HL Mando X

UDS Protocol

$22 & $2E

Group C

YongSeong Lee, HuiJun Song, HyunJun Cho

Agenda

Project Goals

Goal # 1

UDS, DID ($22, $2E) Services Implementation

Goal # 2

Write SW Unit Design spec(SUD)

Goal # 3

Write SW Unit Test report (SUT)

Understanding of Embedded & AUTOSAR

Introduction to Embedded System

- Basic Components
- Architecture
- Processors

Introduction to AUTOSAR

- Background
- Architecture
- Methodology

Embedded System

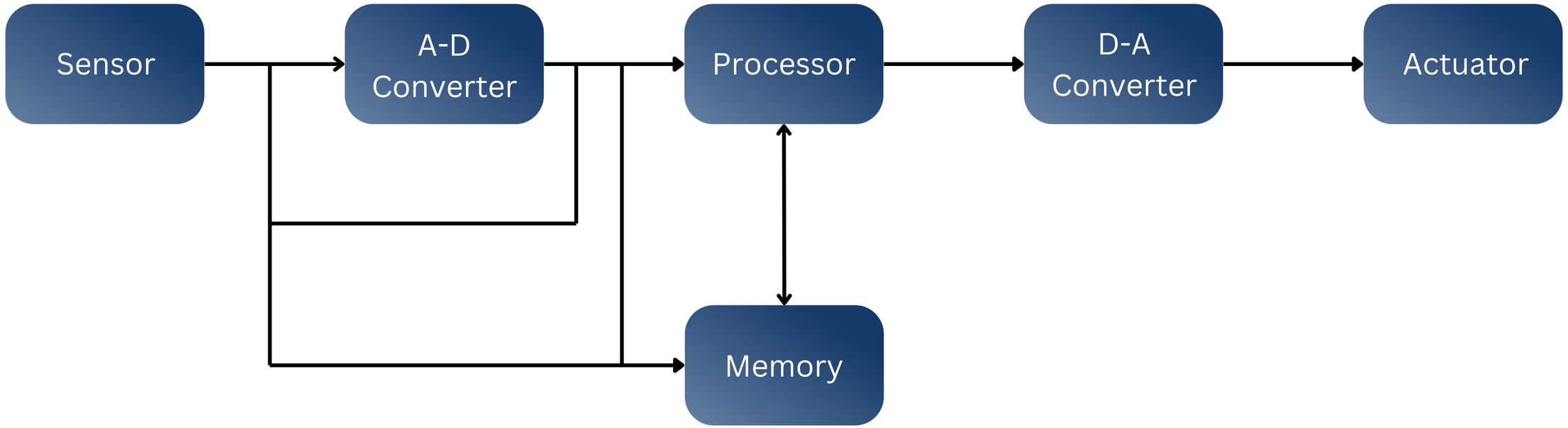
What is Embedded System?

Specialized computer system that is desinged to perform specific task or function

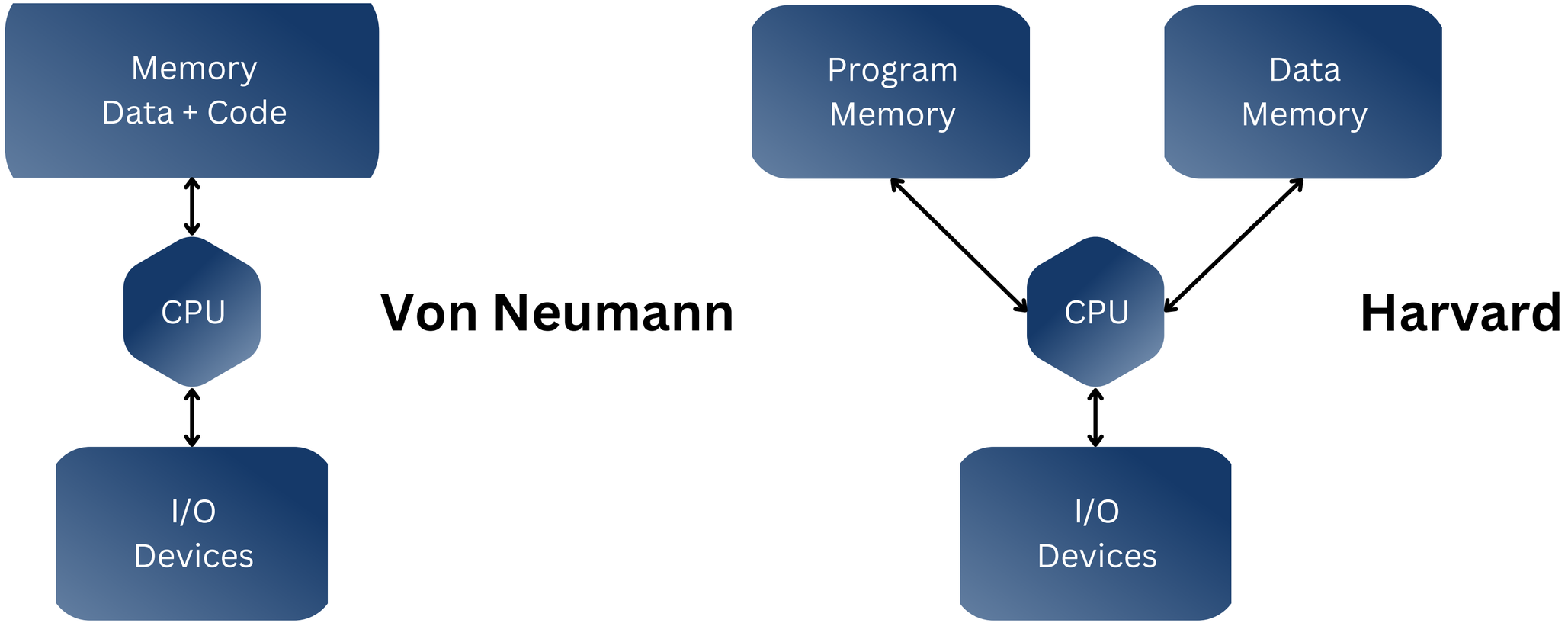
Examples

Washing machines, medical devices
Automotive: ECU, ABS, Airbag, Infotainment

Basic Components

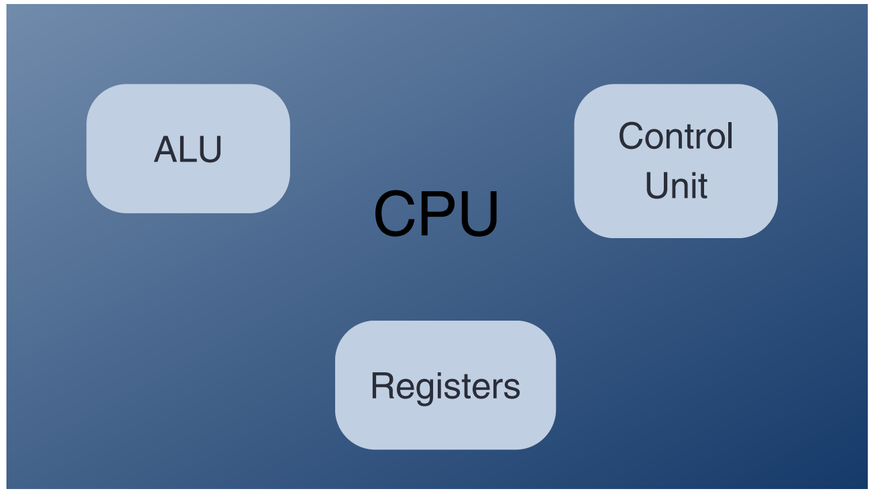


Architecture

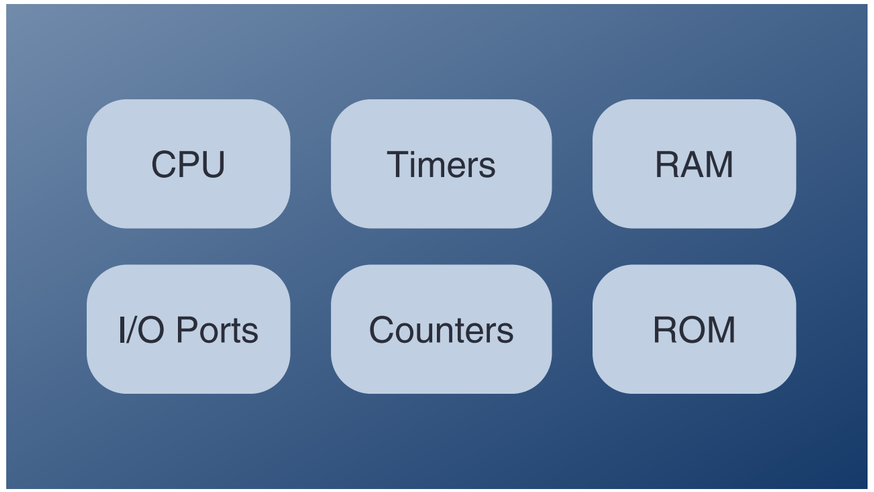


Processors (1)

Microprocessor



Microcontroller



Processor (2)

	Microprocessor	Microcontroller
Function	Process the general task	Both Process and Control the specific task
Memory	No in-built memory	In-built ROM and RAM memories
Application	General purpose	Specific purpose
Complexity	More Complex, Large instructions	Less Complex, Less no. of instructions
Cost	High	Low
Efficiency	Less	More
Architecture	Von Neumann	Harvard

Understanding of Embedded & AUTOSAR

Introduction to Embedded System

- Basic Components
- Architecture
- Processors

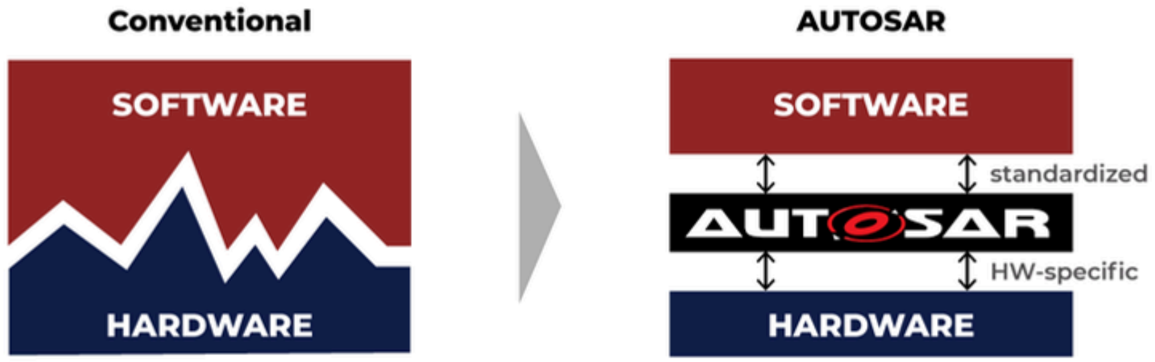
Introduction to AUTOSAR

- Background
- Architecture
- Methodology

Development Backgound

Previous

Hardware manufacturers and software developers were mutually dependent -> Difficult to switch platforms



AUTOSAR (Automotive Open System Architecture)

What is AUTOSAR?

Standardized software architecture for automotive electronic control units (ECUs)

Related Companies




AUTOSAR Benefits (1)

- **Benefit 1**

  Top down approach helps OEMs to design Application Software independent of base software

- **Benefit 2**

  Hierachical structure and moduler design

AUTOSAR Benefits (2)

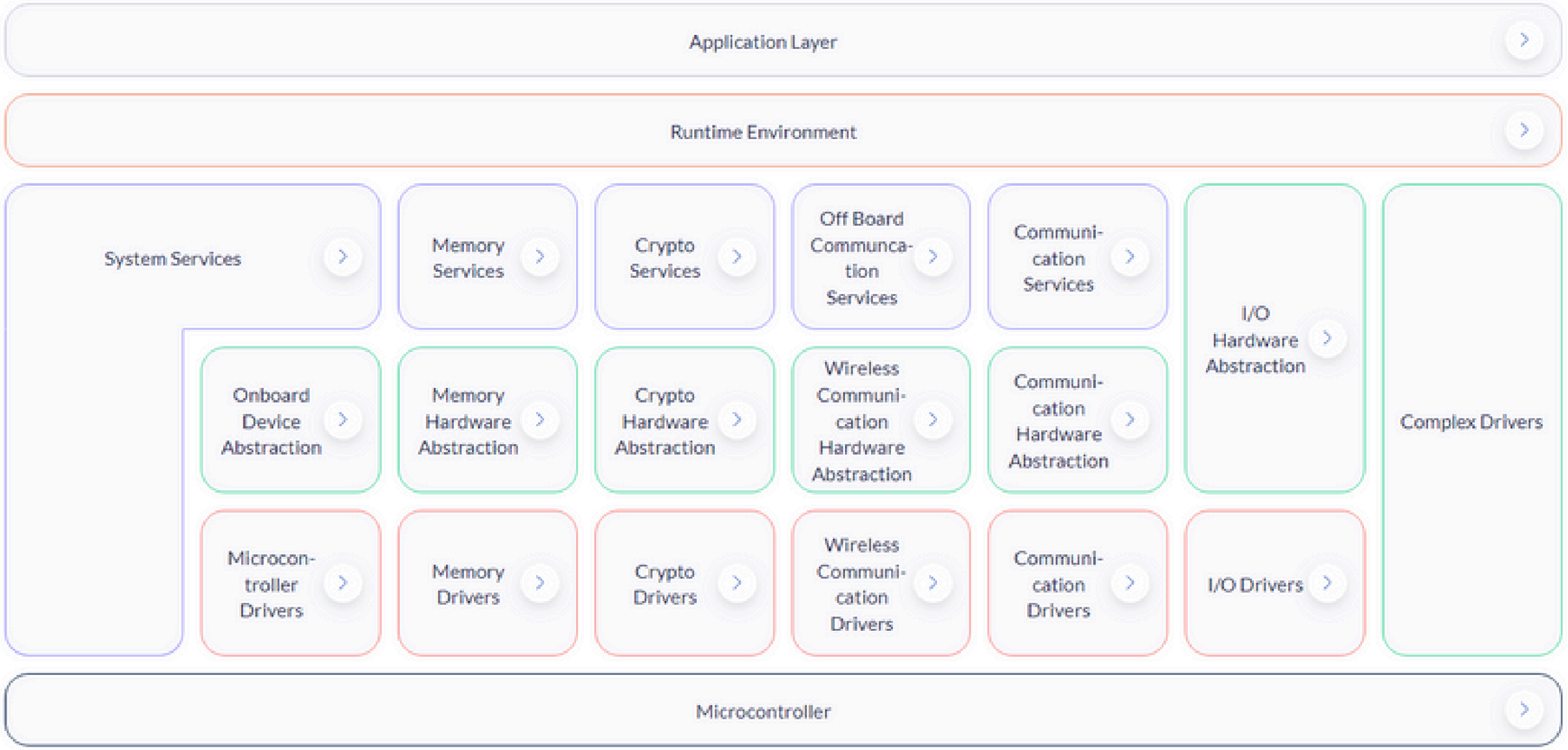
- **Benefit 3**

  Network independent communication mechanisms for applications

- **Benefit 4**

  Userfriendly (Easier for developers)

Architecture



Architecture - BSW Layer

Understanding of CAN Protocol

Introduction to CAN Protocol

- About CAN Protocol
- CAN frame structure
- CAN BUS arbitration

CAN error handling

- CAN error handling

Introduction of CAN protocol

What is CAN?

Controller Area Network

WHY CAN?

The need for a centralized standard communication protocol came because of the increase in the number of electronic devices

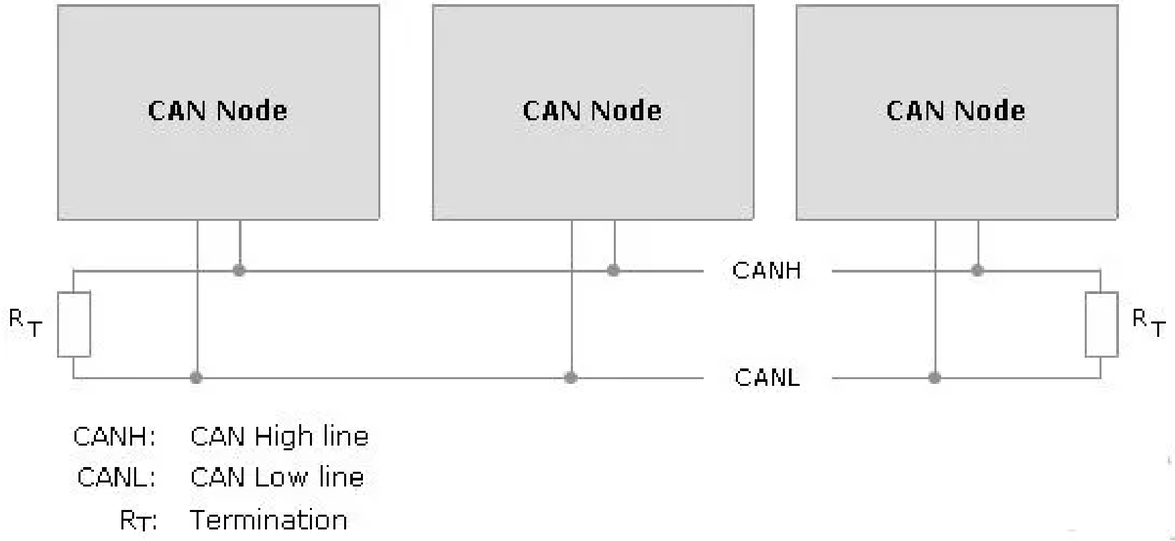
Introduction of CAN protocol

Feature

- Asyncronous serial communication protocol
- Message- oriented
- Error detection and Fault confinement
- High-speed communication
- Broadcast - half-duplex

Introduction of CAN protocol

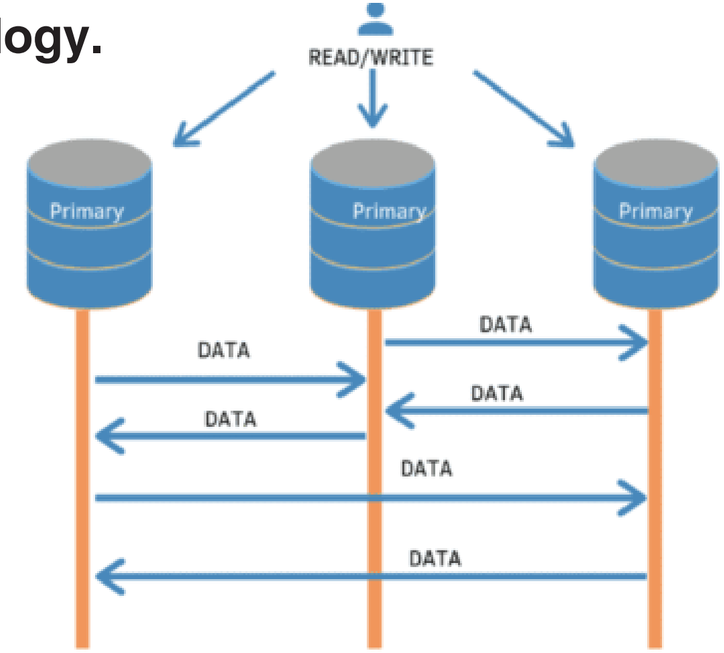
Feature (continued)



CANH: CAN High line
CANL: CAN Low line
R_T: Termination

Multi-Master Architecture

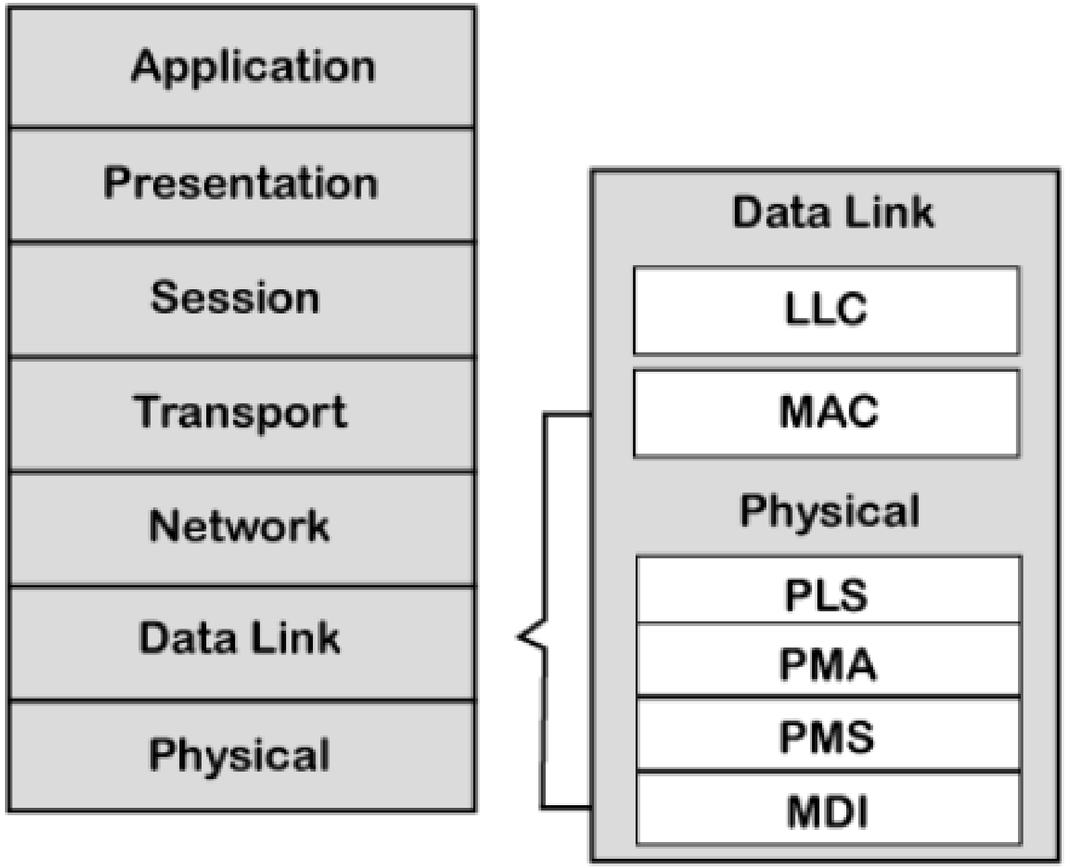
CAN nodes are connected on the CAN bus in the BUS topology.



CAN frame Structure

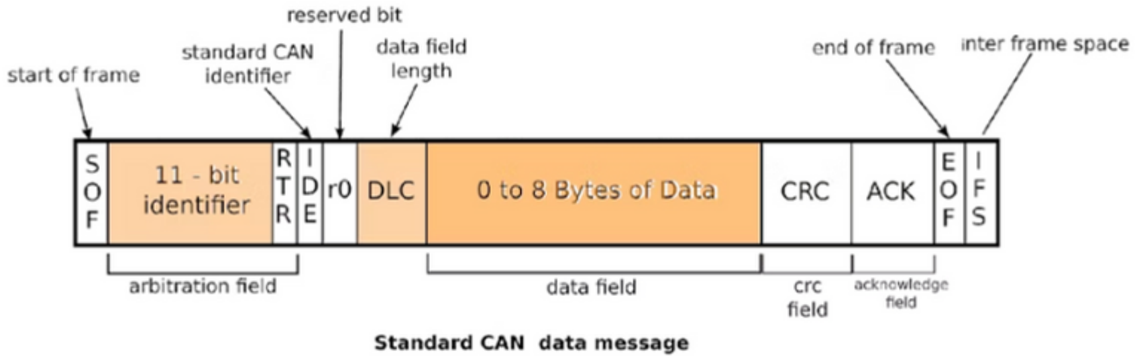
CAN layer architecture

- OSI model partitions the communication system into 7 different layers.
- CAN layered architecture consists of two layers, i.e., data-link layer and physical layer.

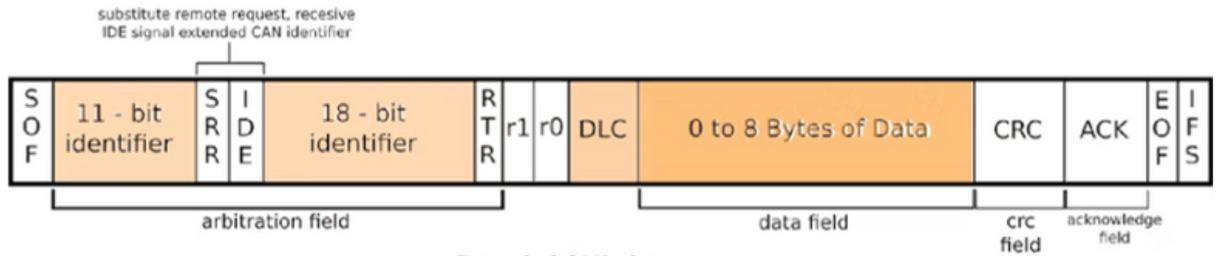


CAN frame Structure

- **CAN framing(standard && extended)**



Standard CAN data message



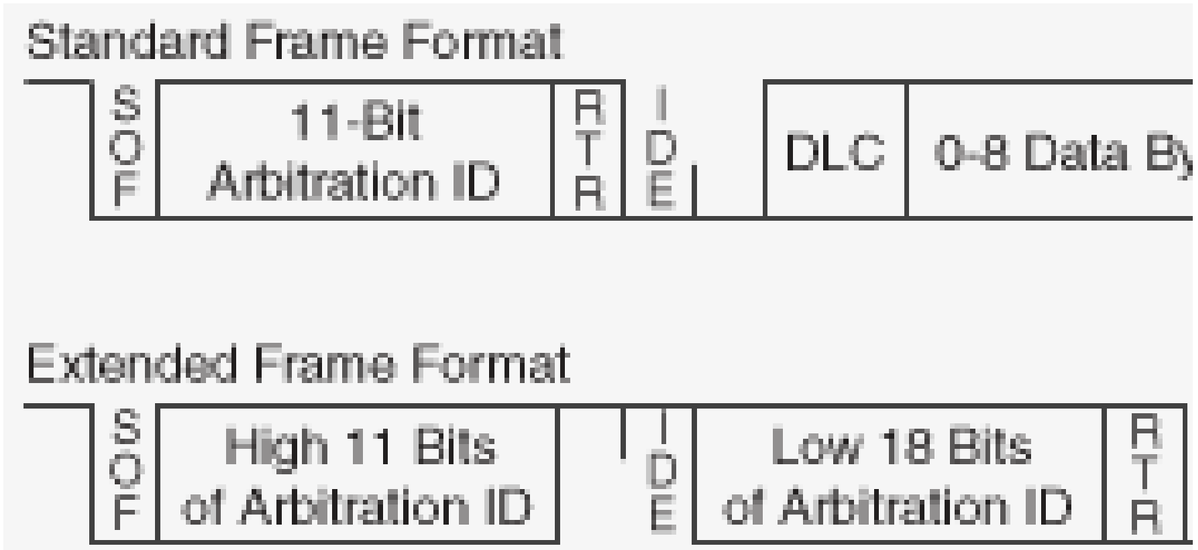
Extended CAN data message

CAN frame Structure

- **Difference standard and extended data frame**

= The arbitration ID

In the standard frame format (also known as 2.0A), the length of the ID is 11 bits. In the extended frame format (also known as 2.0B), the length of the ID is 29 bits.



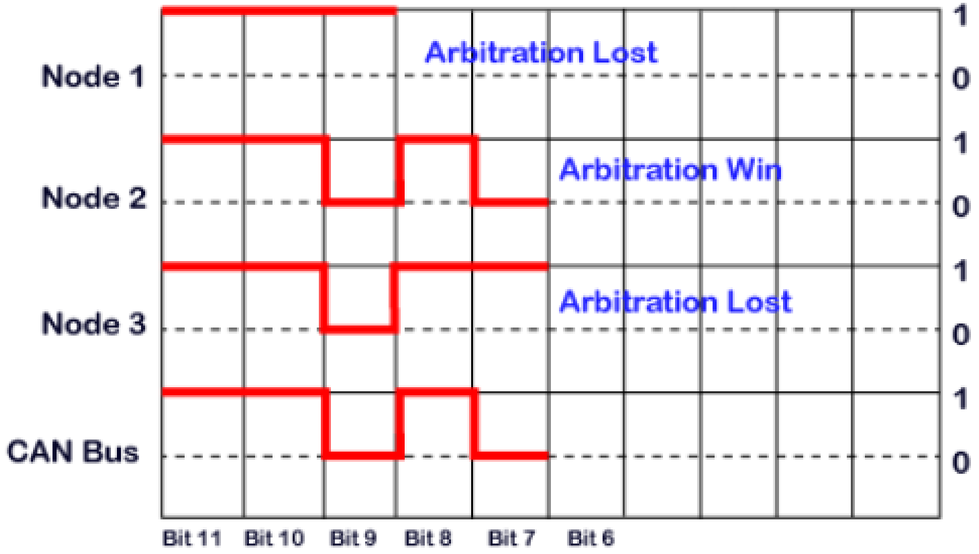
CAN BUS Arbitration

WHY CAN BUS Arbitration

In the CAN protocol, multiple devices can attempt to send data simultaneously, which can lead to collisions. To prevent this, CAN BUS arbitration is necessary.

CAN BUS Arbitration

Example of CAN Arbitration



CAN Node	Identifier (Hex)	Identifier (Binary)
1	0x7F3	11111110011
2	0x6B3	11010110011
3	0x6D9	11011011001

Understanding of CAN Protocol

Introduction to CAN Protocol

- About CAN Protocol
- CAN frame structure
- CAN BUS arbitration

CAN error handling

- CAN error handling

CAN error handling

What are CAN bus errors?

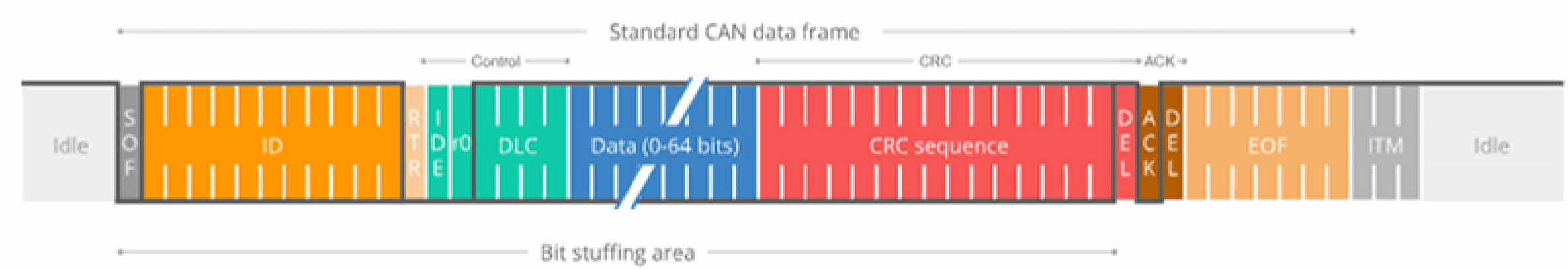
CAN bus errors refers to faults in the CAN bus network, commonly used in automotive and industrial systems for communication between various electronic control units (ECUs).

Why CAN bus errors?

Error handling is vital to the robustness of CAN

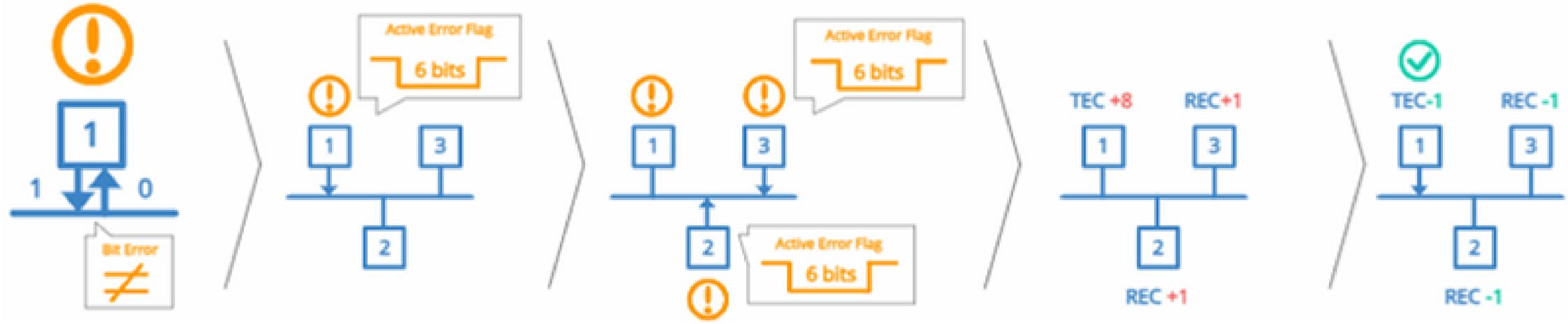
CAN error handling

- **The CAN bus error frame**



CAN error handling

- **How does CAN error handling work**



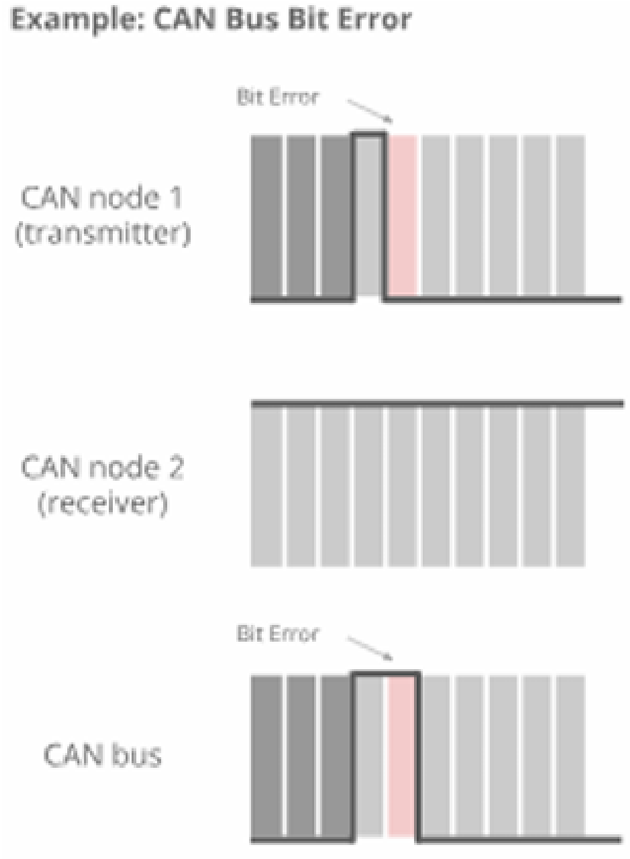
CAN error handling involves two main steps: error detection and error confinement.

CAN error handling

Type of CAN errors

#1 Bit Error

If the bit transmitted and the bit received are not of the same value, a "bit error" occurs.

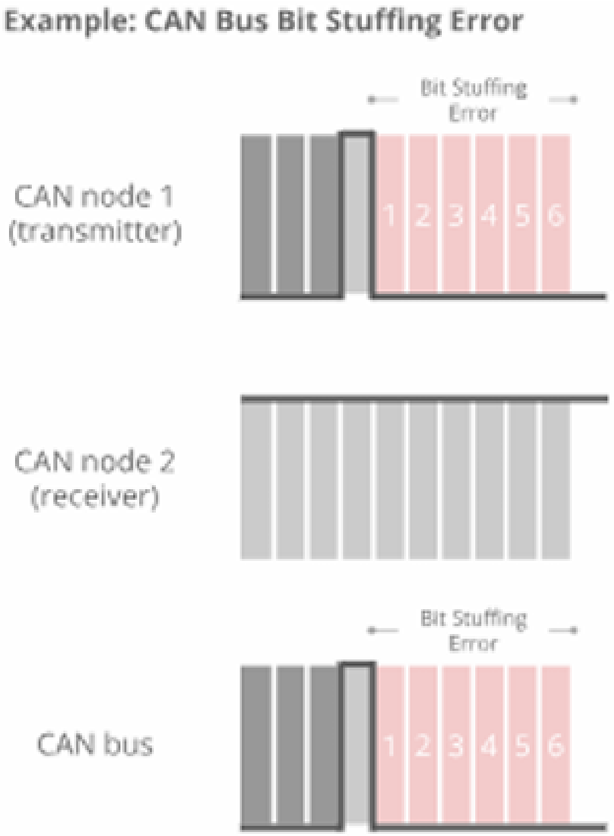


CAN error handling

Type of CAN errors

#2 Bit Stuffing Error

A bit stuffing error occurs if more than five consecutive bits of the same value are detected.

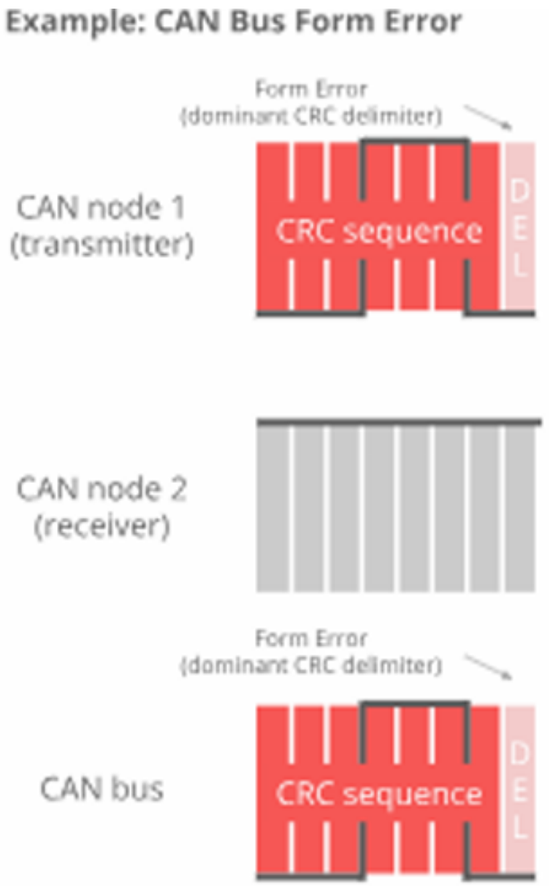


CAN error handling

Type of CAN errors

#3 Form Error

Violation of fixed bit format results in a Form Error. CRC delimeter, Ack delimeter, EOF must be 1s (Recessive) but somehow it is 0 (dominent)

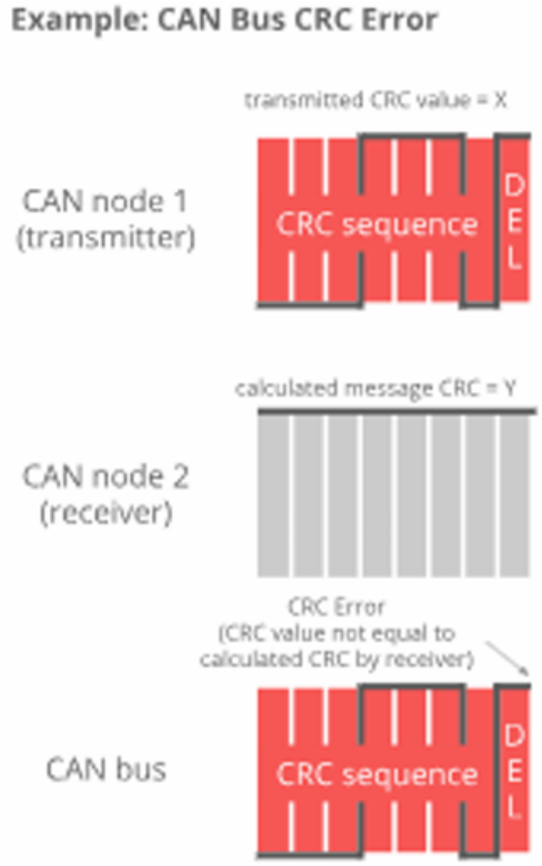


CAN error handling

Type of CAN errors

#4 CRC Error

If the CRC calculation gives different result from what it recieved

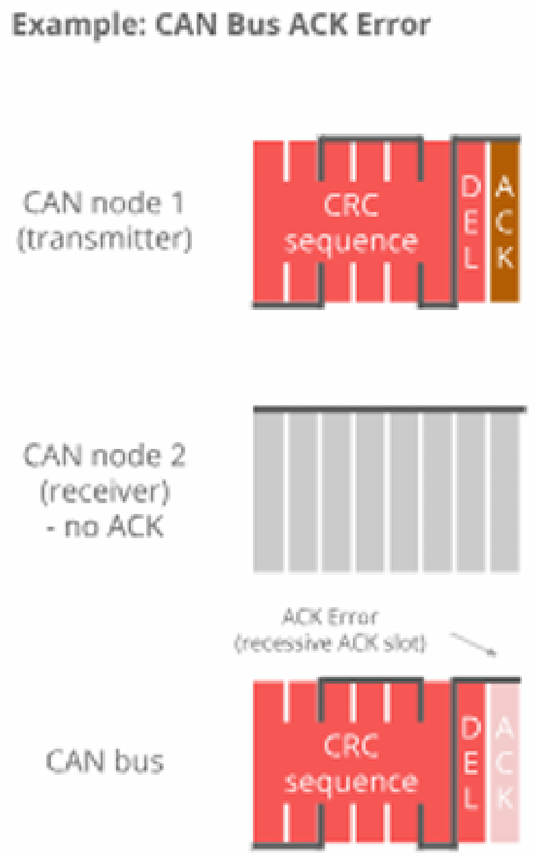


CAN error handling

Type of CAN errors

#5 ACK Error

if the transmitter does not receive this ACK

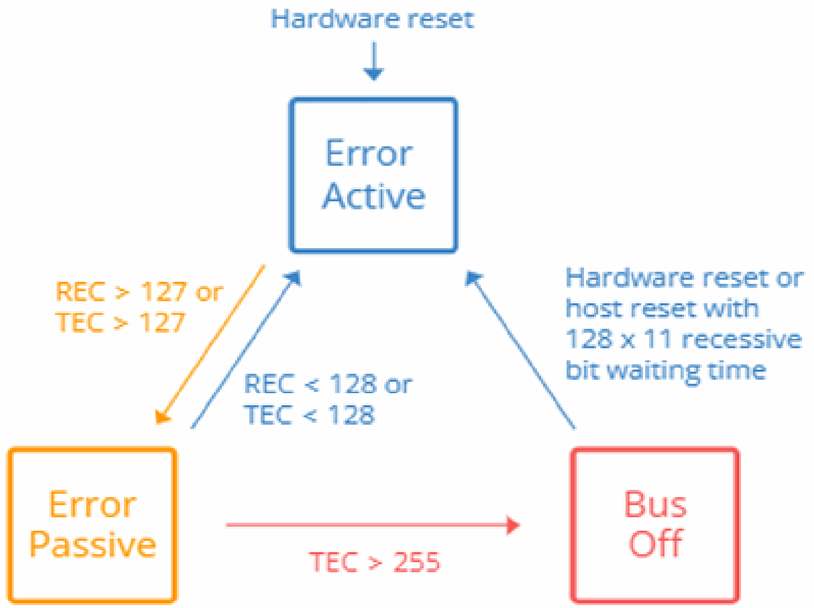


CAN error handling

CAN node states

#1 Error Active

This is the default state of every CAN node, in which it is able to transmit data and raise 'Active Error Flags' when detecting errors



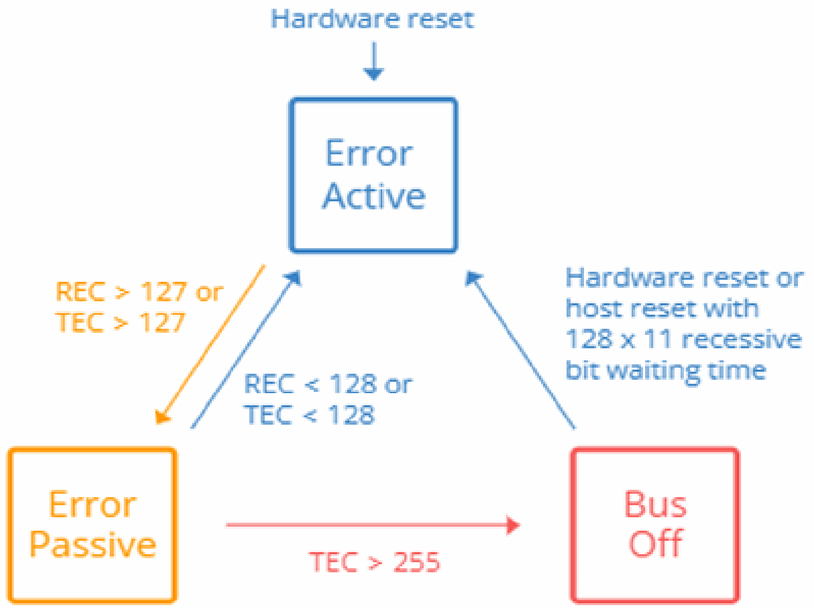
CAN node error tracking via state changes
TEC: Transmit Error Counter, REC: Receive Error Counter

CAN error handling

CAN node states

#2 Error Passive

In this state, the CAN node is still able to transmit data, but it now raises 'Passive Error Flags' when detecting errors



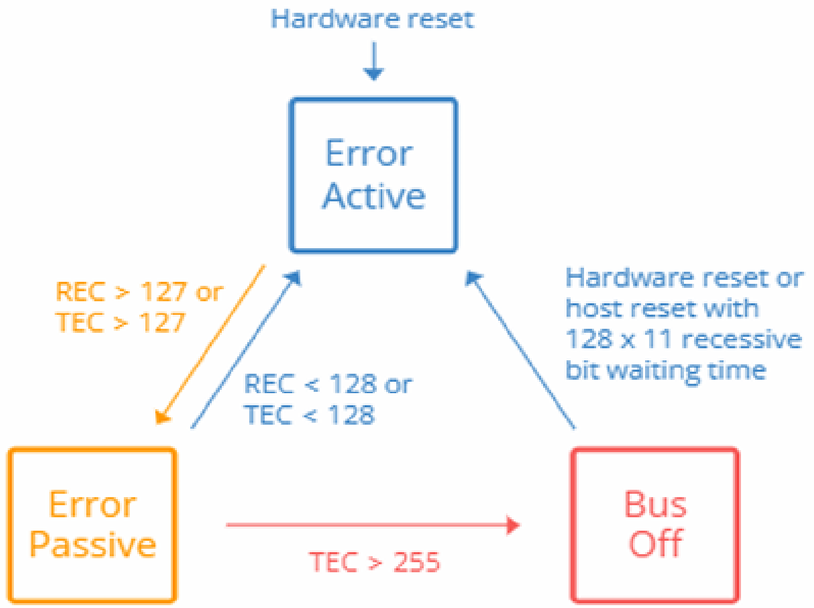
CAN node error tracking via state changes
TEC: Transmit Error Counter, REC: Receive Error Counter

CAN error handling

CAN node states

#3 Bus Off

In this state, the CAN node disconnects itself from the CAN bus and can no longer transmit data or raise error flags



CAN node error tracking via state changes
TEC: Transmit Error Counter, REC: Receive Error Counter

Understanding of UDS protocol

Introduction to UDS

- *UDS request structure*
- *Positive vs Negative responses*

Communication

- Single -frame communication
- Multi-frame communication
- SID

UDS

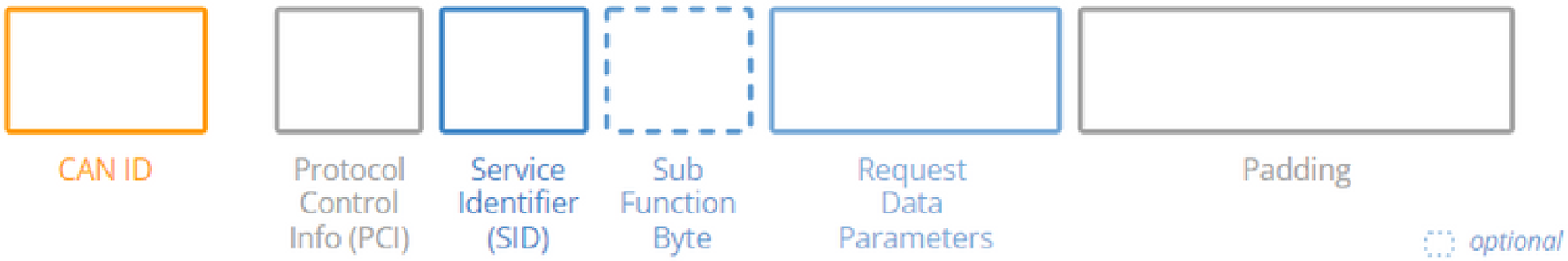
Unified Diagnostic Services (UDS)

A communication protocol used in automotive Electronic Control Units (ECUs) to enable diagnostics, firmware updates, routine testing and more

In practice, UDS communication is performed in a client-server relationship - with the client being a tester-tool and the server being a vehicle ECU.

UDS request structure

- **UDS request frame (using CAN bus as basis)**



UDS is a request based protocol
A diagnostic UDS request on CAN contains various fields

UDS request structure

Protocol Control Information (PCI)

The PCI field is not related to the UDS request itself, but is required for diagnostic UDS requests made on CAN bus

PCI field can be 1–3 bytes long

UDS request structure

UDS Service ID (SID)

Diagnostic Communications M	0x29	0x69	Authentication	Enable more advanced authentication vs. 0x27 (PKI based exchange)
	0x3E	0x7E	Tester Present	Send a "heartbeat" periodically to remain in the current session
	0x83	0xC3	Access Timing Parameters	View/modify timing parameters used in client/server communication
	0x84	0xC4	Secured Data Transmission	Send encrypted data via ISO 15764 (Extended Data Link Security)
	0x85	0xC5	Control DTC Settings	Enable/disable detection of errors (e.g. used during diagnostics)
	0x86	0xC6	Response On Event	Request that an ECU processes a service request if an event happens
	0x87	0xC7	Link Control	Set the baud rate for diagnostic access
Data Transmission	0x22	0x62	Read Data By Identifier	Read data from targeted ECU - e.g. VIN, sensor data values etc.
	0x23	0x63	Read Memory By Address	Read data from physical memory (e.g. to understand software behavior)
	0x24	0x64	Read Scaling Data By Identifier	Read information about how to scale data identifiers
	0x2A	0x6A	Read Data By Identifier Periodic	Request ECU to broadcast sensor data at slow/medium/fast/stop rate
	0x2C	0x6C	Dynamically Define Data Identifier	Define data parameter for use in 0x22 or 0x2A dynamically
	0x2E	0x6E	Write Data By Identifier	Program specific variables determined by data parameters
	0x3D	0x7D	Write Memory By Address	Write information to the ECU's memory
DTCs	0x14	0x54	Clear Diagnostic Information	Delete stored DTCs
	0x19	0x59	Read DTC Information	Read stored DTCs, as well as related information
	0x2F	0x6F	Input Output Control By Identifier	Gain control over ECU analog/digital inputs/outputs
	0x31	0x71	Routine Control	Initiate/stop routines (e.g. self-testing, erasing of flash memory)

UDS request structure

UDS Sub Function Byte

UDS SID (request)	UDS SID (response)	Service	Sub function types
0x10	0x50	Diagnostic Session Control	Diagnostic session type
0x11	0x51	ECU Reset	Reset type
0x27	0x67	Security Access	Security access type
0x28	0x68	Communication Control	Control type
0x3E	0x7E	Tester Present	"Zero sub function"
0x83	0xC3	Access Timing Parameters	Timing parameter access type

The sub-function byte is used as an optional field in the UDS request frame and is utilized only in certain services.

UDS request structure

- **UDS 'Request Data Parameters' - incl. Data Identifier (DID)**

In most UDS services, in addition to the SID and Sub-function byte, there is a DID field for making specific requests, and various Request Data Parameters can be included.

UDS request structure

UDS - standardized data identifiers (DID)

UDS DID (data identifier)	Description
0xF180	Boot software identification
0xF181	Application software identification
0xF182	Application data identification
0xF183	Boot software fingerprint
0xF184	Application software fingerprint
0xF185	Application data fingerprint
0xF186	Active diagnostic session
0xF187	Manufacturer spare part number
0xF188	Manufacturer ECU software number
0xF189	Manufacturer ECU software version

Positive vs Negative responses

Positive Response

a positive response contains the SID + 0x40 as the first byte, followed by the sub-function and the relevant response data.

Positive vs Negative responses

Negative Response

As for the negative response, you receive the error SID (0x7F), the SID of the failed UDS request and the NRC, which will allow you to know further details about the error

Positive vs Negative responses

NRC

- 0x13 :Invalid message length/format
- 0x22:conditions not correct
- 0x31:Request out of range
- 0x33:Security access denied

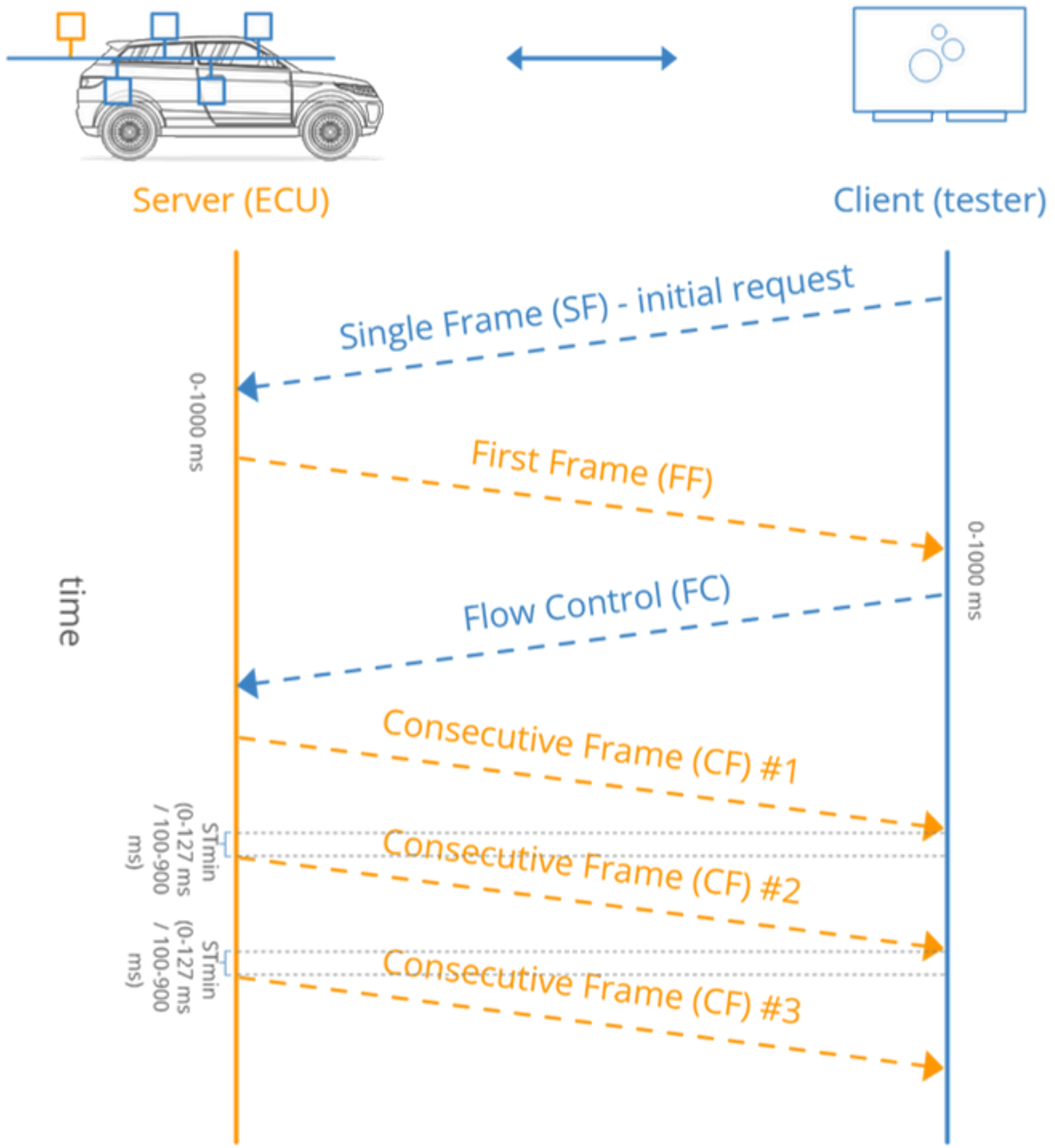
communication

Single-frame communication

a tester tool sends a Single Frame to request data from an ECU. If the response can be contained in a 7-byte payload, the ECU provides a Single Frame response.

Multi-frame communication

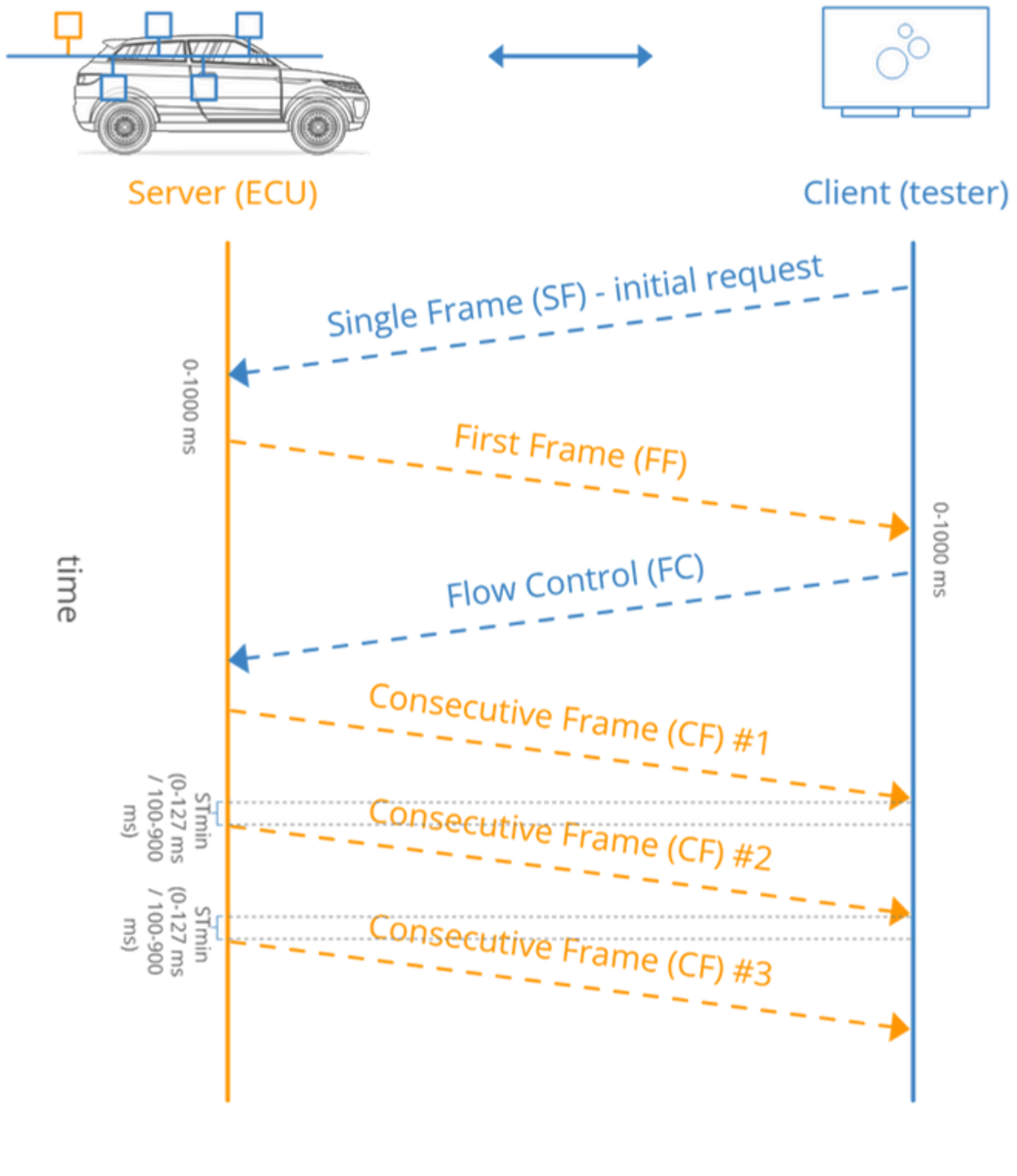
If the payload exceeds 7 bytes, it must be divided into multiple CAN frames.



Multi-frame communication

- **SF (Single Frame) - Contains the UDS request sent to the ECU.**

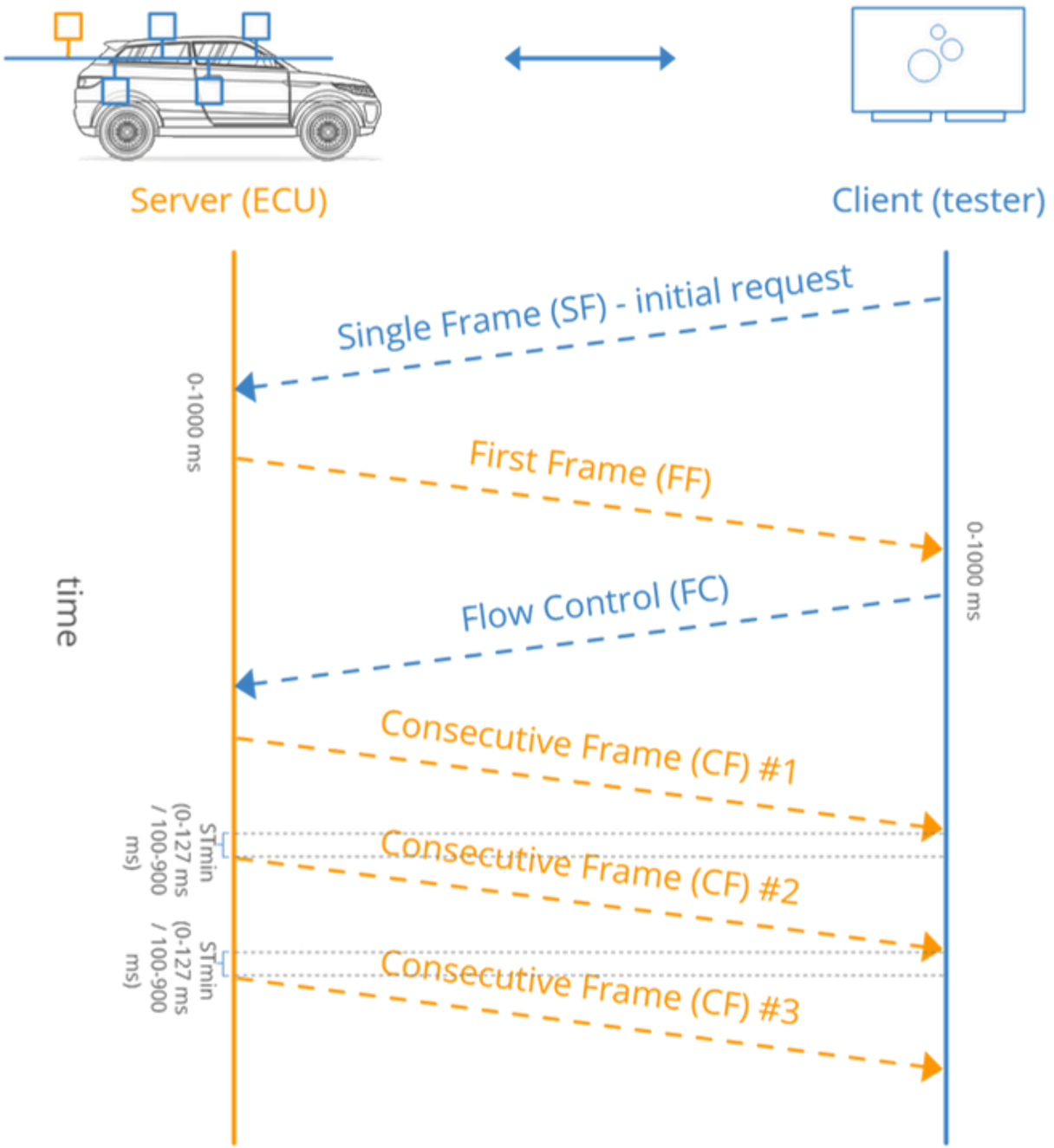
- **FF (First Frame) - The ECU sends this response with the initial part of the response and the total response length. This way, the tester knows more messages are coming.**



Multi-frame communication

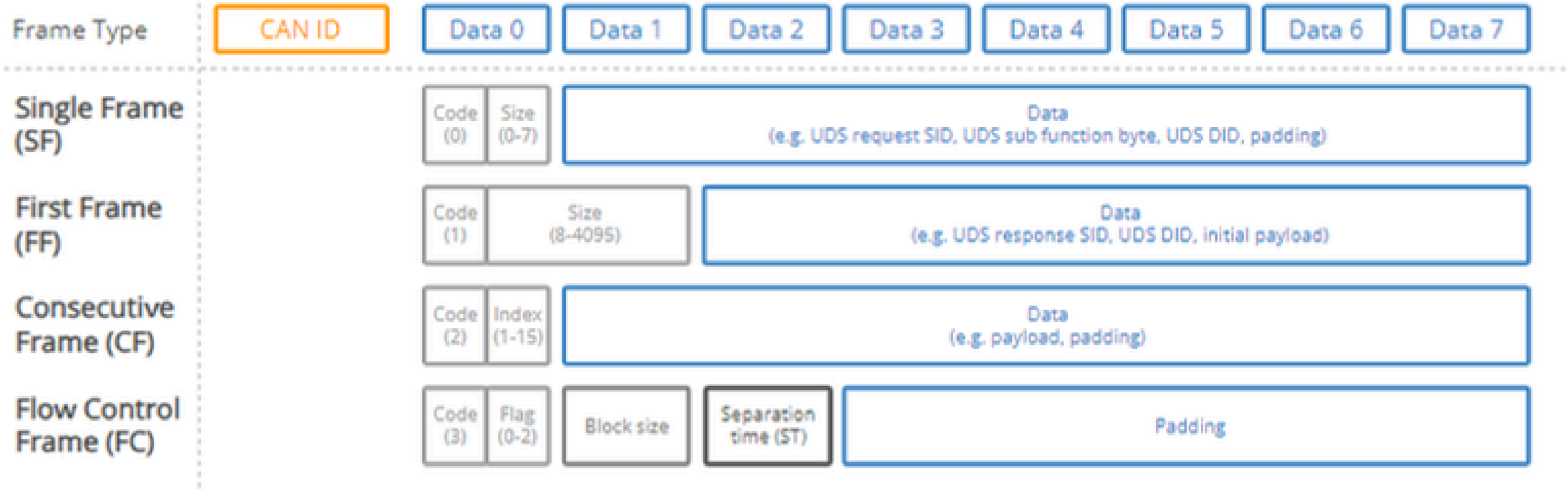
- **FC (Flow Control) - Provides the ECU with parameters that define how the rest of the response will be sent to the tester.**

- **CF (Consecutive Frame) -containing the remaining data payload.**



Multi-frame communication

ISO TP frame types (CAN Bus Transport Protocol, ISO 15765-2)



SID

Common Services

- Session Control (0x10)
- Write DID (0x2E)
- Read DTC (0x19)
- ECU reset(0x11)
- Communication Control(0x28)
- Read DID (0x22)
- Security Access (0x27)

SID

SESSION CONTROL (0X10)

Sub functions

- Default session: 0x01 (Starts)
- Programming: 0x02 (Flash new codes, software)
- Extended: 0x03 (Extended sessions) → Additional functions, security
- Supplier: 0x60 (Mando) → Extra for suppliers

SID

ECU RESET (0X11)

Sub functions

- **Hard reset: 0x01 (Battery disconnect and reconnect)**
- **On/Off reset: 0x02**
- **Soft reset: 0x03 (Restart main program)**

SID

READ DID (0X22)

Sends DID without sub function values

- Read data of an item at DID (Data Identifier – address)

SID

WRITE DID (0X2E)

Sends DID with data to write

- **Write data of an item at DID (Data Identifier – address)**

SID

COMMUNICATION CONTROL (0X28)

Sub functions + Control Byte

- 0x00: Enable Rx, Tx
- 0x01: Enable Rx, Disable Tx
- 0x02: Disable Rx, Enable Tx
- 0x03: Disable Rx, Tx
- Control Byte
  - 0x01: Communication (Exchanging Data)
  - 0x02: Network Mangement messages (Commands for network element
  - 0x03: Mixture

SID

SECURITY ACCESS (0X27)

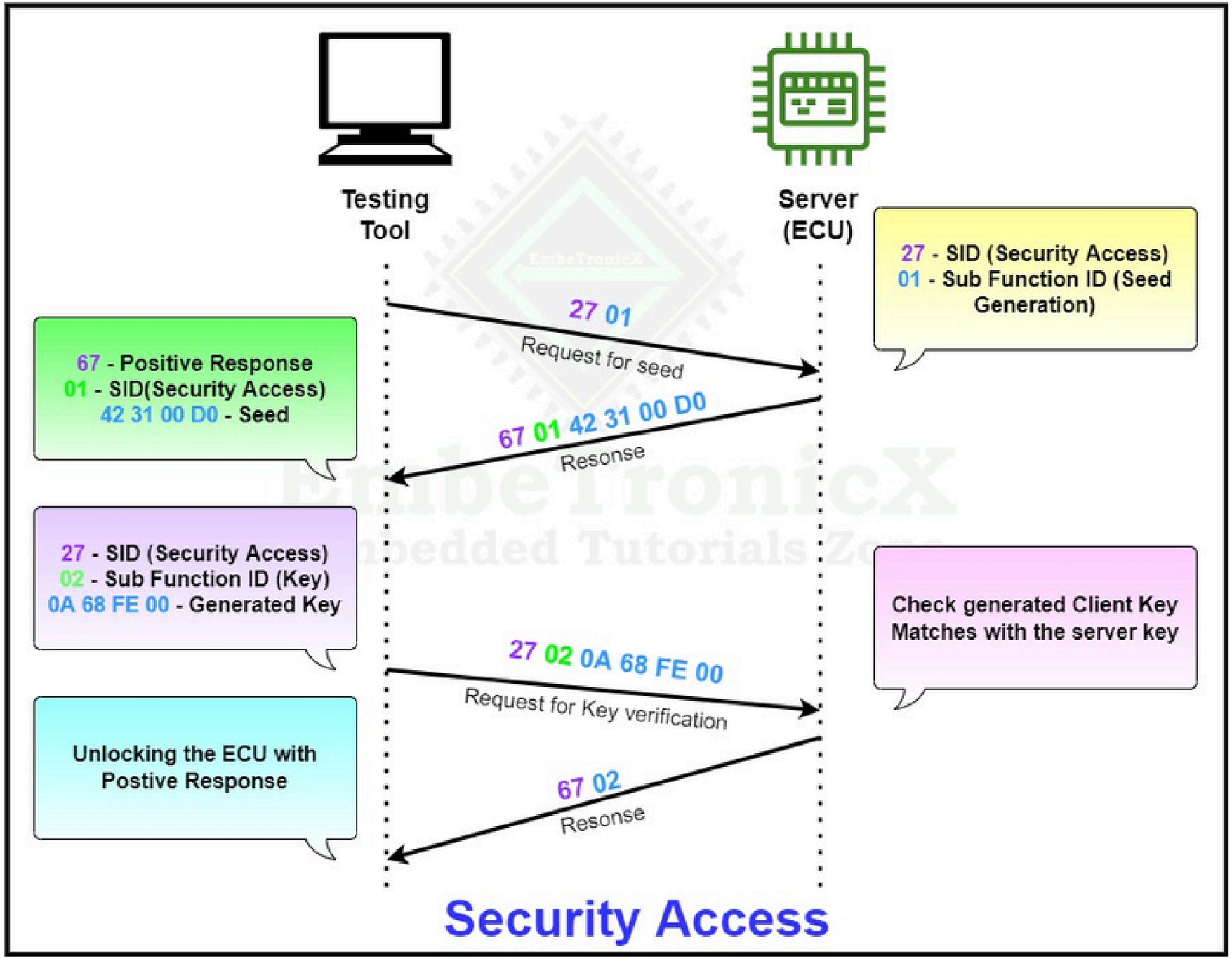
Security access diagnostic service is used to give security access to the UDS protocol services to avoid security breaches.

Some diagnostics data and services can be restricted for safety purposes.

(0X27)

Sub functions

- Security Level N
- Ox(N * 2) – 1 : Seed
- Ox(N * 2) + Key



SID

READ DTC (0X19)

Sub functions

- **0x0A: Report All Supported DTC using mask**
- **0x04: Report DTC Snapshot Record By DTC Number**
- **0x06: Report DTC Extended Data Record By DTC Number**
    - **Provides**

SID

DTC STATUS

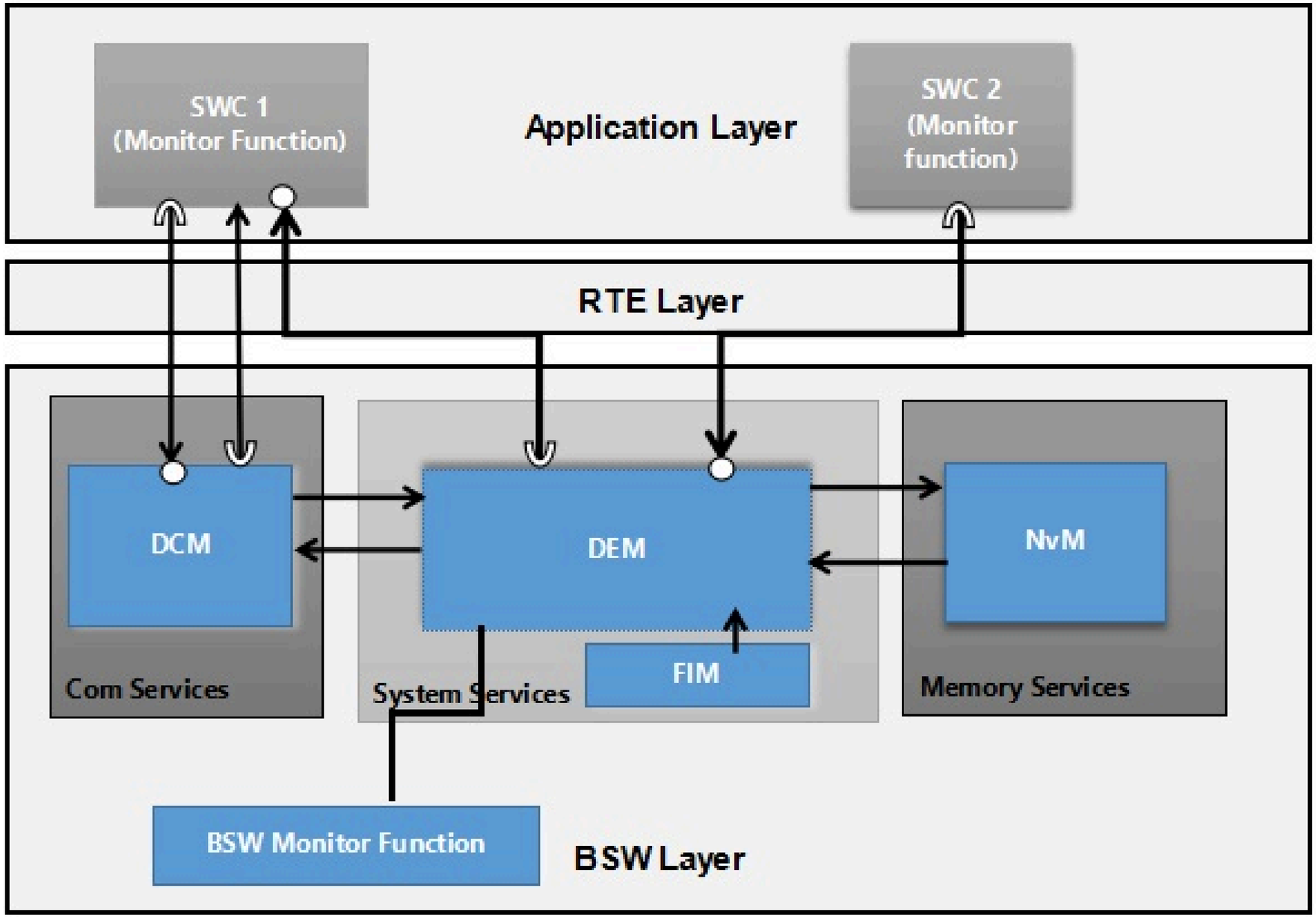
statusOfDTC: bit field name	Bit #	Bit state	Description
testFailed	0	0	DTC is failed at the time of the request.
testFailedThisOperationCycle	1	0	DTC failed on the current operation cycle.
pendingDTC	2	0	DTC failed on the current or previous operation cycle.
confirmedDTC	3	0	DTC is confirmed at the time of the request.
testNotCompletedSinceLastClear	4	1	DTC test has been completed since the last code clear.
testFailedSinceLastClear	5	0	DTC test failed at least once since last code clear.
testNotCompletedThisOperationCycle	6	1	DTC test completed this operation cycle.
warningIndicatorRequested	7	0	Server is not requesting warningIndicator to be active.

Understanding of Diagnostic Stack

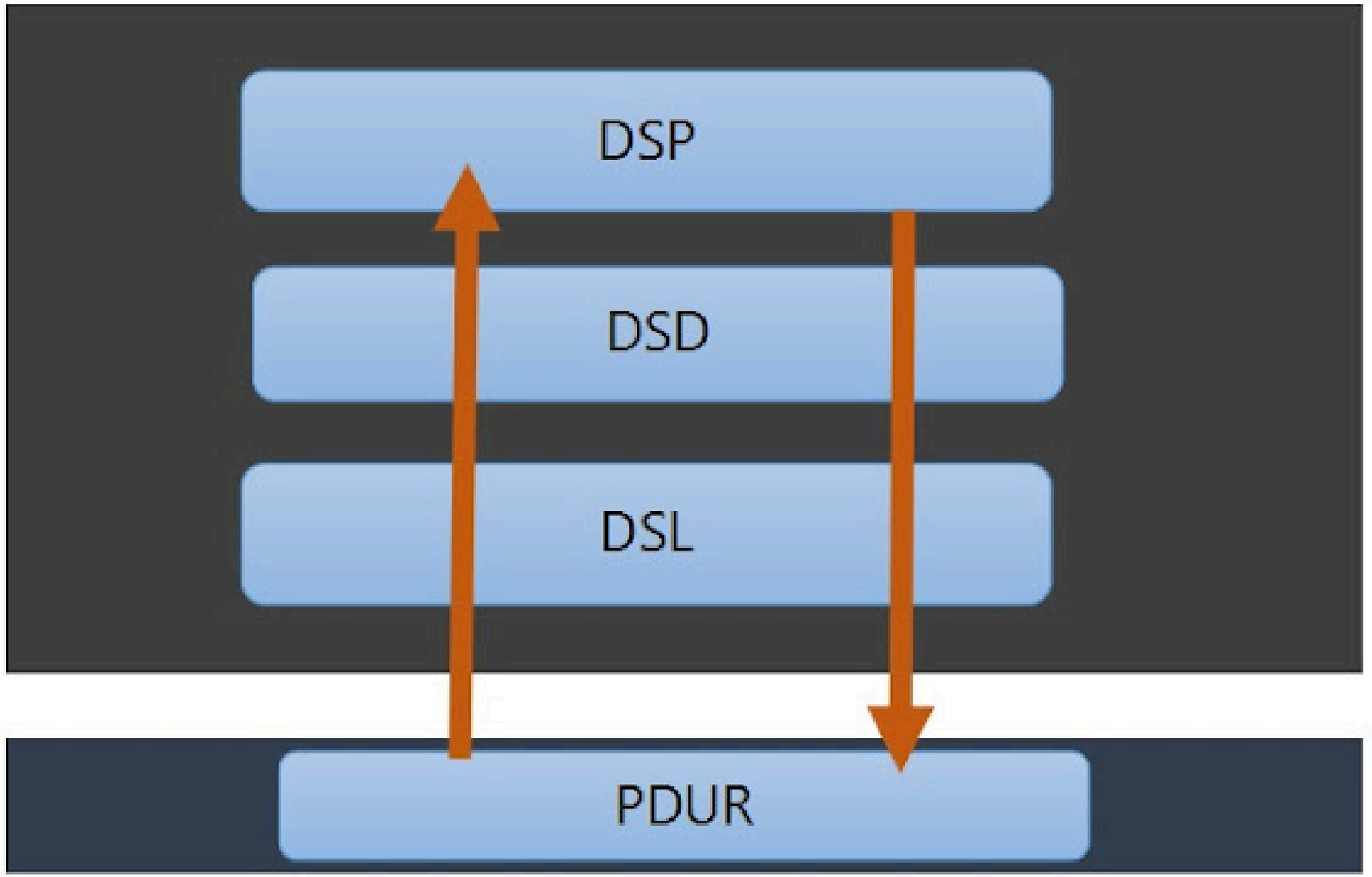
Diagnostic stack

- Overview of Diagnostics Stack of Autosar.
- DEM
- DCM layer

Overview of Diagnostic Stack



DCM layer



Real Project

$22 services

- F2F0
- F2F1
- F2F2
- F2F3

$2E services

- F2F4
- F2F5
- F2F6

Methodology - EB

- Load EPS stack
- Go to DCM -> DSP
- Create DID
  - DID data configuration
  - DID data information
  - DID identifier
  - DID info
- Verify & Generate project

Methodology - Codes

Dcm_Callouts.c

- Add mecros (#define)
  - DID number
  - Return data length
  - On/Off for $22, $2E
- Declare function for positive response
  - Gets static value from id.c
- ReadData function
  - Gets from Dcm_API_Cfg.h
- Check condition function

id.c & id.h

- Declare static values for read DID

Methodology - Tricore

- Use launch batch file for compiling -> Create .elf file
- Target Reset
- Trace .elf file to ECU
- RUN

Methodology - CANoe

- Hardware -> Channel mapping (Check CAN is connected)
- Run
- Analysis -> Trace (Use filter to look nicer)
- Diagnostics -> Console
- 10 01: set to defualt session -> Pos: 50 01 P2server time
- Test other SIDs and look it give positive response

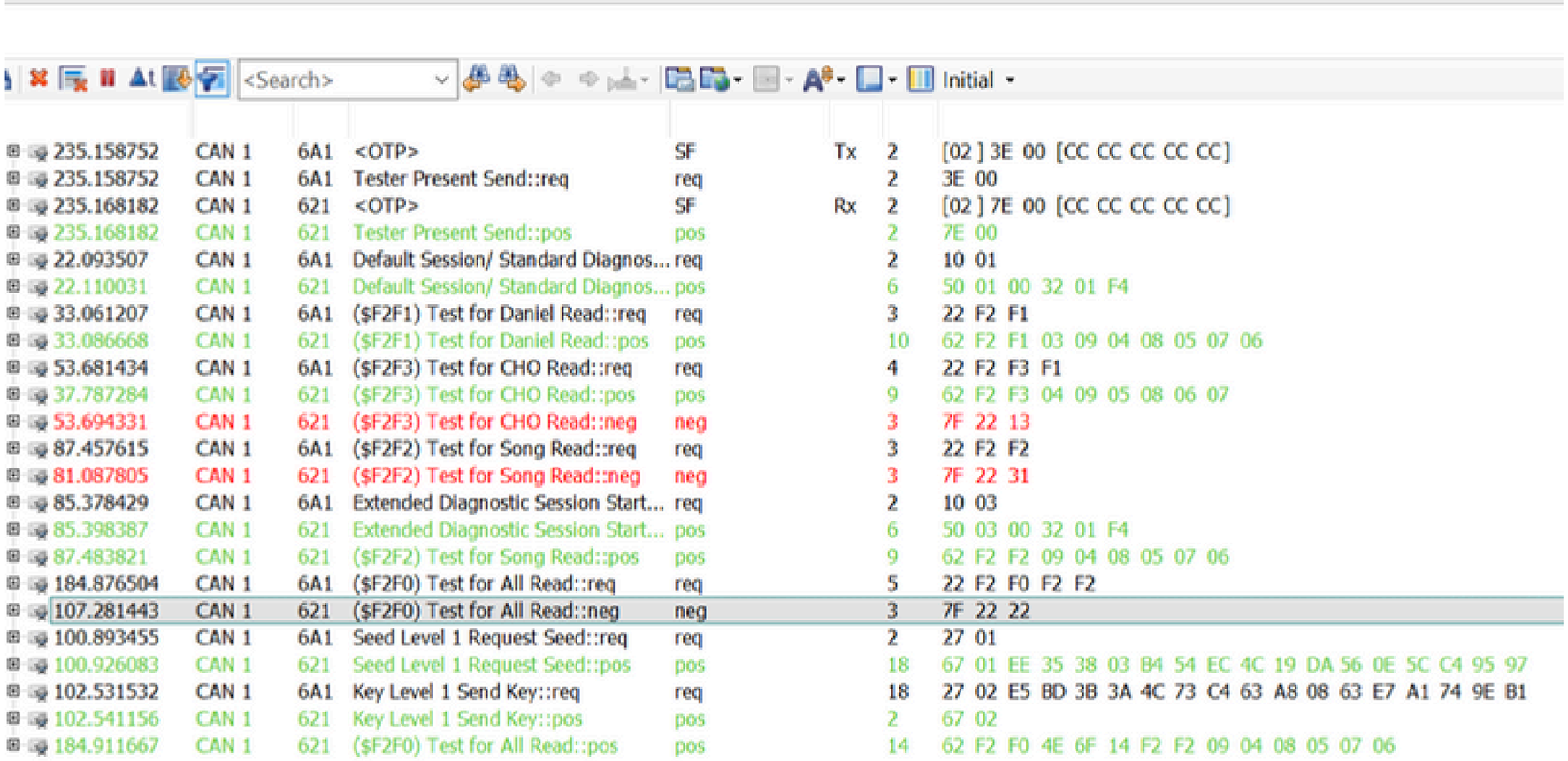
$22 services - SUD

3.1.1 Detailed Description:

1. Service $22 Read Data by Identifier Test cases (Default session, Extended session)

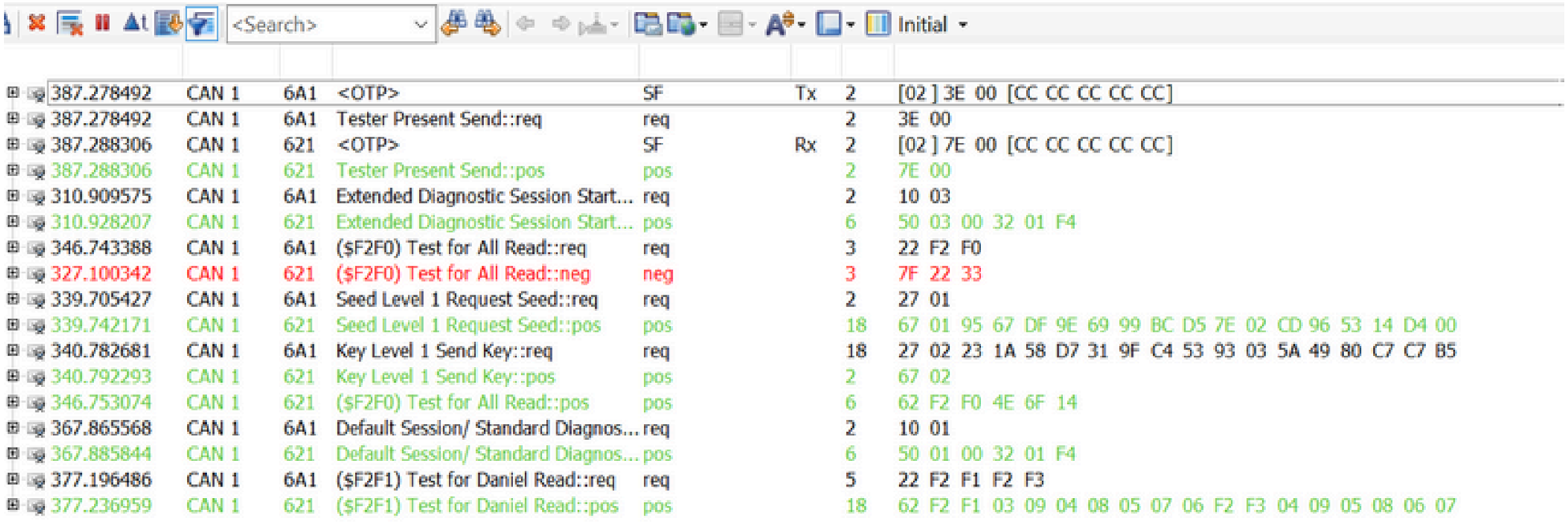
	Detail	Data length	Data type	Allowed Session	Security
F2 F0	: Read EPS and check condition (voltage range in 11 to 13)	3Byte	ASCII	Default, Extended, Supplier	1 level
F2 F1	Read 7Byte Hex decimal	7Byte	HEX	Default, Supplier	None
F2 F2	Read 6Byte Hex decimal	6Byte	HEX	Extended, Supplier	None
F2 F3	Read 6Byte Hex decimal	6Byte	HEX	Default, Extended	None

Test Result (1) - SUT



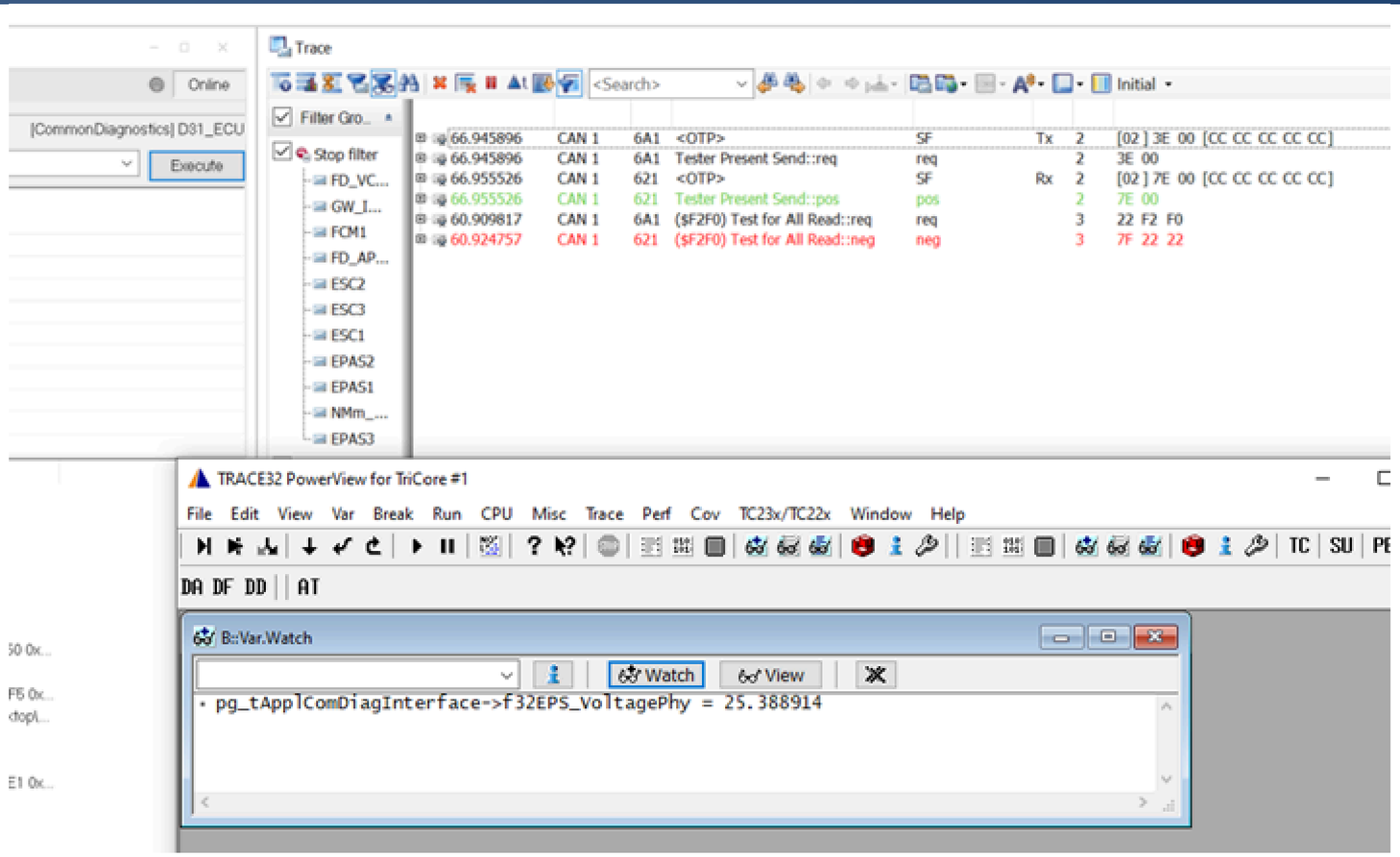
235.158752	CAN 1	6A1	<OTP>	SF	Tx	2	[02] 3E 00 [CC CC CC CC CC]
235.158752	CAN 1	6A1	Tester Present Send::req	req		2	3E 00
235.168182	CAN 1	621	<OTP>	SF	Rx	2	[02] 7E 00 [CC CC CC CC CC]
235.168182	CAN 1	621	Tester Present Send::pos	pos		2	7E 00
22.093507	CAN 1	6A1	Default Session/ Standard Diagnos...	req		2	10 01
22.110031	CAN 1	621	Default Session/ Standard Diagnos...	pos		6	50 01 00 32 01 F4
33.061207	CAN 1	6A1	($F2F1) Test for Daniel Read::req	req		3	22 F2 F1
33.086668	CAN 1	621	($F2F1) Test for Daniel Read::pos	pos		10	62 F2 F1 03 09 04 08 05 07 06
53.681434	CAN 1	6A1	($F2F3) Test for CHO Read::req	req		4	22 F2 F3 F1
37.787284	CAN 1	621	($F2F3) Test for CHO Read::pos	pos		9	62 F2 F3 04 09 05 08 06 07
53.694331	CAN 1	621	($F2F3) Test for CHO Read::neg	neg		3	7F 22 13
87.457615	CAN 1	6A1	($F2F2) Test for Song Read::req	req		3	22 F2 F2
81.087805	CAN 1	621	($F2F2) Test for Song Read::neg	neg		3	7F 22 31
85.378429	CAN 1	6A1	Extended Diagnostic Session Start...	req		2	10 03
85.398387	CAN 1	621	Extended Diagnostic Session Start...	pos		6	50 03 00 32 01 F4
87.483821	CAN 1	621	($F2F2) Test for Song Read::pos	pos		9	62 F2 F2 09 04 08 05 07 06
184.876504	CAN 1	6A1	($F2F0) Test for All Read::req	req		5	22 F2 F0 F2 F2
107.281443	CAN 1	621	($F2F0) Test for All Read::neg	neg		3	7F 22 22
100.893455	CAN 1	6A1	Seed Level 1 Request Seed::req	req		2	27 01
100.926083	CAN 1	621	Seed Level 1 Request Seed::pos	pos		18	67 01 EE 35 38 03 B4 54 EC 4C 19 DA 56 0E 5C C4 95 97
102.531532	CAN 1	6A1	Key Level 1 Send Key::req	req		18	27 02 E5 BD 3B 3A 4C 73 C4 63 A8 08 63 E7 A1 74 9E B1
102.541156	CAN 1	621	Key Level 1 Send Key::pos	pos		2	67 02
184.911667	CAN 1	621	($F2F0) Test for All Read::pos	pos		14	62 F2 F0 4E 6F 14 F2 F2 09 04 08 05 07 06

Test Result (2) - SUT

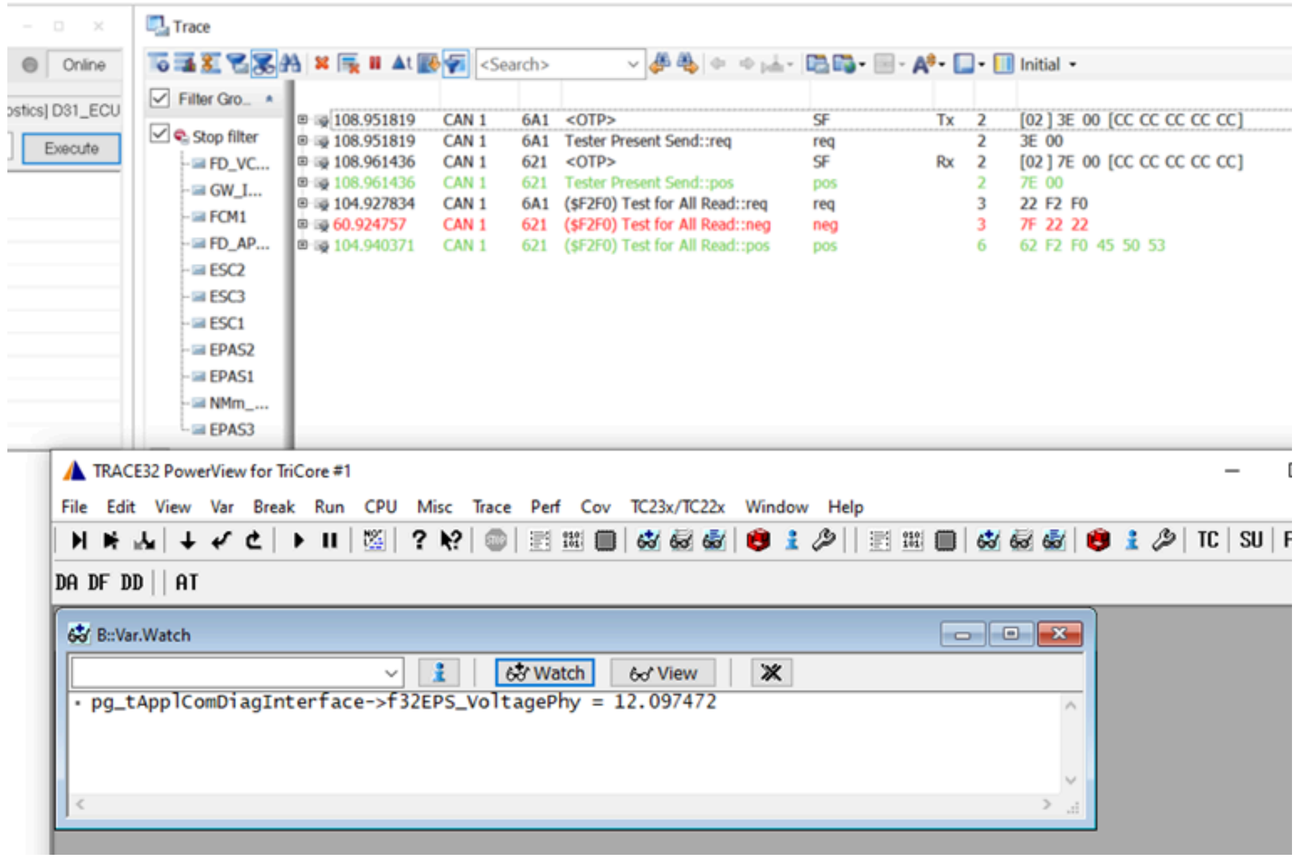


387.278492	CAN 1	6A1	<OTP>	SF	Tx	2	[02] 3E 00 [CC CC CC CC CC]
387.278492	CAN 1	6A1	Tester Present Send::req	req		2	3E 00
387.288306	CAN 1	621	<OTP>	SF	Rx	2	[02] 7E 00 [CC CC CC CC CC]
387.288306	CAN 1	621	Tester Present Send::pos	pos		2	7E 00
310.909575	CAN 1	6A1	Extended Diagnostic Session Start...	req		2	10 03
310.928207	CAN 1	621	Extended Diagnostic Session Start...	pos		6	50 03 00 32 01 F4
346.743388	CAN 1	6A1	($F2F0) Test for All Read::req	req		3	22 F2 F0
327.100342	CAN 1	621	($F2F0) Test for All Read::neg	neg		3	7F 22 33
339.705427	CAN 1	6A1	Seed Level 1 Request Seed::req	req		2	27 01
339.742171	CAN 1	621	Seed Level 1 Request Seed::pos	pos		18	67 01 95 67 DF 9E 69 99 BC D5 7E 02 CD 96 53 14 D4 00
340.782681	CAN 1	6A1	Key Level 1 Send Key::req	req		18	27 02 23 1A 58 D7 31 9F C4 53 93 03 5A 49 80 C7 C7 B5
340.792293	CAN 1	621	Key Level 1 Send Key::pos	pos		2	67 02
346.753074	CAN 1	621	($F2F0) Test for All Read::pos	pos		6	62 F2 F0 4E 6F 14
367.865568	CAN 1	6A1	Default Session/ Standard Diagnos...	req		2	10 01
367.885844	CAN 1	621	Default Session/ Standard Diagnos...	pos		6	50 01 00 32 01 F4
377.196486	CAN 1	6A1	($F2F1) Test for Daniel Read::req	req		5	22 F2 F1 F2 F3
377.236959	CAN 1	621	($F2F1) Test for Daniel Read::pos	pos		18	62 F2 F1 03 09 04 08 05 07 06 F2 F3 04 09 05 08 06 07

Test Result (3) - SUT



Test Result (4) - SUT



Real Project

$22 services

- F2F0
- F2F1
- F2F2
- F2F3

$2E services

- F2F4
- F2F5
- F2F6

Methodology - EB

- Load EPS stack
- Go to DCM -> DSP
- Create DID
- Do same things like $22, but add Write DidInfo, and Write DID Data function
- Go to NvM
- Update NvM Configuration Table

(Max Number of Read and Write Retries,Block Length,Number of NVRAM Copies for Block,Block Identifier,RAM Block Data Address, Write Vertification Data Size, Block Base Number,Fee Target Block Reference)

- Go to Fee (Flash eeprom)
- Update Fee Configuration

  (FeeBlockNumber, FeeBlockSize)
- Verify & Generate project

Methodology - Codes

Dcm_Callouts.c

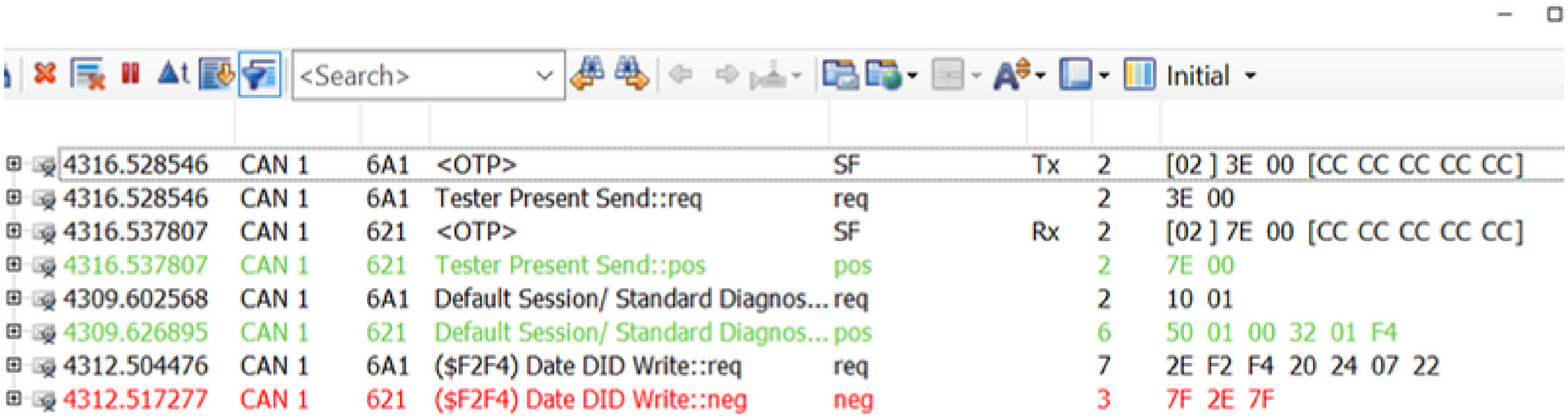
- Add mecros (#define)
  - DID number
  - Return data length
  - On for $22, $2E
- WriteData function
  - Gets from Dcm_API_Cfg.h
- Create global variable to store input data

$2E services - SUD

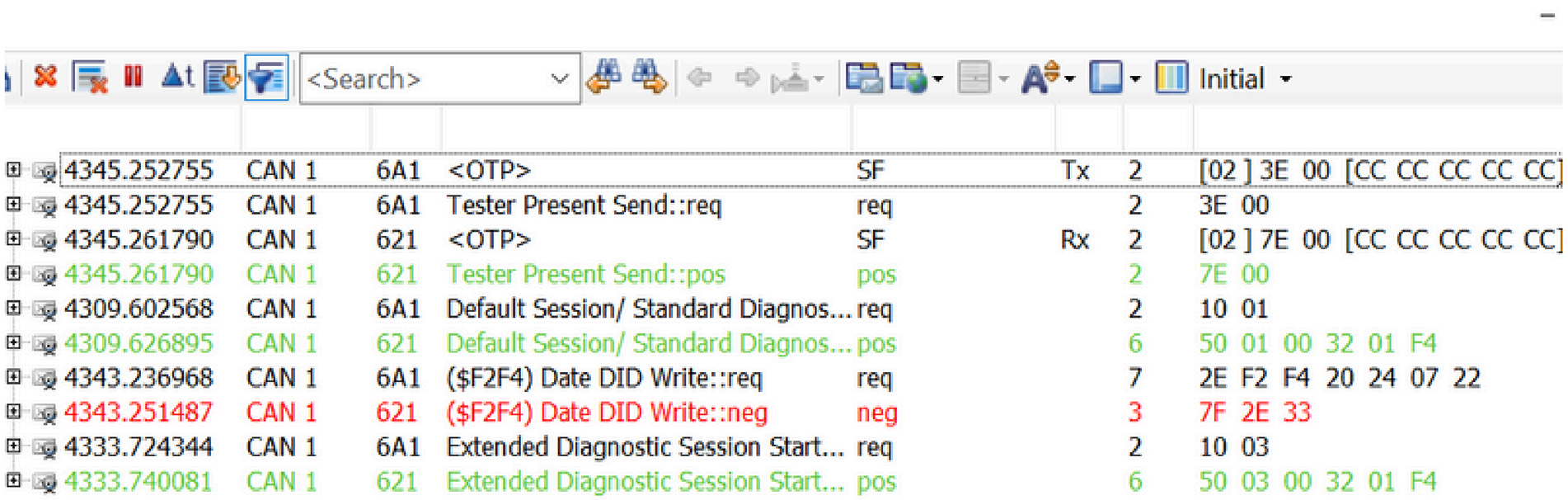
2. Service $2E Write Data by Identifier Test cases (Extended session, Supplier session)

	Detail	Data length	Data type	Allowed Session	Security
F2 F4	Write 4 Byte about date (YY YY MM DD) and check condition (valid range that represents Date)	4Byte	HEX	Extended, Supplier	1 level
F2 F5	Write any 4 Byte	4Byte	HEX	Supplier	1 level
F2 F6	Write 10 Byte about name and if name length is less than 10, fill rest with ' '.	10Byte	ASCII	Extended, Supplier	1 level

Test Result (1) - SUT

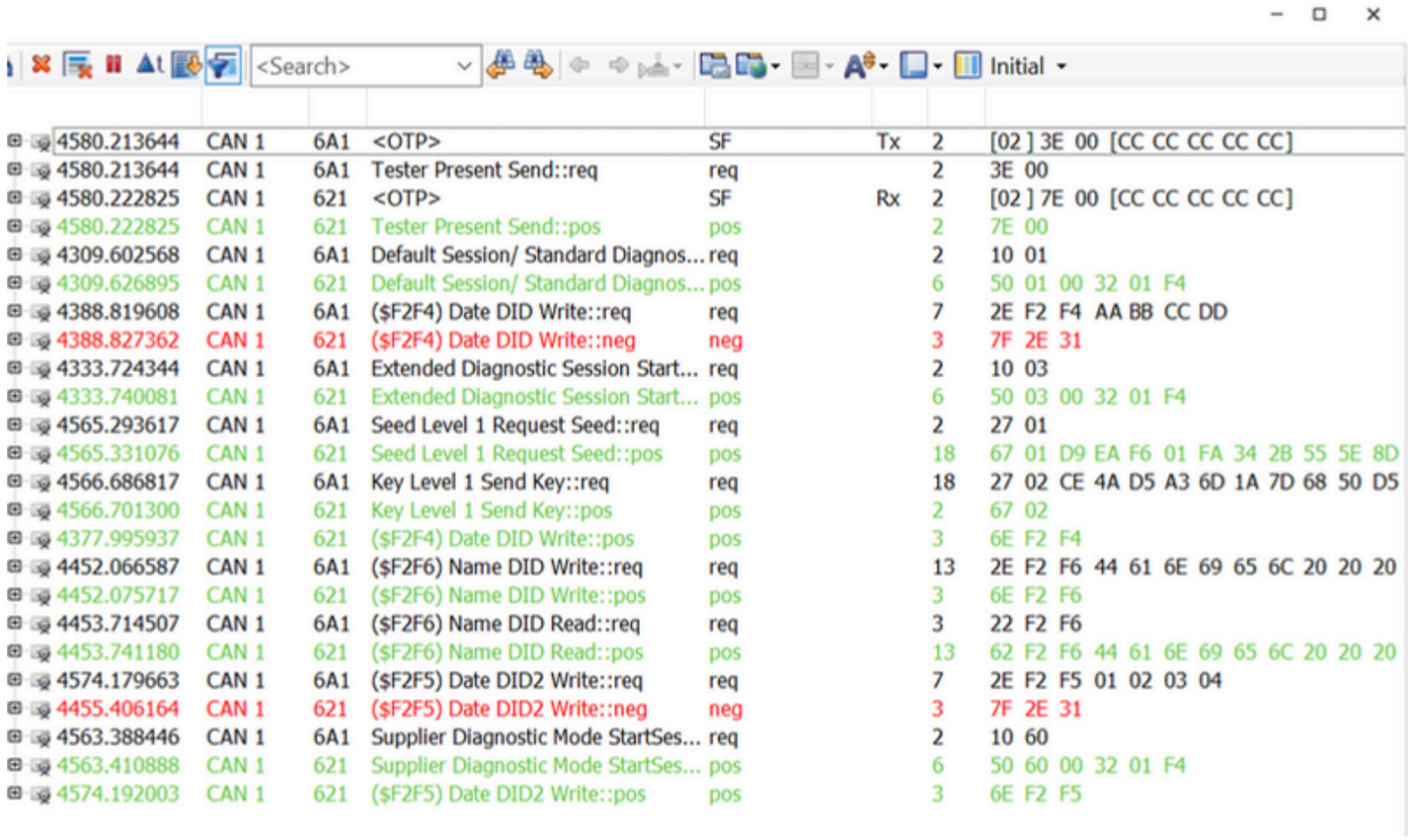


Test Result (2) - SUT



4345.252755	CAN 1	6A1	<OTP>	SF	Tx	2	[02] 3E 00 [CC CC CC CC CC]
4345.252755	CAN 1	6A1	Tester Present Send::req	req		2	3E 00
4345.261790	CAN 1	621	<OTP>	SF	Rx	2	[02] 7E 00 [CC CC CC CC CC]
4345.261790	CAN 1	621	Tester Present Send::pos	pos		2	7E 00
4309.602568	CAN 1	6A1	Default Session/ Standard Diagnos...	req		2	10 01
4309.626895	CAN 1	621	Default Session/ Standard Diagnos...	pos		6	50 01 00 32 01 F4
4343.236968	CAN 1	6A1	($F2F4) Date DID Write::req	req		7	2E F2 F4 20 24 07 22
4343.251487	CAN 1	621	($F2F4) Date DID Write::neg	neg		3	7F 2E 33
4333.724344	CAN 1	6A1	Extended Diagnostic Session Start...	req		2	10 03
4333.740081	CAN 1	621	Extended Diagnostic Session Start...	pos		6	50 03 00 32 01 F4

Test Result (3) - SUT



4580.213644	CAN 1	6A1	<OTP>	SF	Tx	2	[02] 3E 00 [CC CC CC CC CC]
4580.213644	CAN 1	6A1	Tester Present Send::req	req		2	3E 00
4580.222825	CAN 1	621	<OTP>	SF	Rx	2	[02] 7E 00 [CC CC CC CC CC]
4580.222825	CAN 1	621	Tester Present Send::pos	pos		2	7E 00
4309.602568	CAN 1	6A1	Default Session/ Standard Diagnos...	req		2	10 01
4309.626895	CAN 1	621	Default Session/ Standard Diagnos...	pos		6	50 01 00 32 01 F4
4388.819608	CAN 1	6A1	($F2F4) Date DID Write::req	req		7	2E F2 F4 AA BB CC DD
4388.827362	CAN 1	621	($F2F4) Date DID Write::neg	neg		3	7F 2E 31
4333.724344	CAN 1	6A1	Extended Diagnostic Session Start...	req		2	10 03
4333.740081	CAN 1	621	Extended Diagnostic Session Start...	pos		6	50 03 00 32 01 F4
4565.293617	CAN 1	6A1	Seed Level 1 Request Seed::req	req		2	27 01
4565.331076	CAN 1	621	Seed Level 1 Request Seed::pos	pos		18	67 01 D9 EA F6 01 FA 34 2B 55 5E 8D
4566.686817	CAN 1	6A1	Key Level 1 Send Key::req	req		18	27 02 CE 4A D5 A3 6D 1A 7D 68 50 D5
4566.701300	CAN 1	621	Key Level 1 Send Key::pos	pos		2	67 02
4377.995937	CAN 1	621	($F2F4) Date DID Write::pos	pos		3	6E F2 F4
4452.066587	CAN 1	6A1	($F2F6) Name DID Write::req	req		13	2E F2 F6 44 61 6E 69 65 6C 20 20 20
4452.075717	CAN 1	621	($F2F6) Name DID Write::pos	pos		3	6E F2 F6
4453.714507	CAN 1	6A1	($F2F6) Name DID Read::req	req		3	22 F2 F6
4453.741180	CAN 1	621	($F2F6) Name DID Read::pos	pos		13	62 F2 F6 44 61 6E 69 65 6C 20 20 20
4574.179663	CAN 1	6A1	($F2F5) Date DID2 Write::req	req		7	2E F2 F5 01 02 03 04
4455.406164	CAN 1	621	($F2F5) Date DID2 Write::neg	neg		3	7F 2E 31
4563.388446	CAN 1	6A1	Supplier Diagnostic Mode StartSes...	req		2	10 60
4563.410888	CAN 1	621	Supplier Diagnostic Mode StartSes...	pos		6	50 60 00 32 01 F4
4574.192003	CAN 1	621	($F2F5) Date DID2 Write::pos	pos		3	6E F2 F5

Conclusion

Q & A