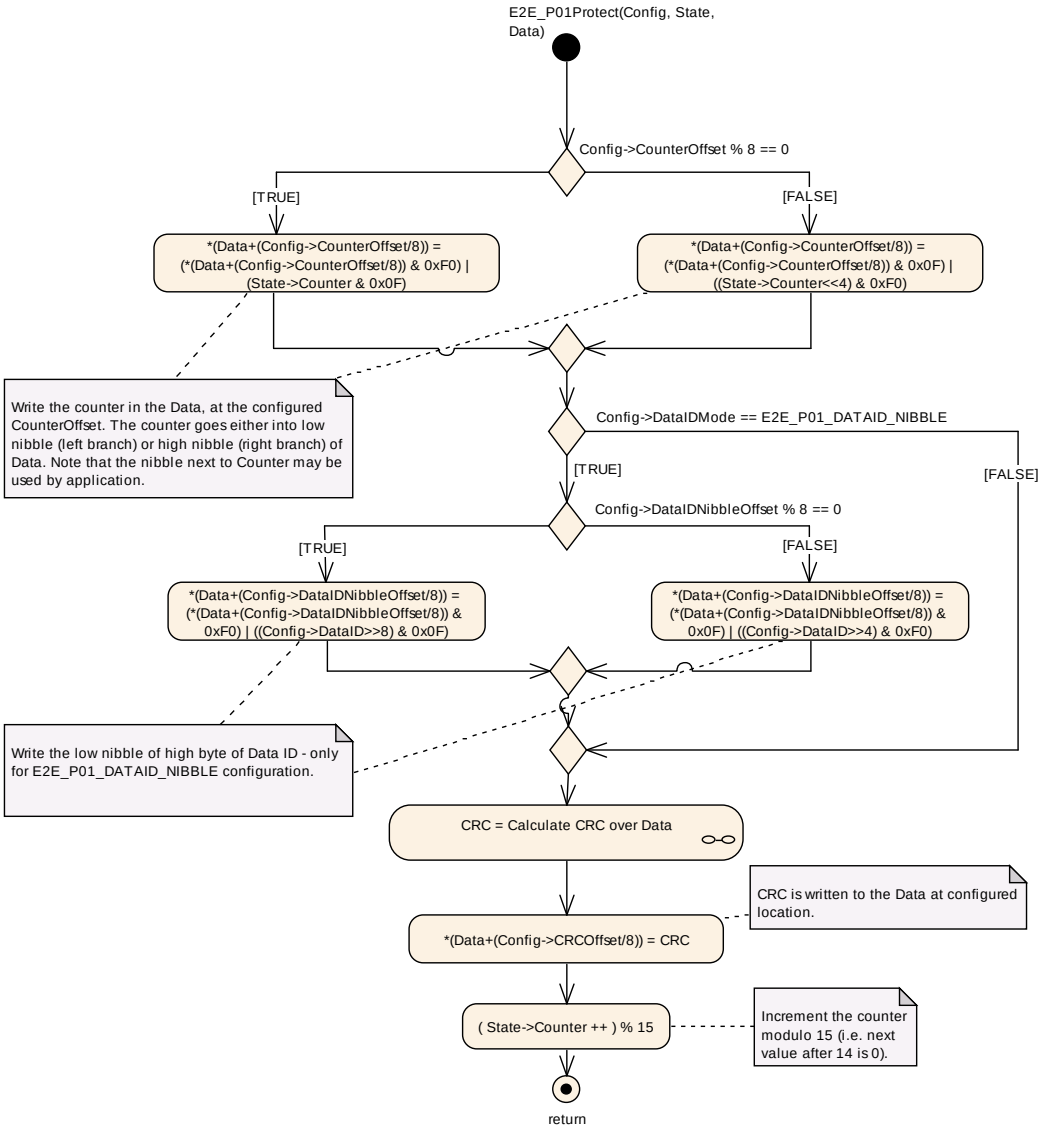

Figure 5.5: E2E_P01Protect()

5.3.8 Calculate CRC

The diagram of the function E2E_P01Protect() (see above chapter) and E2E_P01Check() (see below chapter) have a sub-diagram specifying the calculation of CRC:

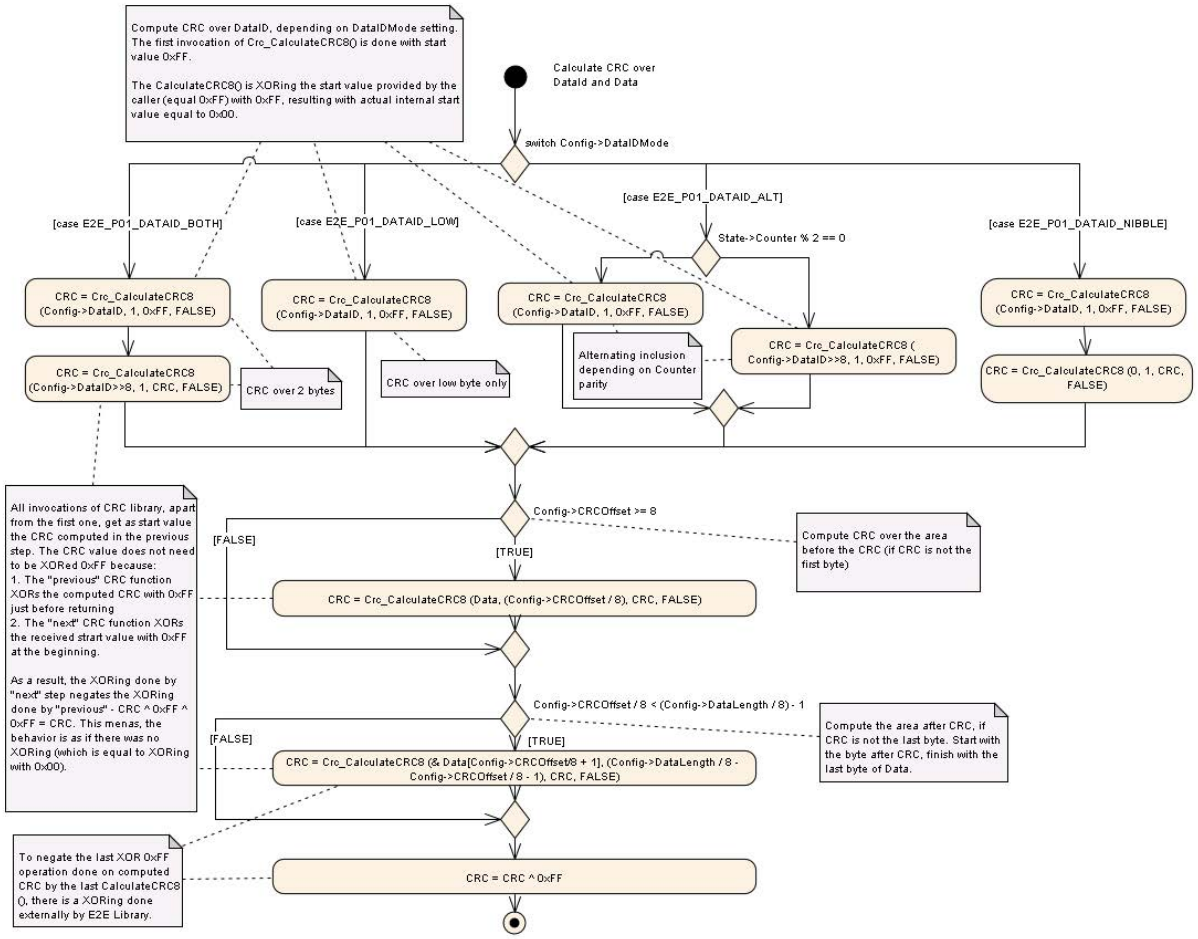

Figure 5.6: Subdiagram „Calculate CRC over Data ID and Data", used by E2E_P01Protect() and E2E_P01Check()

It is important to note that the function Crc_CalculateCRC8 of CRC Supervision / CRC routines have changed is functionality since R4.0, i.e. it is different in R3.2 and >=R4.0:

1. There is an additional parameter Crc_IsFirstCall
2. The function has different start value and different XOR values (changed from 0x00 to OxFF).

This results with a different value of computed CRC of a given buffer.

To have the same results of the functions E2E_P01Protect() and E2E_P02Check() in >=R4.0 and R3.2, while using differently functioning CRC Supervision, E2E «„compensates" different behavior of the CRC Supervision. This results with different invocation of the CRC Supervision by E2E Supervision Figure 5.6 in >=R4.0 and R3.2. This means Figure 5.6 is different in >=R4.0 and R3.2.

5.3.9 E2E_P01Check

[PRS_E2EProtocol_00196] ⌈The function E2E_P01Check shall

1. Check the CRC
2. Check the Data ID nibble, i.e. compare the expected value with the received value (for E2E_P01_DATAID_NIBBLE configuration only)
3. Check the Counter,
4. determine the check Status,as specified by Figure 5.7 and Figure 5.6.

⌋(*RS_E2EProtocol_08528*, *RS_E2EProtocol_08530*)

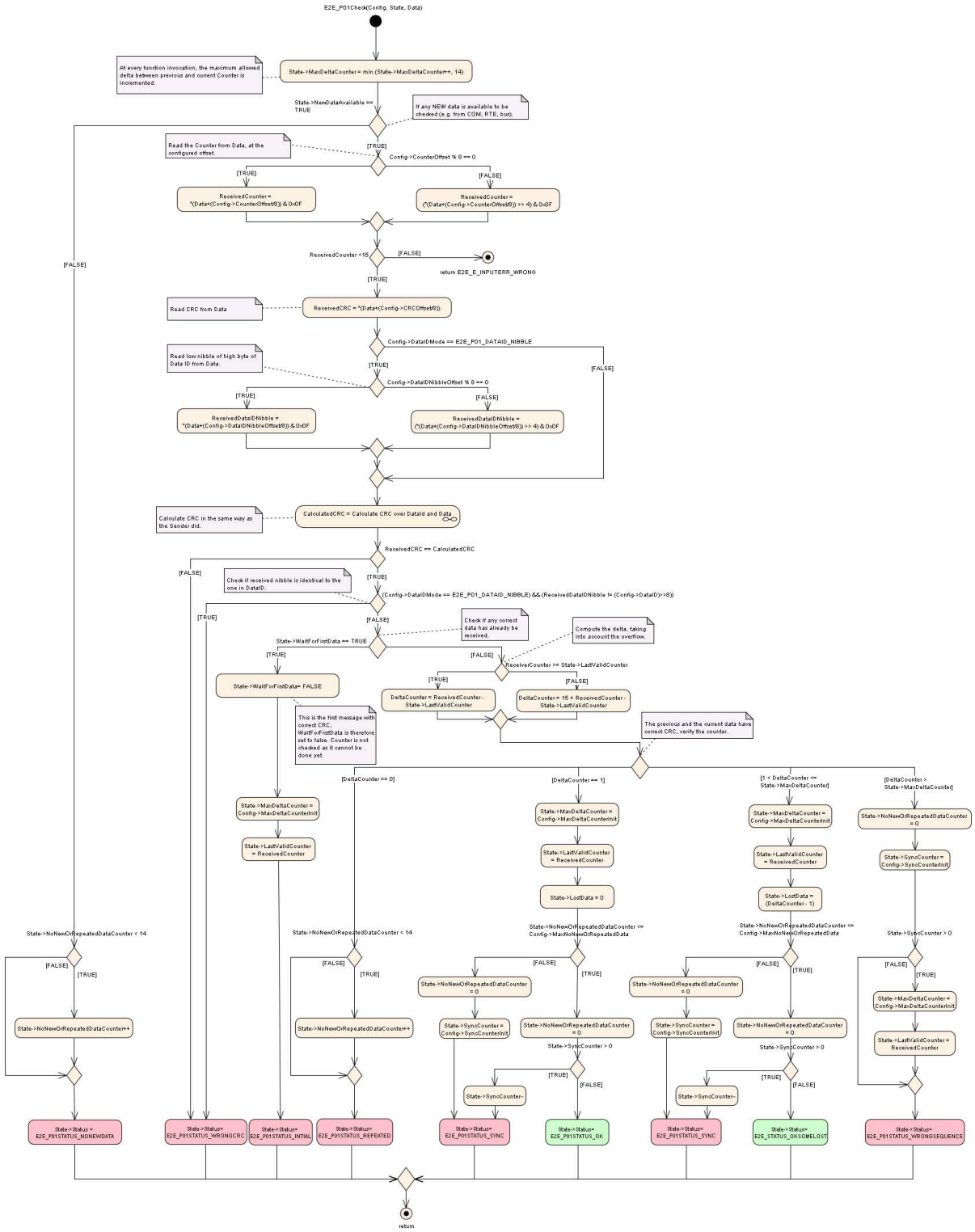

Figure 5.7: E2E_P01Check()

The diagram of the function E2E_P01Check() has a sub-diagram specifying the calculation of CRC, which is shown by Figure 5.6.

5.3.10 E2E Profile 1 Protocol Examples

The default configuration assumed for the following examples, if not otherwise stated to be different:

E2E_P01ConfigType field	Value
CounterOffset	8
CRCOffset	0
DataID	0x01234
DataIDNibbleOffset	12
DataIDMode	E2E_P01DATAID_BOTH
DataLength	64
MaxDeltaCounterInit	1
MaxNoNewOrRepeatedData	15
SyncCounterInit	0

E2E_P01ProtectStateType field	Value
Counter	0

Result data of E2E_P01Protect() with data equals all zeros (0x00), counter = 0:

Byte							
0	1	2	3	4	5	6	7
0xcc	0x00	0x00	0x00	0x00	0x00	0x00	0x00

Result data of E2E_P01Protect() with data equals all zeros (0x00), counter = 1:

Byte							
0	1	2	3	4	5	6	7
0x91	0x01	0x00	0x00	0x00	0x00	0x00	0x00

5.3.10.1 DataIDMode set to E2E_P01DATAID_ALT

Result data of E2E_P01Protect() with data equals all zeros (0x00), counter = 0:

Byte							
0	1	2	3	4	5	6	7
0x5f	0x00	0x00	0x00	0x00	0x00	0x00	0x00

Result data of E2E_P01Protect() with data equals all zeros (0x00), counter = 1:

Byte							
0	1	2	3	4	5	6	7
0x93	0x01	0x00	0x00	0x00	0x00	0x00	0x00

5.3.10.2 DataIDMode set to E2E_P01DATAID_LOW

Result data of E2E_P01Protect() with data equals all zeros (0x00), counter = 0:

Byte							
0	1	2	3	4	5	6	7
0x5f	0x00	0x00	0x00	0x00	0x00	0x00	0x00

Result data of E2E_P01Protect() with data equals all zeros (0x00), counter = 1:

Byte							
0	1	2	3	4	5	6	7
0x02	0x01	0x00	0x00	0x00	0x00	0x00	0x00

5.3.10.3 DataIDMode set to E2E_P01DATAID_NIBBLE

Result data of E2E_P01Protect() with data equals all zeros (0x00), counter = 0:

Byte							
0	1	2	3	4	5	6	7
0x2a	0x10	0x00	0x00	0x00	0x00	0x00	0x00

Result data of E2E_P01Protect() with data equals all zeros (0x00), counter = 1:

Byte							
0	1	2	3	4	5	6	7
0x77	0x11	0x00	0x00	0x00	0x00	0x00	0x00

5.4 Specification of E2E Profile 2 (only for CP)

[PRS_E2EProtocol_00219] ⌈Profile 2 shall provide the following mechanisms:

Mechanism	Description

Sequence Number (Counter)	4bit (explicitly sent) representing numbers from 0 to 15 incremented by 1 on every send request (Bit 0:3 of Data 1) at sender side. The counter is incremented on every call of the E2E_P02Protect() function, i.e. on every transmission request of the SW-C
Message Key used for CRC calculation (Data ID)	8 bit (not explicitly sent) The specific Data ID used to calculate the CRC depends on the value of the Counter and is an element of an pre-defined set of Data IDs (value of the counter as index to select the particular Data ID used for the protection). For every Data element, the List of Data IDs depending on each value of the counter is unique.
Data ID + CRC	Masquerade and incorrect addressing, insertion
Safety Code(CRC)	8 bit explicitly sent (Data 0) Polynomial: 0x2F (x8 + x5 + x3 + x2 + x + 1) Start value: 0xFF Final XOR-value: 0xFF Note: This CRC polynomial is different from the CRC-polynomials used by FlexRay and CAN.

⌋(*RS_E2EProtocol_08529*, *RS_E2EProtocol_08530*, *RS_E2EProtocol_08533*)

The mechanisms provided by Profile 2 enable the detection of the relevant failure modes except message delay (for details see the table in section 5.4):

Since this profile is implemented in a Supervision, the Supervision's E2E_P02Check() function itself cannot ensure to be called in a periodic manner. Thus, a required protection mechanism against undetected message delay (e.g. Timeout) must be implemented in the caller.

The E2E mechanisms can detect the following faults or effects of faults:

E2E Mechanism	**Detected communication faults**
Counter	Repetition, Loss, insertion, incorrect sequence, blocking
Transmission on a regular bases and timeout monitoring using E2E-Library [3]	Loss, delay, blocking
Data ID + CRC	Masquerade and incorrect addressing, insertion
CRC	Corruption, Asymmetric information [4]

[3] Implementation by sender and receiver
[4] for a set of data protected by same CRC

[PRS_E2EProtocol_00117] ⌈E2E Profile 2 shall use the Crc_CalculateCRC8H2F() function of the SWS CRC Supervision for calculating CRC checksums. ⌋ *(RS_E2EProtocol_08531)*

[PRS_E2EProtocol_00118] ⌈E2E Profile 2 shall use 0xFF as the start value CRC_StartValue8 for CRC calculation. ⌋*(RS_E2EProtocol_08528)*

[PRS_E2EProtocol_00119] ⌈In E2E Profile 2, the specific Data ID used to calculate a specific CRC shall be of length 8 bit. ⌋*(RS_E2EProtocol_08528)*

[PRS_E2EProtocol_00120] ⌈In E2E Profile 2, the specific Data ID used for CRC calculation shall be selected from a pre-defined DataIDList[16] using the value of the Counter as an index. ⌋*(RS_E2EProtocol_08528)*

Each data, which is protected by a CRC owns a dedicated DataIDList which is deposited on the sender site and all the receiver sites.

The pre-defined DataIDList[16] is generated offline. In general, there are several factors influencing the contents of DataIDList, e.g:

1. length of the protected data
2. number of protected data elements
3. number of cycles within a masquerading fault has to be detected
4. number of senders and receivers
5. characteristics of the CRC polynomial.

Due to the limited length of the 8bit polynomial, a masquerading fault cannot be detected in a specific cycle when evaluating a received CRC value. Due to the adequate Data IDs in the DataIDList, a masquerading fault can be detected in one of the successive communication cycles.

Due to the underlying rules for the DataIDList, the system design of the application has to take into account that a masquerading fault is detected not until evaluating a certain number of communication cycles.

[PRS_E2EProtocol_00121] ⌈ In E2E Profile 2, the layout of the data buffer (Data) shall be as depicted in below, with a maximum length of 256 bytes (i.e. N=255)

Figure 5.8

⌋*(RS_E2EProtocol_08528)*

[PRS_E2EProtocol_00122] ⌈ In E2E Profile 2, the CRC shall be Data[0]. ⌋ *(RS_E2EProtocol_08528)*

[PRS_E2EProtocol_00123] ⌈ In E2E Profile 2, the Counter shall be the low nibble (Bit 0...Bit 3) of Data[1]. ⌋*(RS_E2EProtocol_08528)*

[PRS_E2EProtocol_00124] ⌈ In E2E Profile 2, the E2E_P02Protect() function shall not modify any bit of Data except the bits representing the CRC and the Counter. ⌋ *(RS_E2EProtocol_08528)*

[PRS_E2EProtocol_00125] ⌈ In E2E Profile 2, the E2E_P02Check() function shall not modify any bit in Data. ⌋*(RS_E2EProtocol_08528)*

5.4.1 E2E_P02Protect

The E2E_P02Protect() function of E2E Profile 2 is called by a SW-C in order to protect its application data against the failure modes as shown in table in section 5.4. E2E_P02Protect() therefore calculates the Counter and the CRC and puts it into the data buffer (Data). A flow chart with the visual description of the function E2E_P02Protect() is depicted in Figure 5.9 and Figure 5.10.

[PRS_E2EProtocol_00126] ⌈ In E2E Profile 2, the E2E_P02Protect() function shall perform the activities as specified in Figure 5.9 and Figure 5.10. ⌋ *(RS_E2EProtocol_08528, RS_E2EProtocol_08536)*

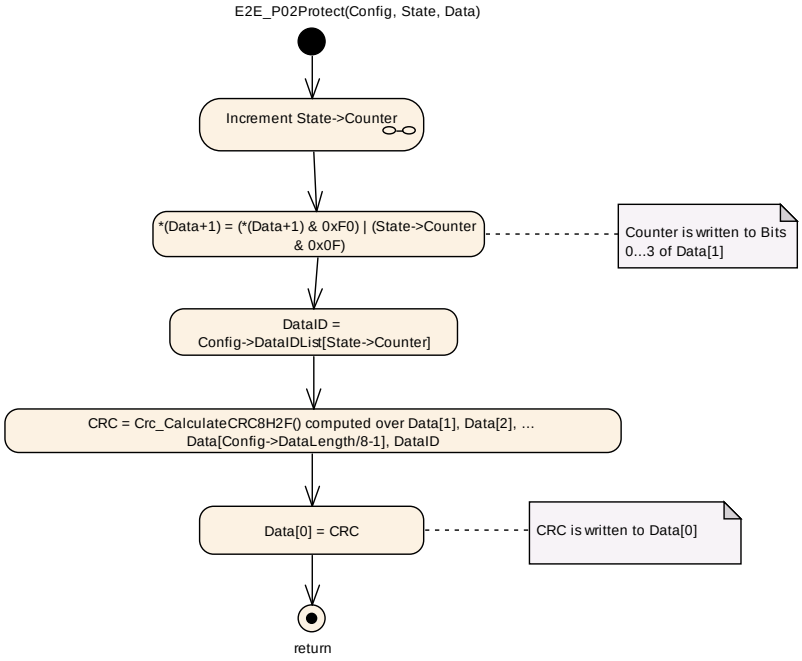

Figure 5.9: E2E_P02Protect()

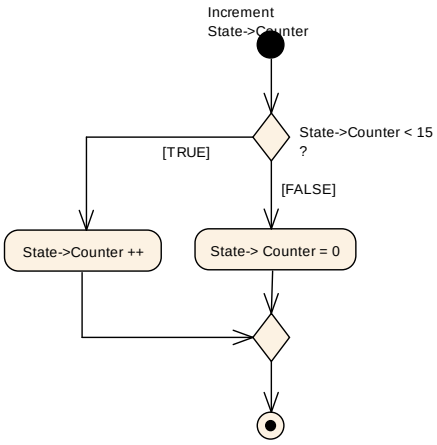

Figure 5.10: Increment Counter

[PRS_E2EProtocol_00127] ⌈ In E2E Profile 2, the E2E_P02Protect() function shall increment the Counter of the state (E2E_P02ProtectStateType) by 1 on every transmission request from the sending SW-C, i.e. on every call of E2E_P02Protect(). ⌋ *(RS_E2EProtocol_08528)*

[PRS_E2EProtocol_00128] ⌈ In E2E Profile 2, the range of the value of the Counter shall be [0...15]. ⌋*(RS_E2EProtocol_08528)*

[PRS_E2EProtocol_00129] ⌈ When the Counter has reached its upper bound of 15 (0xF), it shall restart at 0 for the next call of the E2E_P02Protect() from the sending SW-C. ⌋*(RS_E2EProtocol_08528)*

[PRS_E2EProtocol_00130] ⌈ In E2E Profile 2, the E2E_P02Protect() function shall update the Counter (i.e. low nibble (Bit 0...Bit 3) of Data byte 1) in the data buffer (Data) after incrementing the Counter. ⌋*(RS_E2EProtocol_08528)*

The specific Data ID used for this send request is then determined from a DataIDList[] depending on the value of the Counter (Counter is used as an index to select the Data ID from DataIDList[]). The DataIDList[] is defined in E2E_P02ConfigType.

[PRS_E2EProtocol_00132] ⌈ In E2E Profile 2, after determining the specific Data ID, the E2E_P02Protect() function shall calculate the CRC over Data[1], Data[2], ... Data[Config->DataLength/8-1] of the data buffer (Data) extended with the Data ID. ⌋ *(RS_E2EProtocol_08528)*

[PRS_E2EProtocol_00133] ⌈ In E2E Profile 2, the E2E_P02Protect() function shall update the CRC (i.e. Data[0]) in the data buffer (Data) after computing the CRC. ⌋ *(RS_E2EProtocol_08528)*

The specific Data ID itself is not transmitted on the bus. It is just a virtual message key used for the CRC calculation.

5.4.2 E2E_P02Check

The E2E_P02Check() function is used as an error detection mechanism by a caller in order to check if the received data is correct with respect to the failure modes mentioned in the profile summary.

A flow chart with the visual description of the function E2E_P02Check() is depicted in Figure 5.11, Figure 5.12 and Figure 5.13.

[PRS_E2EProtocol_00134] ⌈ In E2E Profile 2, the E2E_P02Check() function shall perform the activities as specified in Figure 5.11, Figure 5.12 and Figure 5.13. ⌋ *(RS_E2EProtocol_08528, RS_E2EProtocol_08536)*

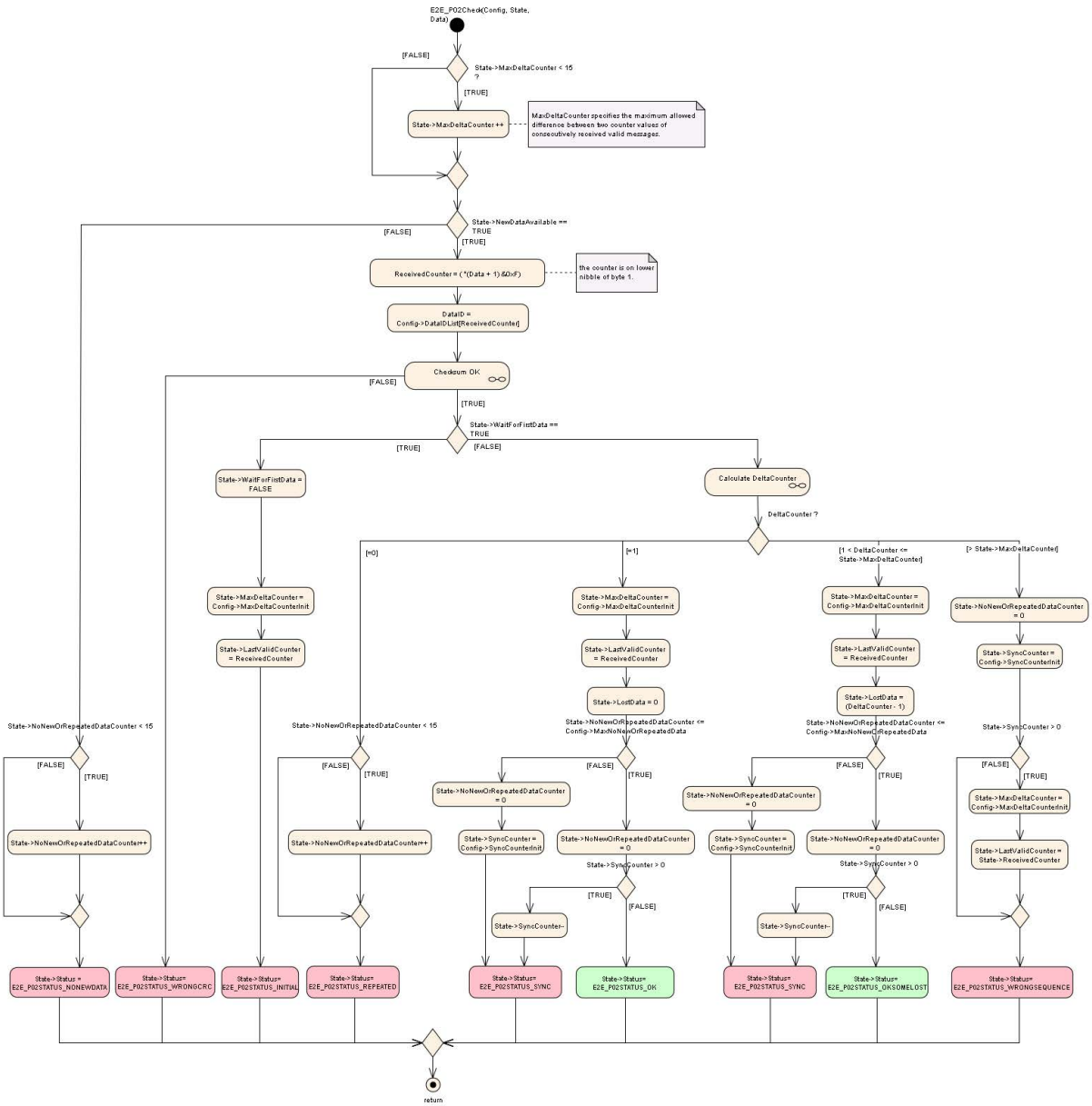

Figure 5.11: E2E_P02Check

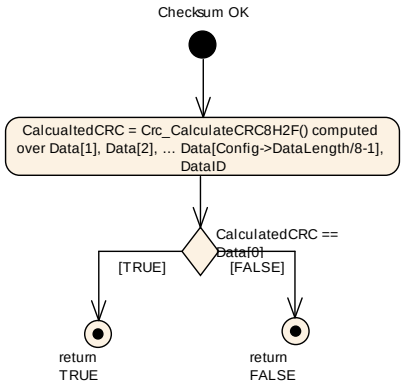

Figure 5.12: Checksum OK

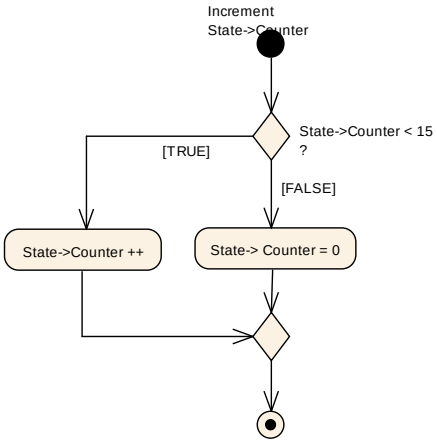

Figure 5.13: Calculate Delta Counter

First, the E2E_P02Check() function increments the value MaxDeltaCounter. MaxDeltaCounter specifies the maximum allowed difference between two Counter values of two consecutively received valid messages. Note: MaxDeltaCounter is used in order to perform a plausibility check for the failure mode re-sequencing. If the flag NewDataAvailable is set, the E2E_P02Check() function continues with the evaluation of the CRC. Otherwise, it returns with Status set to E2E_P02STATUS_NONEWDATA. To evaluate the correctness of the CRC, the following actions are performed:

- The specific Data ID is determined using the value of the Counter as provided in Data.
- Then the CRC is calculated over Data payload extended with the Data ID as last Byte: CalculatedCRC = Crc_CalculateCRC8H2F() calculated over Data[1], Data[2], ... Data[Config->DataLength/8-1], Data ID
- Finally, the check for correctness of the received Data is performed by comparing CalculatedCRC with the value of CRC stored in Data.

In case CRC in Data and CalculatedCRC do not match, the E2E_P02Check() function returns with Status E2E_P02STATUS_WRONGCRC, otherwise it continues with further evaluation steps.

The flag WaitForFirstData specifies if the SW-C expects the first message after startup or after a timeout error. This flag should be set by the SW-C if the SW-C expects the first message e.g. after startup or after reinitialization due to error handling. This flag is allowed to be reset by the E2E_P02Check() function only. The reception of the first message is a special event because no plausibility checks against previously received messages is performed.

If the flag WaitForFirstData is set by the SW-C, E2E_P02Check() does not evaluate the Counter of Data and returns with Status E2E_P02STATUS_INITIAL. However, if the flag WaitForFirstData is reset (the SW-C does not expect the first message) the E2E_P02Check() function evaluates the value of the Counter in Data.

For messages with a received Counter value within a valid range, the E2E_P02Check() function returns either with E2E_P02STATUS_OK or E2E_P02STATUS_OKSOMELOST. In LostData, the number of missing messages since the most recently received valid message is provided to the SW-C.

For messages with a received Counter value outside of a valid range, E2E_P02Check() returns with one of the following states: E2E_P02STATUS_WRONGSEQUENCE or E2E_P02STATUS_REPEATED.

[PRS_E2EProtocol_00135] ⌈ In E2E Profile 2, the local variable DeltaCounter shall be calculated by subtracting LastValidCounter from Counter in Data, considering an overflow due to the range of values [0...15]. ⌋*(RS_E2EProtocol_08528)*

Details on the calculation of DeltaCounter are depicted in Figure 7-12.

[PRS_E2EProtocol_00136] ⌈In E2E Profile 2, MaxDeltaCounter shall specify the maximum allowed difference between two Counter values of two consecutively received valid messages. ⌋*(RS_E2EProtocol_08528)*

[PRS_E2EProtocol_00137] ⌈ In E2E Profile 2, MaxDeltaCounter shall be incremented by 1 every time the E2E_P02Check() function is called, up to the maximum value of 15 (0xF). ⌋*(RS_E2EProtocol_08528)*

[PRS_E2EProtocol_00138] ⌈ In E2E Profile 2, the E2E_P02Check() function shall set Status to E2E_P02STATUS_NONEWDATA if the attribute NewDataAvailable is FALSE. ⌋*(RS_E2EProtocol_08528)*

[PRS_E2EProtocol_00139] ⌈ In E2E Profile 2, the E2E_P02Check() function shall determine the specific Data ID from DataIDList using the Counter of the received Data as index. ⌋*(RS_E2EProtocol_08528)*

[PRS_E2EProtocol_00140] ⌈ In E2E Profile 2, the E2E_P02Check() function shall calculate CalculatedCRC over Data[1], Data[2], ... Data[Config->DataLength/8-1] of the data buffer (Data) extended with the determined Data ID. ⌋*(RS_E2EProtocol_08528)*

[PRS_E2EProtocol_00141] ⌈ In E2E Profile 2, the E2E_P02Check() function shall set Status to E2E_P02STATUS_WRONGCRC if the calculated CalculatedCRC value differs from the value of the CRC in Data.

⌋*(RS_E2EProtocol_08528)*

[PRS_E2EProtocol_00142] ⌈ In E2E Profile 2, the E2E_P02Check() function shall set Status to E2E_P02STATUS_INITIAL if the flag WaitForFirstData is TRUE. ⌋ *(RS_E2EProtocol_08528)*

[PRS_E2EProtocol_00143] ⌈ In E2E Profile 2, the E2E_P02Check() function shall clear the flag WaitForFirstData if it returns with Status E2E_P02STATUS_INITIAL. ⌋ *(RS_E2EProtocol_08528)*

For the first message after start up no plausibility check of the Counter is possible. Thus, at least a minimum number of messages need to be received in order to perform a check of the Counter values and in order to guarantee that at least one correct message was received.

[PRS_E2EProtocol_00145] ⌈ The E2E_P02Check() function shall

- set Status to E2E_P02STATUS_WRONGSEQUENCE; and
- re-initialize SyncCounter with SyncCounterInit

if the calculated value of DeltaCounter exceeds the value of MaxDeltaCounter. ⌋ *(RS_E2EProtocol_08528)*

[PRS_E2EProtocol_00146] ⌈ The E2E_P02Check() function shall set Status to E2E_P02STATUS_REPEATED if the calculated DeltaCounter equals 0. ⌋ *(RS_E2EProtocol_08528)*

[PRS_E2EProtocol_00147] ⌈ The E2E_P02Check() function shall set Status to E2E_P02STATUS_OK if the following conditions are true:

- the calculated DeltaCounter equals 1; and
- the value of the NoNewOrRepeatedDataCounter is less than or equal to MaxNoNewOrRepeatedData (i.e. State –> NoNewOrRepeatedDataCounter <= Config –> MaxNoNewOrRepeatedData); and
- the SyncCounter equals 0.

⌋ *(RS_E2EProtocol_08528)*

[PRS_E2EProtocol_00298] ⌈ The E2E_P02Check() function shall

- re-initialize SyncCounter with SyncCounterInit; and
- set Status to E2E_P02STATUS_SYNC; if the following conditions are true:
- the calculated DeltaCounter is within the parameters of 1 and MaxDeltaCounter (i.e. 1 =/< DeltaCounter =/< MaxDeltaCounter); and
- the value of the NoNewOrRepeatedDataCounter exceeds MaxNoNewOrRepeatedData. (i.e. State NoNewOrRepeatedDataCounter > Config MaxNoNewOrRepeatedData)

⌋ *(RS_E2EProtocol_08528)*

[PRS_E2EProtocol_00299] ⌈ The E2E_P02Check() function shall

- decrement SyncCounter by 1; and
- set Status to E2E_P02STATUS_SYNC if the following conditions are true:
- the calculated DeltaCounter is within the parameters of 1 and MaxDeltaCounter (i.e. 1 =/< DeltaCounter =/< MaxDeltaCounter); and
- the value of the NoNewOrRepeatedDataCounter is less than or equal to MaxNoNewOrRepeatedData (i.e. State NoNewOrRepeatedDataCounter =/< Config MaxNoNewOrRepeatedData); and
- the SyncCounter exceeds 0.

⌋(*RS_E2EProtocol_08528*)

[PRS_E2EProtocol_00148] ⌈ The E2E_P02Check() function shall set Status to E2E_P02STATUS_OKSOMELOST if the following conditions are true:

- the calculated DeltaCounter is greater-than 1 but less-than or equal to MaxDeltaCounter (i.e. 1 < DeltaCounter =/< MaxDeltaCounter); and
- the NoNewOrRepeatedDataCounter is less than or equal to MaxNoNewOrRepeatedData (i.e. State NoNewOrRepeatedDataCounter =/< Config MaxNoNewOrRepeatedData); and
- the SyncCounter equals 0.

⌋(*RS_E2EProtocol_08528*)

[PRS_E2EProtocol_00149] ⌈ The E2E_P02Check() function shall set the value LostData to (DeltaCounter - 1) if the calculated DeltaCounter is greater-than 1 but less-than or equal to MaxDeltaCounter. ⌋(*RS_E2EProtocol_08528*)

[PRS_E2EProtocol_00150] ⌈ The E2E_P02Check() function shall r-initialize MaxDeltaCounter with MaxDeltaCounterInit if it returns one of the following Status:

- E2E_P02STATUS_OK; or
- E2E_P02STATUS_OKSOMELOST; or
- E2E_P02STATUS_INITIAL; or
- E2E_P02STATUS_SYNC; or
- E2E_P02STATUS_WRONGSEQUENCE on condition that SyncCounter exceeds 0 (i.e. SyncCounter > 0).

⌋(*RS_E2EProtocol_08528*)

[PRS_E2EProtocol_00151] ⌈ The E2E_P02Check() function shall set LastValidCounter to Counter of Data if it returns one of the following Status:

- E2E_P02STATUS_OK; or
- E2E_P02STATUS_OKSOMELOST; or

- E2E_P02STATUS_INITIAL; or
- E2E_P02STATUS_SYNC; or
- E2E_P02STATUS_WRONGSEQUENCE on condition that SyncCounter exceeds 0 (i.e. SyncCounter > 0).

⌋(*RS_E2EProtocol_08528*)

[PRS_E2EProtocol_00300] ⌈ The E2E_P02Check() function shall reset the NoNewOrRepeatedDataCounter to 0 if it returns one of the following status:

- E2E_P02STATUS_OK; or
- E2E_P02STATUS_OKSOMELOST; or
- E2E_P02STATUS_SYNC; or
- E2E_P02STATUS_WRONGSEQUENCE

⌋(*RS_E2EProtocol_08528*)

[PRS_E2EProtocol_00301] ⌈ The E2E_P02Check() function shall increment NoNewOrRepeatedDataCounter by 1 if it returns the Status E2E_P02STATUS_NONEWDATA or E2E_P02STATUS_REPEATED up to the maximum value of Counter (i.e. 15 or 0xF). ⌋(*RS_E2EProtocol_08528*)

5.5 Specification of E2E Profile 4

[PRS_E2EProtocol_00372] ⌈ Profile 4 shall provide the following control fields, transmitted at runtime together with the protected data:

Control field	Description
Length	16 bits, to support dynamic-size data.
Counter	16-bits.
CRC	32 bits, polynomial in normal form 0x1F4ACFB13, provided by CRC library. Note: This CRC polynomial is different from the CRC-polynomials used by FlexRay, CAN and LIN and TCPIP.
Data ID	32-bits, unique system-wide.

⌋(*RS_E2EProtocol_08529, RS_E2EProtocol_08530, RS_E2EProtocol_08533*)

The E2E mechanisms can detect the following faults or effects of faults:

Fault	Main safety mechanisms
Repetition of information	Counter
Loss of information	Counter

Delay of information	Counter
Insertion of information	Data ID
Masquerading	Data ID, CRC
Incorrect addressing	Data ID
Incorrect sequence of information	Counter
Corruption of information	CRC
Asymmetric information sent from a sender to multiple receivers	CRC (to detect corruption at any of receivers)
Information from a sender received by only a subset of the receivers	Counter (loss on specific receivers)
Blocking access to a communication channel	Counter (loss or timeout)

Table 5.1: Detectable communication faults using Profile 4

For details of CRC computation, the usage of start values and XOR values see CRC Supervision [7]

5.5.1 Data Layout

5.5.1.1 User data layout

In the E2E Profile 4, the user data layout (of the data to be protected) is not constrained by E2E Profile 4 - there is only a requirement that the length of data to be protected is multiple of 1 byte.

5.5.1.2 Header layout

The header of the E2E Profile 4 has one fixed layout, as follows:

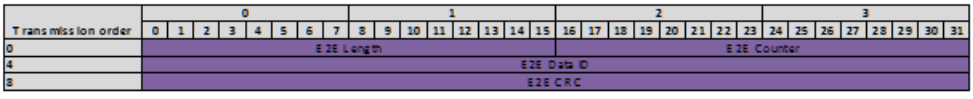

Figure 5.14: Profile 4 Header

The bit numbering shown above represents the order in which bits are transmitted. The E2E header fields (e.g. E2E Counter) are encoded as:

1. Big Endian (most significant byte fist) - imposed by profile
2. LSB Fist (least significant bit within byte first) - imposed by TCPIP bus

For example, the 16 bits of the E2E counter are transmitted in the following order (higher number meaning higher significance): 7 8 9 10 11 12 13 14 15 0 1 2 3 4 5 6 7.

The header can be placed at a specific location in the protected data, by configuring the offset of the entire E2E header.

5.5.2 Counter

In E2E Profile 4, the counter is initialized, incremented, reset and checked by E2E profile. The counter is not manipulated or used by the caller of the E2E Supervision.

[PRS_E2EProtocol_00478] ⌈In E2E Profile 4, on the sender side, for the first transmission request of a data element the counter shall be initialized with 0 and shall be incremented by 1 for every subsequent send request. When the counter reaches the maximum value (0xFF'FF), then it shall restart with 0 for the next send request. ⌋ *(RS_E2EProtocol_08539)*

Note: This specification was previously falsely identified as PRS_E2EProtocol_00324.

Note that the counter value 0xFF'FF is not reseved as a special invalid value, but it is used as a normal counter value.

In E2E Profile 4, on the receiver side, by evaluating the counter of received data against the counter of previously received data, the following is detected:

1. Repetition:

   a. no new data has arrived since last invocation of E2E Supervision check function, b. the data is repeated

2. OK: a. counter is incremented by one (i.e. no data lost), b. counter is incremented more than by one, but still within allowed limits (i.e. some data lost),

3. Wrong sequence: a. counter is incremented more than allowed (i.e. too many data lost).

Case 1 corresponds to the failed alive counter check, and case 3 correspond to failed sequence counter check.

The above requirements are specified in more details by the UML diagrams in the following document sections.

/subsectionData ID

The unique Data IDs are to verify the identity of each transmitted safety-related data element.

[PRS_E2EProtocol_00326] ⌈ In the E2E Profile 4, the Data ID shall be explicitly transmitted, i.e. it shall be the part of the transmitted E2E header.col_08539 counter shall be initialized with 0xFF'FF. ⌋*(RS_E2EProtocol_08539)*

[PRS_E2EProtocol_UC_00327] ⌈ In the E2E profile 4, the Data IDs shall be globally unique within the network of communicating system (made of several ECUs each sending different data). ⌋*(RS_E2EProtocol_08539)*

In case of usage of E2E Supervision for protecting data elements (i.e invocation from RTE), due to multiplicity of communication (1:1 or 1:N), a consumer of a data element expects only a specific data element, which is checked by E2E Supervision using Data ID.

In case of usage of E2E Supervision for protecting I-PDUs (i.e. invocation from COM), the receiver COM espects at a reception only a specific I-PDU, which is checked by E2E Supervision using Data ID.

5.5.3 Length

The Length field is introduced to support variable-size length - the Data [] array storing the serialized data can potentially have a different length in each cycle.

5.5.4 CRC

E2E Profile 4 uses a 32-bit CRC, to ensure a high detection rate and high Hamming Distance.

[PRS_E2EProtocol_00329] ⌈E2E Profile 4 shall use the Crc_CalculateCRC32P4 () function of the SWS CRC Supervision for calculating the CRC. ⌋ *(RS_E2EProtocol_08539, RS_E2EProtocol_08531)*

Note: The CRC used by E2E Profile 4 is different from the CRCs used by FlexRay, CAN and TCP/IP. It is also provided by different software modules (FlexRay, CAN and TCP/IP stack CRCs/checksums are provided by hardware support in Communication Controllers or by communication stack software, but not by CRC Supervision).

[PRS_E2EProtocol_00330] ⌈ In E2E Profile 4, the CRC shall be calculated over the entire E2E header (excluding the CRC bytes) and over the user data. ⌋ *(RS_E2EProtocol_08536)*

5.5.5 Timeout detection

The previously mentioned mechanisms (CRC, Counter, Data ID, Length) enable to check the validity of received data element, when the receiver is executed independently from the data transmission, i.e. when receiver is not blocked waiting for Data Elements or respectively I-PDUs, but instead if the receiver reads the currently available data (i.e. checks if new data is available). Then, by means of the counter, the receiver can detect loss of communication and timeouts.

5.5.6 E2E Profile 4 variants

The E2E Profile 4 variants are specified in TPS System Specification.

5.5.7 E2E_P04Protect

The function E2E_P04Protect() performs the steps as specified by the following eight diagrams in this section.

[PRS_E2EProtocol_00362] ⌈ The function E2E_P04Protect() shall have the following overall behavior:

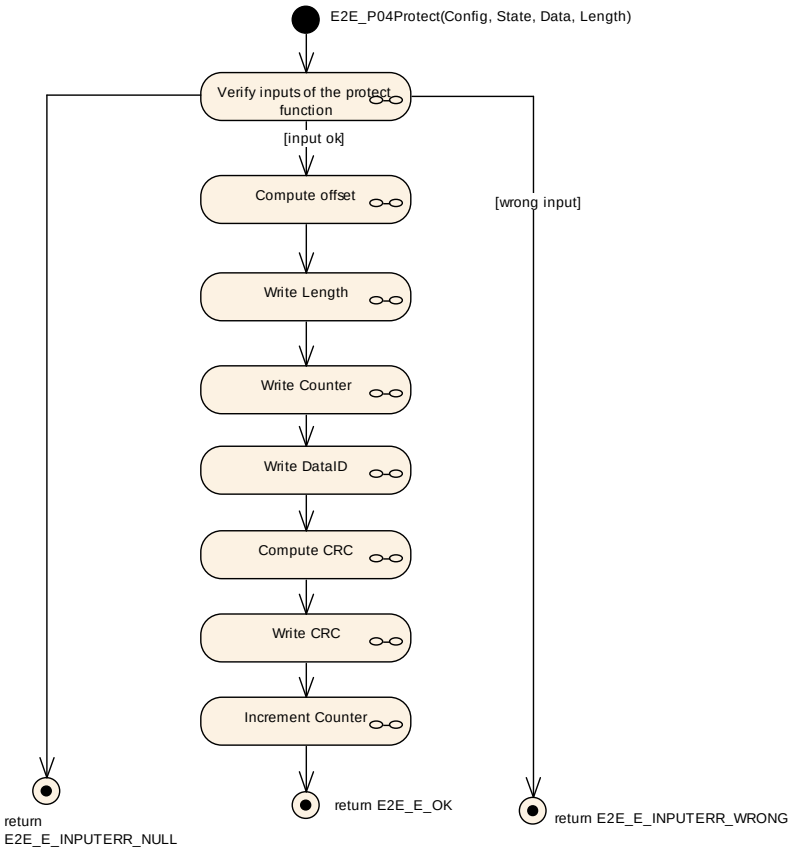

Figure 5.15

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00363] ⌈ The step ”Verify inputs of the protect function” in E2E_P04Protect() shall have the following behavior:

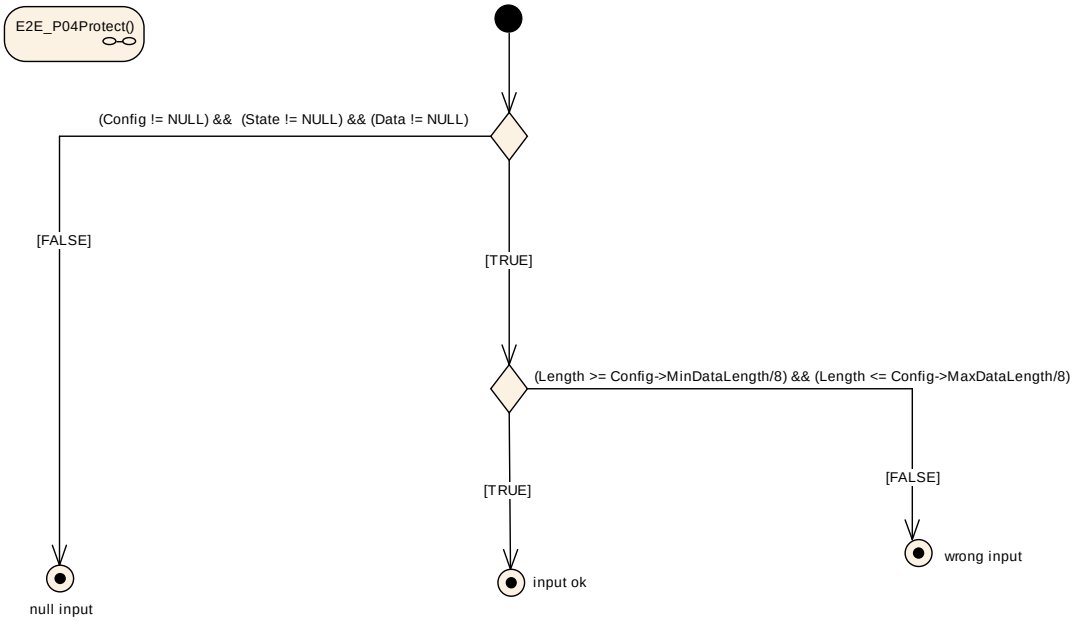

Figure 5.16

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00376] ⌈The step ”Compute offset” in E2E_P04Protect() and E2E_P04Check() shall have the following behavior:

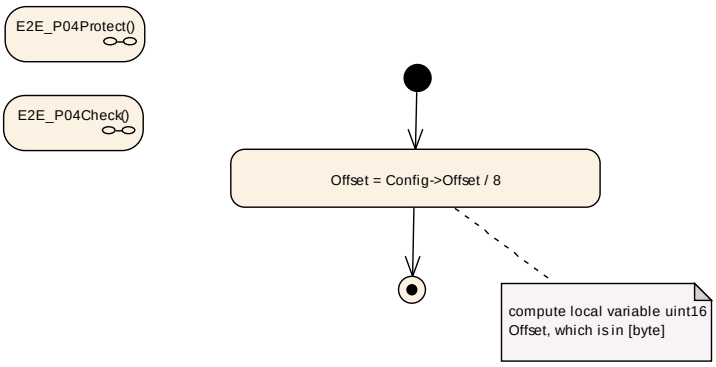

Figure 5.17

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00364] ⌈ The step ”Write Length” in E2E_P04Protect() shall have the following behavior:

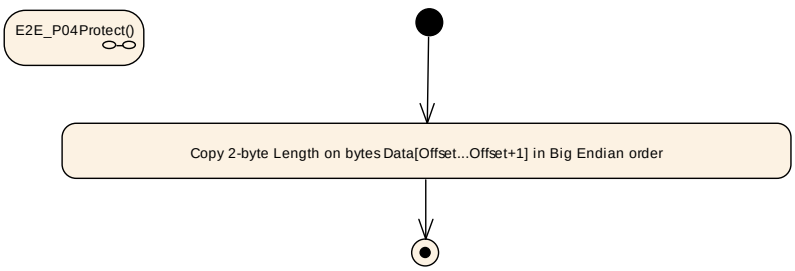

Figure 5.18

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00365] ⌈ The step ”Write Counter” in E2E_P04Protect() shall have the following behavior:

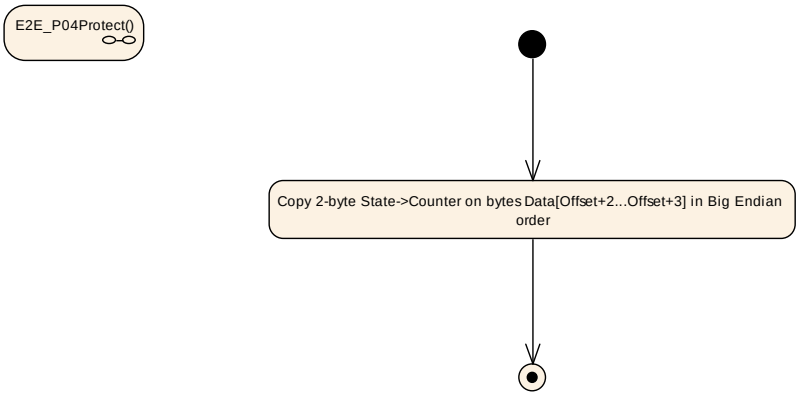

Figure 5.19

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00366] ⌈ The step ”Write DataID”in E2E_P04Protect() shall have the following behavior:

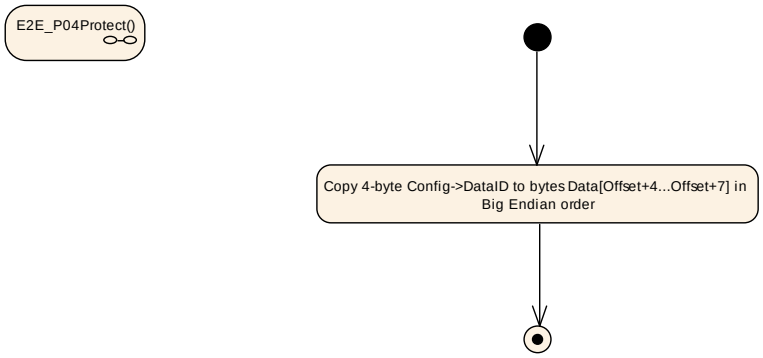

Figure 5.20

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00367] ⌈ The step ”ComputeCRC” in E2E_P04Protect() and in E2E_P04Check() shall have the following behavior:

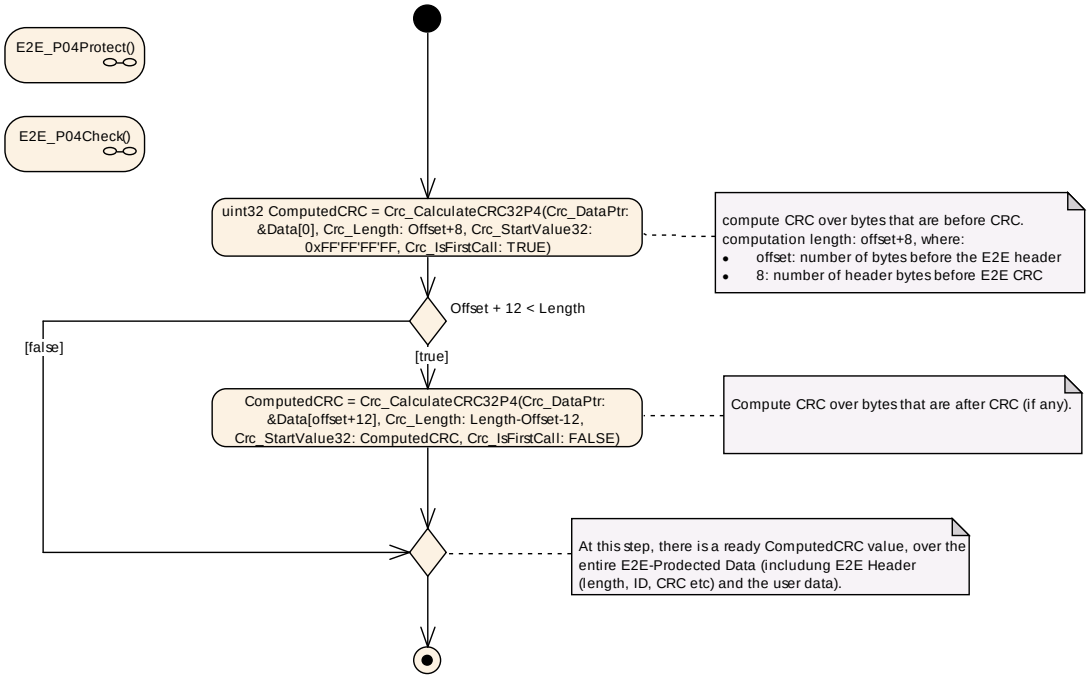

Figure 5.21

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00368] ⌈ The step ”Write CRC”in E2E_P04Protect() shall have the following behavior:

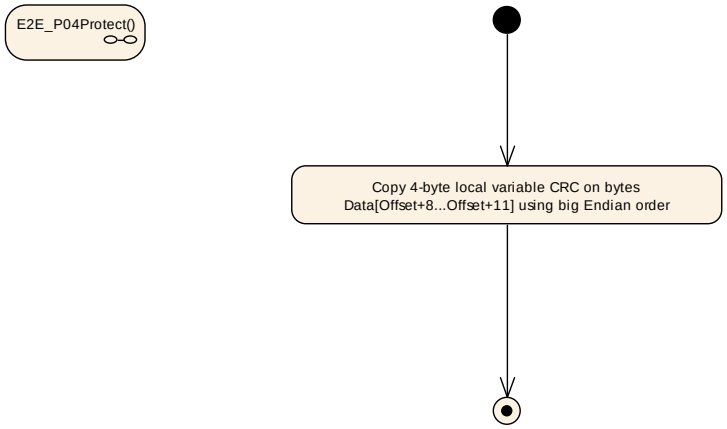

Figure 5.22

⌋(*RS_E2EProtocol_08539*)

[E2E_P04Protect_Increment_Counter] ⌈ The step ”Increment Counter” in E2E_P04Protect() shall have the following behavior:

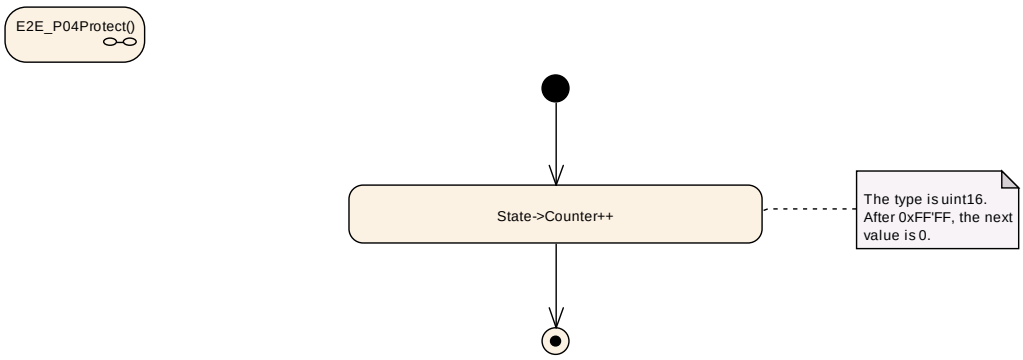

Figure 5.23

⌋(*RS_E2EProtocol_08539*)

5.5.8 E2E_P04Check

The function E2E_P04Check performs the actions as as specified by the following seven diagrams in this section and according to diagram PRS_E2EProtocol_00367.

[PRS_E2EProtocol_00355] ⌈The function E2E_P04Check() shall have the following overall behavior:

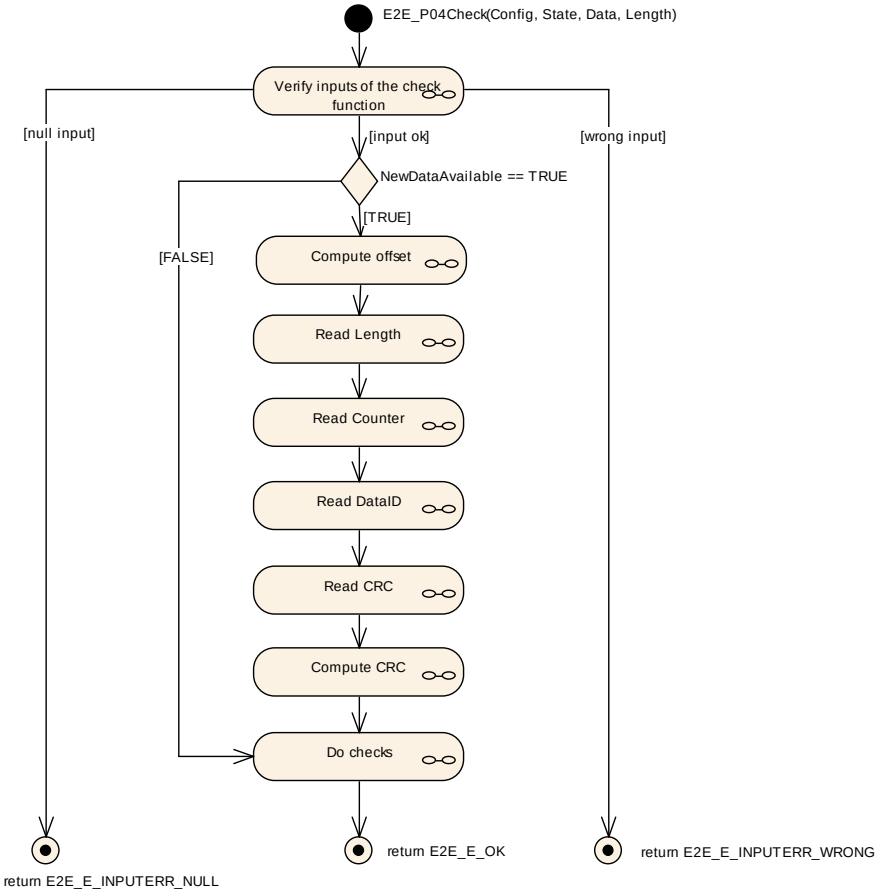

Figure 5.24

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00356] ⌈ The step ”Verify inputs of the check function” in E2E_P04Check() shall have the following behavior:

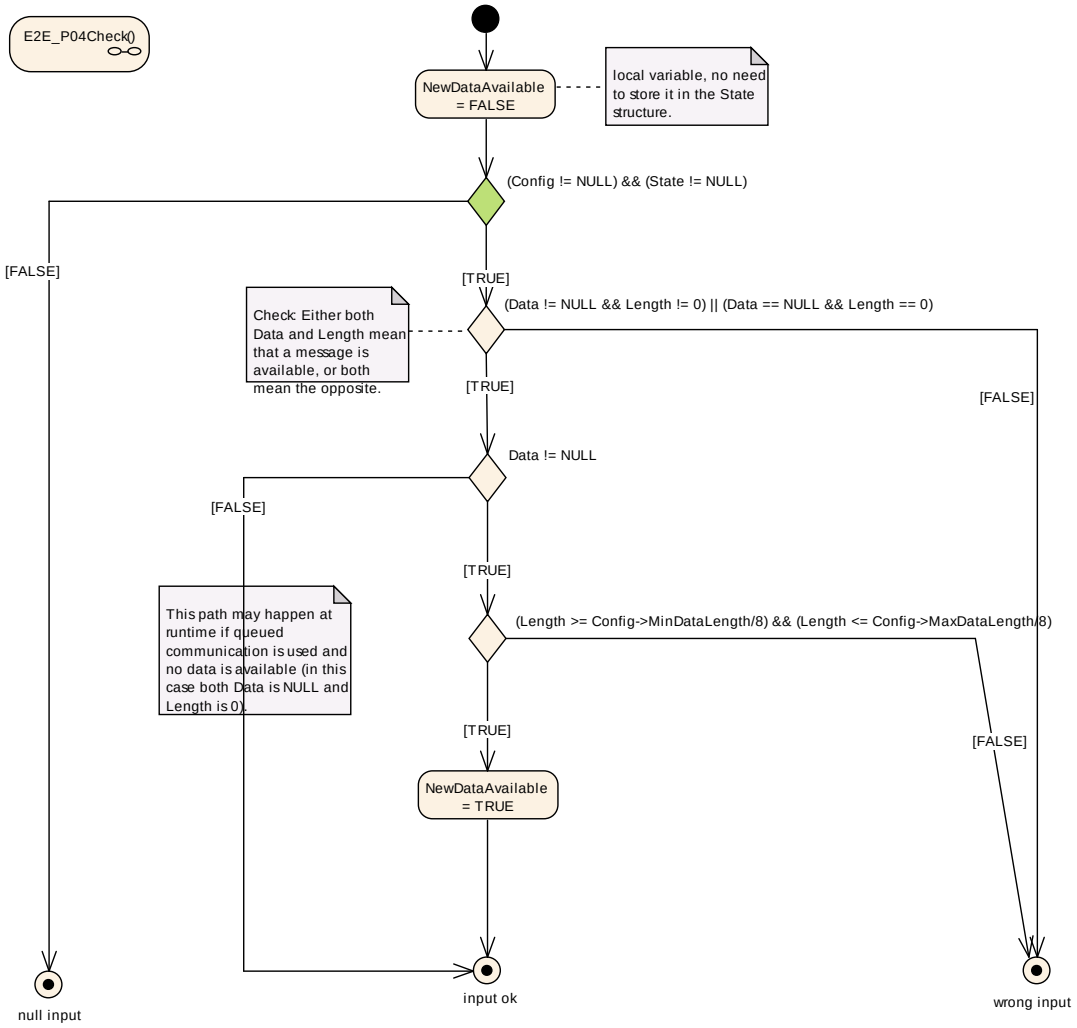

Figure 5.25

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00357] ⌈ The step ”Read Length” in E2E_P04Check() shall have the following behavior:

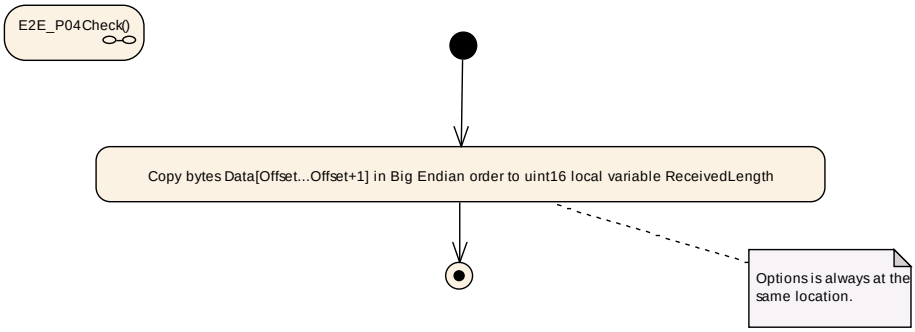

Figure 5.26

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00358] ⌈ The step ”Read Counter” in E2E_P04Check() shall have the following behavior:

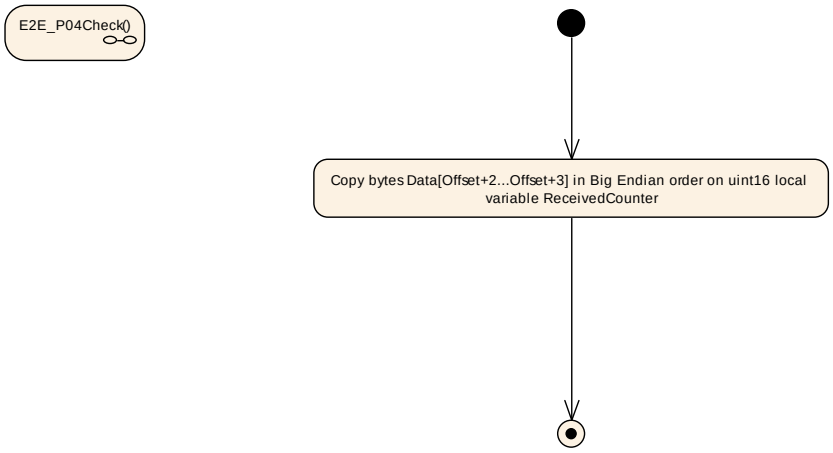

Figure 5.27

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00359] ⌈ The step ”Read DataID ” in E2E_P04Check() shall have the following behavior:

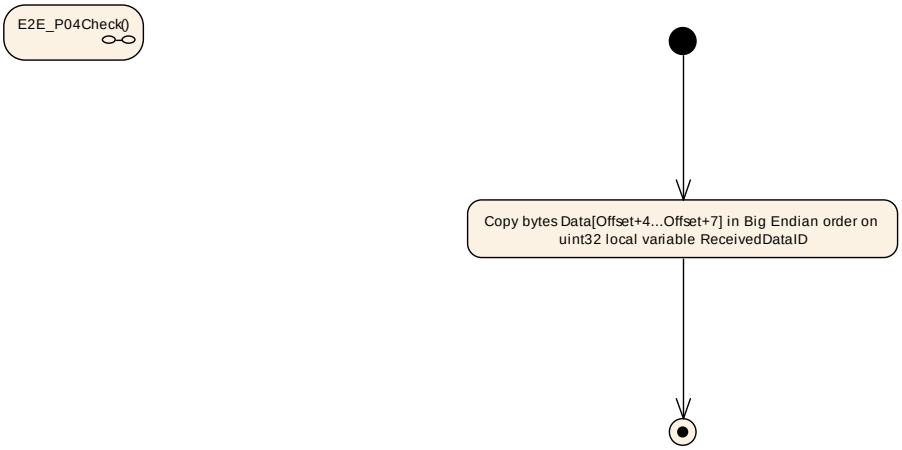

Figure 5.28

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00360] ⌈ The step ”Read CRC” in E2E_P04Check() shall have the following behavior:

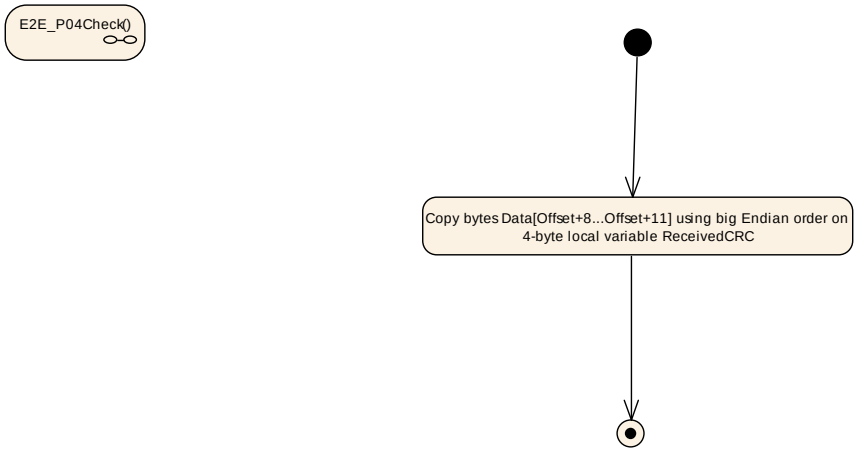

Figure 5.29

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00361] ⌈ The step ”Do Checks” in E2E_P04Check() shall have the following behavior:

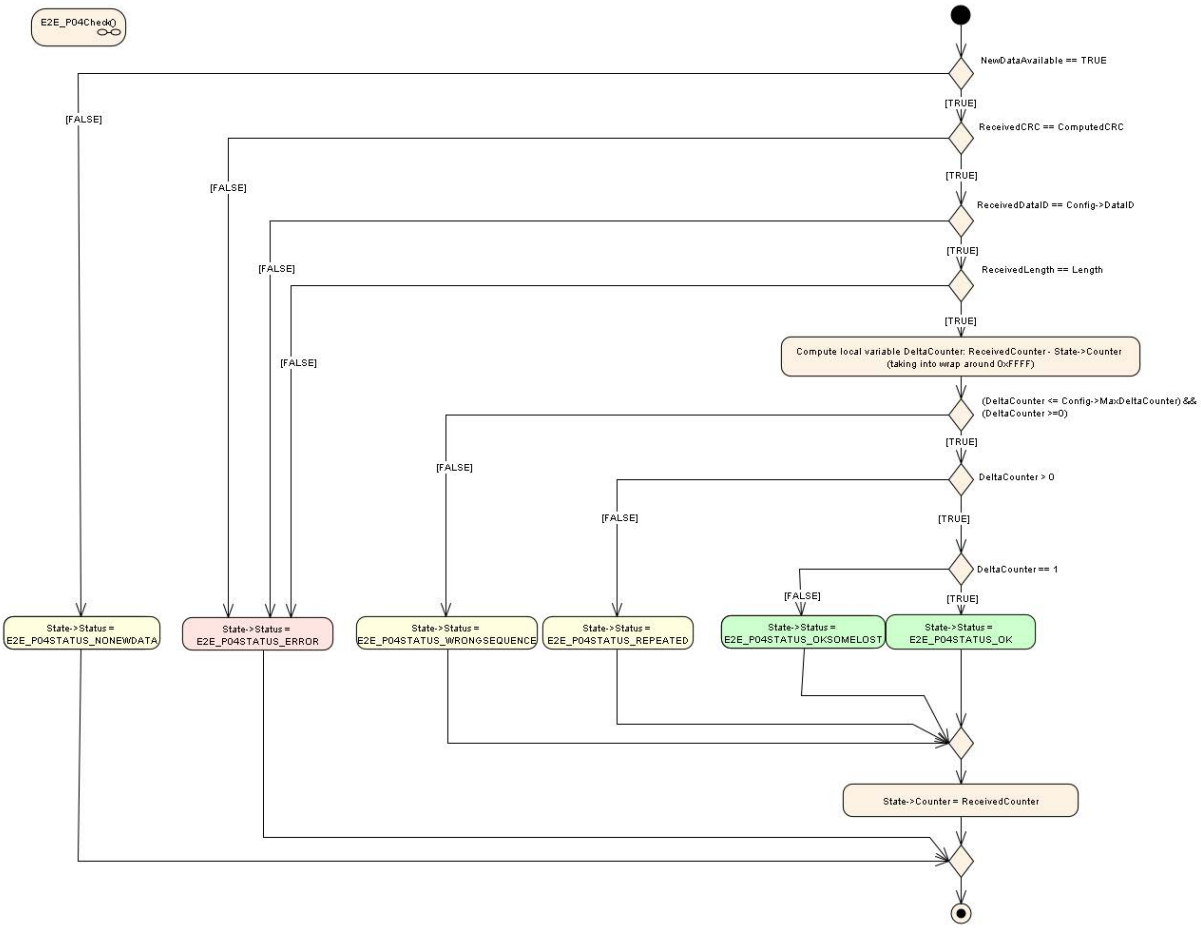

Figure 5.30

⌋(*RS_E2EProtocol_08539*)

5.6 Specification of E2E Profile 5

[PRS_E2EProtocol_00394] ⌈ Profile 5 shall provide the following control fields, transmitted at runtime together with the protected data:

Control field	Description
Counter	8 bits. (explicitly sent)
CRC	16 bits, polynomial in normal form 0x1021 (Autosar notation), provided by CRC library. (explicitly sent)
Data ID	16 bits, unique system-wide. (implicitly sent)

⌋ *(RS_E2EProtocol_08529, RS_E2EProtocol_08530, RS_E2EProtocol_08533)*

The E2E mechanisms can detect the following faults or effects of faults:

Fault	Main safety mechanisms
Repetition of information	Counter
Loss of information	Counter
Delay of information	Counter
Masquerading	Data ID, CRC
Incorrect addressing	Data ID
Incorrect sequence of information	Counter
Corruption of information	CRC
Asymmetric information sent from a sender to multiple receivers	CRC (to detect corruption at any of receivers)
Information from a sender received by only a subset of the receivers	Counter (loss on specific receivers)
Blocking access to a communication channel	Counter (loss or timeout)

Table 5.2: Detectable communication faults using Profile 5

For details of CRC computation, the usage of start values and XOR values see CRC Supervision [7].

5.6.1 Data Layout

5.6.1.1 User data layout

In the E2E Profile 5, the user data layout (of the data to be protected) is not constrained by E2E Profile 5 - there is only a requirement, that the length of data to be protected is multiple of 1 byte.

5.6.1.2 Header layout

The header of the E2E Profile 5 has one fixed layout, as follows:

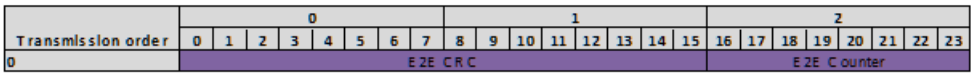

Figure 5.31

The bit numbering shown above represents the order in which bits are transmitted. The E2E header fields (e.g. CRC) are encoded like in CAN and FlexRay, i.e.:

1. Little Endian (least significant byte fist) applicable for both implicit and explicit header fields - imposed by profile
2. MSB Fist (most significant bit within byte first) - imposed by FlexrayCAN bus.

5.6.2 Counter

In E2E Profile 5, the counter is initialized, incremented, reset and checked by E2E profile. The counter is not manipulated or used by the caller of the E2E Supervision.

[PRS_E2EProtocol_00397] ⌈ In E2E Profile 5, on the sender side, for the first transmission request of a data element the counter shall be initialized with 0 and shall be incremented by 1 for every subsequent send request. When the counter reaches the maximum value (0xFF), then it shall restart with 0 for the next send request. ⌋ *(RS_E2EProtocol_08539)*

Note that the counter value 0xFF is not reserved as a special invalid value, but it is used as a normal counter value.

In E2E Profile 5, on the receiver side, by evaluating the counter of received data against the counter of previously received data, the following is detected:

1. Repetition:

   a. no new data has arrived since last invocation of E2E Supervision check function,

   b. the data is repeated

2. OK:

   a. counter is incremented by one (i.e. no data lost),

   b. counter is incremented more than by one, but still within allowed limits (i.e. some data lost),

3. Error: a. counter is incremented more than allowed (i.e. too many data lost).

Case 1 corresponds to the failed alive counter check, and case 3 correspond to failed sequence counter check.

The above requirements are specified in more details by the UML diagrams in the following document sections.

5.6.3 Data ID

The unique Data IDs are to verify the identity of each transmitted safety-related data element.

[PRS_E2EProtocol_00399] ⌈ In the E2E Profile 5, the Data ID shall be implicitly transmitted, by adding the Data ID after the user data in the CRC calculation. ⌋ *(RS_E2EProtocol_08539)*

The Data ID is not a part of the transmitted E2E header (similar to Profile 2 and 6).

[PRS_E2EProtocol_UC_00463] ⌈ In the E2E profile 5, the Data IDs shall be globally unique within the network of communicating system (made of several ECUs each sending different data). ⌋ *(RS_E2EProtocol_08539)*

In case of usage of E2E Supervision for protecting data elements (i.e invocation from RTE), due to multiplicity of communication (1:1 or 1:N), a consumer of a data element expects only a specific data element, which is checked by E2E Supervision using Data ID.

In case of usage of E2E Supervision for protecting I-PDUs (i.e. invocation from COM), the receiver COM expects at a reception only a specific I-PDU, which is checked by E2E Supervision using Data ID.

5.6.4 Length

In Profile 5 there is no explicit transmission of the length.

5.6.5 CRC

E2E Profile 5 uses a 16-bit CRC, to ensure a sufficient detection rate and sufficient Hamming Distance.

[PRS_E2EProtocol_00400] ⌈ E2E Profile 5 shall use the Crc_CalculateCRC16() function of the SWS CRC Supervision for calculating the CRC (Polynomial: 0x1021; Autosar notation). ⌋ *(RS_E2EProtocol_08539, RS_E2EProtocol_08531)*

[PRS_E2EProtocol_00401] ⌈ In E2E Profile 5, the CRC shall be calculated over the entire E2E header (excluding the CRC bytes), including the user data extended at the end with the Data ID. ⌋ *(RS_E2EProtocol_08539, RS_E2EProtocol_08536)*

5.6.6 Timeout detection

The previously mentioned mechanisms (for Profile 5: CRC, Counter, Data ID) enable to check the validity of received data element, when the receiver is executed independently from the data transmission, i.e. when receiver is not blocked waiting for Data Elements or respectively I-PDUs, but instead if the receiver reads the currently available data (i.e. checks if new data is available). Then, by means of the counter, the receiver can detect loss of communication and timeouts.

The attribute State->NewDataAvailable == FALSE means that the transmission medium (e.g RTE) reports that no new data element is available at the transmission medium. The attribute State->Status = E2E_P05STATUS_REPEATED means that the transmission medium (e.g. RTE) provided new valid data element, but this data element has the same counter as the previous valid data element. Both conditions represent an unavailability of valid data that was updated since the previous cycle.

5.6.7 E2E_P05Protect

The function E2E_P05Protect() performs the steps as specified by the following six diagrams in this section.

[PRS_E2EProtocol_00403] ⌈ The function E2E_P05Protect() shall have the following overall behavior:

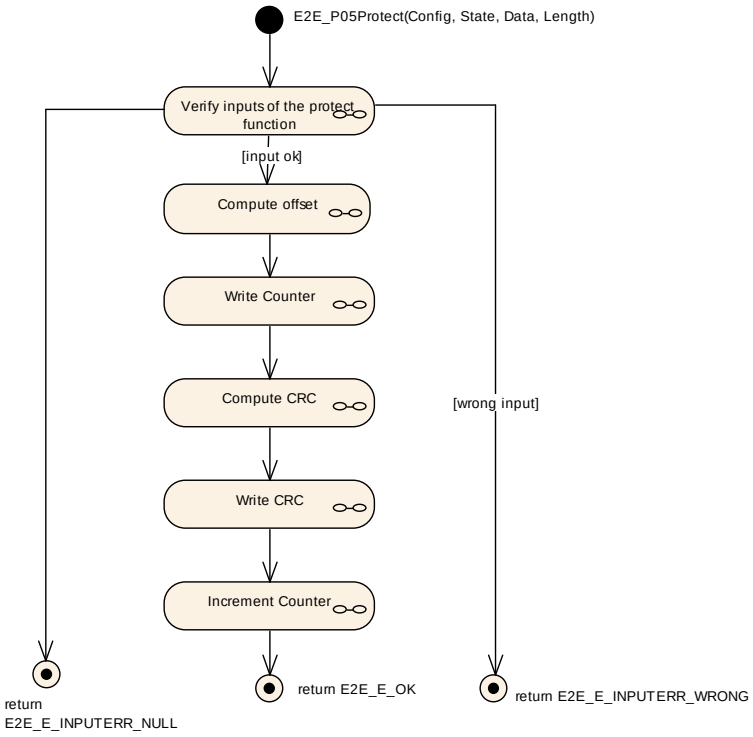

Figure 5.32

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00404] ⌈The step ”Verify inputs of the protect function” in E2E_P05Protect() shall have the following behavior:

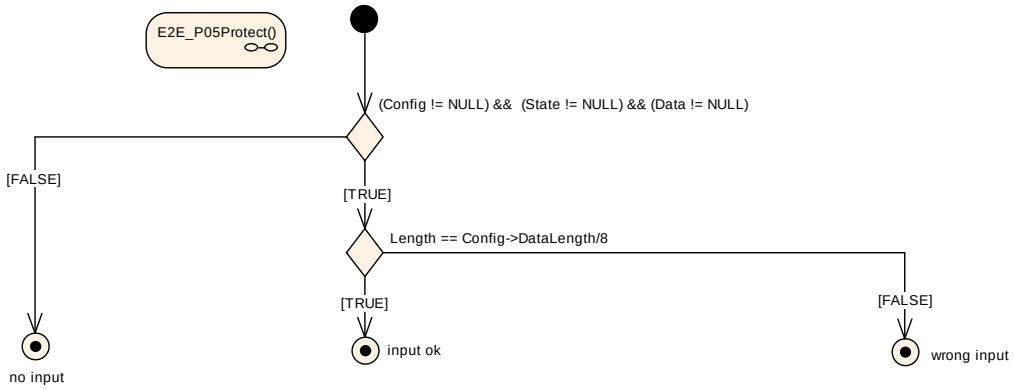

Figure 5.33

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00469] ⌈ The step ”Compute offset” in E2E_P05Protect() and E2E_P05Check() shall have the following behavior:

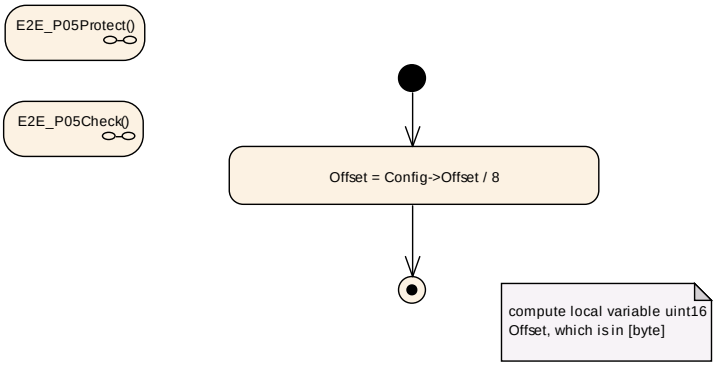

Figure 5.34

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00405] ⌈ The step ”Write Counter” in E2E_P05Protect() shall have the following behavior:

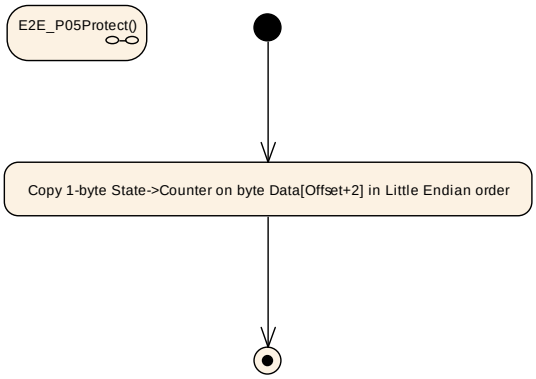

Figure 5.35

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00406] ⌈ The step ”Compute CRC” in E2E_P05Protect() and in E2E_P05Check shall have the following behavior:

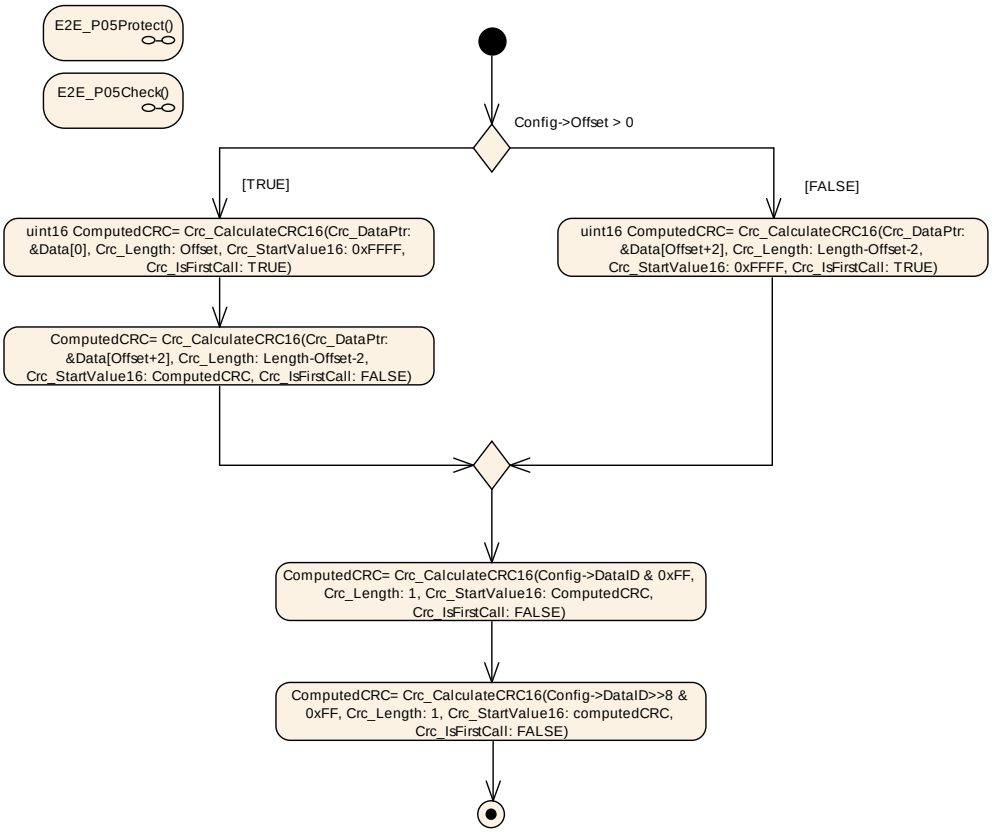

Figure 5.36

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00407] ⌈ The step ”Write CRC” in E2E_P05Protect() shall have the following behavior:

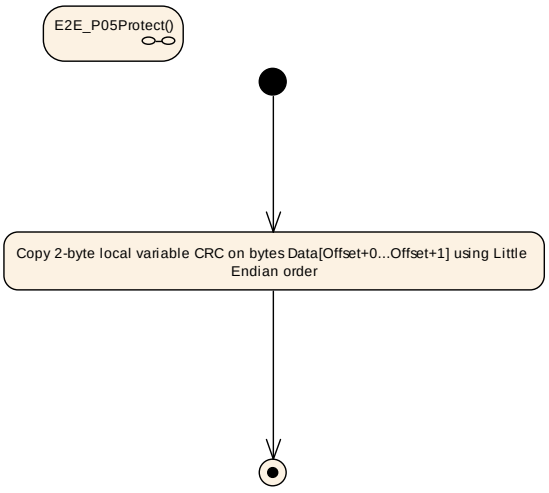

Figure 5.37

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00409] ⌈ The step ”Increment Counter” in E2E_P05Protect() shall have the following behavior:

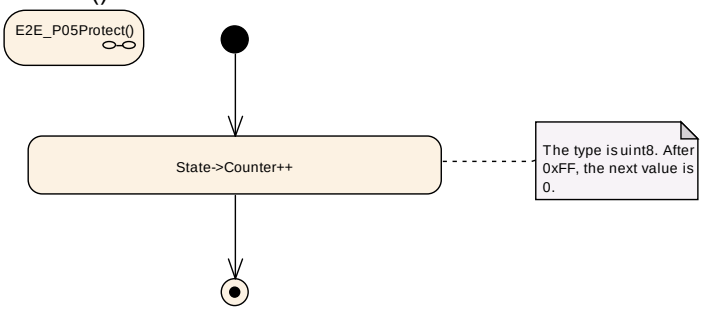

Figure 5.38

⌋(*RS_E2EProtocol_08539*)

5.6.8 E2E_P05Check

The function E2E_P05Check performs the actions as specified by the following six diagrams in this section.

[PRS_E2EProtocol_00411] ⌈ The function E2E_P05Check() shall have the following overall behavior:

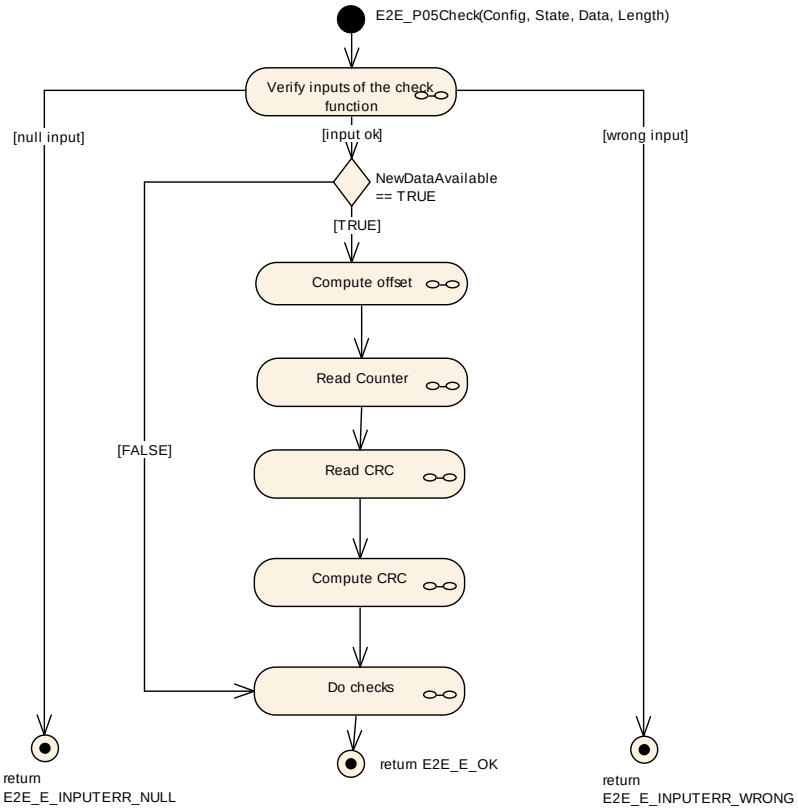

Figure 5.39

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00412] ⌈ The step ”Verify inputs of the check function” in E2E_P05Check() shall have the following behavior:

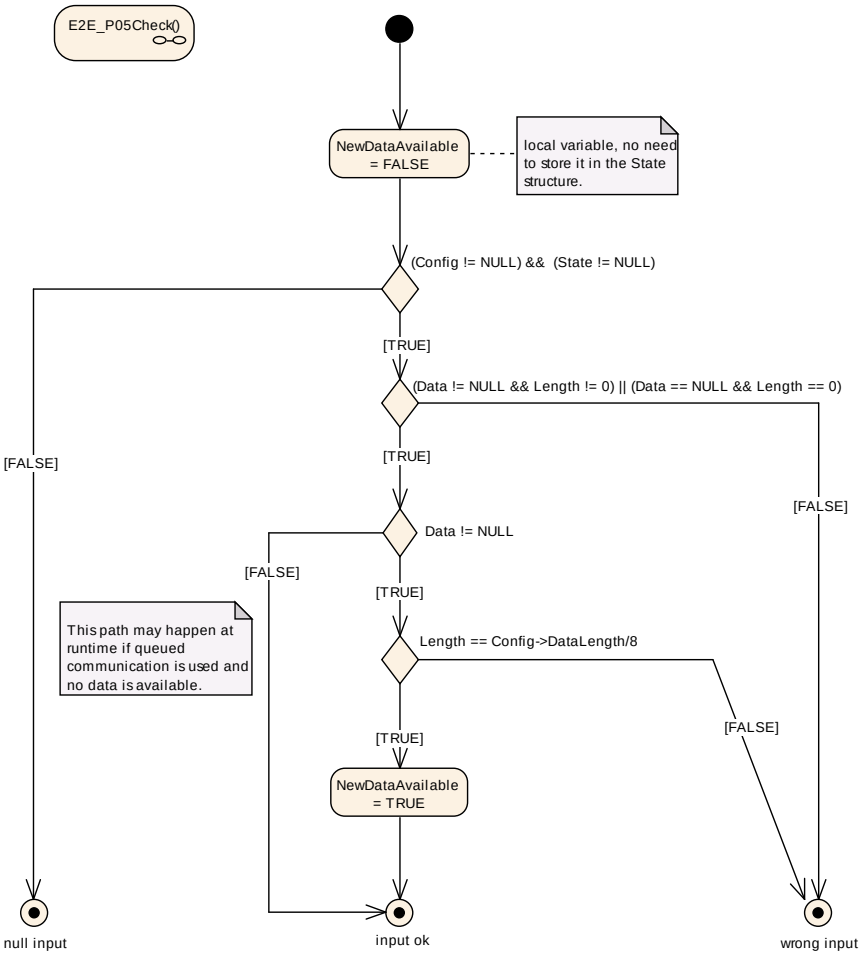

Figure 5.40

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00413] ⌈ The step ”Read Counter” in E2E_P05Check() shall have the following behavior:

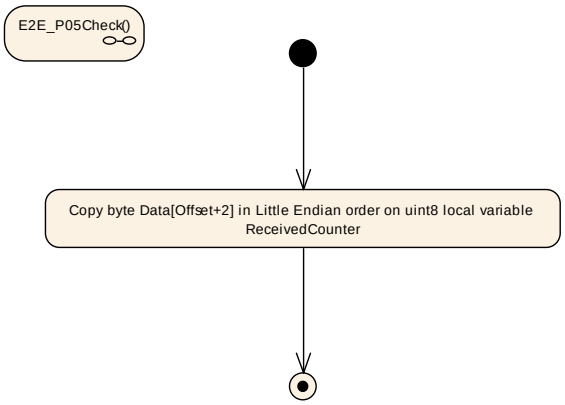

Figure 5.41

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00414] ⌈ The step ”Read CRC” in E2E_P05Check() shall have the following behavior:

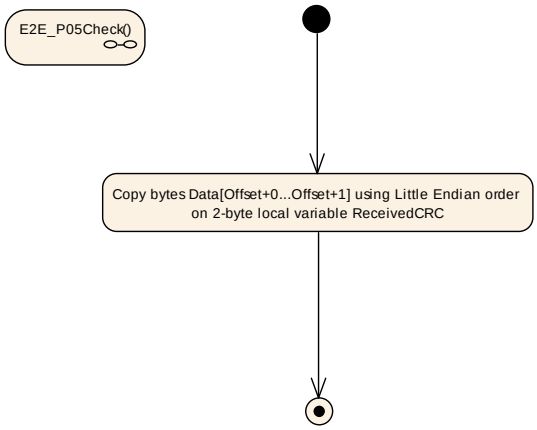

Figure 5.42

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00416] ⌈ The step ”Do Checks’ in E2E_P05Check() shall have the following behavior:

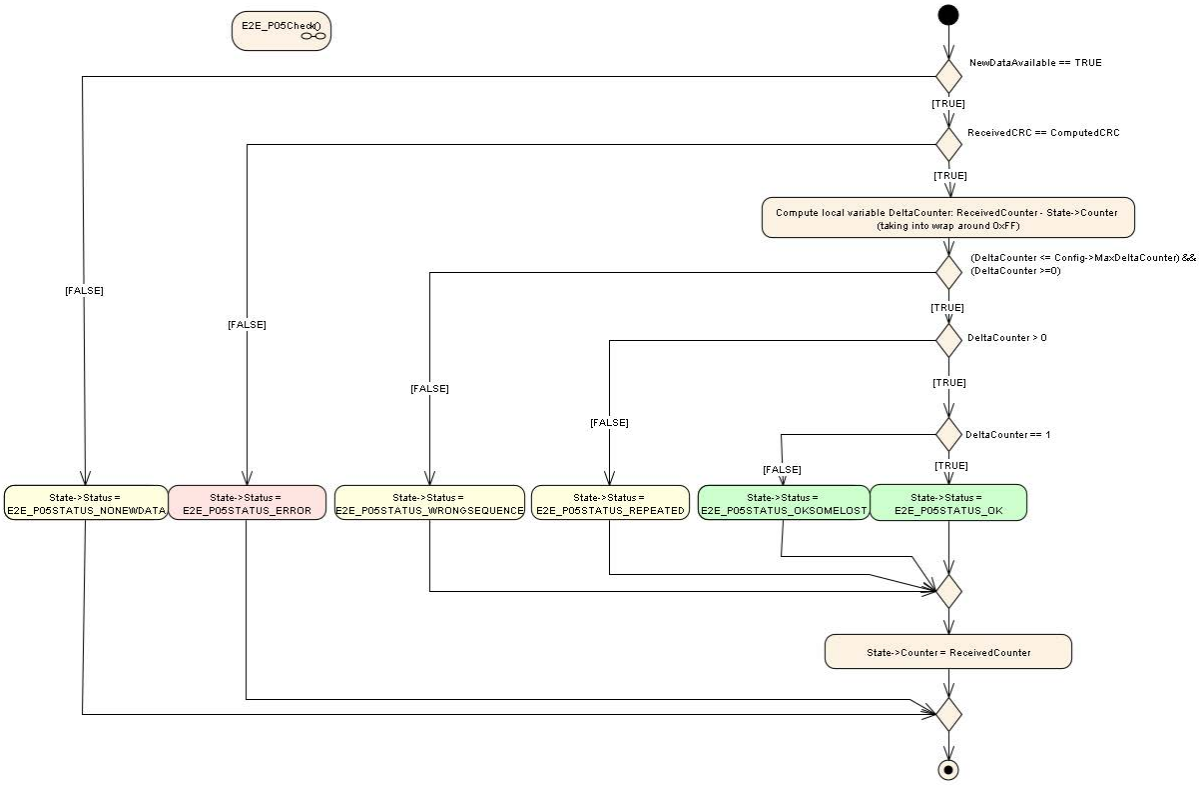

Figure 5.43

⌋(*RS_E2EProtocol_08539*)

5.7 Specification of E2E Profile 6

[PRS_E2EProtocol_00479] ⌈ Profile 6 shall provide the following control fields, transmitted at runtime together with the protected data:

Control field	Description
Length	16 bits, to support dynamic-size data. (explicitly sent)
Counter	8-bits. (explicitly sent)
CRC	16-bits, polynomial in normal form 0x1021 (Autosar notation), provided by CRC library. (explicitly sent)
Data ID	16-bits, unique system-wide. (implicitly sent)

⌋(*RS_E2EProtocol_08529*, *RS_E2EProtocol_08530*, *RS_E2EProtocol_08533*)

The E2E mechanisms can detect the following faults or effects of faults:

Fault	Main safety mechanisms
Repetition of information	Counter
Loss of information	Counter
Delay of information	Counter
Insertion of information	Data ID
Masquerading	Data ID, CRC
Incorrect addressing	Data ID
Incorrect sequence of information	Counter
Corruption of information	CRC
Asymmetric information sent from a sender to multiple receivers	CRC (to detect corruption at any of receivers)
Information from a sender received by only a subset of the receivers	Counter (loss on specific receivers)
Blocking access to a communication channel	Counter (loss or timeout)

Table 5.3: Detectable communication faults using Profile 6

For details of CRC computation, the usage of start values and XOR values see CRC Supervision [7].

5.7.1 Data Layout

5.7.1.1 User data layout

In the E2E Profile 6, the user data layout (of the data to be protected) is not constrained by E2E Profile 6 - there is only a requirement that the length of data to be protected is multiple of 1 byte.

5.7.1.2 Header layout

The header of the E2E Profile 6 has one fixed layout, as follows:

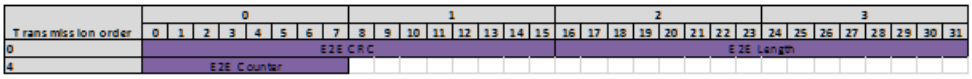

Figure 5.44

The bit numbering shown above represents the order in which bits are transmitted. The E2E header fields (e.g. E2E Counter) are encoded as:

1. Big Endian (most significant byte fist), applicable for both implicit and explicit header fields - imposed by profile
2. LSB Fist (least significant bit within byte first) - imposed by TCP/IP bus

5.7.2 Counter

In E2E Profile 6, the counter is initialized, incremented, reset and checked by E2E profile. The counter is not manipulated or used by the caller of the E2E Supervision.

[PRS_E2EProtocol_00417] ⌈ In E2E Profile 6, on the sender side, for the first transmission request of a data element the counter shall be initialized with 0 and shall be incremented by 1 for every subsequent send request. When the counter reaches the maximum value (0xFF), then it shall restart with 0 for the next send request. ⌋ *(RS_E2EProtocol_08539)*

Note that the counter value 0xFF is not reserved as a special invalid value, but it is used as a normal counter value.

In E2E Profile 6, on the receiver side, by evaluating the counter of received data against the counter of previously received data, the following is detected:

1. Repetition:

   a. no new data has arrived since last invocation of E2E Supervision check function,

   b. the data is repeated

2. OK:

   a. counter is incremented by one (i.e. no data lost),

   b. counter is incremented more than by one, but still within allowed limits (i.e. some data lost),

3. Error: a. counter is incremented more than allowed (i.e. too many data lost).

Case 1 corresponds to the failed alive counter check, and case 3 correspond to failed sequence counter check.

The above requirements are specified in more details by the UML diagrams in the following document sections.

5.7.3 Data ID

The unique Data IDs are to verify the identity of each transmitted safety-related data element.

[PRS_E2EProtocol_00419] ⌈In the E2E Profile 5, the Data ID shall be implicitly transmitted, by adding the Data ID after the user data in the CRC calculation. ⌋ *(RS_E2EProtocol_08539)*

The Data ID is not a part of the transmitted E2E header (similar to Profile 2 and 5).

[PRS_E2EProtocol_UC_00464] ⌈ In the E2E profile 6, the Data IDs shall be globally unique within the network of communicating system (made of several ECUs each sending different data). ⌋*(RS_E2EProtocol_08539)*

In case of usage of E2E Supervision for protecting data elements (i.e invocation from RTE), due to multiplicity of communication (1:1 or 1:N), a consumer of a data element expects only a specific data element, which is checked by E2E Supervision using Data ID.

In case of usage of E2E Supervision for protecting I-PDUs (i.e. invocation from COM), the receiver COM expects at a reception only a specific I-PDU, which is checked by E2E Supervision using Data ID.

5.7.4 Length

In Profile 6 the length field is introduced to support variable-size length - the Data [] array storing the serialized data can potentially have a different length in each cycle. In Profile 6 there is a explicit transmission of the length.

5.7.5 CRC

E2E Profile 6 uses a 16-bit CRC, to ensure a sufficient detection rate and sufficient Hamming Distance.

[PRS_E2EProtocol_00420] ⌈ E2E Profile 6 shall use the Crc_CalculateCRC16() function of the SWS CRC Supervision for calculating the CRC (Polynomial: 0x1021; Autosar notation). ⌋*(RS_E2EProtocol_08539, RS_E2EProtocol_08531)*

[PRS_E2EProtocol_00421] ⌈ In E2E Profile 6, the CRC shall be calculated over the entire E2E header (excluding the CRC bytes), including the user data extended with the Data ID. ⌋*(RS_E2EProtocol_08539, RS_E2EProtocol_08536)*

5.7.6 Timeout detection

The previously mentioned mechanisms (for Profile 6: CRC, Counter, Data ID, Length) enable to check the validity of received data element, when the receiver is executed independently from the data transmission, i.e. when receiver is not blocked waiting for Data Elements or respectively I-PDUs, but instead if the receiver reads the currently available data (i.e. checks if new data is available). Then, by means of the counter, the receiver can detect loss of communication and timeouts.

The attribute State->NewDataAvailable == FALSE means that the transmission medium (e.g RTE) reports that no new data element is available at the transmission medium. The attribute State->Status = E2E_P06STATUS_REPEATED means that the transmission medium (e.g. RTE) provided new valid data element, but this data element has the same counter as the previous valid data element. Both conditions represent an unavailability of valid data that was updated since the previous cycle.

5.7.7 E2E_P06Protect

The function E2E_P06Protect() performs the steps as specified by the following seven diagrams in this section.

[PRS_E2EProtocol_00423] ⌈ The function E2E_P06Protect() shall have the following overall behavior:

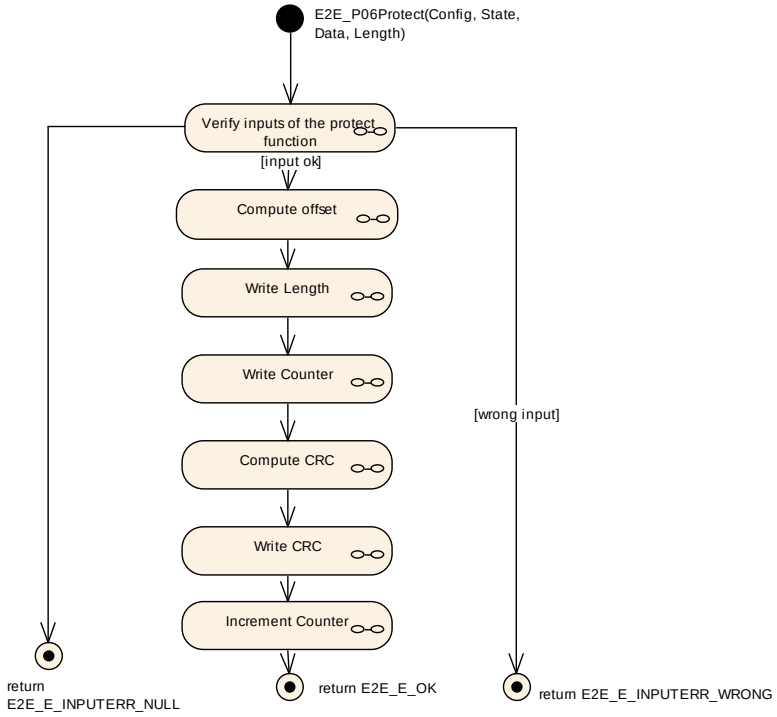

Figure 5.45

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00424] ⌈ The step ”Verify inputs of the protect function” in E2E_P06Protect() shall have the following behavior:

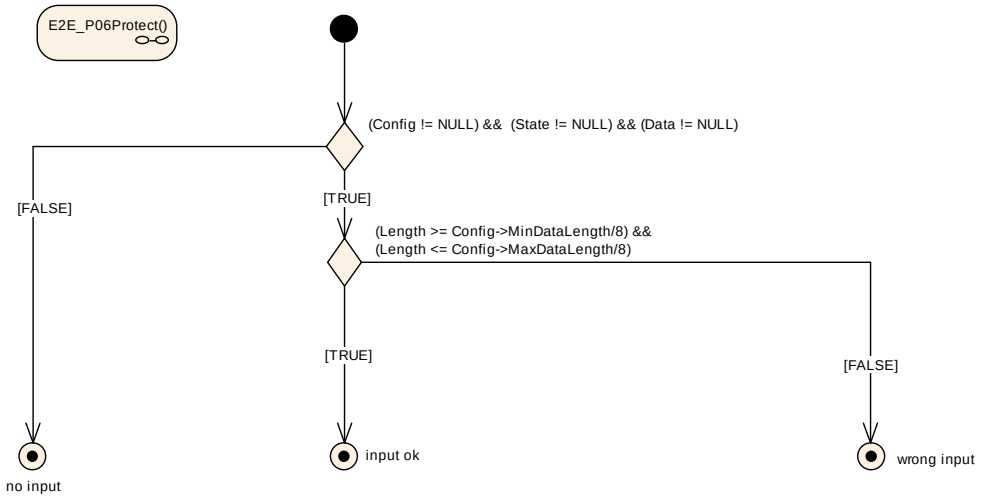

Figure 5.46

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00470] ⌈ The step ”Compute offset” in E2E_P06Protect() and E2E_P06Check() shall have the following behavior:

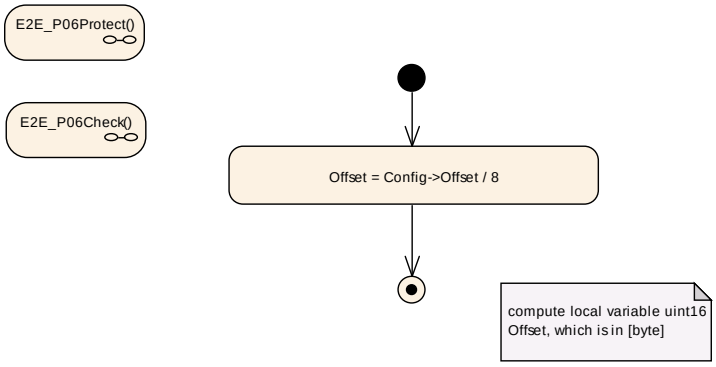

Figure 5.47

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00425] ⌈ The step ”Write Length” in E2E_P06Protect() shall have the following behavior:

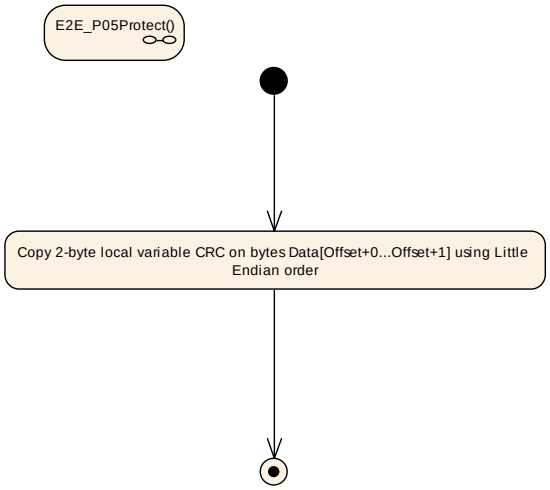

Figure 5.48

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00426] ⌈ The step ”Write Counter” in E2E_P06Protect() shall have the following behavior:

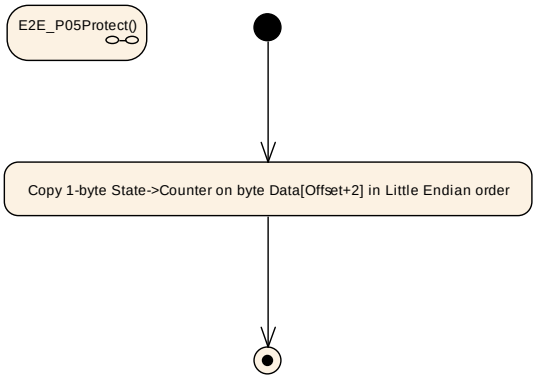

Figure 5.49

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00427] ⌈ The step ”Compute CRC” in E2E_P06Protect() and E2E_P06Check() shall have the following behavior:

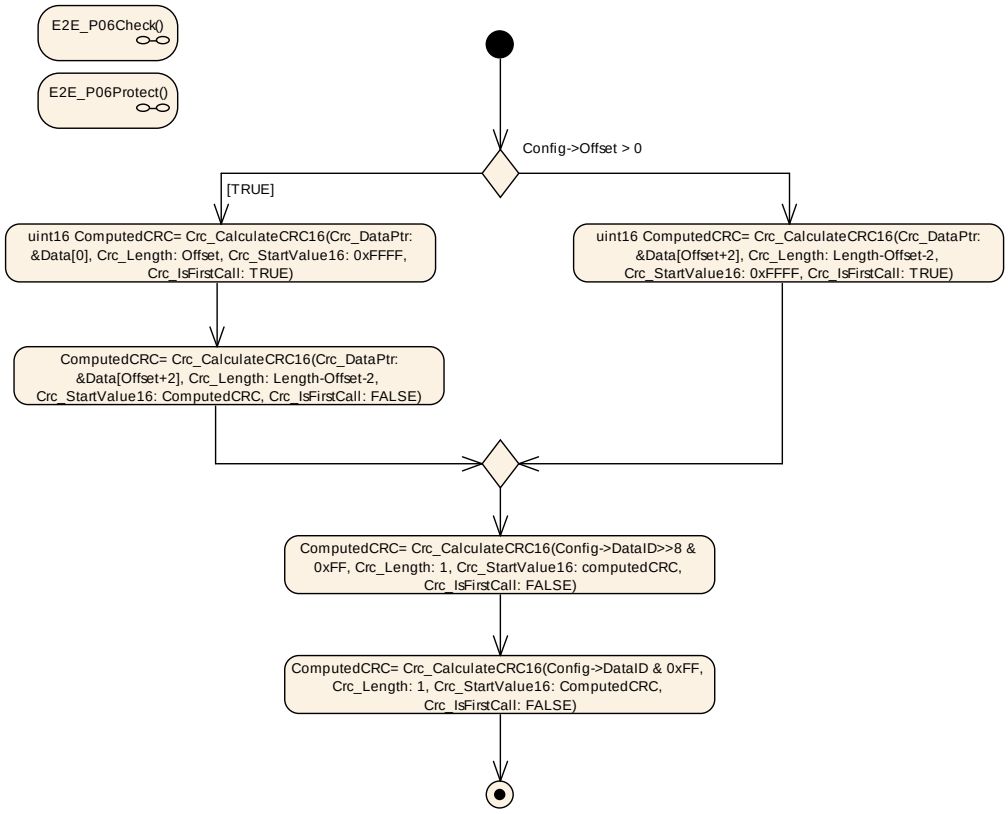

Figure 5.50

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00428] ⌈ The step ”Write CRC” in E2E_P06Protect() shall have the following behavior:

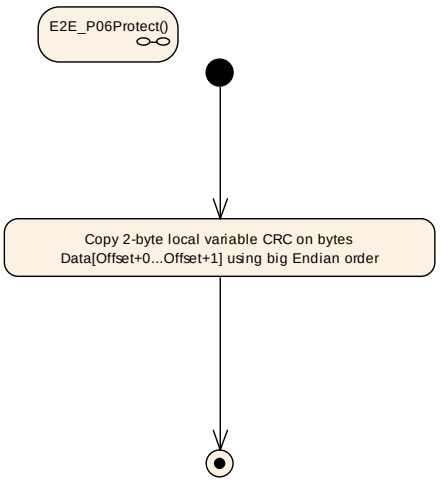

Figure 5.51

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00429] ⌈ The step ”Increment Counter” in E2E_P06Protect() shall have the following behavior:

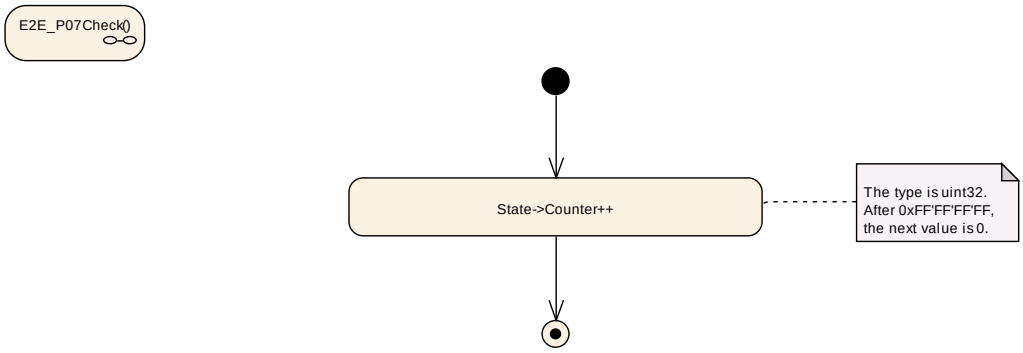

Figure 5.52

⌋(*RS_E2EProtocol_08539*)

5.7.8 E2E_P06Check

The function E2E_P06Check performs the actions as specified by the following seven diagrams in this section.

[PRS_E2EProtocol_00430] ⌈The function E2E_P06Check() shall have the following overall behavior:

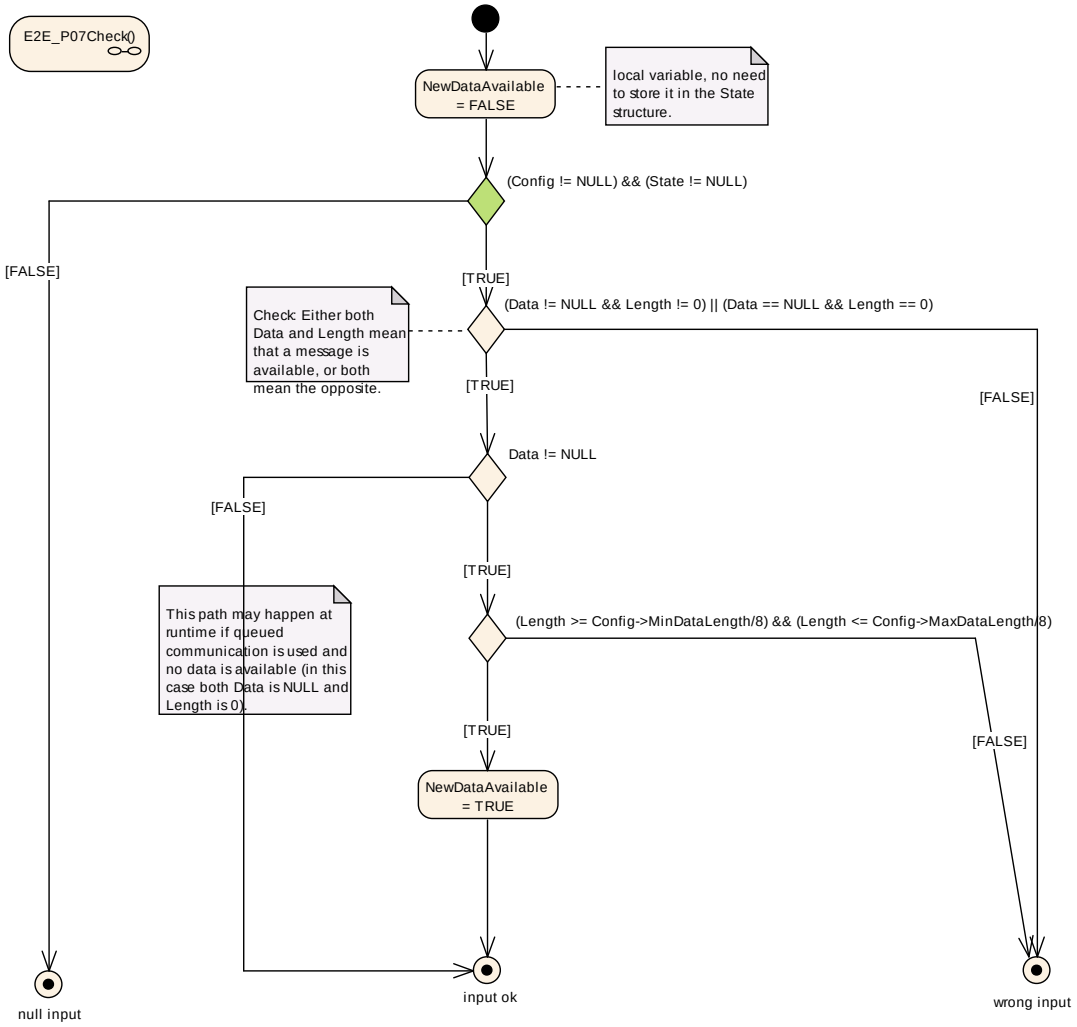

Figure 5.53

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00432] ⌈ The step ”Read Length” in E2E_P06Check() shall have the following behavior:

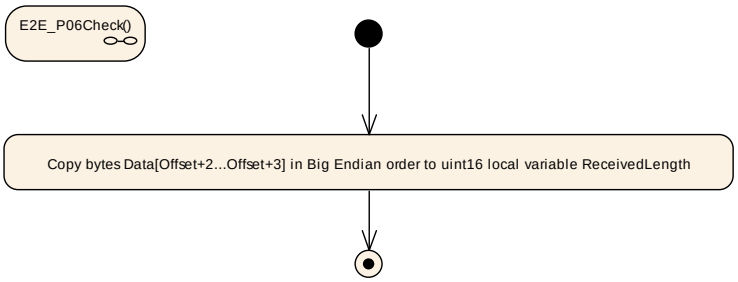

Figure 5.54

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00434] ⌈ The step ”Read CRC” in E2E_P06Check() shall have the following behavior:

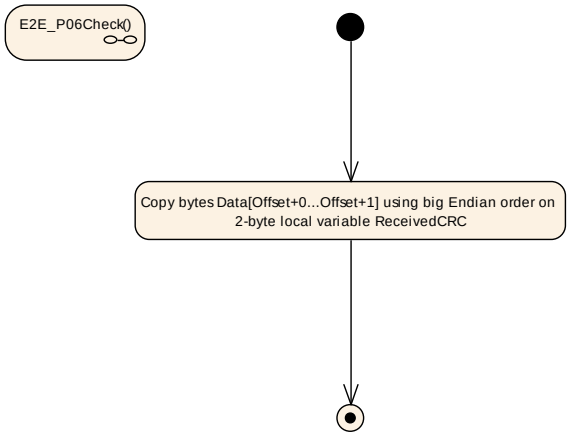

Figure 5.55

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00436] ⌈ The step ”Do Checks” in E2E_P06Check() shall have the following behavior:

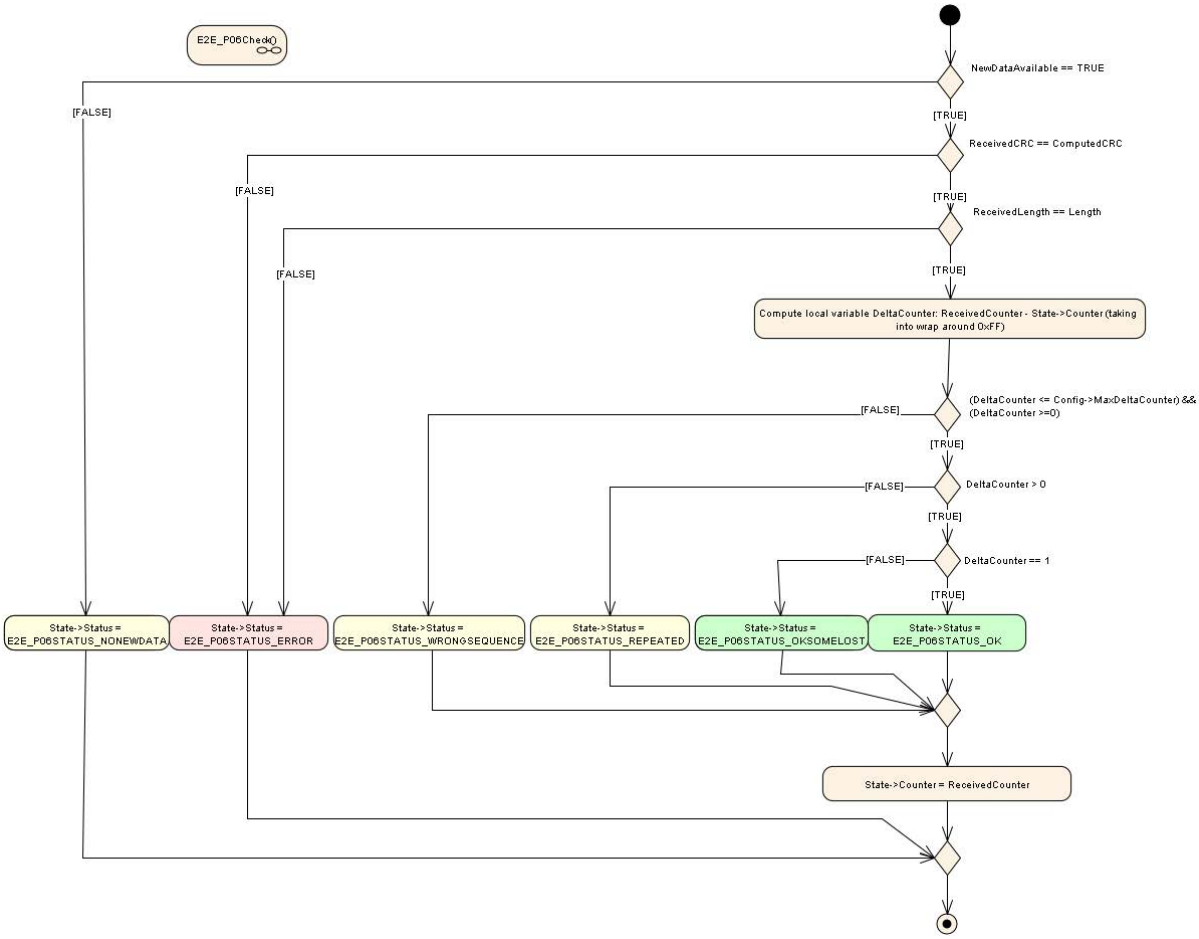

Figure 5.56

⌋(*RS_E2EProtocol_08539*)

5.8 Specification of E2E Profile 7

[PRS_E2EProtocol_00480] ⌈ Profile 7 shall provide the following control fields, transmitted at runtime together with the protected data:

Control field	Description
Length	32 bits, to support dynamic-size data.
Counter	32 bits.
CRC	64 bits, polynomial in normal form 0x42F0E1EBA9EA3693, provided by CRC library. Note: This CRC polynomial is also known as “CRC-64 (ECMA)”.
Data ID	32 bits, unique system-wide.

⌋(*RS_E2EProtocol_08529*, *RS_E2EProtocol_08530*, *RS_E2EProtocol_08533*)

The E2E mechanisms can detect the following faults or effects of faults:

Fault	Main safety mechanisms
Repetition of information	Counter
Loss of information	Counter
Delay of information	Counter
Insertion of information	Data ID, CRC
Masquerading	Data ID, CRC
Incorrect addressing	Data ID
Incorrect sequence of information	Counter
Corruption of information	CRC
Asymmetric information sent from a sender to multiple receivers	CRC (to detect corruption at any of receivers)
Information from a sender received by only a subset of the receivers	Counter (loss on specific receivers)
Blocking access to a communication channel	Counter (loss or timeout)

For details of CRC computation, the usage of start values and XOR values see CRC Supervision [7]

5.8.1 Data Layout

5.8.1.1 User data layout

In the E2E Profile 7, the user data layout (of the data to be protected) is not constrained by E2E Profile 7 - there is only a requirement that the length of data to be protected is multiple of 1 byte.

5.8.1.2 Header layout

The header of the E2E Profile 7 has one fixed layout, as follows:

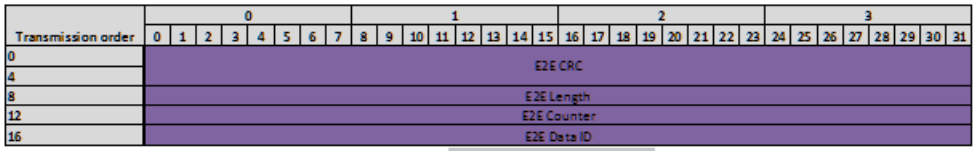

Figure 5.57: Profile 7 Header

The bit numbering shown above represents the order in which bits are transmitted. The E2E header fields (e.g. E2E Counter) are encoded as:

1. Big Endian (most significant byte fist) - imposed by profile
2. LSB Fist (least significant bit within byte first) - imposed by TCPIP bus

For example, the 32 bits of the E2E counter are transmitted in the following order (higher number meaning higher significance): 24 25 26 27 28 29 30 31 16 17 18 19 20 21 22 23 7 8 9 10 11 12 13 14 15 0 1 2 3 4 5 6 7.

The header can be placed at a specific location in the protected data, by configuring the offset of the entire E2E header.

5.8.2 Counter

In E2E Profile 7, the counter is initialized, incremented, reset and checked by E2E profile. The counter is not manipulated or used by the caller of the E2E Supervision.

[PRS_E2EProtocol_00481] ⌈In E2E Profile 7, on the sender side, for the first transmission request of a data element the counter shall be initialized with 0 and shall be incremented by 1 for every subsequent send request. When the counter reaches the maximum value (0xFF'FF'FF'FF), then it shall restart with 0 for the next send request. ⌋*(RS_E2EProtocol_08539)*

Note that the counter value 0xFF'FF'FF'FF is not reseved as a special invalid value, but it is used as a normal counter value.

In E2E Profile 7, on the receiver side, by evaluating the counter of received data against the counter of previously received data, the following is detected:

1. Repetition:

   a. no new data has arrived since last invocation of E2E Supervision check function, b. the data is repeated

2. OK: a. counter is incremented by one (i.e. no data lost), b. counter is incremented more than by one, but still within allowed limits (i.e. some data lost),

3. Wrong sequence: a. counter is incremented more than allowed (i.e. too many data lost).

Case 1 corresponds to the failed alive counter check, and case 3 correspond to failed sequence counter check.

The above requirements are specified in more details by the UML diagrams in the following document sections.

5.8.3 Data ID

The unique Data IDs are to verify the identity of each transmitted safety-related data element.

[PRS_E2EProtocol_00482] ⌈ In the E2E Profile 7, the Data ID shall be explicitly transmitted, i.e. it shall be the part of the transmitted E2E header ⌋*(RS_E2EProtocol_08539)*

There are currently no limitations on the values of Data ID - any values within the addres space of 32 bits are allowed.

[PRS_E2EProtocol_00483] ⌈ In the E2E profile 7, the Data IDs shall be globally unique within the network of communicating system (made of several ECUs each sending different data). ⌋*(RS_E2EProtocol_08539)*

In case of usage of E2E Supervision for protecting data elements (i.e invocation from RTE), due to multiplicity of communication (1:1 or 1:N), a consumer of a data element expects only a specific data element, which is checked by E2E Supervision using Data ID.

In case of usage of E2E Supervision for protecting I-PDUs (i.e. invocation from COM), the receiver COM espects at a reception only a specific I-PDU, which is checked by E2E Supervision using Data ID.

5.8.4 Length

The Length field is introduced to support variable-size length - the Data [] array storing the serialized data can potentially have a different length in each cycle.

5.8.5 CRC

E2E Profile 7 uses a 64-bit CRC, to ensure a high detection rate and high Hamming Distance.

[PRS_E2EProtocol_00484] ⌈E2E Profile 7 shall use the Crc_CalculateCRC64 4 () function of the SWS CRC Supervision for calculating the CRC. ⌋ *(RS_E2EProtocol_08539, RS_E2EProtocol_08531)*

[PRS_E2EProtocol_00485] ⌈ In E2E Profile 7, the CRC shall be calculated over the entire E2E header (excluding the CRC bytes) and over the user data. ⌋ *(RS_E2EProtocol_08536)*

5.8.6 Timeout detection

The previously mentioned mechanisms (CRC, Counter, Data ID, Length) enable to check the validity of received data element, when the receiver is executed independently from the data transmission, i.e. when receiver is not blocked waiting for Data Elements or respectively I-PDUs, but instead if the receiver reads the currently available data (i.e. checks if new data is available). Then, by means of the counter, the receiver can detect loss of communication and timeouts.

5.8.7 E2E Profile 7 variants

The E2E Profile 7 variants are specified in TPS System Specification.

5.8.8 E2E_P07Protect

The function E2E_P07Protect() performs the steps as specified by the following eight diagrams in this section.

[PRS_E2EProtocol_00486] ⌈ The function E2E_P07Protect() shall have the following overall behavior:

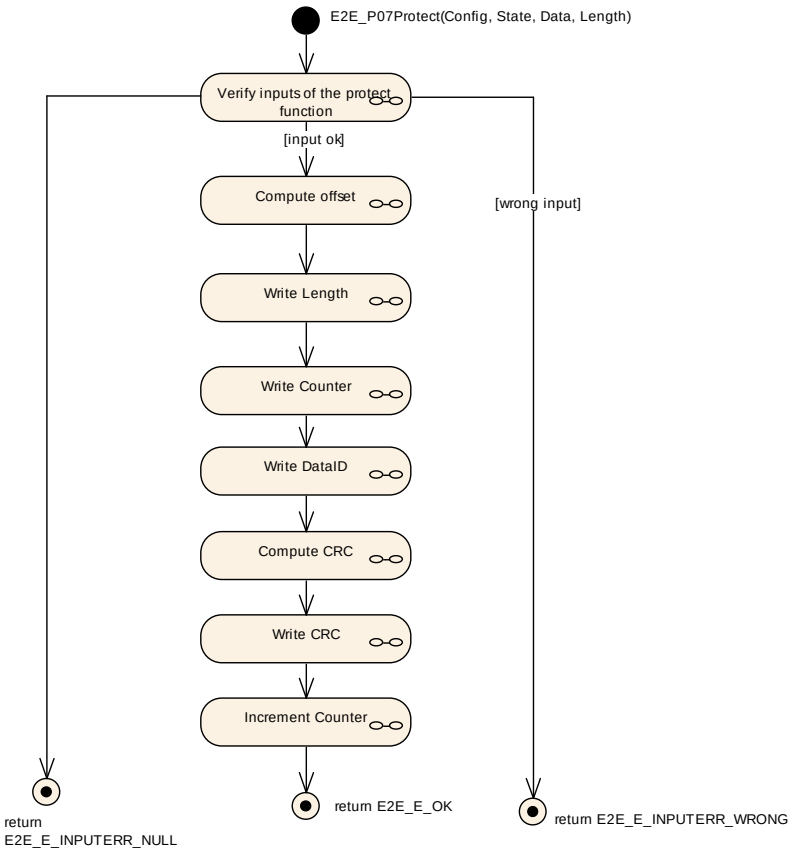

Figure 5.58

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00487] ⌈ The step ”Verify inputs of the protect function” in E2E_P07Protect() shall have the following behavior:

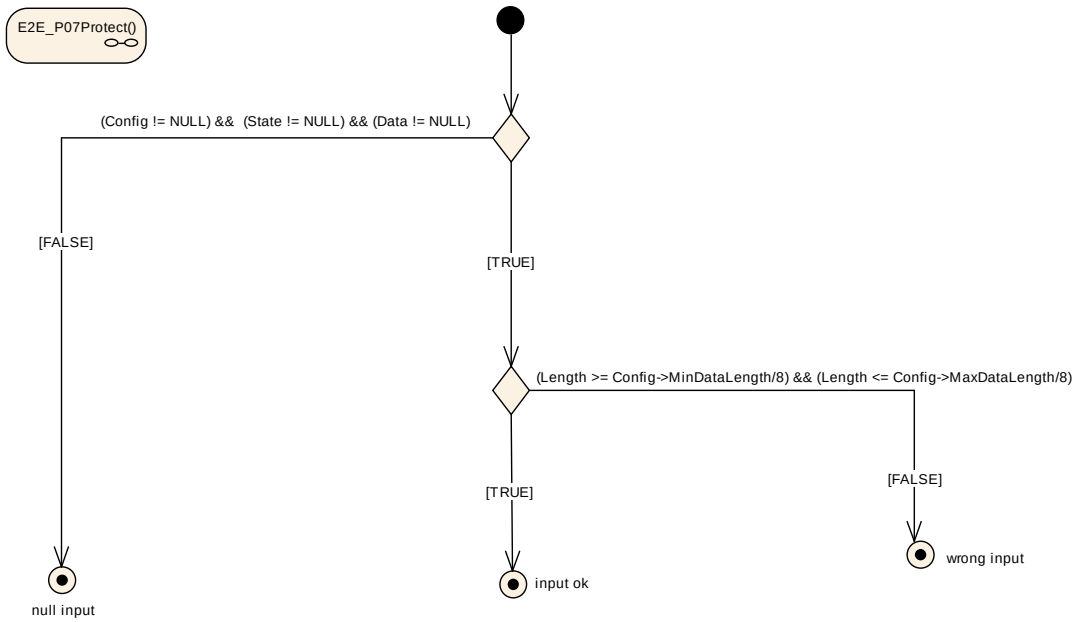

Figure 5.59

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00488] ⌈The step ”Compute offset” in E2E_P07Protect() and E2E_P07Check() shall have the following behavior:

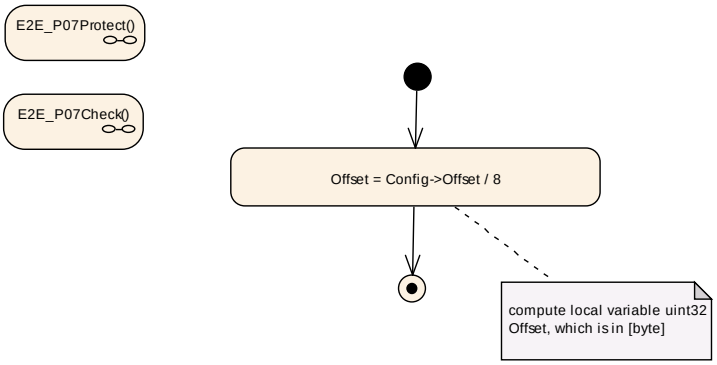

Figure 5.60

⌋()

[PRS_E2EProtocol_00489] ⌈ The step ”Write Length” in E2E_P07Protect() shall have the following behavior:

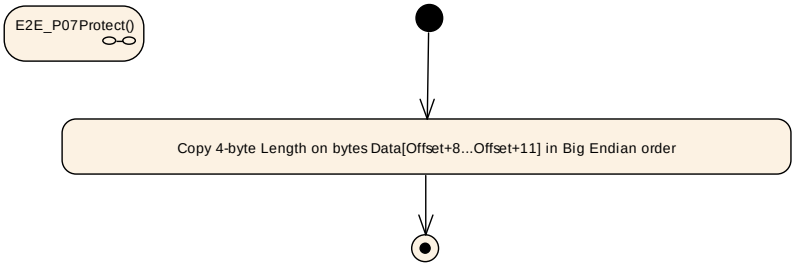

Figure 5.61

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00490] ⌈ The step ”Write Counter” in E2E_P07Protect() shall have the following behavior:

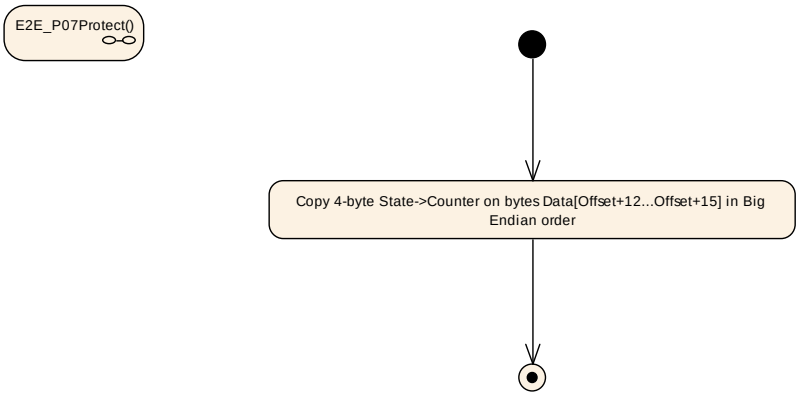

Figure 5.62

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00491] ⌈ The step ”Write DataID”in E2E_P07Protect() shall have the following behavior:

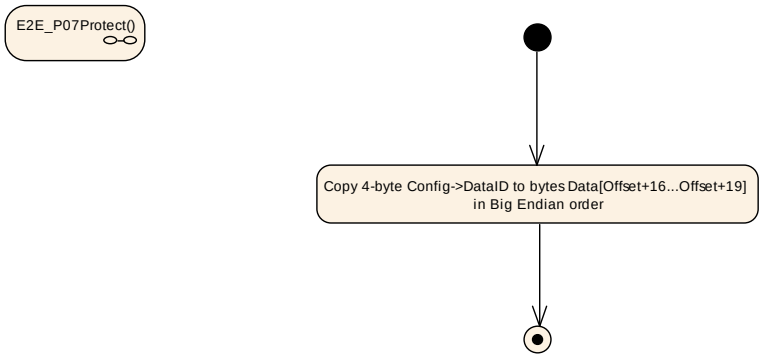

Figure 5.63

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00492] ⌈ The step ”ComputeCRC” in E2E_P07Protect() and in E2E_P07Check() shall have the following behavior:

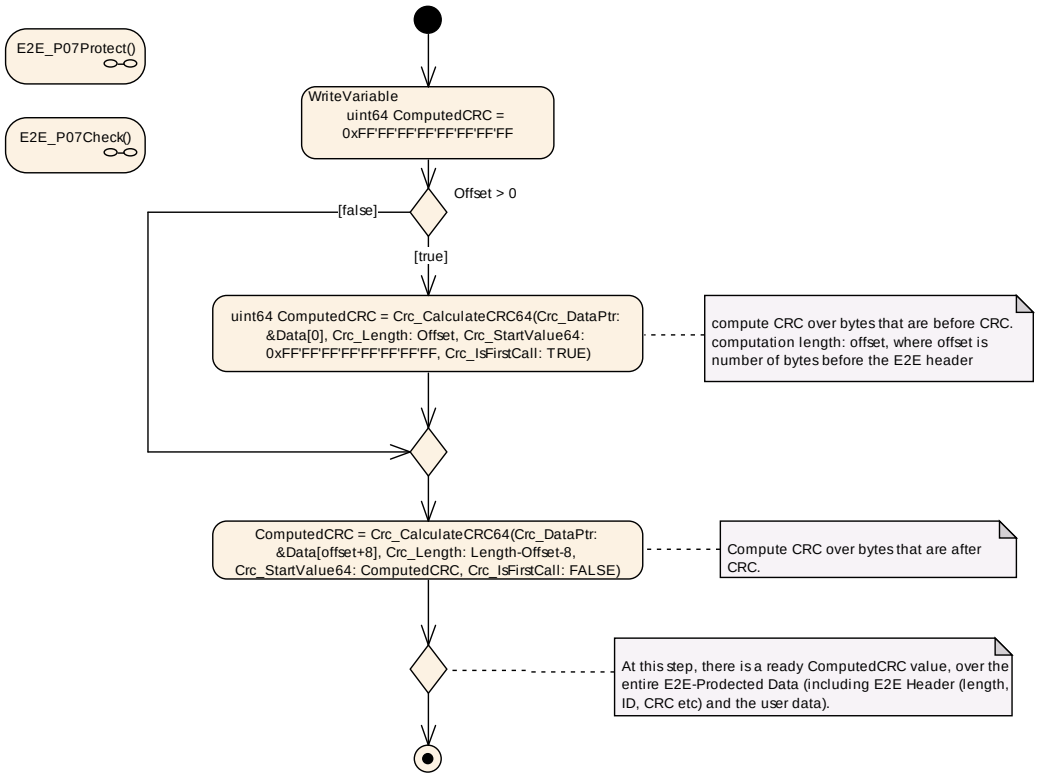

Figure 5.64

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00493] ⌈ The step ”Write CRC”in E2E_P07Protect() shall have the following behavior:

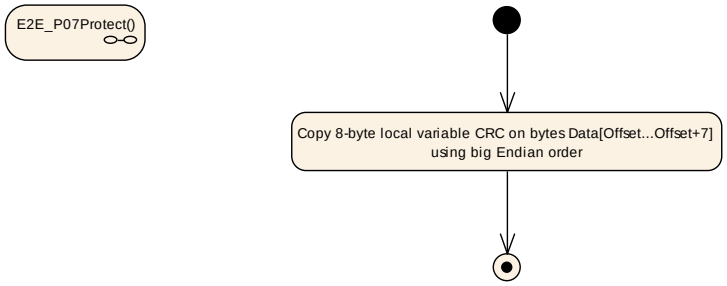

Figure 5.65

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00494] ⌈ The step ”Increment Counter” in E2E_P07Protect() shall have the following behavior:

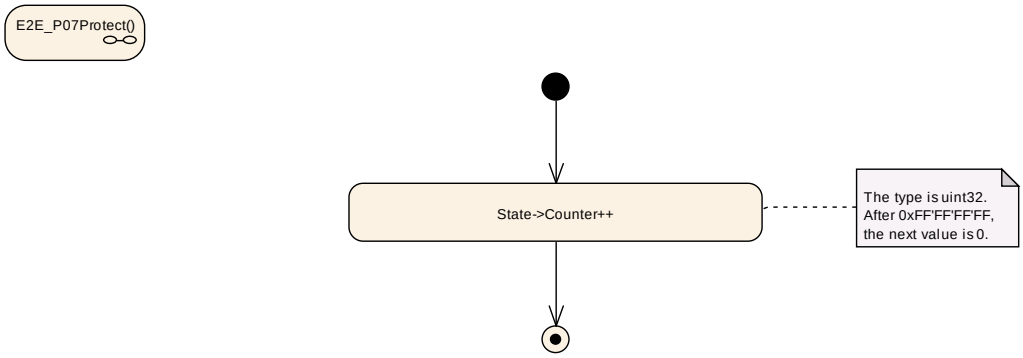

Figure 5.66

⌋(*RS_E2EProtocol_08539*)

5.8.9 E2E_P07Check

The function E2E_P07Check performs the actions as as specified by the following seven diagrams in this section and according to diagram PRS_E2EProtocol_00492.

[PRS_E2EProtocol_00495] ⌈The function E2E_P07Check() shall have the following overall behavior:

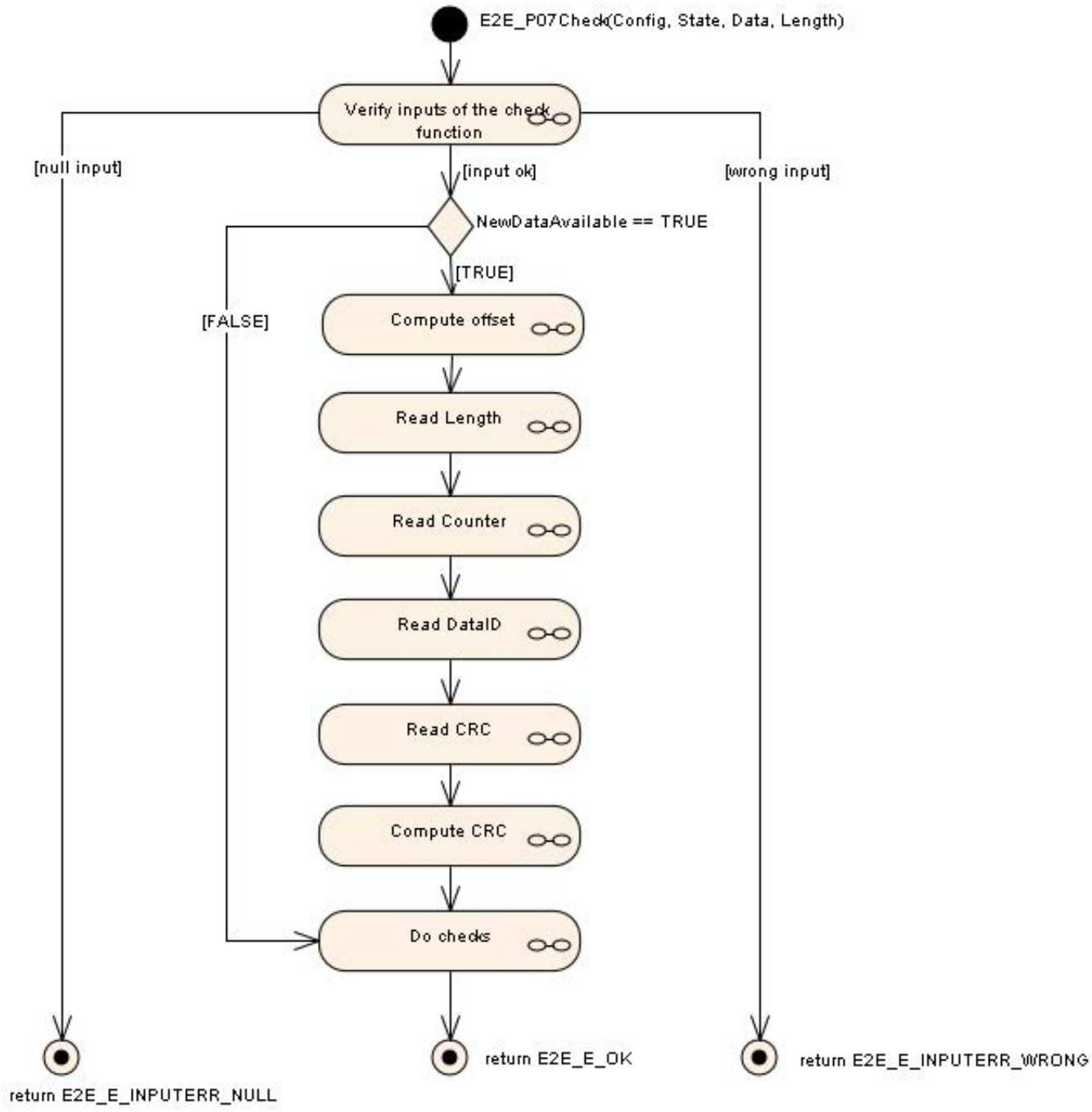

Figure 5.67

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00496] ⌈ The step ”Verify inputs of the check function” in E2E_P07Check() shall have the following behavior:

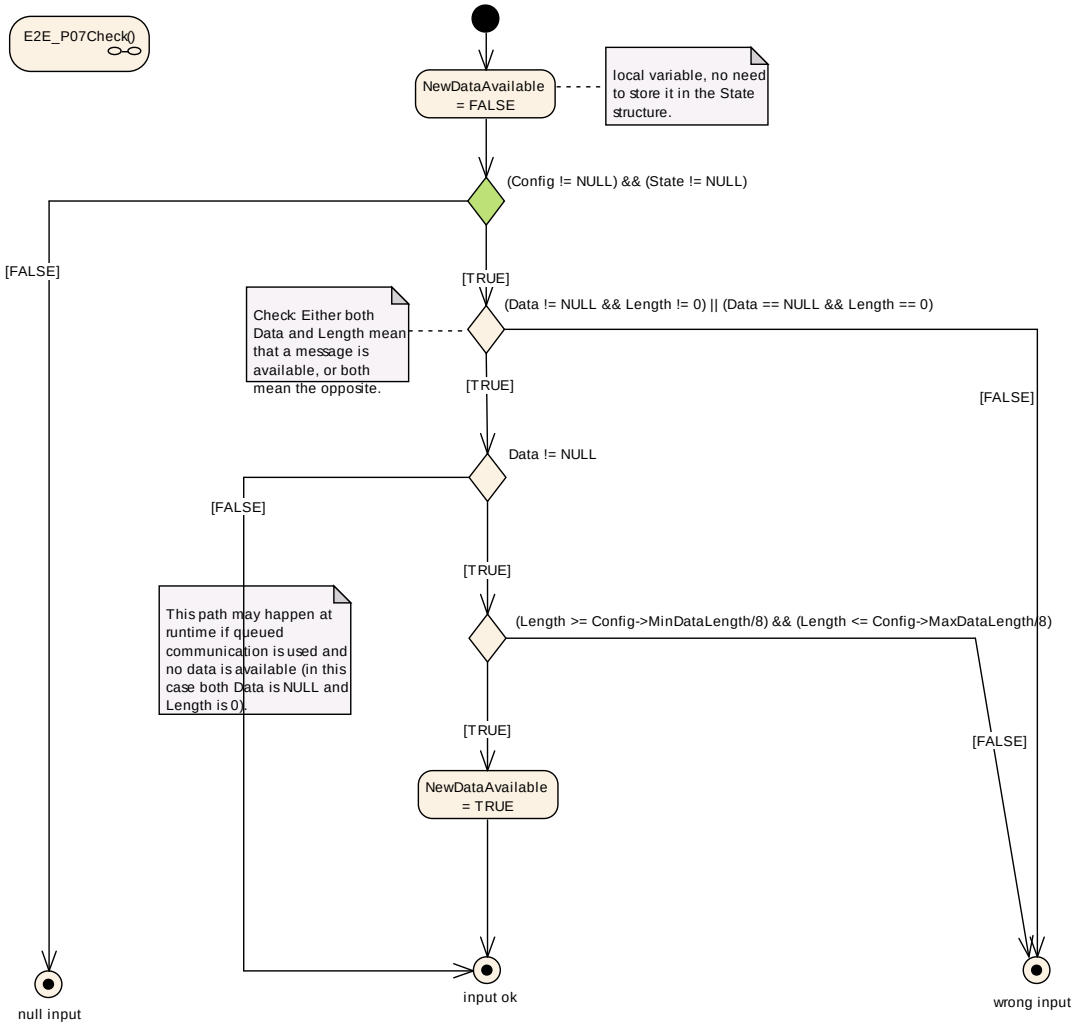

Figure 5.68

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00497] ⌈ The step ”Read Length” in E2E_P07Check() shall have the following behavior:

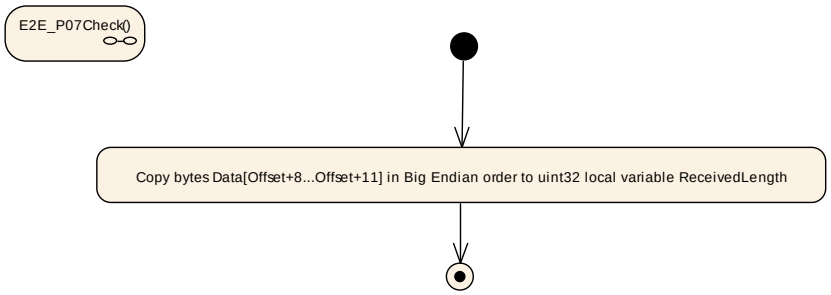

Figure 5.69

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00498] ⌈ The step ”Read Counter” in E2E_P07Check() shall have the following behavior:

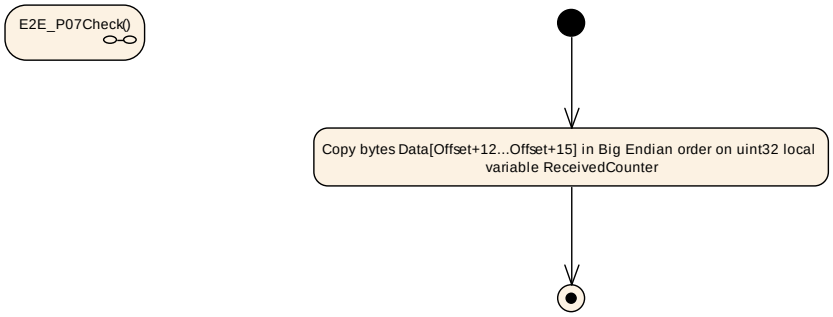

Figure 5.70

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00499] ⌈ The step ”Read DataID ” in E2E_P07Check() shall have the following behavior:

Figure 5.71

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00500] ⌈ The step ”Read CRC” in E2E_P07Check() shall have the following behavior:

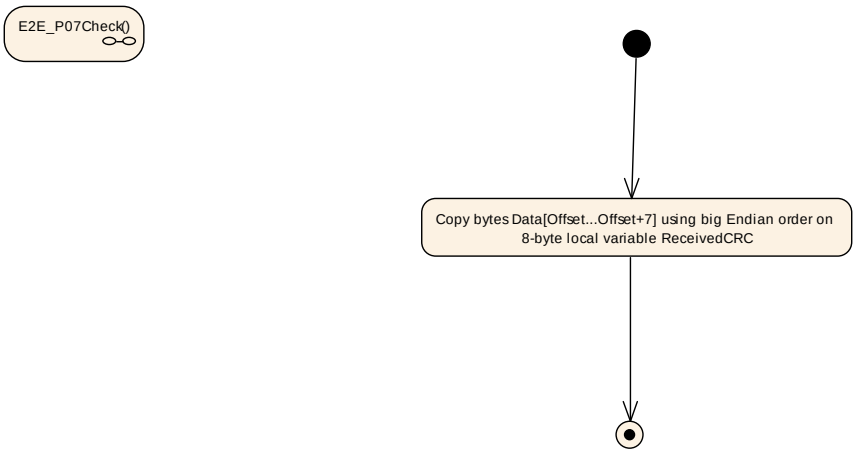

Figure 5.72

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00501] ⌈ The step ”Do Checks” in E2E_P07Check() shall have the following behavior:

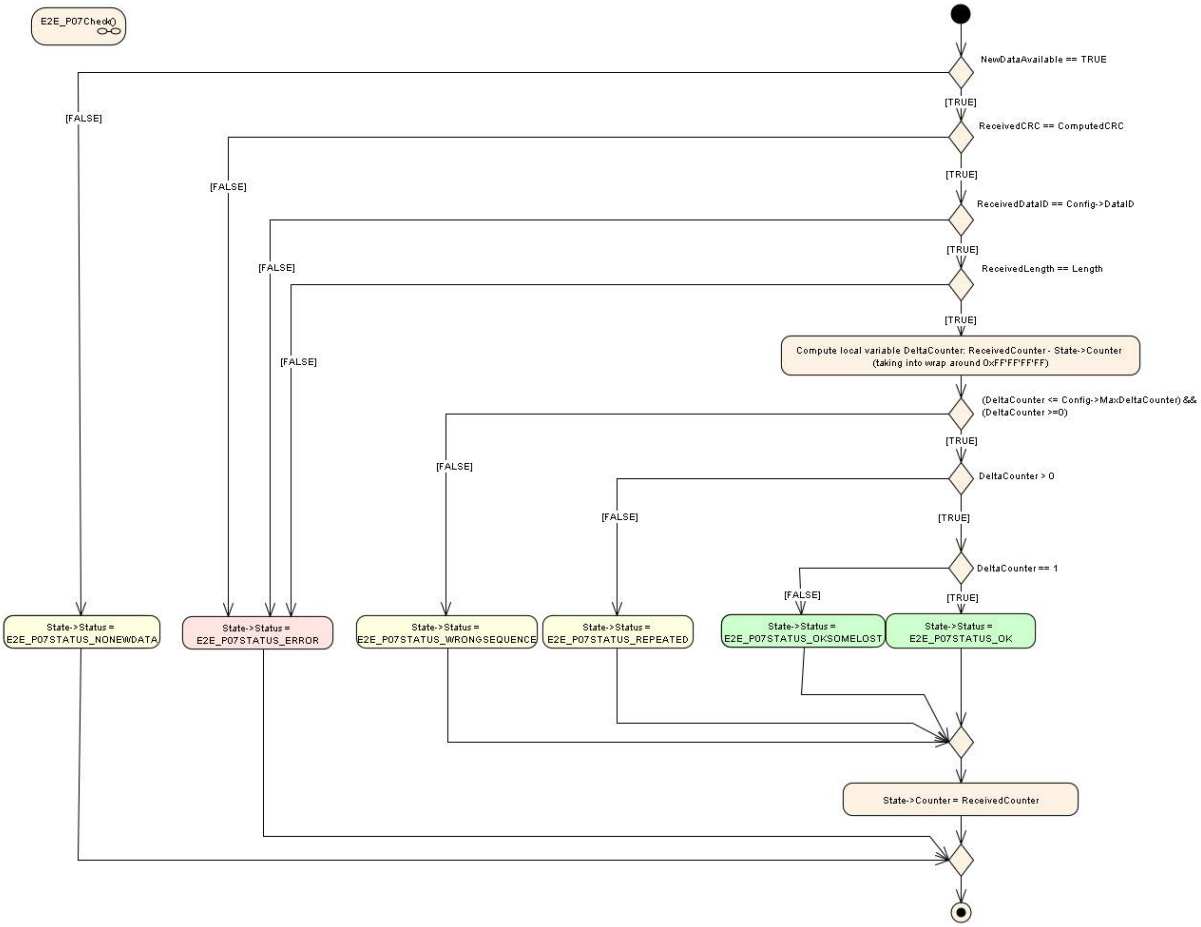

Figure 5.73

⌋(*RS_E2EProtocol_08539*)

5.9 Specification of E2E Profile 11

Profile 11 is bus-compatible to profile 1, but provides "new" profile behavior similar to profiles 4 to 7 on receiver side. Moreover, some legacy DataIDModes that are by now obsolete are omitted.

[PRS_E2EProtocol_00503] ⌈ Profile 11 shall provide the following control fields, transmitted at runtime together with the protected data:

Control field	Description
Counter	4 bits. (explicitly sent)
CRC	8 bits, CRC-8-SAE J1850, provided by CRC library. (explicitly sent)
Data ID	16 bits or 12 bit, unique system-wide. (either implicitly sent (16 bits) or partly explicitly sent (12 bits; 4 bits explicitly and 8 bits implicitly sent))

⌋(*RS_E2EProtocol_08529*, *RS_E2EProtocol_08530*, *RS_E2EProtocol_08533*)

The E2E mechanisms can detect the following faults or effects of faults:

Fault	Main safety mechanisms
Repetition of information	Counter
Loss of information	Counter
Delay of information	Counter
Insertion of information	Data ID
Masquerading	Data ID, CRC
Incorrect addressing	Data ID
Incorrect sequence of information	Counter
Corruption of information	CRC
Asymmetric information sent from a sender to multiple receivers	CRC (to detect corruption at any of receivers)
Information from a sender received by only a subset of receivers and the receivers	Counter (loss on specific receivers)
Blocking access to a communication channel	Counter (loss or timeout)

For details of CRC computation, the usage of start values and XOR values see CRC Supervision [7].

5.9.1 Data Layout

5.9.1.1 User data layout

In the E2E Profile 11, the user data layout (of the data to be protected) is not constrained by E2E Profile 11 - there is only a requirement, that the length of data to be protected is multiple of 1 byte.

5.9.1.2 Header layout

Profile 11 is backward compatible to the bus-layout of profile 1. This means that while all the header fields are configurable, the profile variants of profile 1 are also applicable. Namely, profile 1 variant 1A and variant 1C.

Byte Order	0								1							
Transmission Order	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Bit Order	7	6	5	4	3	2	1	0	15	14	12	12	11	10	9	8
	E2E CRC								DataIDNibble				Counter			

Figure 5.74

The figure above shows Profile 11 variant 11C where the configuration is given as: The E2E header fields (e.g. CRC) are encoded like in CAN and FlexRay, i.e.:

1. CRCOffset = 0
2. CounterOffset = 8 by FlexrayCAN bus.
3. DataIDNibbleOffset = 12

For Profile 11 Variant 11A, DataIDNibble is not used. Instead, user data can be placed there.

[PRS_E2EProtocol_00540] ⌈ The E2E Profile variant 11A is defined as follows:

1. CRC is the 0th byte in the signal group (i.e. starts with bit offset 0)
2. Alive counter is located in lowest 4 bits of 1st byte (i.e. starts with bit offset 8)
3. E2E_P11DataIDMode = E2E_P11_DATAID_BOTH
4. SignalIPdu.unusedBitPattern = 0xFF.

⌋ *(RS_E2EProtocol_08528)*

Below is an example compliant to 11A:

Byte Order	0								1							
Transmission Order	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Bit Order	7	6	5	4	3	2	1	0	15	14	12	12	11	10	9	8
0	E2E CRC												Counter			

Figure 5.75

[PRS_E2EProtocol_00541] ⌈The E2E Profile variant 11C is defined as follows:

1. CRC is the 0th byte in the signal group (i.e. starts with bit offset 0)
2. Alive counter is located in lowest 4 bits of 1st byte (i.e. starts with bit offset 8)
3. The Data ID nibble is located in the highest 4 bits of 1st byte (i.e. starts with bit offset 12)
4. E2E_P11DataIDMode = E2E_P11_DATAID_NIBBLE
5. SignalIPdu.unusedBitPattern = 0xFF

⌋ *(SRS_E2E_08528))*

E2E Profile variant 11C relates to Configuration of E2E Profile 11 configuration setting 11C in system template (system template is more specific).

The transmission order shown above represents the order in which bits are transmitted. For comparability to the figures of profile 1, also the bit order is given. The E2E header fields (e.g. CRC) are encoded like in CAN and FlexRay, i.e.:

1. Little Endian (least significant byte fist) applicable for both implicit and explicit header fields - imposed by profile
2. MSB Fist (most significant bit within byte first) - imposed by Flexray/CAN bus.

5.9.2 Counter

In E2E Profile 11, the counter is initialized, incremented, reset and checked by E2E profile. The counter is not manipulated or used by the caller of the E2E Supervision.

[PRS_E2EProtocol_00504] ⌈ In E2E Profile 11, on the sender side, for the first transmission request of a data element the counter shall be initialized with 0 and shall be incremented by 1 for every subsequent send request. When the counter reaches the maximum value (0x0E), then it shall restart with 0 for the next send request. ⌋ *(RS_E2EProtocol_08539)*

Note that the counter value 0x0F is reserved as a special invalid value, and must never be used by the E2E profile 11.

In E2E Profile 11, on the receiver side, by evaluating the counter of received data against the counter of previously received data, the following is detected:

1. Repetition:

   a. no new data has arrived since last invocation of E2E Supervision check function,

   b. the data is repeated

2. OK:

   a. counter is incremented by one (i.e. no data lost),

   b. counter is incremented more than by one, but still within allowed limits (i.e. some data lost),

3. Error: a. counter is incremented more than allowed (i.e. too many data lost).

Case 1 corresponds to the failed alive counter check, and case 3 correspond to failed sequence counter check.

The above requirements are specified in more details by the UML diagrams in the following document sections.

5.9.3 Data ID

The unique Data IDs are to verify the identity of each transmitted safety-related data element.

There are two supported modes how the Data ID is used:

1. E2E_P11_DATAID_BOTH: both bytes of the 16 bit Data ID are used in the CRC calculation: first the low byte and then the high byte.
2. E2E_P11_DATAID_NIBBLE:

the high nibble of high byte of DataID is not used (it is 0x0), as the DataID is limited to 12 bits,

the low nibble of high byte of DataID is transmitted explicitly and covered by CRC calculation when computing the CRC over Data.

the low byte is not transmitted, but it is included in the CRC computation as start value.

[PRS_E2EProtocol_0507] ⌈ In the E2E profile 11, the Data IDs shall be globally unique within the network of communicating system (made of several ECUs each sending different data). ⌋*(RS_E2EProtocol_08539)*

In case of usage of E2E Supervision for protecting data elements (i.e invocation from RTE), due to multiplicity of communication (1:1 or 1:N), a consumer of a data element expects only a specific data element, which is checked by E2E Supervision using Data ID.

In case of usage of E2E Supervision for protecting I-PDUs (i.e. invocation from COM), the receiver COM expects at a reception only a specific I-PDU, which is checked by E2E Supervision using Data ID.

5.9.4 Length

In Profile 11 there is no explicit transmission of the length.

5.9.5 CRC

E2E Profile 11 uses a 8-bit CRC, to ensure a sufficient detection rate and sufficient Hamming Distance.

[PRS_E2EProtocol_00508] ⌈ E2E Profile 11 shall use the Crc_CalculateCRC8 function of the SWS CRC Supervision for calculating the CRC (CRC-8-SAE J1850). ⌋ *(RS_E2EProtocol_08539, RS_E2EProtocol_08531)*

[PRS_E2EProtocol_00505] ⌈ In the E2E Profile 11 with DataIDMode set to E2E_P11_DATAID_BOTH, the Data ID shall be implicitly transmitted, by adding first the Data ID low byte, then the Data ID high byte before the user data in the CRC calculation ⌋*(RS_E2EProtocol_08539)*

[PRS_E2EProtocol_00506] ⌈In E2E Profile 11 with DataIDMode set to E2E_P01_DATAID_NIBBLE, the lower nibble of the high byte of the DataID shall be placed in the transmitted data at bit position DataIDNibbleOffset, and the CRC calculation shall be done by first calculating over the low byte of the Data ID, then a 0-byte, and then the user data. ⌋*(RS_E2EProtocol_08539)*

Note: the byte containing the CRC is always omitted from the CRC calculation.

5.9.6 Timeout detection

The previously mentioned mechanisms (for Profile 11: CRC, Counter, Data ID) enable to check the validity of received data element, when the receiver is executed independently from the data transmission, i.e. when receiver is not blocked waiting for Data Elements or respectively I-PDUs, but instead if the receiver reads the currently available data (i.e. checks if new data is available). Then, by means of the counter, the receiver can detect loss of communication and timeouts.

The attribute State->NewDataAvailable == E2E_P11STATUS_NONEWDATA means that the transmission medium (e.g RTE) reports that no new data element is available at the transmission medium. The attribute State->Status = E2E_P11STATUS_REPEATED means that the transmission medium (e.g. RTE) provided new valid data element, but this data element has the same counter as the previous valid data element. Both conditions represent an unavailability of valid data that was updated since the previous cycle.

5.9.7 E2E_P11Protect

The function E2E_P11Protect() performs the steps as specified by the following six diagrams in this section.

[PRS_E2EProtocol_00509] ⌈ The function E2E_P11Protect() shall have the following overall behavior:

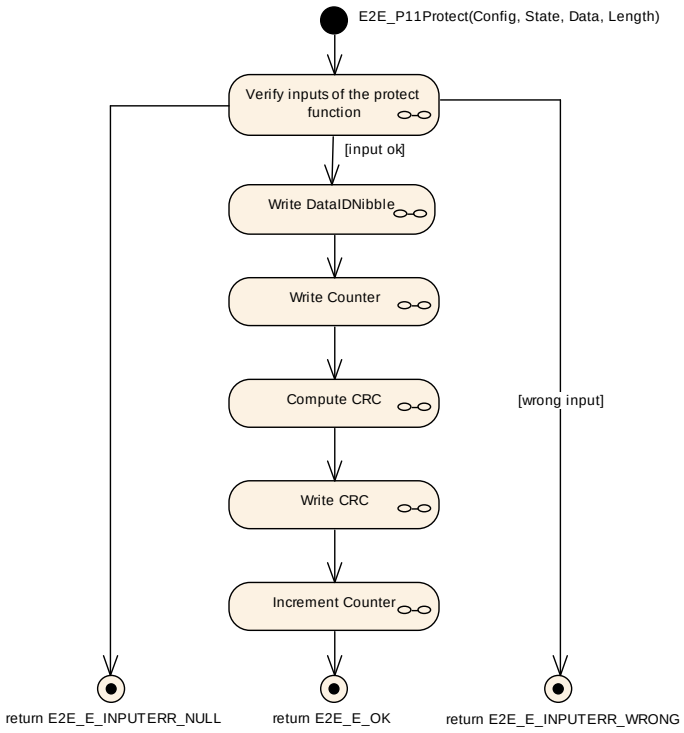

Figure 5.76

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00510] ⌈The step ”Verify inputs of the protect function” in E2E_P11Protect() shall have the following behavior:

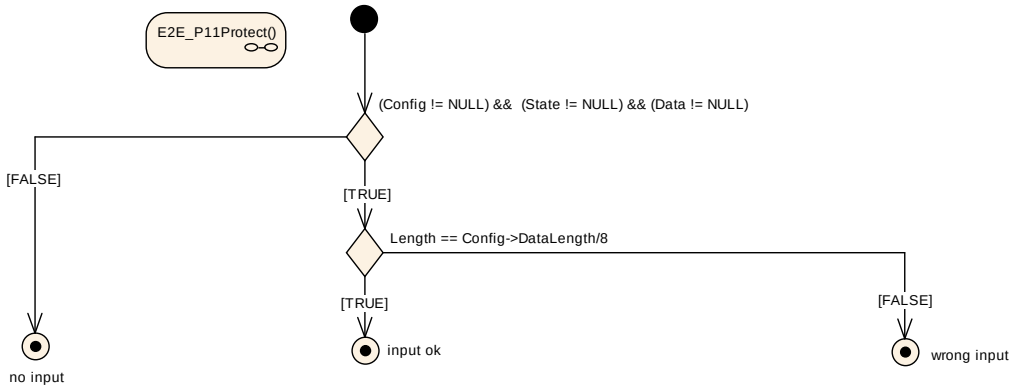

Figure 5.77

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00511] ⌈ The step „Write DataIDNibble” in E2E_P11Protect() shall have the following behavior:

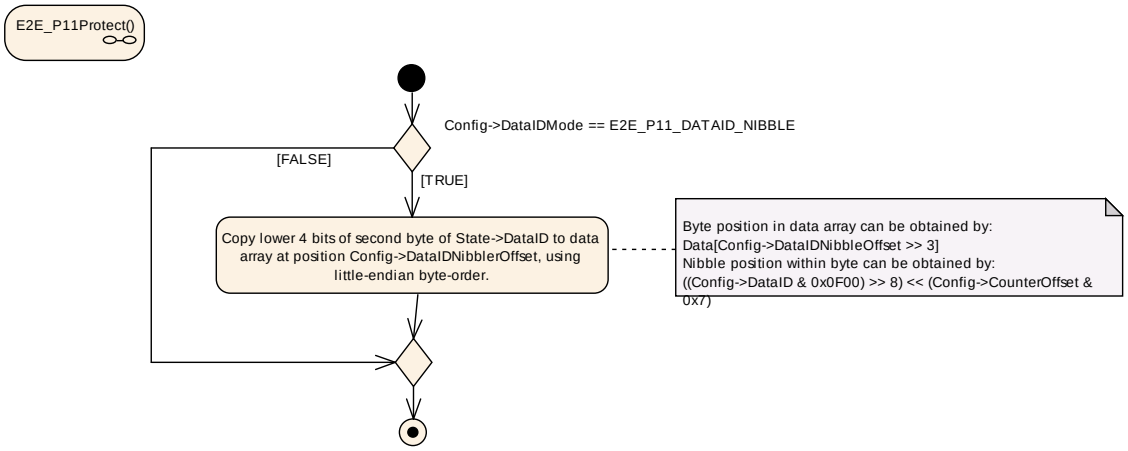

Figure 5.78

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00512] ⌈ The step ”Write Counter” in E2E_P11Protect() shall have the following behavior:

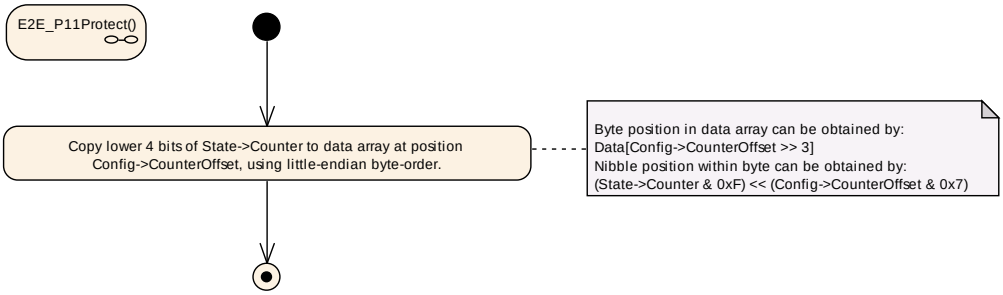

Figure 5.79

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00513] ⌈ The step ”Compute CRC” in E2E_P11Protect() and in E2E_P11Check shall have the following behavior:

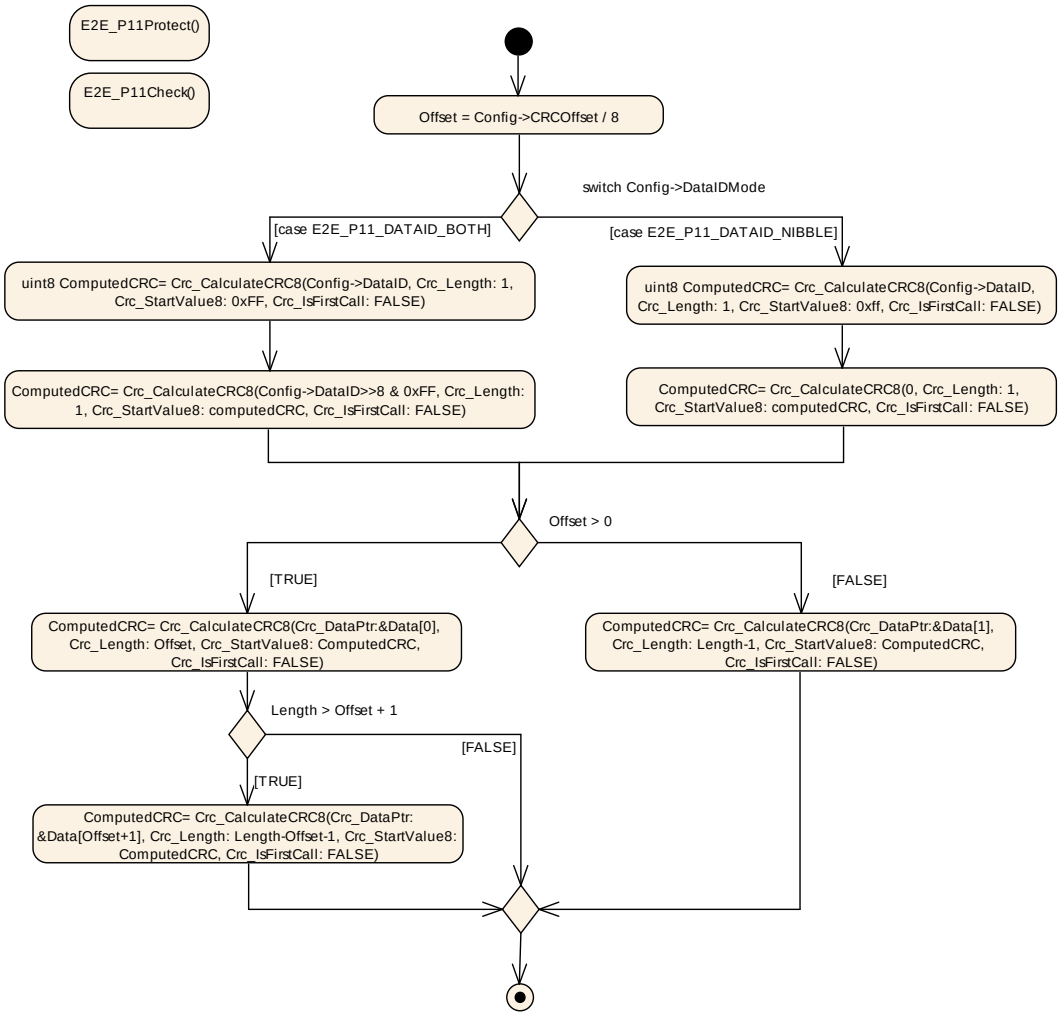

Figure 5.80

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00514] ⌈ The step ”Write CRC” in E2E_P11Protect() shall have the following behavior:

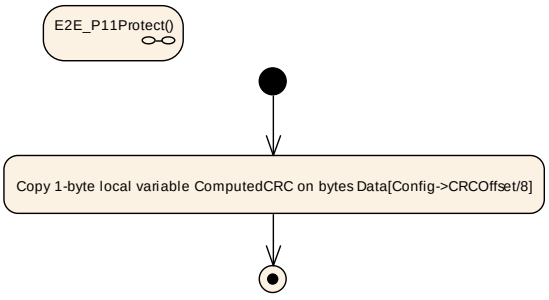

Figure 5.81

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00515] ⌈ The step ”Increment Counter” in E2E_P11Protect() shall have the following behavior:

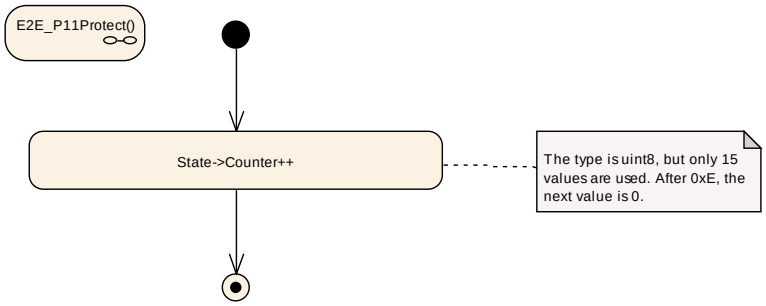

Figure 5.82

⌋(*RS_E2EProtocol_08539*)

5.9.8 E2E_P11Check

The function E2E_P11Check performs the actions as specified by the following six diagrams in this section.

[PRS_E2EProtocol_00516] ⌈ The function E2E_P11Check() shall have the following overall behavior:

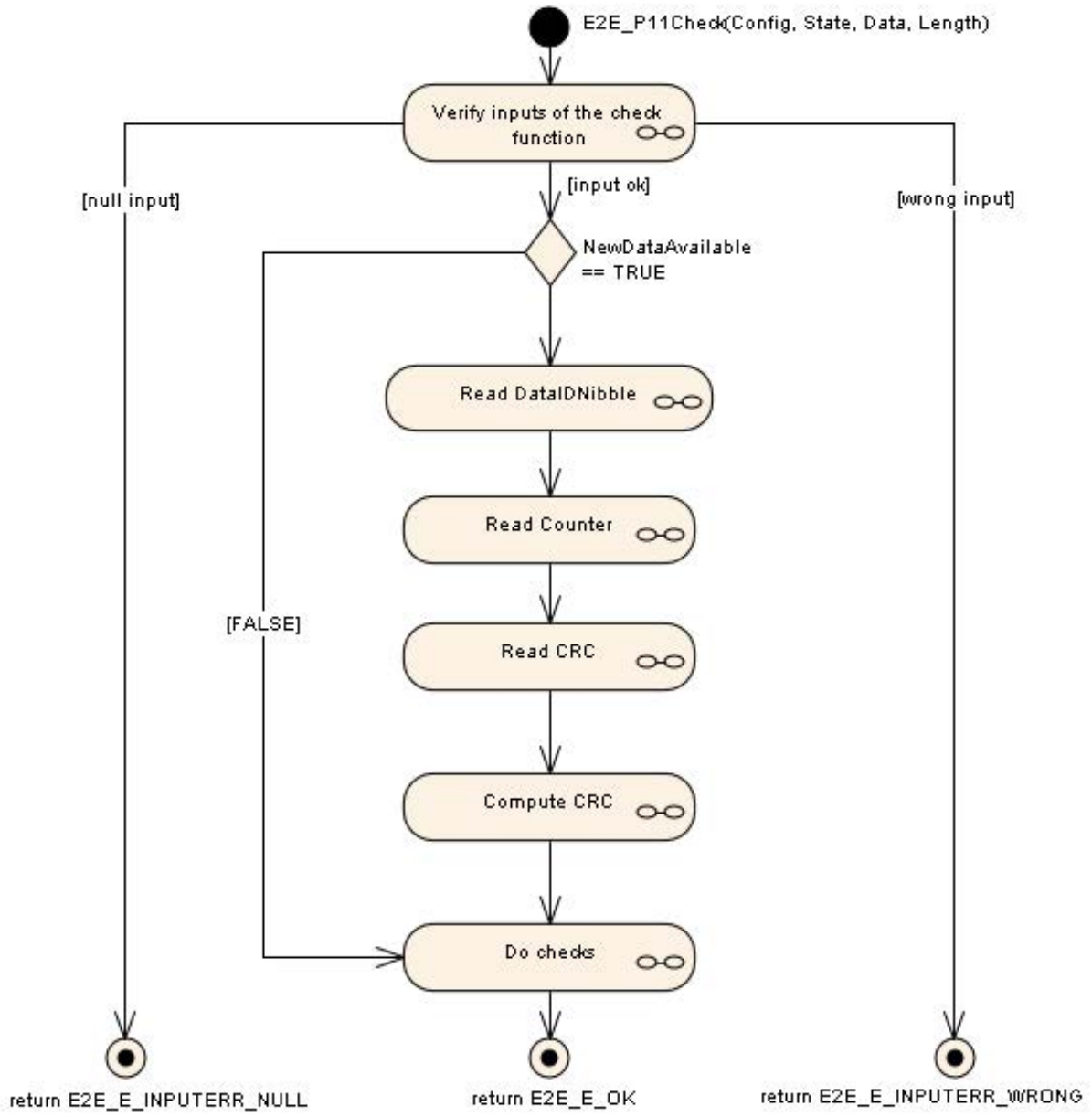

Figure 5.83

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00517] ⌈ The step ”Verify inputs of the check function” in E2E_P11Check() shall have the following behavior:

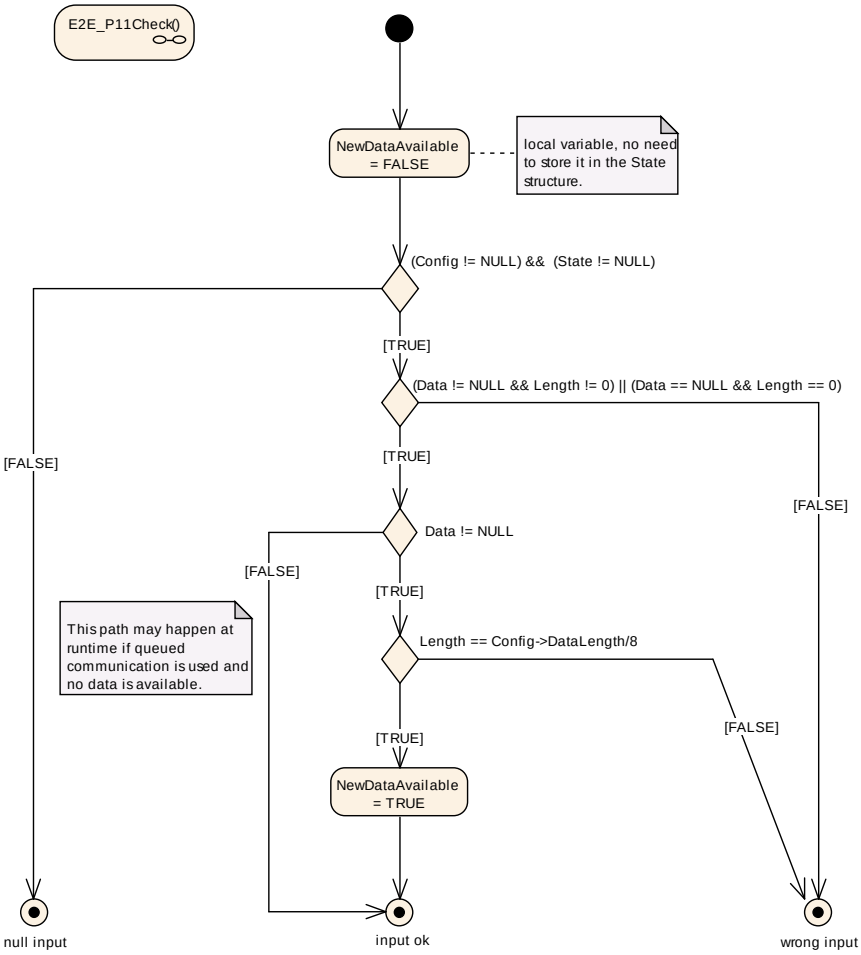

Figure 5.84

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00582] ⌈ The step ”Read Counter” in E2E_P11Check() shall have the following behavior:

fix me figure ⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00518] ⌈ The step "Read Counter" in E2E_P11Check() shall have the following behavior:

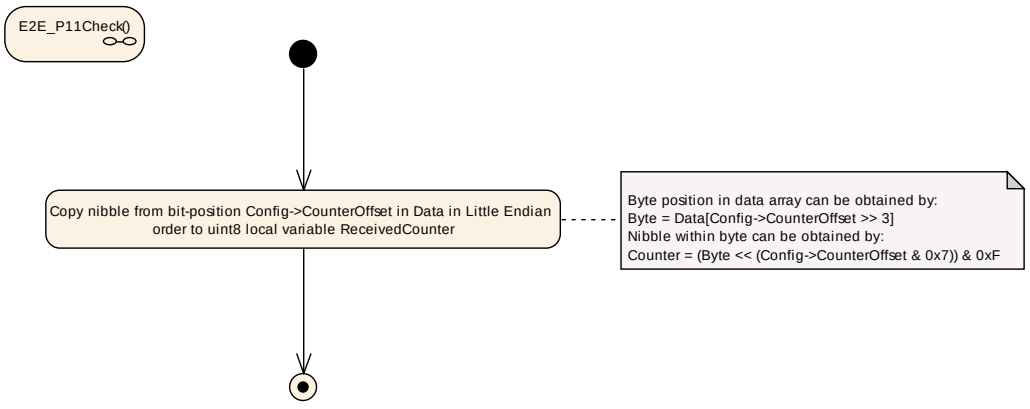

Figure 5.85

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00519] ⌈ The step ”Read CRC” in E2E_P11Check() shall have the following behavior:

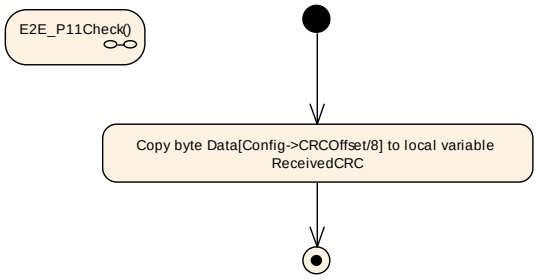

Figure 5.86

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00521] ⌈ The step ”Do Checks’ in E2E_P11Check() shall have the following behavior:

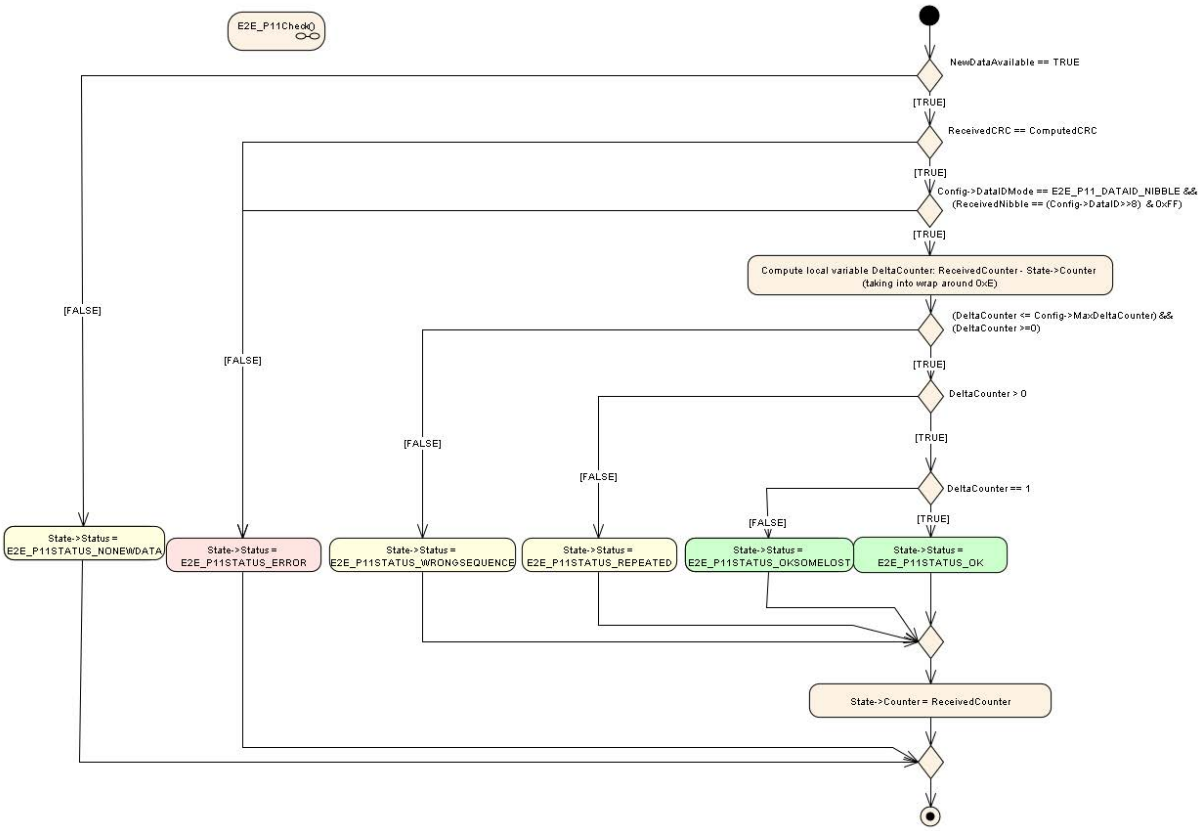

Figure 5.87

⌋(*RS_E2EProtocol_08539*)

5.10 Specification of E2E Profile 22

[PRS_E2EProtocol_00522] ⌈ Profile 22 shall provide the following control fields, transmitted at runtime together with the protected data:

Control field	Description
Counter	4 bits. (explicitly sent)
CRC	8 bits, polynomial in normal form 0x2F (Autosar notation), provided by CRC library. (explicitly sent)
Data ID List	16 8 bits values, linked to Counter value. Effectively 16 different values, one for each counter value. The Data ID List must be unique system-wide.

⌋*(RS_E2EProtocol_08529, RS_E2EProtocol_08530, RS_E2EProtocol_08533)*

The E2E mechanisms can detect the following faults or effects of faults:

E2E Mechanism	Detected communication faults
Counter	Repetition, loss, insertion, incorrect sequence, blocking
Transmission on a regular bases and timeout monitoring using E2E-Library [5]	Loss, delay, blocking
Data ID + CRC	Masquerade and icorrect addressing, insertion
CRC	Corruption, asymmetric information [6]

For details of CRC computation, the usage of start values and XOR values see CRC Supervision [7].

5.10.1 Data Layout

5.10.1.1 User data layout

In the E2E Profile 22, the user data layout (of the data to be protected) is not constrained by E2E Profile 22. The total length of transmitted data must be a multiple of 8 bit (full bytes). Also, as the header only used 12 bit, there are 4 bit unused and available for user data in the byte where the 4 bit of the counter are placed.

[5] Implementation by sender and receiver
[6] for a set of data protected by same CRC

5.10.1.2 Header layout

Profile 22 is backward compatible to the bus-layout of profile 2. In addition, the configuration field offset can be used to offset the header fields, then breaking with backward-compatibility to profile 2 bus-layout.

Byte Order	0								1							
Transmission Order	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Bit Order	7	6	5	4	3	2	1	0	15	14	12	12	11	10	9	8
0	E2E CRC												Counter			

Figure 5.88: E2E Profile22 header with offset 0.

The figure above shows Profile 22 with offset configured with 0. Offset is always given in bit and a multiple of 8 (full bytes).

The transmission order shown above represents the order in which bits are transmitted. For comparability to the figures of profile 2, also the bit order is given. The E2E header fields (e.g. CRC) are encoded like in CAN and FlexRay, i.e.:

1. Little Endian (least significant byte fist) applicable for both implicit and explicit header fields - imposed by profile
2. MSB Fist (most significant bit within byte first) - imposed by Flexray/CAN bus.

5.10.2 Counter

In E2E Profile 22, the counter is initialized, incremented, reset and checked by E2E profile check and protect functions. The counter is not manipulated or used by the caller of the E2E Supervision. .

[PRS_E2EProtocol_00523] ⌈ In E2E Profile 22, on the sender side, for the first transmission request of a data element the counter shall be initialized with 0 and shall be incremented by 1 for every subsequent send request. When the counter reaches the maximum value (0x0F), then it shall restart with 0 for the next send request. ⌋ *(RS_E2EProtocol_08539)*

Note that the counter value 0x0F is not reserved as a special invalid value.

In E2E Profile 22, on the receiver side, by evaluating the counter of received data against the counter of previously received data, the following is detected:

1. Repetition:

   a. no new data has arrived since last invocation of E2E Supervision check function,

   b. the data is repeated

2. OK:

a. counter is incremented by one (i.e. no data lost),

b. counter is incremented more than by one, but still within allowed limits (i.e. some data lost),

3. Error: a. counter is incremented more than allowed (i.e. too many data lost).

Case 1 corresponds to the failed alive counter check, and case 3 correspond to failed sequence counter check.

The above requirements are specified in more details by the UML diagrams in the following document sections.

5.10.3 Data ID

The unique Data ID List is used to verify the identity of each transmitted safety-related data element.

[PRS_E2EProtocol_00524] ⌈ In the E2E Profile 22, the Data ID shall be implicitly transmitted, by adding the Data ID after the user data in the CRC calculation. ⌋ *(RS_E2EProtocol_08539)*

[PRS_E2EProtocol_00525] ⌈ In the E2E profiles 2 and 22, the Data ID Lists shall be globally unique within the network of communicating system (made of several ECUs each sending different data.) ⌋*(RS_E2EProtocol_08539)*

In case of usage of E2E Supervision for protecting data elements (i.e invocation from RTE), due to multiplicity of communication (1:1 or 1:N), a consumer of a data element expects only a specific data element, which is checked by E2E Supervision using Data ID.

In case of usage of E2E Supervision for protecting I-PDUs (i.e. invocation from COM), the receiver COM expects at a reception only a specific I-PDU, which is checked by E2E Supervision using Data ID.

5.10.4 Length

In Profile 22 there is no explicit transmission of the length.

5.10.5 CRC

E2E Profile 22 uses an 8-bit CRC, to ensure a sufficient detection rate and sufficient Hamming Distance. The CRC polynomial is the same as used in profile 2.

[PRS_E2EProtocol_00526] ⌈ E2E Profile 22 shall use the Crc_CalculateCRC8H2F() function of the SWS CRC Supervision for calculating the CRC (Polynomial 0x2F, see also SWS_E2E_00117) ⌋*(RS_E2EProtocol_08539, RS_E2EProtocol_08531)*

[PRS_E2EProtocol_00527] ⌈ In E2E Profile 22, the CRC shall be calculated over the entire E2E header (excluding the CRC bytes), including the user data extended at the end with the coresponding Data ID from the Data ID List. ⌋*(RS_E2EProtocol_08539, RS_E2EProtocol_08536)*

5.10.6 Timeout detection

The previously mentioned mechanisms (for Profile 22: CRC, Counter, Data ID) enable to check the validity of received data element, when the receiver is executed independently from the data transmission, i.e. when receiver is not blocked waiting for Data Elements or respectively I-PDUs, but instead if the receiver reads the currently available data (i.e. checks if new data is available). Then, by means of the counter, the receiver can detect loss of communication and timeouts.

The attribute State->Status = E2E_P22STATUS_NONEWDATA means that the transmission medium (e.g RTE) reported that no new data element is available at the transmission medium. The attribute State->Status = E2E_P22STATUS_REPEATED means that the transmission medium (e.g. RTE) provided new valid data element, but this data element has the same counter as the previous valid data element. Both conditions represent an unavailability of valid data that was updated since the previous cycle.

5.10.7 E2E_P22Protect

The function E2E_P22Protect() performs the steps as specified by the following diagrams in this section.

[PRS_E2EProtocol_00528] ⌈ The function E2E_P22Protect() shall have the following overall behavior:

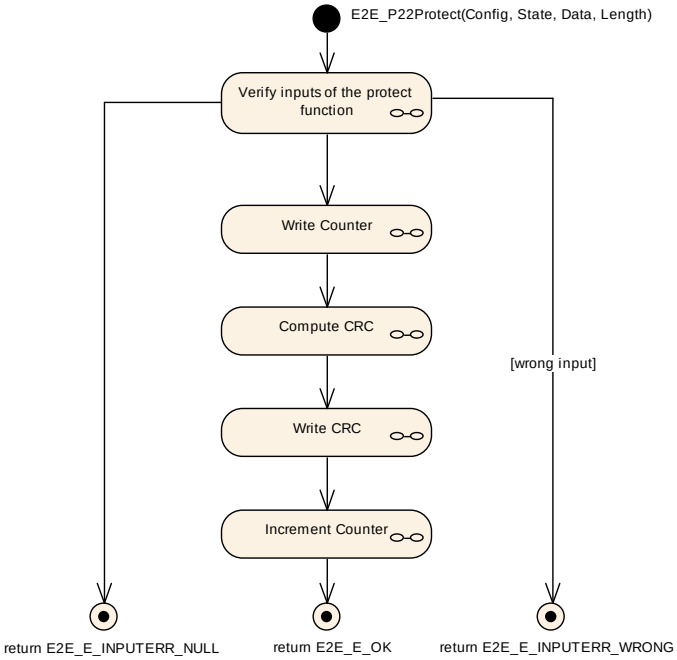

Figure 5.89

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00529] ⌈The step ”Verify inputs of the protect function” in E2E_P22Protect() shall have the following behavior:

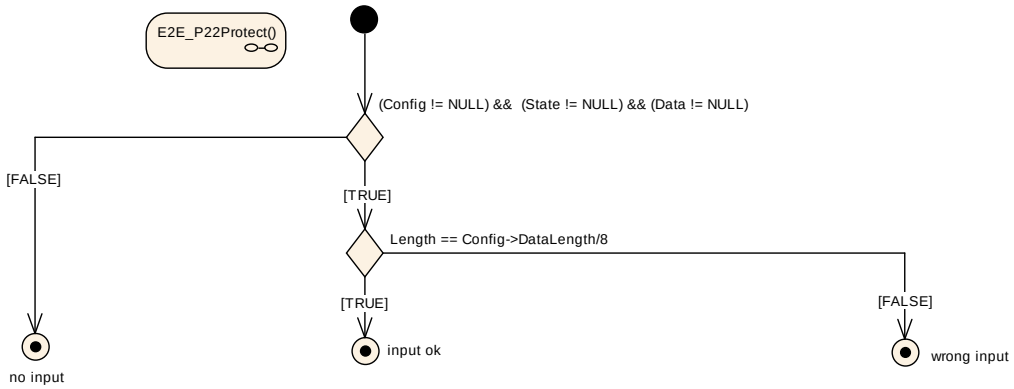

Figure 5.90

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00530] ⌈The step ”Write Counter” in E2E_P22Protect() shall have the following behavior:

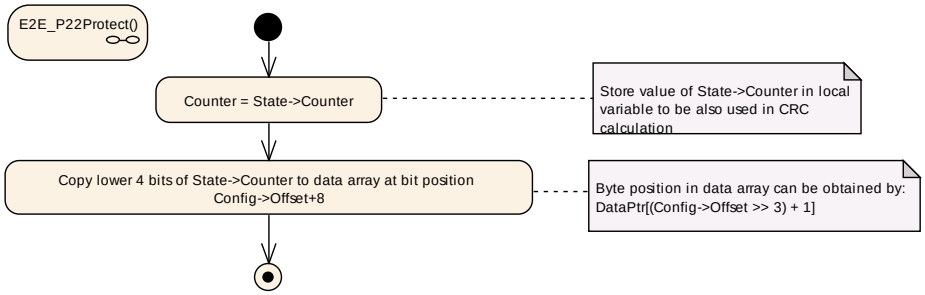

Figure 5.91

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00531] ⌈ The step ”Compute CRC” in E2E_P22Protect() and in E2E_P22Check shall have the following behavior:

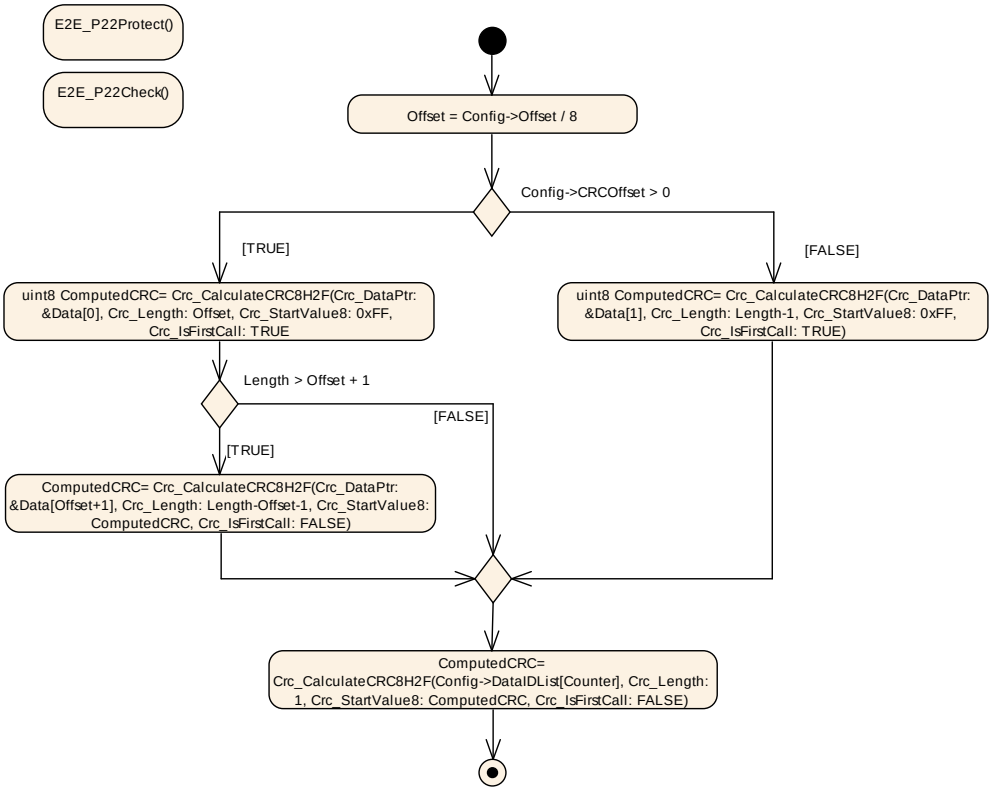

Figure 5.92

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00532] ⌈ The step ”Write CRC” in E2E_P22Protect() shall have the following behavior:

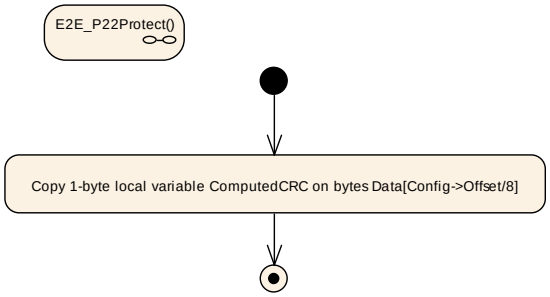

Figure 5.93

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00533] ⌈ The step ”Increment Counter” in E2E_P22Protect() shall have the following behavior:

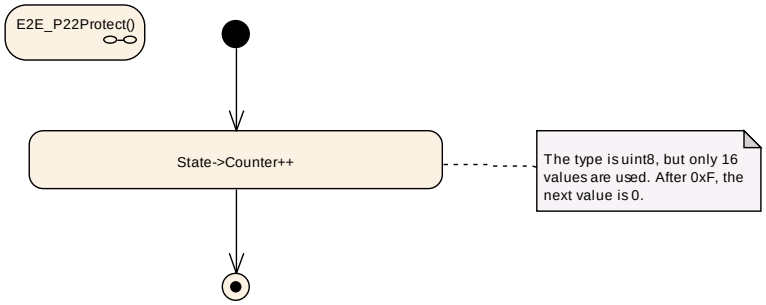

Figure 5.94

⌋(*RS_E2EProtocol_08539*)

5.10.8 E2E_P22Check

The function E2E_P22Check performs the actions as specified by the following six diagrams in this section.

[PRS_E2EProtocol_00534] ⌈ The function E2E_P22Check() shall have the following overall behavior:

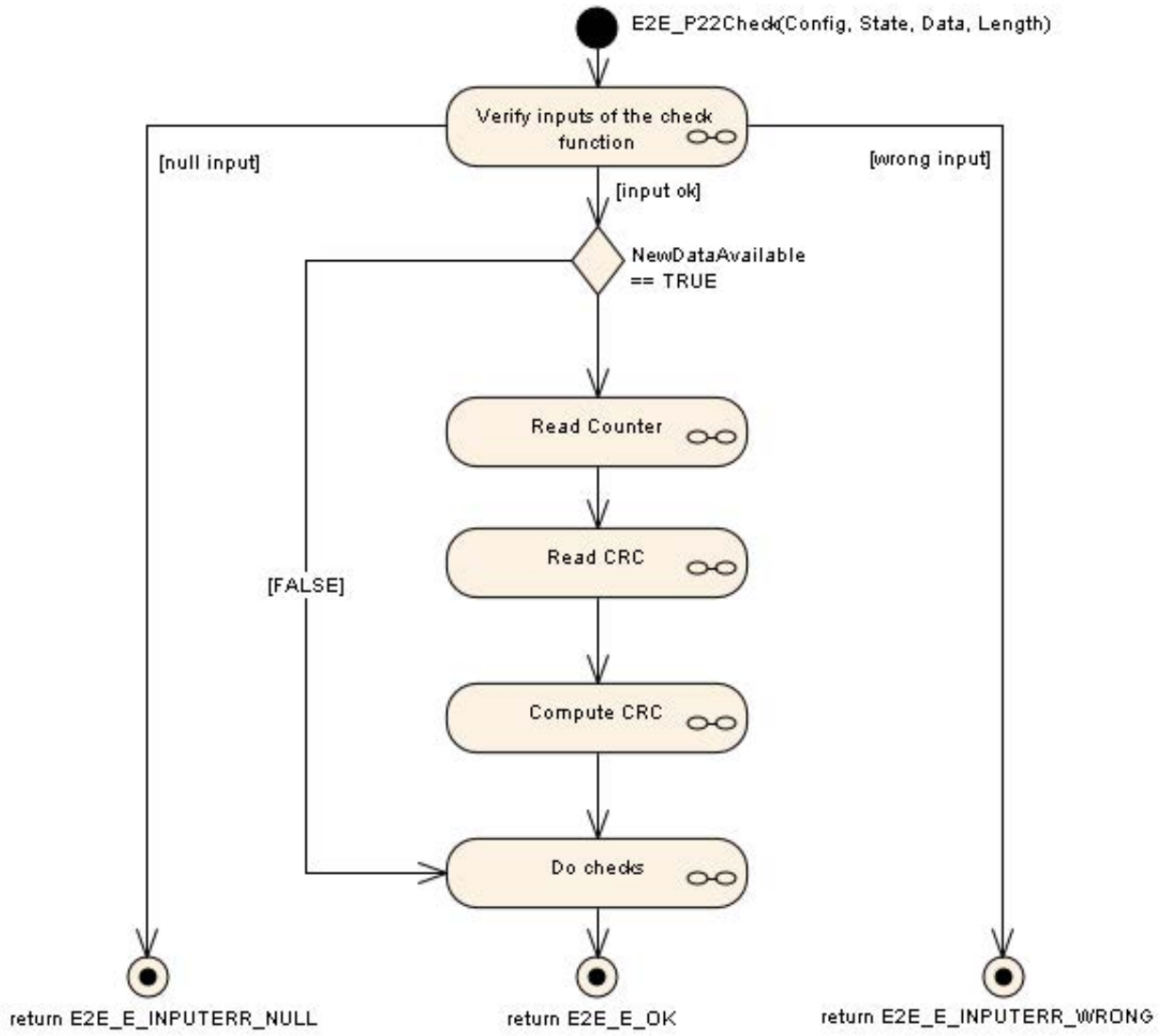

Figure 5.95

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00535] ⌈ The step ”Verify inputs of the check function” in E2E_P22Check() shall have the following behavior:

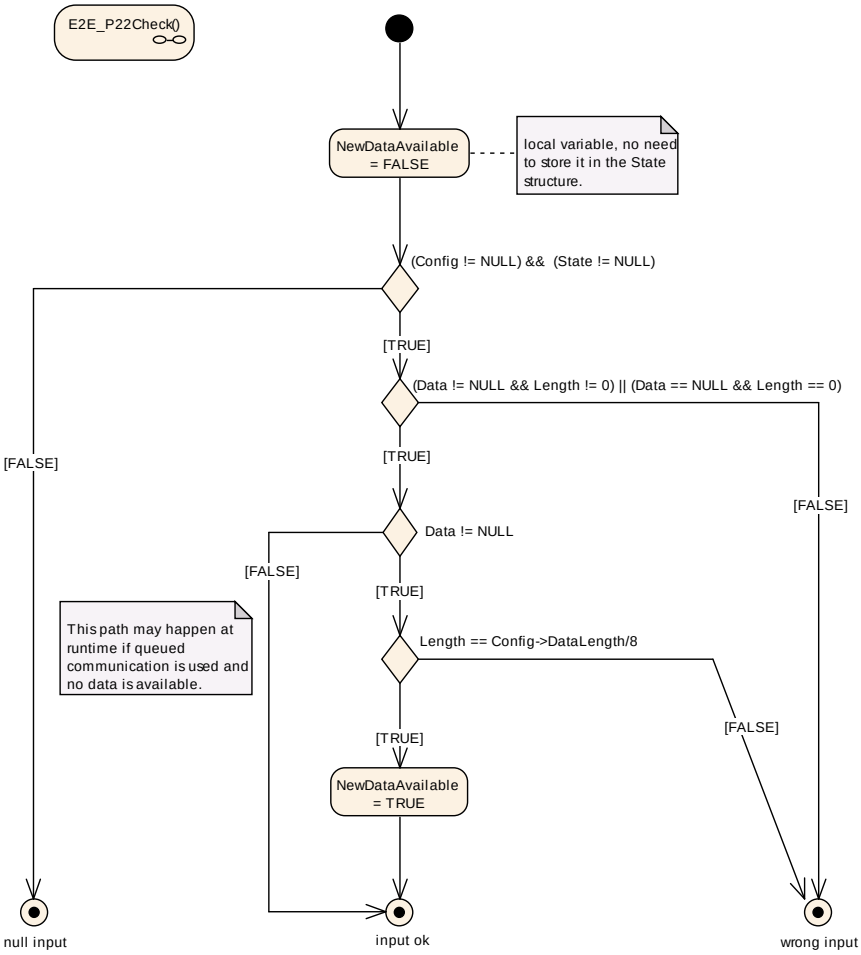

Figure 5.96

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00536] ⌈ The step ”Read Counter” in E2E_P22Check() shall have the following behavior:

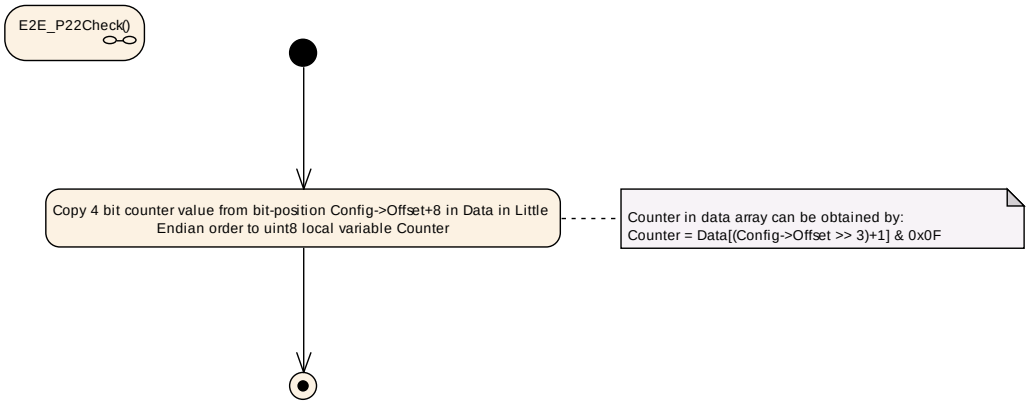

Figure 5.97

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00537] ⌈ The step ”Read CRC” in E2E_P22Check() shall have the following behavior:

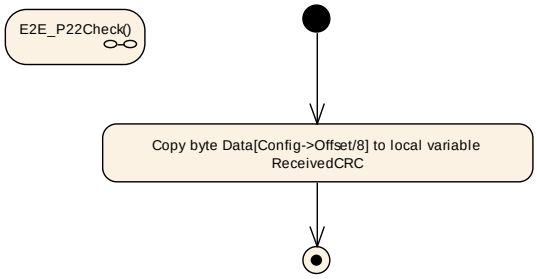

Figure 5.98

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00539] ⌈ The step ”Do Checks’ in E2E_P22Check() shall have the following behavior:

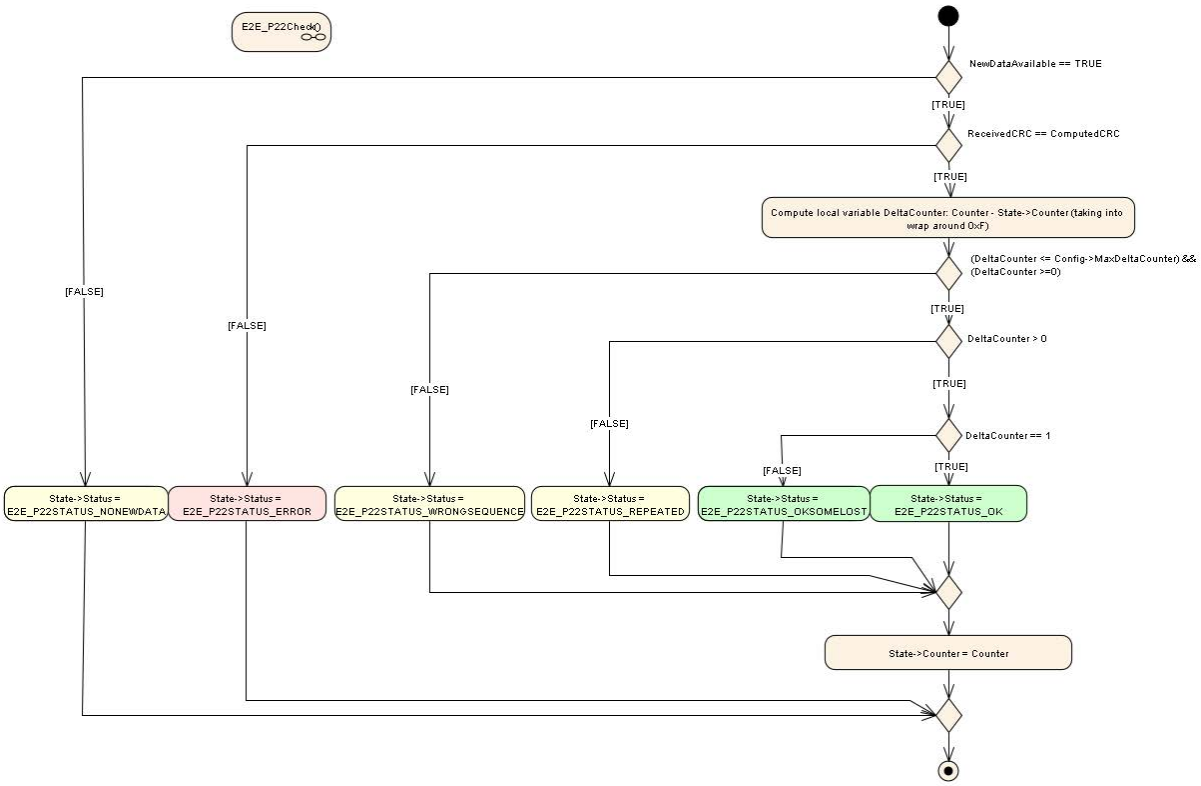

Figure 5.99

⌋(*RS_E2EProtocol_08539*)

5.11 Specification of E2E state machine

The E2E Profile check()-functions verifies data in one cycle. This function only determines if data in that cycle are correct or not. In contrary, the state machine builds up a state out of several results of check() function within a reception window, which is then provided to the consumer (RTE/SWC/COM).

The state machine is applicable for all E2E profiles. Profiles P01 and P02 can be configured to work together with the state machine. However, the behavior of P01/P02 alone, regardless how it is configured, is different to the behavior of P01/P02 + state machine.

5.11.1 Overview of the state machine

The diagram below summarizes the state machine.

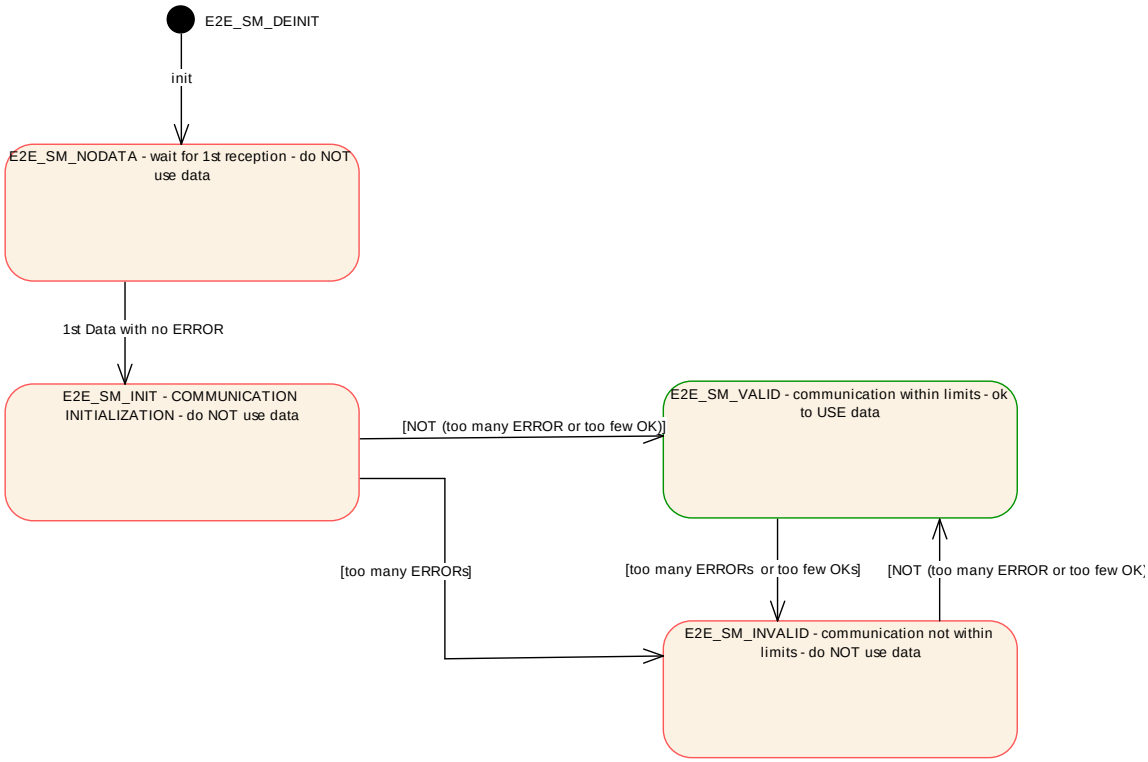

Figure 5.100

5.11.2 State machine specification

[PRS_E2EProtocol_00354] ⌈ The E2E state machine shall be implemented by the functions E2E_SMCheck() and E2E_SMCheckInit() ⌋*(RS_E2EProtocol_08539)*

[PRS_E2EProtocol_00345] ⌈ The E2E State machine shall have the following behavior with respect to the function E2E_SMCheck():

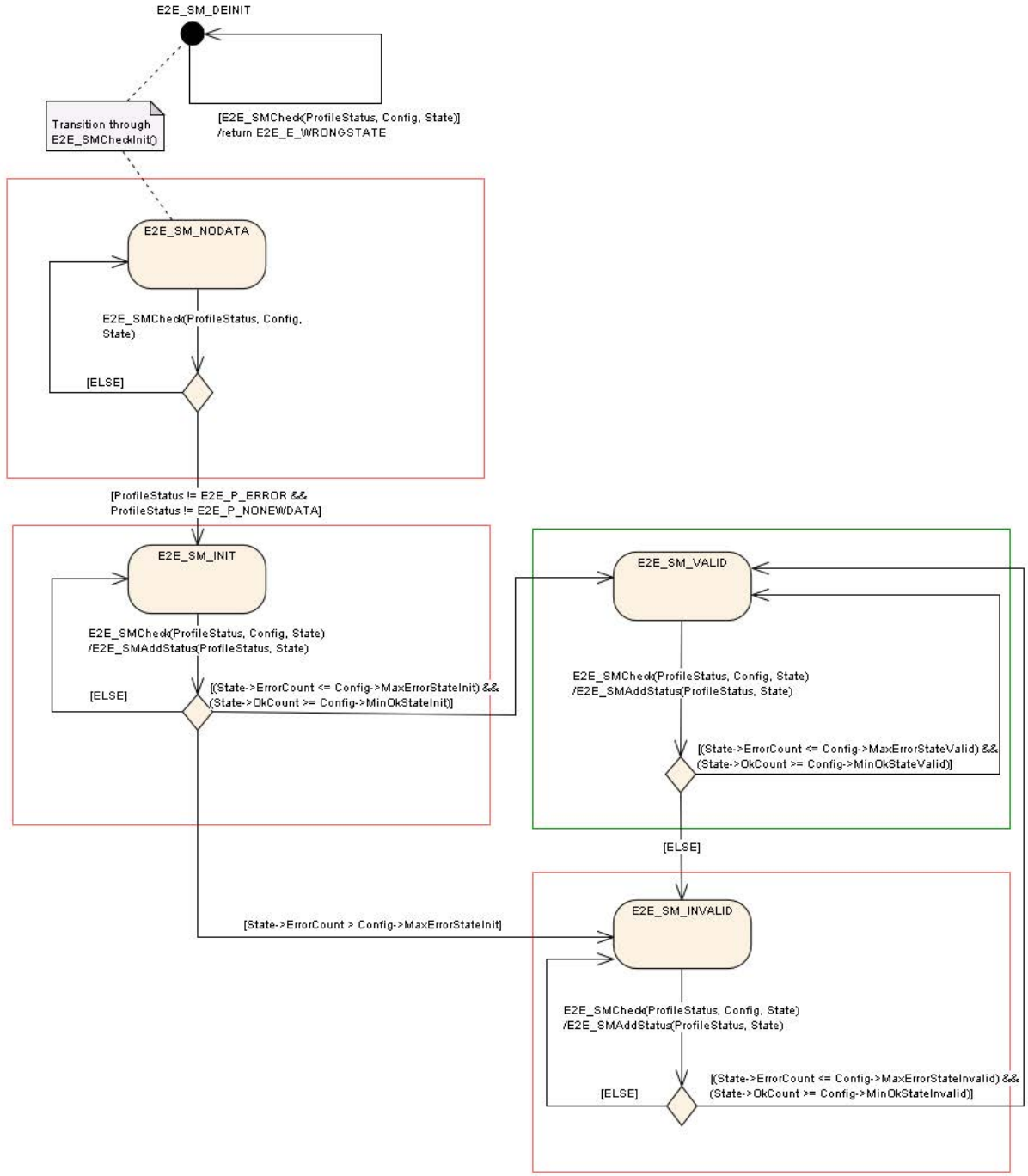

Figure 5.101

This shall be understood as follows:

1. The current state (e.g. E2E_SM_VALID) is stored in State->SMState
2. At every invocation of E2E_SMCheck, the ProfileStatus is processed (as shown by logical step E2E_SMAddStatus()

3. After that, there is an examination of two counters: State->ErrorCount and State->OKCount. Depending on their values, there is a transition to a new state, stored in State->SMState.

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00466] ⌈ The step E2E_SMAddStatus(ProfileStatus, State) in E2E_SMCheck() shall have the following behavior:

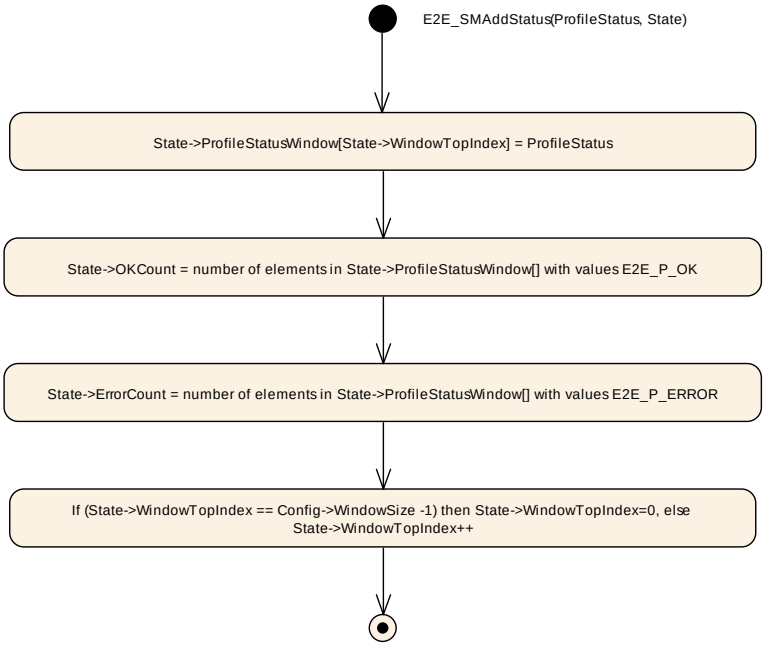

Figure 5.102

⌋(*RS_E2EProtocol_08539*)

E2E_SMAddStatus is just a logical step in the algorithm, it may (but it does not have to be) implemented a a separate function. It is not a module API function.

[PRS_E2EProtocol_00375] ⌈ The E2E State machine shall have the following behavior with respect to the function E2E_SMCheckInit():

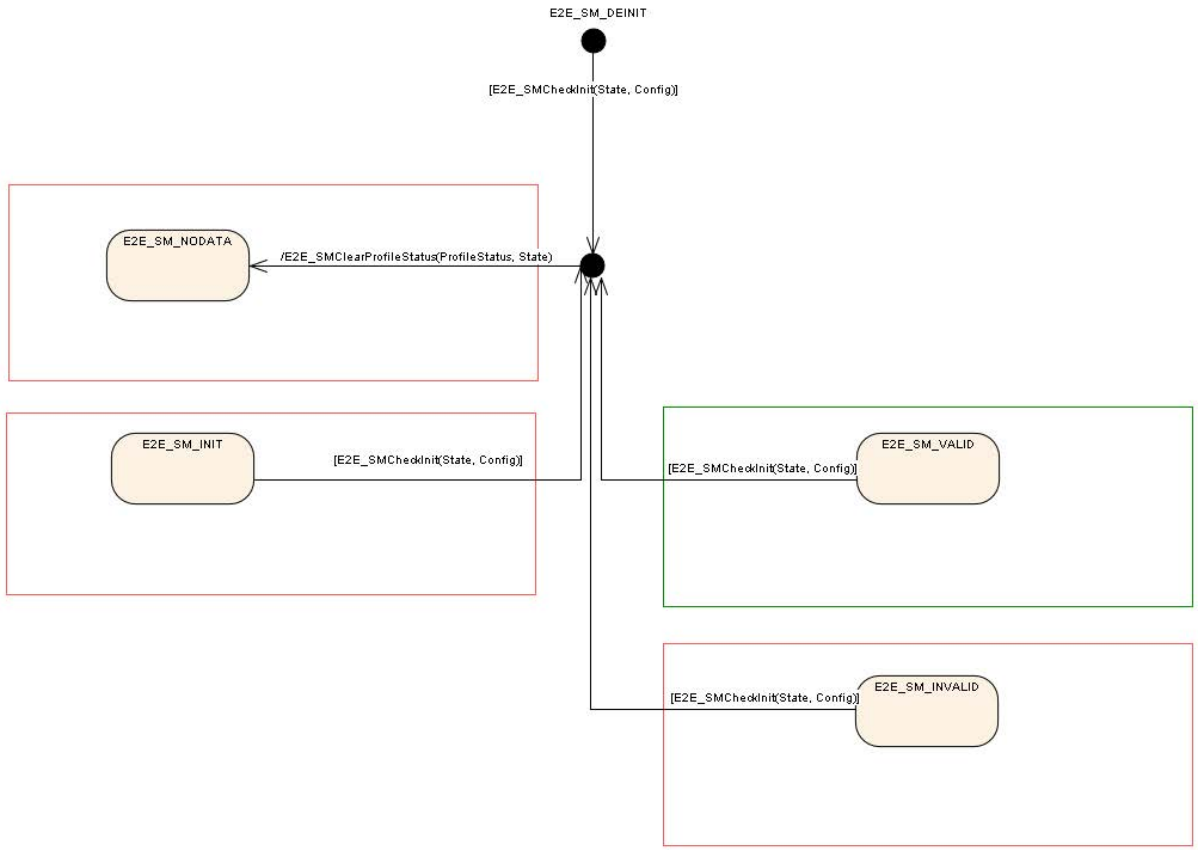

Figure 5.103

⌋(*RS_E2EProtocol_08539*)

[PRS_E2EProtocol_00467] ⌈ The step E2E_SMClearStatus(ProfileStatus, State) in E2E_SMCheck() shall have the following behavior:

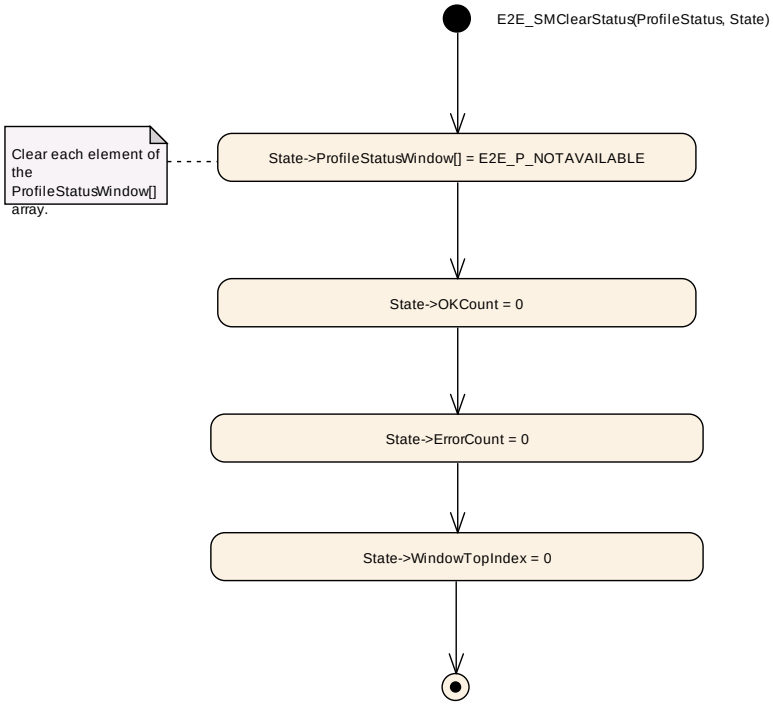

Figure 5.104

⌋(*SRS_BSW_00003*)

5.12 Basic Concepts of CRC Codes

5.12.1 Mathematical Description

Let D be a bitwise representation of data with a total number of n bit, i.e.

$$D = (d_{n-1}, d_{n-2}, d_{n-3}, ..., d_1, d_0),$$

with $d_0, d_1, ... = 0b, 1b$. The corresponding Redundant Code C is represented by $n + k$ bit as

$$C = (D, R) = (d_{n-1}, d_{n-2}, d_{n-3}, ..., d_2, d_1, d_0, r_{k-1}, ..., r_2, r_1, r_0)$$

with r_0, r_1, ... = 0b,1b and R = ($r_{k-1}, ..., r_2, r_1, r_0$) The code is simply a concatenation of the data and the redundant part. (For our application, we will chose $k = 16, 32$ and n as a multiple of 16 resp. 32).

CRC-Algorithms are related to polynomials with coefficients in the finite field of two element, using arithmetic operations $\oplus$ and * according to the following tables.

The $\oplus$ operation is identified as the binary operation exclusive-or, that is usually available in the ALU of any CPU.

+	0b	1b		*	0b	1b
0b	0b	1b		0b	0b	0b
1b	1b	0b		1b	0b	1b

For simplicity, we will write ab instead of $a * b$

We introduce some examples for polynomials with coefficients in the field of two elements and give the simplified notation of it.

$$(ex.1)p_1(X) = 1bX^3 + 0bX^2 + 1bX^1 + 0bX^0 = X^3 + X$$

$$(ex.2)p_2(X) = 1bX^2 + 1bX^1 + 1bX^0 = X^2 + X^1 + 1b$$

Any code word, represented by $n+k$ bit can be mapped to a polynomial of order $n+k-1$ with coefficients in the field of two elements. We use the intuitive mapping of the bits i.e.

$$C(X) = d_n -_1 X^(k+n-1) + d_n -_2 X^(k+n-2) + ...$$
$$+d_2X^(k+2) + d_1X^(k+1) + d_0X^k + r_k -_1 X^(k-1) + r\_k_2X^(k-2) + ...r_1X + r_0$$

$$C(X) = X^k(d_n -_1 X^(n-1) + d_(n-2)X^(n-2) + ...$$
$$+d_2X^2 + d_1X^1 + d_0) + r_(k-1)X^(k-1) + r_(k-2)Xk^-2 + ...r_1X + r_0$$

$$C(X) = X^kD(X) \oplus R(X)$$

This mapping is one-to-one.

A certain space CRCG of Cyclic Redundant Code Polynomials is defined to be a multiple of a given Generator Polynomial $G(X) = X^k + g_k - 1X^(k-1) + g_(k-2)X^(k-2) + ... + g_2X^2 + g_1X + g_0$. By definition, for any code polynomial C(X) in CRCG there is a polyno-mial M(X) with

$$C(X) = G(X)M(X)$$

.

For a fixed irreducible (i.e. prime-) polynomial G(X), the mapping M(X) –> C(X) is one-to-one. Now, how are data of a given codeword verified? This is basically a division of polynomials, using the Euclidian Algorithm. In practice, we are not interested in M(X),

but in the remainder of the division, C(X) mod G(X). For a correct code word C, this remainder has to be zero, C(X) mod G(X) = 0. If this is not the case - there is an error in the codeword. Given G(X) has some additional algebraic prop-erties, one can determine the error-location and correct the codeword.

Calculating the code word from the data can also be done with the Euclidian Algo-rithm. For a given data polynomial $D(x) = d_(n-1)X^(n-1) + d_(n-2)X^(n-2) + ... + d_1X^1 + d_0$ and the corresponding code polynomial C(X) we have

$$C(X) = XkD(X) \oplus R(X) = M(X)G(X)$$

Performing the operation „ mod G(X)" on both sides, one obtains

$$0 = C(X)modG(X) = [X^kD(X)]modG(X) \oplus R(X)modG(X)$$

(*)

We denote that the order of the Polynomial R(X) is less than the order of G(X), so the modulo division gives zero with remainder R(X):

$$R(X)modG(X) = R(X)$$

.

For polynomial R(X) with coefficients in the finite field with two elements we have the remarkable property R(X) + R(X) = 0. If we add R(X) on both sides of equation (*) we obtain

$$R(X) = XkD(X)modG(X)$$

.

The important implication is that the redundant part of the requested code can be determined by using the Euclidian Algorithm for polynomials. At present, any CRC calculation method is a more or less sophisticated variation of this basic algorithm.

Up to this point, the propositions on CRC Codes are summarized as follows:

1. The construction principle of CRC Codes is based on polynomials with coef-ficients in the finite field of two elements. The $\oplus$ operation of this field is iden-tical to the binary operation „ xor "(exclusive or)
2. There is a natural mapping of bit-sequences into this space of polynomials.
3. Both calculation and verification of the CRC code polynomial is based on di-vision modulo a given generator polynomial.
4. This generator polynomial has to have certain algebraic properties in order to achieve error-detection and eventually error-correction.

5.12.2 Euclidian Algorithm for Binary Polynomials and Bit-Sequences

Given a Polynomial $P_n(X) = p_nX^n + p_(n-1)X^(n-1) + ... + p_2X^2 + p_1X + p_0$ with coefficients in the finite field of two elements. Let $Q(X) = X^k + q_(k-1)X^(k-1) + q_(k-2)X^(k-2) + ... + q_2X^2 + q_1X + q_0$ be another polynomial of exact order k>0. Let $R_n(X)$ be the remainder of the polynomial division of maximum order $k-1$ and $M_n(X)$ corresponding so that

$$Rn(X) \oplus Mn(X)Q(X) = Pn(X)$$

.

Euclidian Algorithm - Recursive

(Termination of recursion) If $n < k$, then choose $R_n(X) = P_n(X) and M_n = 0.$

(Recursion n+1 –> n) Let $P_n + 1(X)$ be of maximum order $n+1$.

If $n+1 >= k$calculate $P_n(X) = P_(n+1)(X) - p_(n+1)Q(X)X^(n-k+1)$. This polynomial is of maximum order n. Then

$$P_(n+1)(X) mod Q(X) = P_n(X) mod Q(X)$$

.

Proof of recursion

Choose $R_(n+1)(X) = P_(n+1)(X)$ mod $Q(X) and M_(n+1)(X)$ so that $R_(n+1)(X) \oplus M_(n+1)(X)Q(X) = P_(n+1)(X)$.

Then $R_(n+1)(X) - R_n(X) = P_(n+1)(X) - M_(n+1)(X)Q(X) - P_n(X) \oplus M_n(X)Q(X)$.

With $P_(n+1)(X) - P_n(X) = p(n+1)Q(X)X^(n-k+1)$ we obtain

$R_(n+1)(X) - R_n(X) = p_(n+1)Q(X)X^(n-k+1) + M_n(X)Q(X) - M_(n+1)(X)Q(X)$

$R_(n+1)(X) - R_n(X) = Q(X)[p_(n+1)X^(n-k+1) + M_n(X) - M_(n+1)(X)]$

On the left side, there is a polynomial of maximum order k-1. On the right side Q(X) is of exact order k. This implies that both sides are trivial and equal to zero. One obtains

$R_(n+1)(X) = R_n(X)$ (1) $M_(n+1)(X) = M_n(X) + p_(n+1)X^(n-k+1)$ (2)

Example

$P(X) = P_4(X) = X^4 + X^2 + X + 1b$; $Q(X) = X^2 + X + 1b$; $n = 4; k = 2$ $P_3(X) = X^4 + X^2 + X + 1b - 1b(X^2 + X + 1b)X^2 = X^3 + X + 1b.$ $P_2(X) = X^3 + X + 1b - 1bX(X^2 + X + 1b) = X2 + 1b.$ $P_1(X) = X^2 + 1 - 1b(X^2 + X + 1) = X$ $R(X) = P(X)$ mod $Q(X) = R1(X) = P1(X) = X.$

5.12.3 CRC calculation, Variations and Parameter

Based on the Euclidian Algorithm, some variations have been developed in order to improve the calculation performance. All these variations do not improve the capability to detect or correct errors the so-called Hamming Distance of the resulting code is determined only by the generator polynomial. Variations simply optimize for different implementing ALUs.

CRC-Calculation methods are characterized as follows:

1. Rule for Mapping of Data to a bit sequence $(d_(n-1), d_(n-2), d_(n-3), ..., d_1, d_0)$ and the corresponding data polynomial D(X) (standard or reflected data).
2. Generator polynomial G(X)
3. Start value and corresponding Polynomial S(X)
4. Appendix A(X), also called XOR-value for modifying the final result.
5. Rule for mapping the resulting CRC-remainder R(X) to codeword. (Standard or reflected data)

The calculation itself is organized in the following steps

- Map Data to D(X)
- Perform Euclidian Algorithm on $X^k D(X) + X^(n-k-1) S(X) + A(X)$and determine $R(X) = [X^k D(X) + X^(n-k-1) S(X) + A(X)]$ mod $G(X)$
- Map D(X), R(X) to codeword

5.13 CRC Standard Parameters

This section gives a rough overview on the standard parameters that are commonly used for 8-bit, 16-bit and 32-bit CRC calculation.

- CRC result width: Defines the result data width of the CRC calculation.
- Polynomial: Defines the generator polynomial which is used for the CRC algorithm.
- Initial value: Defines the start condition for the CRC algorithm.
- Input data reflected: Defines whether the bits of each input byte are reflected before being processed (see definition be-low).
- Result data reflected: Similar to „Input data reflected ” this parameter de-fines whether the bits of the CRC result are reflect-ed (see definition below). The result is reflected over 8-bit for a CRC8, over 16-bit for a CRC16 and over 32-bit for a CRC32.

- XOR value: This Value is XORed to the final register value before the value is returned as the official check-sum.
- Check: This field is a check value that can be used as a weak validator of implementations of the algorithm. The field contains the checksum obtained when the ASCII values '1' '2' '3' '4' '5' '6' '7' '8' '9' corresponding to values 31h 32h 33h 34h 35h 36h 37h 38h 39h is fed through the specified algorithm.
- Magic check: The CRC checking process calculates the CRC over the entire data block, including the CRC re-sult. An error-free data block will always result in the unique constant polynomial (magic check) -representing the CRC-result XORed with 'XOR value'-regardless of the data block content

Example of magic check: calculation of SAE-J1850 CRC8 (see detailed parameters in PRS_E2EProtocol_CRC_00030) over data bytes 00h 00h 00h 00h:

- CRC generation: CRC over 00h 00h 00h 00h, start value FFh:
  - CRC-result = 59h
- CRC check: CRC over 00h 00h 00h 00h 59h, start value FFh:
  - CRC-result = 3Bh
  - Magic check = CRC-result XORed with 'XOR value': C4h = 3Bh xor FFh

Data reflection: It is a reflection on a bit basis where data bits are written in the reverse order. The formula is:

$$reflect_n(X) \sum_{i=0}^{n-1} x_i 2^{n-i-1}$$

where x is the data and n the number of data bits. E.g. The $reflection_8$ of $2D_{16}$(n =8)

E.g. The $reflection_8$of $2D_{16}$ (n =8) $00101101_2$is $B4_{16}(10110100_2)$

The $reflection_{16}$ of 12345678_{16} (n =16) $(0001001000110100010101100111000_2)$ is $1E6A2C48_{16}$ $(0011110011010100010110001001000_2)$.

The $reflection_{32}$ of $123456789ABCDEF0$ (n=32) $(0001001000110100010101100111000$ $1001101010111100110111011110000_2)$ is $0F7B3D591E6A2C48_{16}$ $(0000111101111011001111010101100100011110011010100010110001001000_2)$.

The $reflection_8$ of $123456789ABCDEF0$ (n =8) $(0001001000110100010101100111$ $1000100110101011110011011110111000_2)$ is $84C2A6E195D3B7F0_{16}$ $(1000010011000010101001101110000110010101110100111011011111110000_2)$.

5.13.1 8-bit CRC calculation

5.13.1.1 8-bit SAE J1850 CRC Calculation

[PRS_E2EProtocol_CRC_00030] ⌈ The Crc_CalculateCRC8() function of the CRC module shall implement the CRC8 routine based on the SAE-J1850 CRC8 Stand-ard:

CRC result width:	8 bits
Polynomial:	1Dh
Initial value:	FFh
Input data reflected:	No
Result data reflected:	No
XOR value:	FFh
Check:	4Bh
Magic check:	C4h

⌋()

[PRS_E2EProtocol_CRC_00052] ⌈ The Crc_CalculateCRC8() function of the CRC module shall provide the following CRC results:

Data bytes (hexadecimal)									CRC
00	00	00	00						59
F2	01	83							37
0F	AA	00	55						79
00	FF	55	11						B8
33	22	55	AA	BB	CC	DD	EE	FF	CB
92	6B	55							8C
FF	FF	FF	FF						74

⌋()

5.13.1.2 8-bit 0x2F polynomial CRC Calculation

[PRS_E2EProtocol_CRC_00042] ⌈ The Crc_CalculateCRC8H2F() function of the CRC module shall implement the CRC8 routine based on the generator polynomial 0x2F:

CRC result width:	8 bits
Polynomial:	2Fh
Initial value:	FFh
Input data reflected:	No
Result data reflected:	No

XOR value:	FFh
Check:	DFh
Magic check:	42h

⌋()

[PRS_E2EProtocol_CRC_00053] ⌈ The Crc_CalculateCRC8H2F() function of the CRC module shall provide the following CRC results:

Data bytes (hexadecimal)									CRC
00	00	00	00						12
F2	01	83							C2
0F	AA	00	55						C6
00	FF	55	11						77
33	22	55	AA	BB	CC	DD	EE	FF	11
92	6B	55							33
FF	FF	FF	FF						6C

⌋()

5.13.2 16-bit CRC calculation

5.13.2.1 16-bit CCITT-FALSE CRC16

[PRS_E2EProtocol_CRC_00002] ⌈ The CRC module shall implement the CRC16 routine based on the CCITT-FALSE CRC16 Standard: Note concerning the standard document [8]: The computed FCS is equal to CRC16 XOR FFFFh when the frame is built (first complement of the CCITT-FALSE CRC16). For the verification, the CRC16 (CCITT-FALSE) is computed on the same data + FCS, and the resulting value is always 1D0Fh. Note that, if during the verification, the check would have been done on data + CRC16 (i.e. FCS XOR FFFFh) the resulting value would have been 0000h that is the

CRC result width:	16 bits
Polynomial:	1021h
Initial value:	FFFFh
Input data reflected:	No
Result data reflected:	No
XOR value:	0000h
Check:	29B1h
Magic check:	0000h

⌋(*SRS_LIBS_08525*)

[PRS_E2EProtocol_CRC_00054] ⌈ The Crc_CalculateCRC16() function of the CRC module shall provide the following CRC results:

Data bytes (hexadecimal)									CRC
00	00	00	00						84C0
F2	01	83							D374
0F	AA	00	55						2023
00	FF	55	11						B8F9
33	22	55	AA	BB	CC	DD	EE	FF	F53F
92	6B	55							0745
FF	FF	FF	FF						1D0F

⌋()

5.13.3 32-bit CRC calculation

5.13.3.1 32-bit Ethernet CRC Calculation

[PRS_E2EProtocol_CRC_00003] ⌈ The CRC module shall implement the CRC32 routine based on the IEEE-802.3 CRC32 Ethernet Standard:

CRC result width:	32 bits
Polynomial:	04C11DB7h
Initial value:	FFFFFFFFh
Input data reflected:	Yes
Result data reflected:	Yes
XOR value:	FFFFFFFFh
Check:	CBF43926h
Magic check*:	DEBB20E3h

***Important note:** To match the magic check value, the CRC must be appended in little endian format, i.e. low significant byte first. This is due to the reflections of the input and the result ⌋(*SRS_LIBS_08525*)

[PRS_E2EProtocol_CRC_00055] ⌈ The Crc_CalculateCRC32() function of the CRC module shall provide the following CRC results:

Data bytes (hexadecimal)									CRC
00	00	00	00						2144DF1C
F2	01	83							24AB9D77
0F	AA	00	55						B6C9B287
00	FF	55	11						32A06212

33	22	55	AA	BB	CC	DD	EE	FF	B0AE863D
92	6B	55							9CDEA29B
FF	FF	FF	FF						FFFFFFFF

⌋()

5.13.3.2 32-bit 0xF4ACFB13 polynomial CRC calculation

This 32-bit CRC function is described in [14]. It has an advantage with respect to the Ethernet CRC - it has a Hamming Distance of 6 up to 4kB. **[PRS_E2EProtocol_CRC_00056]** ⌈ The CRC module shall implement the CRC32 routine using the 0x1'F4'AC'FB'13 (0xF4'AC'FB'13) polynomial:

CRC result width:	32 bits
Polynomial:	F4'AC'FB'13h
IInitial value:	FFFFFFFFh
Input data reflected:	Yes
Result data reflected:	Yes
XOR value:	FFFFFFFFh
Check:	16'97'D0'6Ah
Magic check*:	90'4C'DD'BFh
Hamming distance:	6, up to 4096 bytes (including CRC)

***Important note:** To match the magic check value, the CRC must be appended in little endian format, i.e. low significant byte first. This is due to the reflections of the input and the result.

⌋()

There are three notations for encoding the polynomial, so to clarify, all three notations are shown:

1	Polynomial as binary	0001'0100'0010'1111' 0000'1110'0001'1110' 1011'1010'1001'1110' 1010'0011'0110'1001' 0011
2	Normal representation with high bit	01'42'F0'E1'EB'A9 EA'36'93h
3	Normal representation	42'F0'E1'EB'A9'EA'36'93h
4	Reversed reciprocal representation (=Koopman representation)	A1'78'70'F5'D4'F5'1B'49h

Notes:

1. Normal representation with high bit = hex representation of polynomial as bi-nary
2. Normal representation with high bit = Koopman representation * 2 + 1

[PRS_E2EProtocol_CRC_00057] ⌈ The Crc_CalculateCRC32P4() function of the CRC module shall provide the following CRC results:

Data bytes (hexadecimal)									CRC
00	00	00	00						6FB32240h
F2	01	83							4F721A25h
0F	AA	00	55						20662DF8h
00	FF	55	11						9BD7996Eh
33	22	55	AA	BB	CC	DD	EE	FF	A65A343Dh
92	6B	55							EE688A78h
FF	FF	FF	FF						FFFFFFFFh

⌋(*SRS_LIBS_08525*)

5.13.4 64-bit CRC calculation

5.13.4.1 64-bit ECMA polynomial CRC calculation

This 62-bit CRC function is described in [15]. It has a good hamming distance of 4, for long data (see below).

[PRS_E2EProtocol_CRC_00062] ⌈ The CRC module shall implement the CRC64 routine using the polynomial: 0x1'42'F0'E1'EB'A9'EA'36'93 (0x42'F0'E1'EB'A9'EA'36'93)

CRC result width:	64 bits
Polynomial:	42'F0'E1'EB'A9'EA'36'93h
Initial value:	FFFFFFFFFFFFFFFFh
Input data reflected:	Yes
Result data reflected:	Yes
XOR value:	FFFFFFFFFFFFFFFFh
Check:	99'5D'C9'BB'DF'19'39'FAh
Magic check*:	49'95'8C'9A'BD'7D'35'3Fh
Hamming distance:	4, up to almost 8 GB

***Important note:** To match the magic check value, the CRC must be appended in little endian format, i.e. low significant byte first. This is due to the reflections of the input and the result. ⌋(*SRS_LIBS_08525*)

There are three notations for encoding the polynomial, so to clarify, all three nota-tions are shown:

1	Polynomial as binary	0001’0100’0010’1111’ 0000’1110’0001’1110’ 1011’1010’1001’1110’ 1010’0011’0110’1001’ 0011
2	Normal representation with high bit	01’42’F0’E1’EB’A9’EA’36’93h
3	Normal representation	42’F0’E1’EB’A9’EA’36’93h
4	Reversed reciprocal representation (=Koopman representation)	A1’78’70’F5’D4’F5’1B’49h

Notes:

1. Normal representation with high bit = hex representation of polynomial as bi-nary
2. Normal representation with high bit = Koopman representation * 2 + 1

[PRS_E2EProtocol_CRC_00063] ⌈ The Crc_CalculateCRC64() function of the CRC module shall provide the following CRC results:

Data bytes (hexadecimal)									CRC
00	00	00	00						F4A586351E1B9F4Bh
F2	01	83							319C27668164F1C6h
0F	AA	00	55						54C5D0F7667C1575h
00	FF	55	11						A63822BE7E0704E6h
33	22	55	AA	BB	CC	DD	EE	FF	701ECEB219A8E5D5h
92	6B	55							5FAA96A9B59F3E4Eh
FF	FF	FF	FF						FFFFFFFF00000000h

⌋ *(SRS_LIBS_08525)*

6 E2E API specification

This chapter defines an abstract API of E2E supervision. E2E is supposed to be invoked by middleware, but the results of checks are visible to the application, so this chapter is split into two parts.

6.1 API of middleware to applications

The API to the applications is imposed by the middleware (e.g. RTE or ARA). E2E provides an additional output object providing E2E check results.

[PRS_E2EProtocol_USE_00321] ⌈ The middleware shall provide, for each exchanged dataRecord, a set of functions:

- middleware_send(in dataRecord)
- middleware_receive(out dataRecord, out e2eResult)

⌋(*RS_E2E_08534*)

[PRS_E2EProtocol_00322] ⌈ The e2eResult shall contain pieces of information:

- e2eStatus: Profile-independent status of the reception on one single Data in one cycle. Possible values are: OK, REPEATED, WRONGSEQUENCE, NOTAVAILABLE, NONEWDATA.
- e2eState: Status of the communication channel exchanging the data. Possible values are: VALID, DEINIT, NODATA, INIT, INVALID.

⌋(*RS_E2E_08534*)

6.2 API of E2E

The E2E interface is independent from any middleware. It is supposed to be used for vsomeip, but it could work for any other middleware or software services, e.g. a database requesting to protect its data.

The interface between the middleware and E2E operates on the serialized data, where: E2E adds E2E header (sender side) and E2E check E2E header (receiver side).

[PRS_E2EProtocol_00323] ⌈ E2E protocol shall provide the following interface:

- E2E_check(in dataID, inout serializedData)
- E2E_protect(in dataID, inout serializedData): e2eResult

where:

- dataID is a unique identifier of the exchanged data/information. In case of multiple instantiation, each single instance gets typically a separate dataID, even if the same type of information is transmitted
- serializedData - vector/array of serialized data, where E2E header is located, next to serialized data
- e2eResult - result of E2E checks, see previous section for the definition.

⌋(*RS_E2E_08534*)

The middleware is responsible to provide an adaptation to E2E functional interface.

[PRS_E2EProtocol_00318] ⌈ The middleware shall determine the DataID of the currently exchanged information. ⌋(*RS_E2E_08534*)

For example, in case of vsomeip, it needs to determine DataID based on serviceid/eventid/instanceid tuple.

[PRS_E2EProtocol_00319] ⌈ The middleware invoke E2E functions providing them the DataID together with the data. ⌋*(RS_E2E_08534)*

[PRS_E2EProtocol_00320] ⌈ On the receiver side, the middleware shall provide the e2eResult determined by E2E to the receiver. ⌋*(RS_E2E_08534)*

7 Configuration Parameters

E2E supervision has the following configuration options for each protected data. Note that it is platform-specific how middleware associates a middleware communication channel (e.g. I-PDU or event) with E2E communication protection.

For each DataID, which uniquely represents data exchanged, there is a set of configuration options.

[PRS_E2EProtocol_00324] ⌈ The following options shall be available for a E2E-protected data:

Parameters	Profile	Description
dataID	1, 4, 5, 6, 7, 11	This represents a unique numerical identifier. Note: ID is used for protection against masquerading. The details concerning the maximum number of values (this information is specific for each E2E profile) applicable for this attribute are controlled by a semantic constraint that depends on the category of the EndToEnd-Protection. dataId is used as a unique identifier of a configuration object. One dataId can appear only once in the configuration.
profileName	all	This represents the identification of the concrete E2E profile. Possible profiles: 1 (only CP), 2 (only CP), 4, 5, 6, 7, 11, 22.
dataLength	1, 2, 5, 11, 22	For fixed size data: length of data in bits.
minDataLength	4, 6, 7	For variable size data: minimum length of data in bits.
maxDataLength	4, 6, 7	For variable size data: maximum length of data in bits.
dataIdList	2, 22	List of 16 dataID values, where a a different value is transmitted depending on counter value.
dataIdMode	1, 11	This attribute describes the inclusion mode that is used to include the two-byte Data ID in E2E communication protection.
offset	2, 4, 5, 6, 7, 22	Offset of the E2E header in the Data[] array in bits.
counterOffset	1, 11	Offset of the counter in the Data[] array in bits.
crcOffset	1, 11	Offset of the CRC in the Data[] array in bits.
dataIdNibbleOffset	1, 11	Offset of the dataID nibble in the Data[] array in bits.
maxDeltaCounter	4, 5, 6, 7, 11, 12	Maximum allowed difference between the counter value of the current message and the previous valid message.
Parameters of legacy profiles		
maxDeltaCounterInit	1, 2	Initial maximum allowed gap between two counter values of two consecutively received valid Data. The maxDeltaCounter is increased on each reception try but only reset when receiving a valid message. This is to compensate for and tolerate lost messages.
maxNoNewOrRepeatedData	1, 2	The maximum amount of missing or repeated Data which the receiver does not expect to exceed under normal communication conditions.
syncCounterInit	1, 2	The number of messages required for validating the consistency of the counter after exceeding the maxNoNewOrRepeatedData threshold.

profileBehavior	1, 2	Mapping of specific profile status values to unified profileStatus. False: legacy behavior, as before AUTOSAR Classic Platform Release 4.2,True: mapping according to new profiles (profile 4 and newer) interpretation of status, introduced in AUTOSAR Classic Platform Release 4.2.
Parameters of E2E State Machine		
windowSize	Size of the monitoring window (ProfileStatus circular buffer) for the state machine.	
maxErrorStateInit	Maximum number of checks in which ProfileStatus equal to E2E_P_ERROR was determined, within the last WindowSize checks, for the state E2E_SM_INIT.	
maxErrorStateInvalid	Maximum number of checks in which ProfileStatus equal to E2E_P_ERROR was determined, within the last WindowSize checks, for the state E2E_SM_INVALID.	
maxErrorStateValid	Maximum number of checks in which ProfileStatus equal to E2E_P_ERROR was determined, within the last WindowSize checks, for the state E2E_SM_VALID.	
minOkStateInit	Minimum number of checks in which ProfileStatus equal to E2E_P_OK was determined, within the last WindowSize checks, for the state E2E_SM_INIT.	
minOkStateInvalid	Minimum number of checks in which ProfileStatus equal to E2E_P_OK was determined, within the last WindowSize checks, for the state E2E_SM_INVALID.	
minOkStateValid	Minimum number of checks in which ProfileStatus equal to E2E_P_OK was determined, within the last WindowSize checks, for the state E2E_SM_VALID.	

⌋(*RS_E2E_08534*)

8 Protocol usage and guidelines

This chapter contains requirements on usage of E2E Supervision when designing and implementing safety-related systems, which are depending on E2E communication protection and which are not directly related to some specific functionality. Note that chapter 5 also provides several requirements on usage.

8.1 Periodic use of E2E check

[PRS_E2EProtocol_USE_00325] ⌈ The E2E check function shall be invoked at least once within FTTI (FTTI is for the safety goals from which the requirements for this E2E checks are derived). ⌋(*RS_E2E_08528*)

8.2 Error handling

The E2E Supervision itself does not handle detected communication errors. It only detects such errors for single received data elements and returns this information to the callers (e.g. SW-Cs), which have to react appropriately. A general standardization of the error handing of an application is usually not possible.

[PRS_E2EProtocol_USE_00235] ⌈ The user (caller) of E2E Supervision, in particular the receiver, shall provide the error handling mechanisms for the faults detected by the E2E Supervision. ⌋(*RS_E2E_08528*)

8.3 Maximal lengths of Data, communication buses

The length of the message and the achieved hamming distance for a given CRC are related. To ensure the required diagnostic coverage the maximum length of data elements protected by a CRC needs to be selected appropriately. The E2E profiles are intended to protect inter-ECU communication with lengths as listed in the table below

E2E Profile	Max applicable length including control fields for inter-ECU communication
E2E Profile 1 and 11	32B
E2E Profile 2 and 22	32B
E2E Profile 4	4 kB
E2E Profile 5	4 kB
E2E Profile 6	4 kB
E2E Profile 7	4 MB

In E2E Profiles 1 and 2, the Hamming Distance is 2, up to the given lengths. Due to 8 bit CRC, the burst error detection is up to 8 bits.

[PRS_E2EProtocol_UC_00051] ⌈ In case of inter-ECU communication over FlexRay, the length of the complete Data (including application data, CRC and counter) protected by E2E Profile 1 or E2E Profile 2 should not exceed 32 bytes. ⌋*(RS_E2E_08528)*

This requirement only contains a reasonable maximum length evaluated during the design of the E2E profiles. The responsibility to ensure the adequacy of the implemented E2E communication protection using E2E Supervision for a particular system remains by the user.

[PRS_E2EProtocol_UC_00061] ⌈ In case of CAN or LIN the length of the complete data element (including application data, CRC and counter) protected by E2E Profile 1 should not exceed 8 bytes. ⌋*(RS_E2E_08528)*

[PRS_E2EProtocol_UC_00351] ⌈ The length of the complete Data (including application data and E2E header) protected by E2E Profile 4, 5 or 6 shall not exceed 4kB. ⌋ *(RS_E2E_08528)*

[PRS_E2EProtocol_UC_00316] ⌈ The length of the complete Data (including application data and E2E header) protected by E2E Profile 7 shall not exceed 4MB. ⌋ *(RS_E2E_08528)*

[PRS_E2EProtocol_UC_00236] ⌈ When using E2E Supervision, the designer of the functional or technical safety concept of a particular system using E2E Supervision shall evaluate the maximum permitted length of the protected Data in that system, to ensure an appropriate error detection capability. ⌋*(RS_E2E_08539)*

Thus, the specific maximum lengths for a particular system may be shorter (or maybe in some rare cases even longer) than the recommended maximum applicable lengths defined for the E2E Profiles.

If the protected data length exceeds the network bus frame limit (or payload limit), the data can be segmented on the sender side after the E2E communication protection, and be assembled on the receiver side before the E2E evaluation. The possible faults happening during segmentation/desegmentation can be considered as "corruption of information".

[PRS_E2EProtocol_UC_00170] ⌈ When designing the functional or technical safety concept of a particular system any user of E2E shall ensure that the transmission of one undetected erroneous data element in a sequence of data elements between sender and receiver, protected with profile 1, 11, 2, 22, will not directly lead to the violation of a safety goal of this system. ⌋*()*

In other words, SW-C shall be able to tolerate the reception of one erroneous data element, which error was not detected by the E2E Supervision. What is not required is that an SW-C tolerates two consecutive undetected erroneous data elements, because it is enough unlikely that two consecutive Data are wrong AND that for both Data the error remains undetected by the E2E Supervision.

When using LIN as the underlying communication network the residual error rate on protocol level is several orders of magnitude higher (compared to FlexRay and CAN) for the same bit error rate on the bus. The LIN checksum compared to the protocol CRC of FlexRay (CRC-24) and CAN (CRC-15) has different properties (e.g. hamming distance) resulting in a higher number of undetected errors coming from the bus (e.g. due to EMV). In order to achieve a maximum allowed residual error rate on application level, different error detection capabilities of the application CRC may be necessary, depending on the strength of the protection on the bus protocol level.

[PRS_E2EProtocol_UC_00237] ⌈ Any user of E2E Supervision shall ensure, that within one implementation of a communication network every safety-related Data, protected by E2E Supervision, has a unique Data ID (E2E Profiles 1, 4, 5, 6, 7, 11) or a unique DataIDList[] (E2E Profiles 2, 22). ⌋*(RS_E2E_08528)*

E2E Profile 1 with E2E_P01DataIDMode = E2E_P01_DATAID_BOTH and E2E Profile 11 with E2E_P11DataIDMode = E2E_P11_DATAID_BOTH uses an implicit 2-byte Data ID, over which CRC8 is calculated. As a CRC over two different 2-byte numbers may result with the same CRC, some precautions must be taken by the user. See PRS_E2EProtocol_USE_00072 and PRS_E2EProtocol_USE_00073.