ZÁPADOČESKÁ UNIVERZITA V PLZNI
FAKULTA ELEKTROTECHNICKÁ

KATEDRA ELEKTROENERGETIKY A EKOLOGIE

DIPLOMOVÁ PRÁCE

Vzorové příklady pro simulační nástroj CANoe

Bc. Jan Jendek **2021**

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta elektrotechnická
Akademický rok: 2020/2021

ZADÁNÍ DIPLOMOVÉ PRÁCE
(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Jan JENDEK**
Osobní číslo:	**E18N0007K**
Studijní program:	**N2644 Aplikovaná elektrotechnika**
Studijní obor:	**Aplikovaná elektrotechnika**
Téma práce:	**Vzorové příklady pro simulační nástroj CANoe**
Zadávající katedra:	**Katedra elektroenergetiky**

Zásady pro vypracování

Vytvořte soubor vzorových příkladů pro simulační nástroj CANoe.

1. Seznamte se s možnostmi simulačního nástroje od firmy Vector Informatik GmbH.
2. Navrhněte soubor příkladů, demonstrující možnosti tohoto nástroje. Soubor příkladů by měl obsahovat příklady demonstrující minimálně následující funkčnost: - monitorování a analýza komunikace - využití databáze správ - tvorba grafických rozhraní - simulace uzlu s využitím programovacího jazyku CAPL.
3. Navržené a vytvořené příklady detailně zdokumentujte.

Rozsah diplomové práce:	**40 – 60 stran**
Rozsah grafických prací:	**podle doporučení vedoucího**
Forma zpracování diplomové práce:	**tištěná/elektronická**

Seznam doporučené literatury:

1. CANoe_ProductInformation_EN.pdf dostupná na https://www.vector.com/

Vedoucí diplomové práce:	**Ing. Kamil Kosturik, Ph.D.** Katedra elektroniky a informačních technologií
Datum zadání diplomové práce:	**9. října 2020**
Termín odevzdání diplomové práce:	**27. května 2021**

L.S.

Prof. Ing. Zdeněk Peroutka, Ph.D.
děkan

Doc. Ing. Karel Noháč, Ph.D.
vedoucí katedry

V Plzni dne 9. října 2020

Abstrakt

Předkládaná diplomová práce se zabývá popisem simulačního nástroje CANoe od německé společnosti Vector Informatik. První část diplomové práce je zaměřena na popis základních automobilových sběrnic podporovaných simulačním nástrojem CANoe. Druhá část předkládané práce obsahuje popis jednotlivých funkcionalit, který nástroj umožňuje používat. Tyto funkcionality byly využity a zpracovány do dvou praktických příkladů, které demonstrují reálné využití simulačního nástroje v automobilovém průmyslu. První ze zmíněných příkladů využívá simulační nástroj jako diagnostické prostředí. V druhém příkladu je popsána simulace pro usínání a probouzení elektronických řídicích jednotek, které mezi sebou komunikují. Poslední částí práce je celkové zhodnocení dosažených výsledků.

Klíčová slova

Průmyslová komunikační sběrnice, komunikační sběrnice v automobilech, CAN bus, simulační nástroj, Vector CANoe, elektronická řídicí jednotka, diagnostika elektronické řídicí jednotky.

Abstract

The presented diploma thesis is focused on the description of the simulation tool CANoe from the German company Vector Informatik. The first part of the diploma thesis deals with the description of basic automotive buses supported by the simulation tool CANoe. Another content of the work is a description of the individual functionalities that the tool allows. These functionalities were utilized and applied into two practical examples, which demonstrate the practical use cases in the automotive industry. The first example does show using the tool as a diagnostic environment. Within the second example there is developed the simulation to control states („sleep“ and „wake up“) of electronic control units that communicate with each other. The last part of the work is the overall evaluation of the achieved results.

Key words

Industry communication bus, automotive communication bus, CAN bus, simulation tool, Vector CANoe, electronic control unit, diagnosis of electronic control unit.

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně s použitím odborné literatury a pramenů uvedených v seznamu, který je součástí této diplomové práce.

Dále prohlašuji, že veškerý software, použitý při řešení této bakalářské/diplomové práce, je legální.

.....

podpis

V Plzni dne 27.5.2021 Jan Jendek

(Nepovinná část)

Poděkování

Tímto bych rád poděkoval vedoucímu bakalářské práce Ing. Kamilu Kosturikovi, Ph.D. za cenné profesionální rady, připomínky a metodické vedení práce.

Obsah

Seznam symbolů a zkratek

Zkratka	***Popis***
ABS	Anti-lock Braking System
ARXML	AUTOSAR XML
AUTOSAR	Automotive Open System Architecture
CAN	Controller Area Network
CAN-FD	Controller Area Network Flexible Data-rate
CAPL	Communication Access Programming Language
ECU	Electronic Control Unit
ETH	Ethernet
FIBEX	Field Bus Exchange Format
LDF	LIN Description File
LIN	Local Interconnect Network
MOST	Media Oriented Systems Transport
NM	Network Management
OEM	Original Equipment Manufacturer
PC	Personal Computer
PCIe	Peripheral Component Interconnect Express
PDU	Process Data Unit
RT	Real-Time
RTE	RunTime Environment
USB	Universal Serial Bus
XML	eXtensible Markup Language

Seznam obrázků

Seznam tabulek

Úvod

Simulační nástroj CANoe je vyvíjen německou společností Vector Informatik již od roku 1992 pro použití v automobilovém průmyslu jako nástroj pro testování a simulaci elektrických řídících jednotek. Vzhledem k narůstajícímu počtu elektronických systému pro komunikaci a zároveň vzhledem k rostoucí komplexitě funkční i nefunkčních požadavků na takové systémy, bylo třeba vývoje softwaru, který bude schopen analyzovat, zobrazit, popřípadě nasimulovat a otestovat celou komunikaci probíhající mezi všemi jednotkami v autě. První licence, která nesla označení verze CANoe 1.0, byla prodána v roce 1996. Nejnovější verze nese označení CANoe 14.0, tedy po přibližně 25 letech dalšího vývoje.

Designéři, vývojáři ECU, systémoví integrátoři a testeři využívají tento mocný nástroj ve všech fázích vývojového procesu celého projektu. Schopnost měření v reálném čase je bezpochyby významným benefitem pro analýzu dat. Data lze analyzovat online, popřípadě offline v rámci simulace vlastní komunikace. Pro online analýzu je třeba mít k analyzované komunikační sběrnici připojený hardware od společnosti Vector. Následná offline analýza probíhá z dat nahraných během online měření (analýzy). CANoe podporuje dále propojení s dalšími užitečnými nástroji jako jsou MATLAB a Simulink pomocí DLL knihoven. Tato schopnost kooperace a možnost se funkčně integrovat s rozhraními dalších nástrojů umožňuje širší využití CANoe.

Díky podpoře standardních sběrnicových systému LIN, CAN, CAN-FD, FlexRay, MOST, Ethernet, J708, K-Line, A429, WLAN a dále podpoře specializovaných či proprietárních protokolů založených na sběrnici CAN (CANopen, J939, ISO 11783, GMLAN, CANaero) je široké využití a nasazení nejen v automobilovém průmyslu běžnou součástí celého životního cyklu produktu.

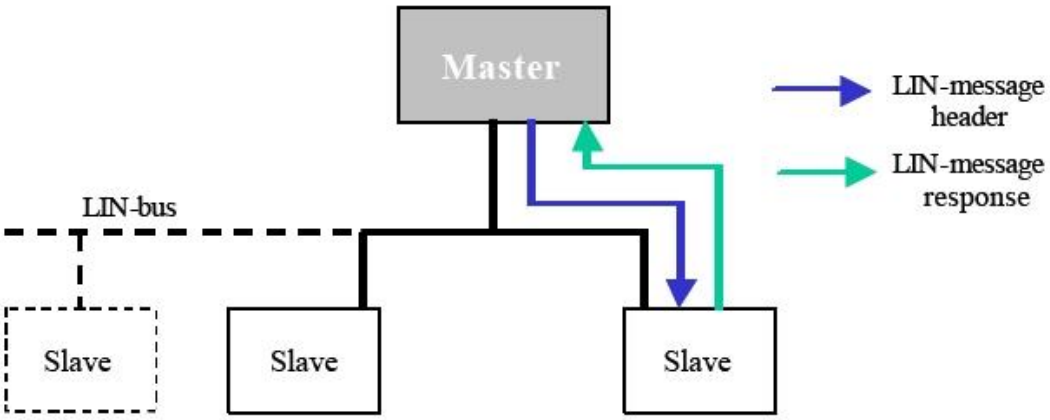
1.1 Základní rozdělení a popis principu podporovaných sběrnic

1.1.1 LIN

Jednovodičová asynchronní sběrnice Local Interconnect Network (LIN) byla navržena pro ovládání jednodušších zařízení (ovládání klimatizace, stahování oken, polohování sedadel). Její předností je jednoduchost a finanční dostupnost Díky nenáročným požadavkům na konstrukci a na zpracování komunikačního protokolu se jedná levnější řešení v porovnání s pořizovacími náklady na realizaci komunikace např. po sběrnici CAN, která však poskytuje spolehlivější a rychlejší datovou komunikaci. LIN sběrnice umožňuje přenášet data rychlostmi od 2400 až do 19200 bit/s. [16]

Sběrnice pracuje na principu single-master / multiple-slave. Jedná se o model komunikace, kde jedno zařízení nebo proces (master) řídí komunikaci mezi ostatními zařízeními. Standardem je doporučeno připojovat maximálně 16 komunikačních uzlů (jednotek) na jednu sběrnici. [16]

K funkčnosti sběrnice je dostačující běžný jednočipový mikropočítač, který disponuje řadičem pro asynchronní sériovou komunikaci UART / SCI. Jednotky slave nevyžadují ke své činnosti přesný krystalový oscilátor, ale vystačí si s levnými RC oscilátory. Rozhraní a fyzická vrstva jsou realizovány budičem LIN (LIN transceiver), který převádí TTL úrovně signálu na napěťové úrovně od 0 až 12V v souladu s normou. [16]

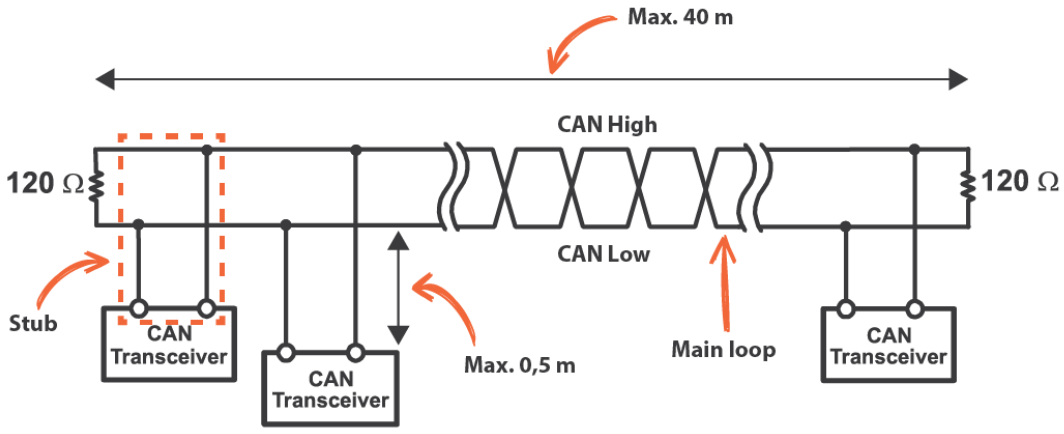


Obr. 1: LIN bus převzato z [11]

1.1.2 CAN

Sériová datová sběrnice CAN je nejpoužívanější sběrnicí v automobilovém průmyslu. Zajišťuje velkou míru zabezpečení proti chybám. Používá se pro důležité funkční jednotky v automobilu a také pro vnitřní komunikační síť senzorů. Dále se dá použít ve sféře průmyslové automatizace, kde je ovšem její využití menší než v automobilovém průmyslu. Přenosová rychlost sběrnice CAN může dosahovat až 1 Mbit/s. [16]

Protokol CAN je typu multi master, kde každý uzel sběrnice může být master a řídit tak chování jiných uzlů. Není tedy nutné řídit celou síť z jednoho nadřazeného uzlu, což přináší zjednodušení řízení a zvyšuje spolehlivost (při poruše jednoho uzlu může zbytek sítě pracovat dál). Komunikace po sběrnici probíhá mezi dvěma uzly pomocí datové a signalizační zprávy. Signalizaci chyb zajišťují speciální zprávy – zpráva o přetížení a chybová zpráva. Vysílané zprávy po sběrnici jsou přijímány všemi uzly připojenými na sběrnici, protože neobsahují žádnou informací o cílovém uzlu. Zprávy jsou uvozeny identifikátorem, který má délku 11 bitů (standardní formát) nebo 29 bitů (rozšířený formát). [16]



***Obr. 2: CAN bus* převzato z [12]**

Identifikátor definuje prioritu přenášené zprávy. Definice identifikátoru (jeho hodnota unikátní na dané sběrnici) tudíž souvisí s významem přenášeného obsahu zprávy, např. otáčky motoru, otáčky kola, informace o teplotě oleje a další. Řídicí jednotka bitově vyhodnocuje identifikátory přenášených zpráv na základě seznamu identifikátorů akceptovatelných zpráv Zpráva s akceptovaným identifikátorem je pak příslušnou jednotkou

přijata a přenášena data jsou následně zpracována. Zprávy s neakceptovatelnými identifikátory jsou ignorovány. Nejnižší hodnota identifikátoru má největší prioritu a naopak. Zprávy se shodnou prioritou se nemohou z principu vyskytovat na jedné sběrnici. V okamžiku, když je sběrnice volná a data jsou připravena k přenosu, začne každá jednotka (stanice) s odesíláním své zprávy. [16]

1.1.3 FlexRay

Sběrnice FlexRay vznikla v konsorciu firem Freescale Semiconductor (od roku 2015 akvizicí firmy NXP Semicondutors), BMW, Philips (NXP Semiconductors), Daimler, Chrysler a také s přispěním firmy Robert Bosch GmbH. Vysokorychlostní deterministická sběrnice FlexRay by měla být využita pro aplikace vyžadující největší možnou provozní datovou rychlost, spolehlivost a vysoký stupeň zabezpečení (např. řídicí jednotky pro řízení podvozku a ovládání brzdového systému ABS). Aktuální verze FlexRay protokolu je 3.0.1 vydaná v roce 2010 a stala se ISO standardem. Standard je nyní dostupný pod označením ISO 17458. [16]

FlexRay komunikační protokol poskytuje větší přenosovou rychlost než hojně používané sběrnice typu LIN a CAN. Rychlost je srovnatelná s Ethernetem komunikujícím rychlostí až 10 Mbit/s. Další nespornou výhodou je dvoukanálová struktura a vyšší odolnost proti elektromagnetickému rušení. Dvoukanálovou strukturou je myšlen návrh a zapojení dvou komunikačních kanálů, kde druhý kanál může sloužit jako záložní (duplicitní), který se stává procesně aktivním v případě selhání (výpadku) komunikace na prvním kanálu. Je možné však používat tyto dva kanály současně, a tím zvýšit přenosovou rychlost až na 20 MBit/s. [16]

Systém FlexRay je složen z několika dalších uzlů a prostřednictvím fyzického přenosového média propojuje všechny zbylé uzly. Sběrnice může být založena na mnoho síťových topologiích, mezi které patří například aktivní hvězda, kaskádová hvězda, dvoukanálová sběrnice nebo kombinace těchto zmíněných topologií. FlexRay je postavena na „Time-triggered communication" architektuře. Znamená to, že veškeré akce na sběrnici jsou statické a časově definované v tomto systému. Princip časové kontroly umožňuje deterministickou datovou komunikaci, a díky tomu je možná snadná implementace různých koncepcí na ní založené, jako je odolnost proti chybám, která je dosažena synchronním spouštěním akcí. [16]

Pro implementaci "Time-triggered" komunikace se využívá metoda TDMA (Time Division Multiple Access), která díky definici pevných časových slotů nedovoluje nekontrolovatelně přistupovat ke sběrnici jako v případě CAN komunikace. Přenos dat mezi FlexRay uzly odpovídá přesně nadefinovanému komunikačnímu plánu – v konkrétními časovému slotu je povoleno odesílání specifické zprávy, která přísluší právě k tomuto slotu. [16]

1.2 AUTOSAR standard a konsorcium

S rostoucí složitostí softwaru v moderních vozidlech bylo potřeba vyvinout referenční architekturu pro software elektronické řídící jednotky. Tato architektura je nazývána AUTOSAR (Automotive Open System Architecture). Roku 2003 byla založena a inicializována předními výrobci a dodavateli automobilového průmyslu. Jedná se o inovativní cestu pro elektronické systémy zlepšující jejich výkon, bezpečnost a šetrnost k životnímu prostředí.Standard AUTOSAR slouží jako platforma, na které jsou implementovány budoucí aplikace vozidel. AUTOSAR umožňuje využití více služeb, hospodárnost a usnadňuje orientaci ve složitých strukturách subsystémů a komponent, dále systematicky podporuje zvýšení jednotek ECU v elektronickém systému, funkce softwaru a rozmanitější sadu hardwaru.

Architektura AUTOSAR Classic je standardem pro elektronické řídící jednotky v reálném čase. Rozlišujeme tři softwarové vrstvy běžící na mikro kontrolérů v architektuře AUTOSAR.

Základní software: většinou se jedná o standardizované moduly, které slouží ke spuštění horní funkční částí softwarové vrstvy a k obsluze HW periférií dané elektronické řídicí jednotky.

Runtime environment (RTE): zajišťuje výměnu informací mezi jednotlivými komponentami elektronických řídících jednotek a mezi základním softwarem a dedikovanými aplikačními komponentami, jako jedna ze SW částí patří do SW vrstvy tzv. middleware.

Aplikační vrstva – aplikační komponenty: sestává z komponent aplikačního SW, které realizují konkrétní funkci (funkcionality) systému.

Dalším výstupem konsorcia AUROSAR je např. specifický datový formát, který je dále popsán v 2.5.1 ARXML – AUTOSAR XML.

2 CANoe

2.1 Systémové požadavky

Systémové požadavky pro simulační nástroj CANoe rozděluje výrobce na minimální a doporoučené dle následující tabulky:

Komponenty	Minimální	Doporučené
Procesor	Intel compatible 2- jádra	Intel Core i7- 4 jádra
Paměť	4 GB	32 GB
Místo na pevném disku	20 GB /SDD/NVMe	80 GB HDD/SSD
Operační systém	Windows 10 64 bit	Windows 7 64 bit Windows 8 64 bit Windows 10 64 bit

Tab. 1: Vector CANoe – systémové požadavky

2.2 Rozdělení licencí

Nástroj CANoe spadá pod:

- EULA – licence pro koncového uživatele.
- ELA – podniková licence

Licence CANoe rozdělujeme dále na tři varianty:

PEX – Tato varianta je nejlevnější, tudíž obsahuje nejméně dostupných funkcionalit. Funkcionality jsou omezené, a to na práci pouze s grafickým rozhraním a dále lze spouštět automatizované testy a generovat reporty. Využití může být ve výrobě, kde je potřeba licencí pro více počítačů – v důsledku tedy takové nasazení vede k minimalizaci nákladů. Příkladem nasazení může být např. plošné testování, kde se využívají stále stejné konfigurace, tedy není požadováno programování skriptů a vytváření nových konfigurací.

RUN – Licence RUN umožňuje využití širších funkcionalit programu. Velká část funkcionalit je již dostupná vyjma možnosti modifikace simulací. Chování uzlů na sběrnici lze simulovat, a to prostřednictvím CAPL (Communication Access Programming Language) skriptů. Čtení statistik a dat ze sběrnice je u této licence možné a lze otestovat celou ECU v interakci s předepsanou zbývající simulací sběrnice.

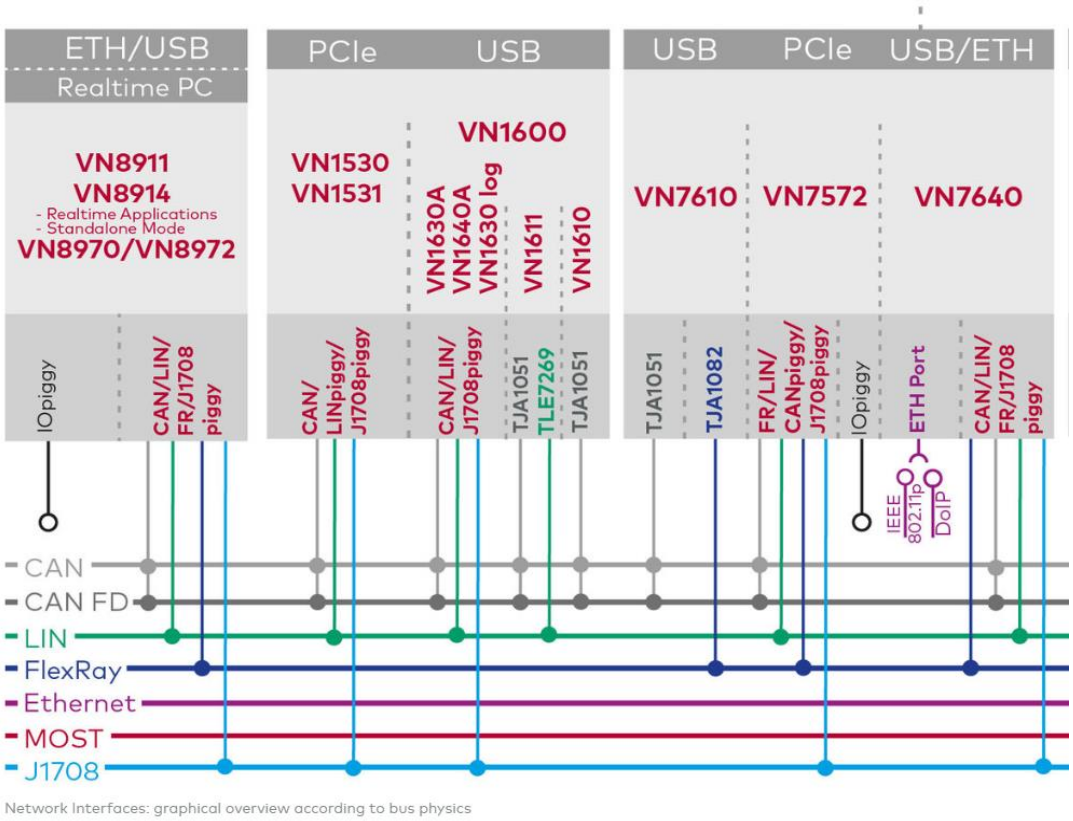
FULL – Plná licence je cenově nákladná, a proto by se její užití v širším spektru mělo zvážit. Samozřejmostí je možnost využití všech funkcionalit, které jsou v simulačním nástroji dostupné.

Licenční politika je dále rozdělována na podporu jednotlivých sběrnic a komunikačních protokolů – standardně dostupná podpora pro sběrnice FlexRay, Ethernet, LIN, kdežto např. podporu protokolu XCP je už potřeba dokoupit nad rámec standardní licence.

2.3 Vector hardware rozhraní

K využití simulačního nástroje CANoe je zapotřebí rozhraní, které realizuje připojení měřícího a simulačního prostředí, a tedy přijímá a vysílá reálná data na sběrnici. Jak uvádějí předchozí kapitoly, lze měřit jak reálná data, tak data simulovaná. Simulovaná data mohou být generovaná prostřednictvím CAPL skriptů, popř. znovu použita původně nahraná data z reálné sběrnice. Rozhraní je vázáno s licencí. Bez licence nebude rozhraní komunikovat se softwarovým nástrojem.

Rozhraní se liší dle různých kritérií na vícero variant. Varianty se odvíjí od podporované komunikační sběrnice. Dále se rozlišuje rozhraní dle způsobu připojení k PC, a to buď přes USB, nebo přes PCIe, popř. po Ethernetu. Škála výrobcem nabízených rozhraní je velmi široká a pestrá, následující text se však bude zaměřovat především na nejběžnější zařízení pro automobilový průmysl. Obr. 3: Vector HW rozhraní – rozdělení poskytuje přehledné rozdělení jednotlivých rozhraních.



Obr. 3: Vector HW rozhraní – rozdělení převzato z [1]

2.4 Rozdělení jednotlivých rozhraní

Následující kapitoly se zabývají některými vybranými typy rozhraní. Jedná se o často využívané zástupce z různých částí nabízeného spektra produktů určených jak pro případy použití, kde je dostačující základní funkcionalita (běžné diagnostické operace s konkrétní ECU), až po složitější způsoby nasazení (komplexní logování dat na sběrnici, nebo analýza s možností spuštění simulace).

2.4.1 Modelové řada VN16xx

Pro program CANoe, CANalyzer, CANape, vFlash a i pro zákaznické aplikace je zařízení VN1600 cenově dostupným a funkčně dostačujícím nástrojem určeným pro testování a diagnostiku komunikace po sběrnici v laboratoři, za jízdy přímo ve vozidle nebo staticky v servisní garáži. Od jednoduchých až složité simulace sběrnice, diagnostické, kalibrační a flash-programovací (nahrání programu a sady parametrů do programové paměti) úlohy je rozhraní VN 1600 ideální volbou. Další nespornou výhodou je podpora umožňující přístup z více programů běžících na PC stanici paralelně na stejném kanálu a na stejném zařízení.

2.4.1.1 Přehled funkcionalit rozhraní modelové řady VN16xx

Hardware (rozhraní) VN1600 lze rozdělit do více skupin:

VN1610:

Obr. 4: VN1610 CAN rozhraní převzato z [13]

Toto rozhraní se používá výhradně pro CAN komunikaci. Poskytuje velmi jednoduché zapojení s PC přes USB, není potřeba externího napájení, které některé zařízení potřebují. Pro sledování komunikace na sběrnici lze využít dva dostupné budiče (2x CAN-high speed 1051 cap transciever) přímo vestavěné v hardwarovém zařízení.

Dalšími rozhraními, které nesou označení VN1611 a VN7610, se tato práce detailněji nezabývá. Jsou v principu stejné jako rozhraní zmíněné výše. Rozdíly jsou v podpoře různých typů komunikačních sběrnic. Zařízení VN1611 podporuje CAN a dále i LIN, a to prostřednictvím budiče typu transceiver LIN 7269cap (s kapacitním oddělením).

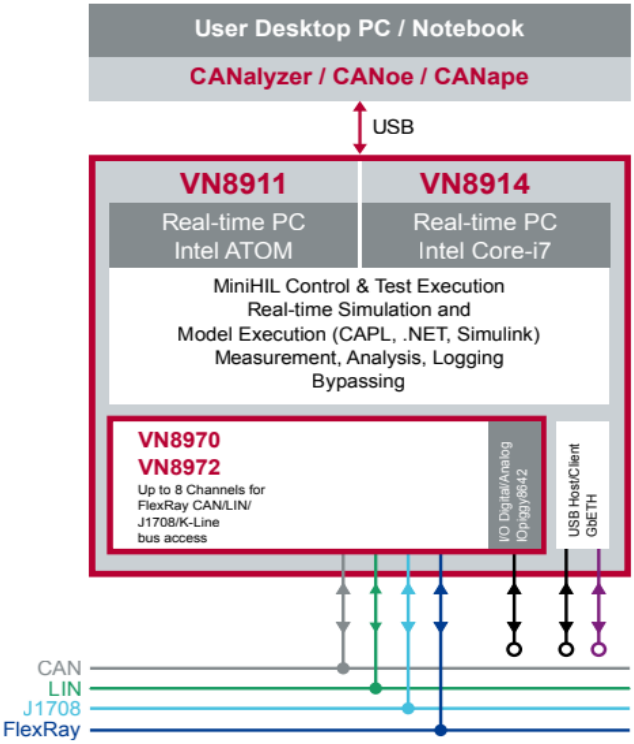
VN1630

Produktová řada označená VN1630A nabízí podporu obou sběrnic CAN a LIN. Napájení samotného rozhraní je opět provedeno skrze USB. Výhodou oproti VN1610 je možnost využití až čtyř kanálu současně, a to až do 100% zatížení sběrnice. Dále pak umožňuje připojení velmi rychlými přenosovými rychlostmi, konkrétně pro CAN až 1 Mbit/s, pro CAN-FD až 8 Mbit/s a pro LIN až 330kbit/s. Dále je dostupný autonomní režim (stand-alone mode) zařízení (rozhraní), během kterého je možné např. ukládat data naměřené ze sběrnice přímo na SD/SDHC kartu vloženou do těla rozhraní. Využití může být například během měření za jízdy přímo ve vozidle pro monitorování a následnou analýzu datové komunikace na sběrnicích vozidla.

Obr. 5: VN1630A CAN rozhraní převzato z [13]

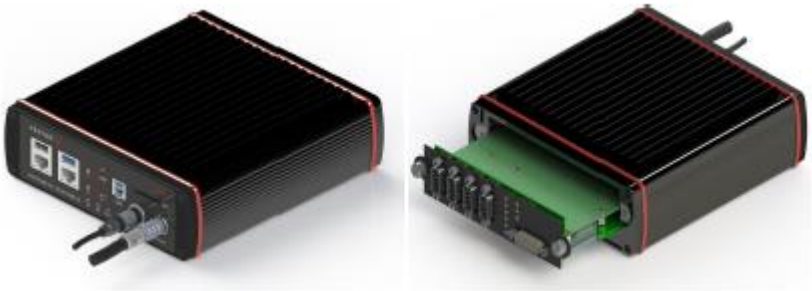
2.4.2 Modelová řada VN89xx

Rozhraní řady VN89xx je výbornou volbou pro vysoce výkonné aplikace zvláště v kombinací s využitím simulačního nástroje CANoe. Klíčová funkce těchto rozhraních je možnost nasazení a vlastního provozu bez připojení jakéhokoli uživatelského PC, neboť zařízení funguje jako samotné počítač pracující v režimu reálného času, tudíž bez negativních dopadů, které je možné pozorovat v případě běhu uživatelské aplikace (např. CANoe) na PC s běžným operačním systémem. V situacích, kde jsou přísné požadavky na přesnost časování, musí být hardware monitorující a případně i stimulující komunikaci na sběrnici schopen pracovat s nízkou latencí.



Obr. 6: Princip měření s real-time PC převzato z [14]

Z Obr. 6: Princip měření s real-time PC je zřejmá možnost propojené rozhraní řady VN89xx přes USB s aplikací CANoe spuštěné na běžném PC. Při využiti rozhraní s aplikací CANoe dochází k provádění testovacích funkcí a simulací v reálném čase na rozhraní, grafické a uživatelské rozhraní je díky tomu odděleno. Konfigurace simulace a vyhodnocení (analýza) naměřených dat se provádí v „offline“ režimu na běžném PC (CANoe), zatímco samotná exekutiva simulačního a testovacího jádro je zajištěna počítačem reálného času, který je vestavěný přímo v šasí VN89xx rozhraní (CANoe RT – RealTime). Spojení mezi PC aplikací CANoe a rozhraním VN89xx je realizována variantně přes USB, popř. Ethernet.



Obr. 7: VN8900 rozhraní převzato z [14]

Podobně jako je popsáno v 2.4.1, také pro vyšší modelovou řadu VN89xx platí, že umí pracovat v samostatném režimu (stand-alone mode). Simulace nebo skripty nakonfigurované v CANoe mohou být zapsány přímo do trvalé paměti, které je vestavěna ve VN89xx. Po zapnutí napájení nebo restartování zařízení se automaticky spustí nahraná konfigurace.

Výhodou je podpora dostupných základních sběrnic jako jsou CAN, CAN-FD, FlexRay, LIN, J1708 a K-Line. Díky své modularitě je možnost vyměnit na zařízení budiče nazývané také jako "Piggybacks": Ty nám zajistí zmíněnou podporu sběrnic, protože v základním dodaném modulu nemusí být například. FlexRay/LIN budič, ale jen nejpoužívanější CAN.

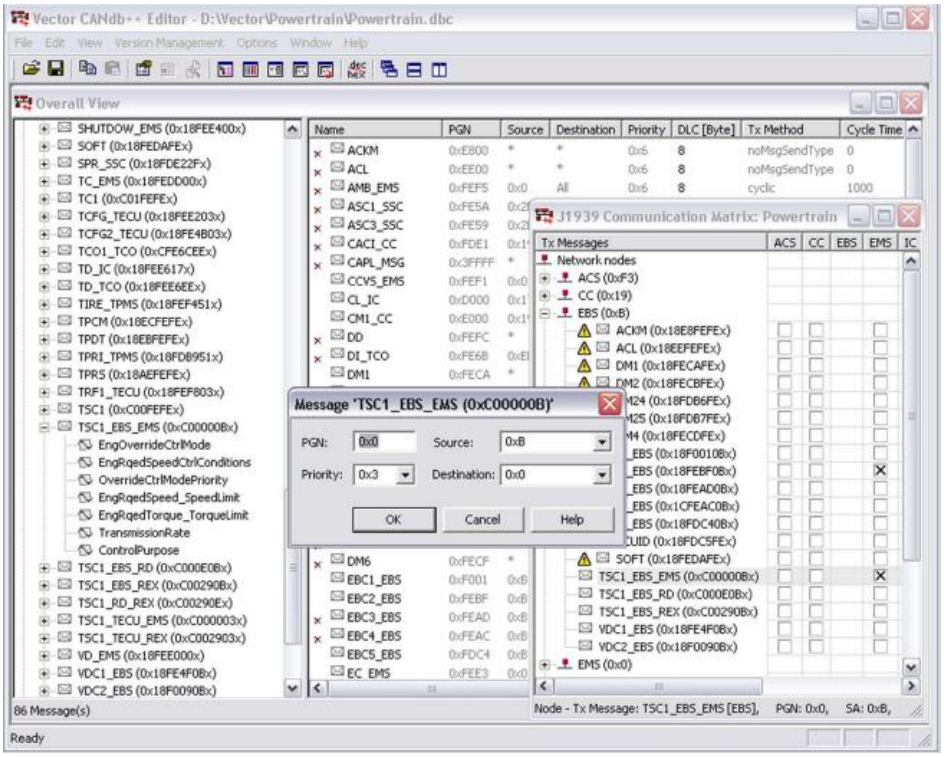
Obr. 8: Vector Piggybacks převzato z [13]

2.5 Komunikační matice – databáze elementů

Databáze popisují vlastnosti (obecný popis funkce a základních parametrů) jednotlivých ECU (Electronic Control Unit) připojených ke sběrnici a dále pak především jejich formát komunikačních zpráv a signálů, které jsou těmito zprávami přenášeny. V simulačním nástroji CANoe lze tyto databáze symbolicky zobrazit a dále využít.

Databáze jsou děleny podle podporované komunikační sběrnice a jsou definovány specifickým datovým (souborovým) formátem. Pro popis sběrnice CAN se používá databáze DBC, sběrnice LIN je popsána datovým formátem LDF, sběrnice MOST obecným XML a pro složitější sběrnice FlexRay je určen dedikovaný formát FIBEX. Obecná snaha unifikace přístupu k širokému množství sběrnic instalovaných ve vozidlech vyústila v definici komunikační matice prostřednictvím ARXML souboru, kterým lze definovat parametry a elementy různých sběrnic jako jsou CAN, FlexRay a Ethernet.

Na Obr. 9: CANdb++ editor pro databáze převzato z [1] lze vidět vzorový příklad CAN databáze pro příslušné zprávy a signály. Pro vytvoření a úpravu CAN databází slouží CANdb editor, která je součástí instalačního balíčku simulačního nástroje CANoe.



Obr. 9: CANdb++ editor pro databáze převzato z [1]

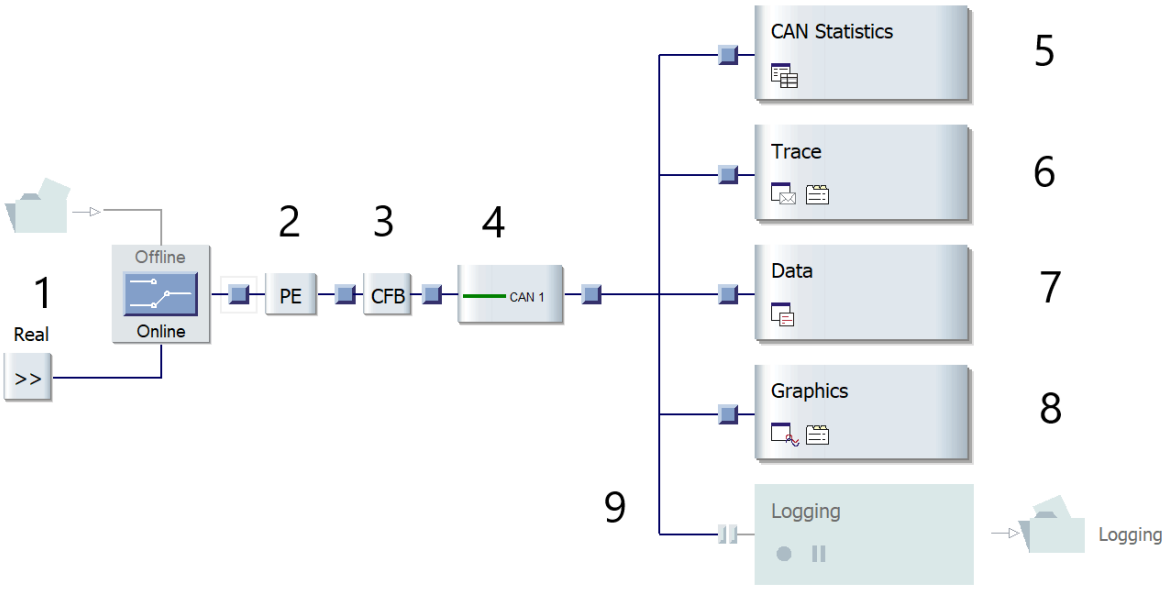
2.5.1 ARXML – AUTOSAR XML

Soubor ARXML je konfigurační datový formát, který vycházející z obecného XML formátu, specifikovaný a standardizovaný konsorciem AUTOSAR. Konfigurační soubory ARXML obsahují informace o popisu systému (či subsystému vozidla), konfiguraci elektronických řídicích jednotky, nebo popisu softwarové komponenty. Formát ARXML byl vyvinut za účelem standardizace výměny dat mezi partnery pro vývoj softwaru pro automobilový průmysl. Soubory jsou vytvářeny výrobci automobilů (OEM) a poskytovány dodavatelům a vývojářům dílčích řešení a komponent. Informace v každém souboru se proto můžou lišit.

3 Analýza

3.1 Nastavení měření

Adekvátní nastavení je nezbytné realizovat před každým měřením konkrétní konfigurace. Obr. 10: Nastavení měření (measurement setup) naznačuje tok měřených a nahrávaných dat směrem od zdroje dat po příjemce.



Obr. 10: Nastavení měření (measurement setup)

Číslování následující podkapitol odpovídá indexům funkčních bloků na Obr. 10: Nastavení měření (measurement setup).

3.1.1 Měření – zdroj dat

Zdroj dat může být online nebo offline. Online zdroj dat slouží pro analýzu reálných dat na sběrnici přes připojené rozhraní (např.VN1610). Offline zdroje dat mohou být soubory se zaznamenanými daty uložených v souborových formátech s příponami *.asc, *.blf, *.mdf, nebo *.mf4.

3.1.2 Měření – filtr proměnných

PASS/STOP variable filtr slouží pro filtrování jak databázových proměnných, tak i systémových.

3.1.3 Měření – filtr CAN zpráv

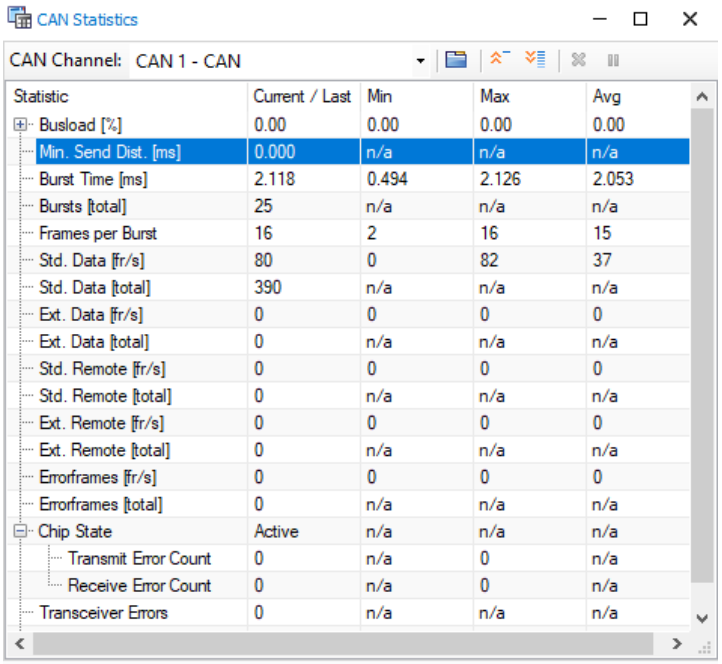
CAN filtr je jeden z nejdůležitějších filtrů. Lze vyfiltrovat celé databáze nebo jednotlivé vybrané zprávy z databáze. Dále lze také filtrovat zprávy na základě indikátoru. To je užitečné v případě, kdy není k dispozici dostupná databáze zpráv, ale jen reálná data.

3.1.4 Měření – filtr CAN kanálu

Channel filtr vyfiltruje potřebná data z vybraných kanálů. Využití je zejména při více kanálovém měření.

3.1.5 Měření – statistika provozu na CAN sběrnici

Okno CAN statistic zobrazuje statistické informace o aktivitách během měření na vybraných sběrnicích CAN, LIN nebo FlexRay - Obr. 11: Statistické okno (statistic window).



Statistic	Current / Last	Min	Max	Avg
Busload [%]	0.00	0.00	0.00	0.00
Min. Send Dist. [ms]	0.000	n/a	n/a	n/a
Burst Time [ms]	2.118	0.494	2.126	2.053
Bursts [total]	25	n/a	n/a	n/a
Frames per Burst	16	2	16	15
Std. Data [fr/s]	80	0	82	37
Std. Data [total]	390	n/a	n/a	n/a
Ext. Data [fr/s]	0	0	0	0
Ext. Data [total]	0	n/a	n/a	n/a
Std. Remote [fr/s]	0	0	0	0
Std. Remote [total]	0	n/a	n/a	n/a
Ext. Remote [fr/s]	0	0	0	0
Ext. Remote [total]	0	n/a	n/a	n/a
Errorframes [fr/s]	0	0	0	0
Errorframes [total]	0	n/a	n/a	n/a
Chip State	Active	n/a	n/a	n/a
Transmit Error Count	0	n/a	0	n/a
Receive Error Count	0	n/a	0	n/a
Transceiver Errors	0	n/a	n/a	n/a

Obr. 11: Statistické okno (statistic window) převzato z [1]

Statistické okno zahrnuje další potřebné údaje k analýze datového provozu na sběrnici.

- Bus load neboli zatížení celé sběrnice může být velice užitečné pro analýzu chování všech elektronických řídicích jednotek v automobilu, kdy jejich vysoké celkové zatížení může způsobovat výpadky v datové komunikaci a v důsledku vést k chybnému chování komunikačních uzlů.

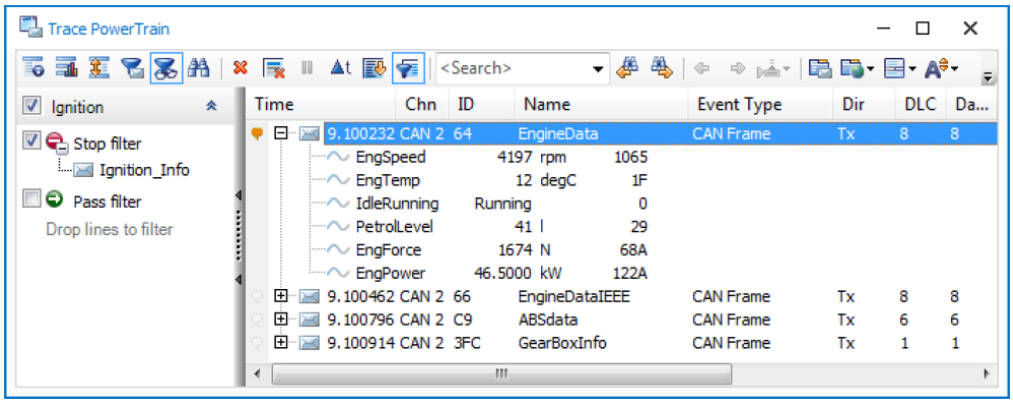
- Burst time označuje čas požadovaný procesem pro jeho provedení.

- Frame-bursting je funkce komunikačního protokolu používaná ve vrstvě spojení v komunikačních sítích ke změně charakteristik přenosu, aby bylo možné těžit z vyšší propustnosti. Jedná se o techniku, která se někdy používá v komunikačních protokolech pro sdílená média k dosažení vyšší propustnosti tím, že umožňuje vysílači posílat sérii rámců za sebou, aniž by se vzdal kontroly nad přenosovým médiem. [17]

- Error frame neboli chybový rámec je speciální zpráva, která porušuje formát zprávy CAN. Pokud uzel zjistí chybu ve jakékoliv přijaté zprávě, dojde k vyslání chybového rámce na sběrnici. Následně i ostatní uzly na sběrnici začnou vysílat chybový rámec.

3.1.6 Měření – analýza zpráv a signálů

Sledovací okno (trace window) slouží pro analýzu zpráv a signálů přítomných na sběrnici - Obr. 12: Sledovací okno (trace window) převzato z [1]

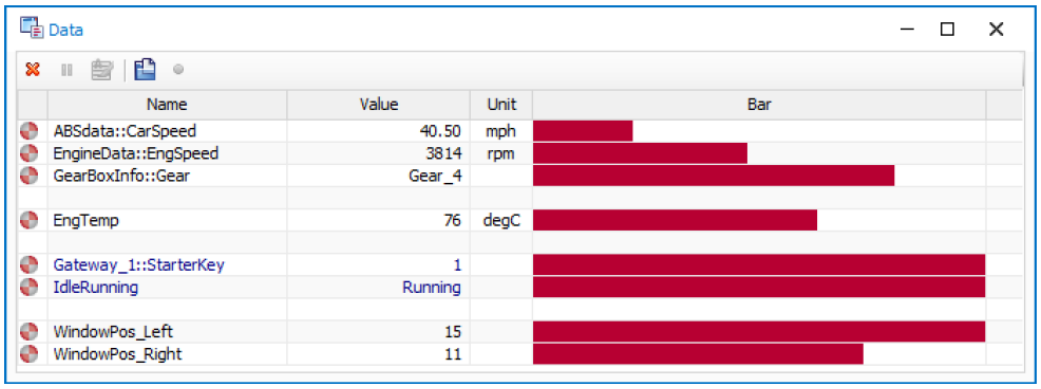


Obr. 12: Sledovací okno (trace window) převzato z [1]

Každá zpráva přítomná na sběrnici má svojí časovou značku, identifikátor, kanál, název, směr, délku a data. Název zprávy a popis signálu je viditelný a dostupný díky CAN databázi, např. ve formátu ARXML. Při nastavení měření se nastavují globální filtry sloužící k propouštění nebo blokování potřebných dat, jak je popsáno v předchozích kapitolách. Filtry v sledovacím okně fungují na jiném principu. Jsou užitečné pro vyfiltrování potřebných dat pouze pro analýzu dat. Data a události, které jsou pro daný scénář a případ použití důležité, lze barevně zvýraznit.

3.1.7 Měření – datová analýza (hodnoty signálů)

Datové okno (data window) slouží pro zobrazení hodnot přenášených ve zprávách, resp. v konkrétních signálech. Zobrazení poskytuje informace např. o aktuální hodnotě signálu, jednotce fyzikální hodnoty příslušného signálu, dále pak grafické znázornění aktuální hodnoty - Obr. 12: Sledovací okno (trace window) převzato z [1].



Obr. 13: Datové okno (data window) převzato z [1]

Měření – grafická analýza (časové průběhy hodnot signálů)

Grafické okno (graphic window) slouží ke grafickému zobrazení hodnot signálů, dat prostředí a diagnostických parametrů. V grafickém zobrazení lze použít, či konfigurovat- Obr. 14: Grafické okno (graphic window) převzato z [1]:

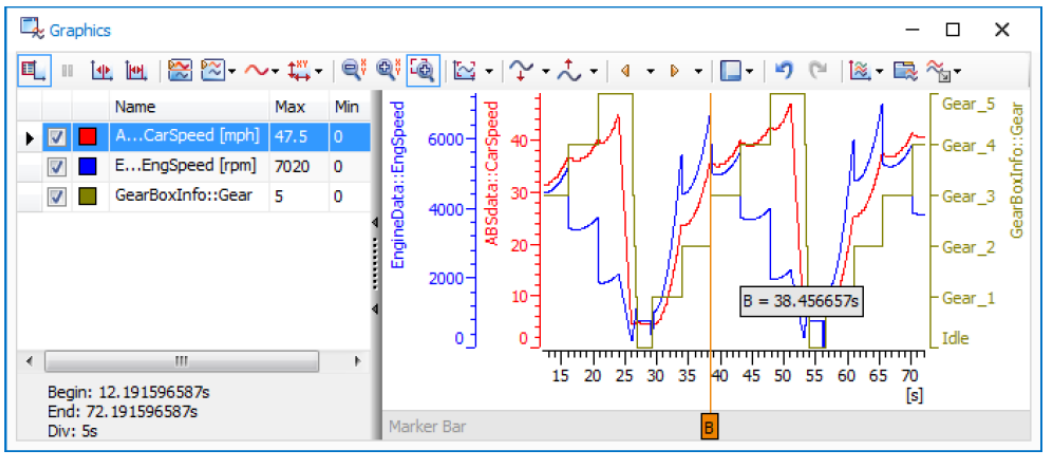
1. značky pro měření absolutních nebo relativních hodnot;

2. nastavení legendy, osyX a osyY;

3. statistické údaje;

4. minimální, maximální, směrodatné odchylky je také možné;

5. synchronizace se sledovacím oknem pro lepší přehlednost a orientaci během analýzy dat.



Obr. 14: Grafické okno (graphic window) převzato z [1]

3.1.8 Měření – nahrávání naměřených dat

Nahrávání (Logging) umožňuje zaznamenat přijatá data, která mohou být využita pro pozdější analýzu. V nahrávacím okně lze nastavit spuštění nahrávání měření pro celou simulaci, pro určitý časový úsek nebo manuální spuštění přes vybrané klávesové zkratky. Samozřejmostí je možnost výběru cesty pro ukládání nahraných dat. Možnost filtrování určitého typu dat jako jsou chybové, interní, diagnostické, komunikační události je dostupná v tomto okně.

4 CANoe – simulační nástroj

Vývoj komplexního systému, jakým je právě automobil, probíhá současně (paralelně) prováděným vývojem a testováním jednotlivých subsystémů, které tvoří dílčí funkční celky v závěru integrovaného systému. Na začátku vývojového cyklu celého systému je tedy nezbytné definovat chování jednotlivých subsystémů, tudíž specifikovat jejich funkční (jakou funkci musí subsystém realizovat) a nefunkční (za jakých podmínek, resp. při splnění všeobecných kritérií jako např. výpočetní náročnost, spotřeba systémových zdrojů apod. musí fungovat) požadavky na takové subsystémy.

Díky specifikaci chování jednotlivých subsystémů a definici rozhraní mezi těmito subsystémy je možné simulovat chování celého systému. Taková simulace systému pak umožňuje vytvořit vývojové a testovací prostředí dílčího subsystému, tzv. testovaného zařízení, tj. device under test (DUT), které může být např. v terminologii firmy Vector Informatik označováno jako tzv. system under test (SUT).

Testovaný subsystém se obecně neomezuje jen na datovou komunikaci (jako např. sběrnice CAN), ale také interaguje se svým prostředím prostřednictvím ostatním informačních kanálu (vstupy a výstupy analogových a digitálních signálů). Za tímto účelem lze simulaci podle potřeby rozšířit o reálná rozhraní pro I/O řadiče, akční členy a senzory. Skutečné akční členy a senzory však nelze vždy použít, a proto se obvykle používají náhradní obvody, např. formou zátěže. [18]

V závislosti na rozsahu požadované úrovně („hloubce“) simulace CANoe mohou ovlivňovat chování simulace také definice faktorů prostředí a fyzikálních účinků. [18]

Konkrétní elementy sestaveného modelu (systému), který je během simulace spuštěn, vytváří reálné nebo simulované vzájemně komunikující uzly vzájemně datové komunikace. CANoe umožnuje generaci simulačního prostředí ze souboru ARXML, čímž jsou vytvořeny modely (dílčí simulace) všech zainteresovaných elektronických řídicích jednotek a jejich zpráv prostřednictvím CANoe funkce “Model Generation Wizard”. Další alternativou je pak definice simulačního prostředí prostřednictvím procedur v jazycích CAPL a C, či prostředky z knihovny .NET (C#) firmy Microsoft

4.1 CANoe – interaktivní generátor

CANoe nabízí rozsáhlou sadu knihoven protokolů pro vytváření simulace komunikačních uzlů sběrnice. Umožňuje implementovat takové funkce jako např. správa sítě na základě konkrétního standardu OEM nebo AUTOSAR. Použití standardizovaných transportních protokolů zjednodušuje proces nastavování simulačních nebo testovacích aplikací, protože tyto služby jsou již plně integrovány.

Dále je možné generovat chybové stavy v datové komunikaci za účelem testování chování konkrétní implementace komunikačního ovladače příslušné ECU. CANoe disponuje tzv. interakčními vrstvami, které umožňuji oddělení signálové vrstvy komunikačního protokolu od aplikační vrstvy, a tím poskytují jistou míru (úroveň) abstrakce z pohledu chování vlastní simulace aplikační logiky konfigurovatelné v souladu se specifikací konkrétního OEM.

CANoe podporuje následující standardy protokolů: [1]

- diagnostické protokoly: KWP2000 a UDS (ISO 14229);
- správa sítě (NM, tedy network management): AUTOSAR, OSEK-NM;
- transportní protokoly (TP): ISO / DIS 15765-2, CMDT (J1939), BAM (J1939), TP pro MOST, LIN a FlexRay.

V současné době CANoe podporuje také specifická rozšíření TP, NM a IL pro 20 OEM, jako např. BMW, Daimler, VAG, Audi, Ford, Toyota, Fiat, Porsche, či Renault. Tato rozšíření usnadňují generování celé zbývající simulace sběrnice, včetně NM a TP. Tím také odpadá manuální psaní kódu - např. pro odesílací model. Jako základ zde slouží správně parametrizovaný soubor popisu sítě, ve kterém je nakonfigurováno celé chování odesílání, a který také definuje takové informace, jaké uzly účastnící se NM. Průvodce generováním modelu je pak schopen z takového souboru s popisem vygenerovat celou zbývající simulaci sběrnice pro CANoe. [1]

4.2 CAPL skript

Communication Acces Programming Language neboli CAPL je programovací jazyk vyvinutý společnosti Vector Informatik pro interní využití simulačního nástroje CANoe. Základní charakteristikou pro tento programovací jazyk je blízkost syntaxe k programovacímu jazyku C. Programovací jazyk CAPL však poskytuje vyšší míru abstrakce, tedy uživatelského zjednodušení oproti C např. v možnosti, že CANoe přebírá kontrolu nad všemi událostmi na sběrnici a je tedy možné s nimi přímo pracovat jako s datovými strukturami. Další podpora spočívá v symbolickém přístupu ke všem informacím (objektům a elementům) z databází (např. ve formátu ARXML) jako jsou zprávy, signály ve své fyzické podobě. Nechybí zde ani podpora rozšíření o externí knihovny .DLL.

Použití CAPL scriptu lze rozdělit na dvě témata:

1. Simulace: jak již bylo zmíněno, pomocí CAPL skriptů lze vytvořit úplnou simulaci zbývající sběrnice (koexistující s DUT), což vývojáře zbavuje denní rutinní práce. Databázové soubory DBC, LDF, FIBEX nebo ARXML jsou nejčastěji udržovány a spravovány centrálně a dodávané zákazníkem. Pomocí dostupných DDL knihoven (Node layer DDLs) mohou být databázové informace využity bez nutnosti definice dalšího uživatelské kódu.

2. Testování: programovací jazyk CAPL se dá také využit pro účely automatizovaného testování a jejích následné vyhodnocení. Pro vytvoření testovací struktury postačí pár řádků kódu.

4.3 CAPL prohlížeč

Důležitou roli v programovacím jazyku CAPL hraje prohlížeč, který nabízí pokročilé funkce vývojového prostředí:

- automatická kontrola syntaxe a dokončování kódu;
- konfigurovatelné zvýraznění syntaxe;
- skládání funkčních bloků;
- seznam funkcí s přímým kopírováním do zdrojového kódu;
- pomoc s odkazy na funkce;
- databázové objekty jsou zobrazeny ve stromovém zobrazení.

Ladění (debuggování) je možné v CAPL prohlížeči, avšak má to své úskalí. Debuggování lze používat jen v simulovaném režimu, protože pro tento účel je simulace zastavena. Při použití skutečného hardwaru debuggovat nelze, protože události odesílané pomocí hardwaru je třeba následně vyhodnotit.

5 CANoe – testovací nástroj

CANoe se využívá i na jiné účely, než je simulace. Lze ji využít také na kompletní testování elektronických řídích jednotek. Testy jsou využívány k ověřeních kroků individuálního vývoje (dílčích změn). Dále k provádění regresních testů a testování prototypů. V CANoe jsou sekvenční testovací toky implementovány jako testovací jednotky nebo testovací moduly, které jsou dále klasifikovány do testovacích skupin a jednotlivých testů. Jednotlivé jednotky a moduly lze spustit kdykoli během měření.

Testovací úlohy mohou být implementovány jednoduše a flexibilně, sestava testovacích funkcí obsahuje komponenty popsané v následujících kapitolách.

5.1 CANoe – testovací jednotky

Obsahují soubor s příponou *.dat, který obsahuje implementaci testů (např. skripty programované v CAPL / C#, testovací tabulky, testovací diagramy nebo soubory parametrů). Testovací jednotka může obsahovat například implementaci testu pro konkrétní ECU funkce. Testovací jednotky jsou spravovány v rámci testovacích konfigurací. Zkušební konfigurace může obsahovat [1… n] testovacích jednotek, které jsou postupně prováděny po sobě jdoucím způsobem. Nastavení systému CANoe na druhé straně může obsahovat jakýko li počet různých testovacích konfigurací, které lze provádět paralelně. Spustitelné testovací jednotky jsou vyvíjeny v dal39m n8stroji firmy Vector Informatik, tzv. vTESTstudio. Samostatný produkt vTESTstudio poskytuje následující koncepty pro efektivní design testů:

- správa parametrů a variant;
- klasifikační stromy pro efektivní generování testovacích dat;
- možnost využití interních a externích knihoven pro vysokou opětovnou použitelnost;
- sledovatelnost změn externích požadavků a specifikací testovacích případů;

- doplňkové nástroje pro připojení k externím aplikacím pro správu testovacích dat (např. IBM Rational Quality Manager, PTC)

- Produkt vTESTstudio nabízí různé metody návrhu pro implementaci testu:

- tabulkové;

- programovatelné (např. CAPL, C#);

- grafické (např. sekvenční testovací diagram, nebo stavový diagram).

5.2 CANoe – testovací moduly

Testovací moduly mohou být implementovány v XML, CAPL nebo .NET. V modulech XML jsou testy sestavovány z předdefinovaných testovacích vzorů a je snadné je parametrizovat pomocí vstupních a výstupních vektorů. Testovací moduly CAPL a .NET jsou naproti tomu naprogramovány, a proto jsou velmi flexibilní kontrola kontroly toku. Testovací moduly .NET lze pohodlně vyvíjet v C # nebo VB.NET. Jiný popis formuláře lze kombinovat podle požadavků. Testovací moduly jsou spravovány v testovacích prostředích. Tato testovací prostředí obsahují testovací moduly i další funkční bloky pro provedení testu. Testovací prostředí se ukládají odděleně od konfigurace systému CANoe. Proto je snadné je znovu použít v různých projektech. Testovací moduly by se měly používat pouze v kontextu menších testovacích balíčků nebo pro rychlou kontrolu jednotlivých funkce testovaného subjektu. U všech ostatních případů použití je doporučeno realizovat organizaci v projektu vTESTstudio, kde testy mohou být navrženy a řízeny velmi efektivně.

Souběžně s prováděním testu lze zkontrolovat další stavy systému, například shodu s dobami cyklu jednotlivých zpráv. Tato omezení se automaticky přidají k vyhodnocení testu.

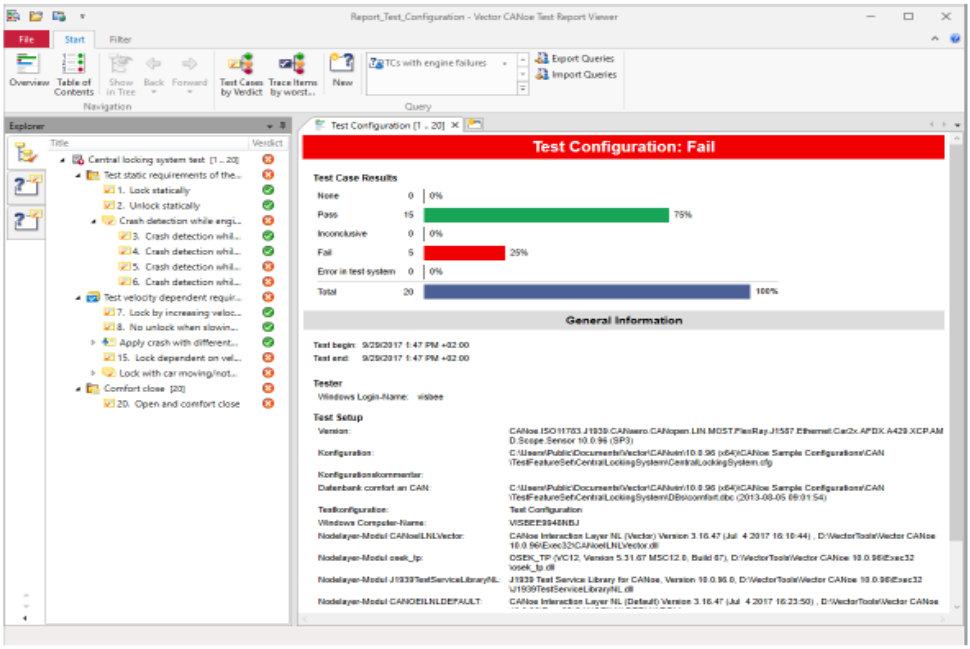
Knihovna testovacích služeb obsahuje kolekci připravených testovacích funkcí, které zjednodušují proces nastavování testů. Používají se v testovacích modulech a jsou parametrizovány prostřednictvím databáze. Například je možné sledovat doby cyklu zpráv, reakční dobu ECU od okamžiku přijetí zprávy do odeslání odpovědi zprávy nebo platnosti

hodnot signálu a diagnostických parametrů, nebo vyhodnotit kvalitu testovaných ECU na základě statistického vyhodnocení testů, například dle počtu hlášených odchylek za testovací období.

Po provedení testovacího modulu nebo testovací jednotky se vygeneruje rozsáhlý testovací protokol, který obsahuje informace o výsledku každého testovacího případu (test case), dále pak doprovodné uživatelské údaje, či doplňkové informace ve formě snímků např. oken s měřenými veličinami. Pro protokol o testu jsou k dispozici následující výstupní formáty:

- VTESTREPORT: CANoe zapisuje výsledky do velmi efektivního formátu souboru, který umožňuje uživatelům provádět rychlou a komplexní analýzu. Vytvořené soubory VTESTREPORT lze vizualizovat a analyzovat později pomocí prohlížeče reportů (CANoe Test Report Viewer). Tento nástroj je součástí instalace CANoe a lze jej stáhnout z webových stránek firmy Vector Informatik bez dodatečných nákladů.

- XML: CANoe zapíše výsledky do flexibilně dále zpracovatelného souboru XML. XML soubor je doprovázen příslušnou definicí šablony ve formátu XSLT.



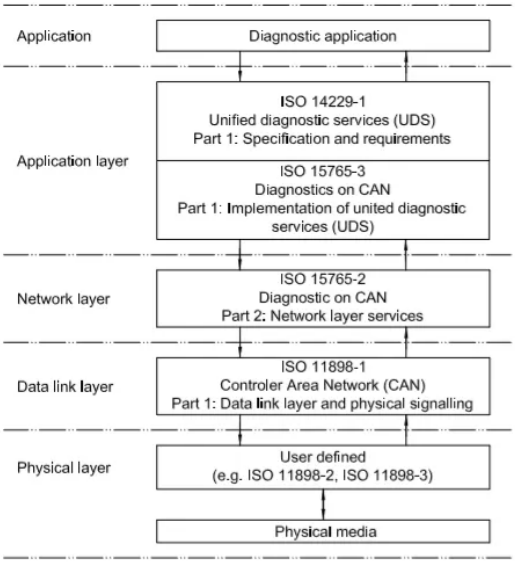
Obr. 15: Vector CANoe test report viewer převzato z [1]

6 CANoe – diagnostický nástroj

Simulační nástroj CANoe se dá využít také jako diagnostické zařízení. Sada diagnostických funkcí dodávaných s CANoe aplikací poskytuje podporu při analýze diagnostických standardů UDS a KWP2000. Diagnostické standardy jsou specifikovány v normě ISO 14229.

- UDS – Unified Diagnostic Services je diagnostický protokol používaný v elektronických řídicích jednotkách v automobilovém průmyslu po celém světě. Každý výrobce automobilů (OEM) je zavázán plnit požadavky normy ISO 14229 ve svých produktech. Tudíž je zajištěna snadná kompatibilita přístupu všech diagnostikovatelných elektronických jednotek v automobilu.

- KWP2000 – Protokol je založen na ISO 15765 a popisuje vrstvy 3. až 7. vrstvy referenčního modelu OSI a je určen pro diagnostické služby. Na konci 20. století nabízel protokol KWP2000 možnost flashování a programování elektronických řídících jednotek. V dnešní době je nahrazován UDS protokolem.



Obr. 16: OSI model pro UDS / KWP200 převzato z [15]

Z Obr. 16: OSI model pro UDS / KWP200 převzato z [15] je patrné, že UDS protokol pracuje na páté (relační) a sedmé (aplikační) vrstvě OSI modelu. Protokol CAN pracuje na první (fyzické) a druhé (linkové) vrstvě. V podstatě UDS protokol využívá ke komunikaci mezi serverem a klientem CAN protokol.

Služby poskytované prostřednictvím diagnostických protokolů jsou fundamentální pro celý životní cyklus produktu, tedy např. automobilu.

Automobil obsahuje mnoho elektronických řídících jednotek, které mezi sebou komunikují a předávají si reálná data např. informace o otáčkách motoru, teplotě oleje automatické převodovky, stavu DPF, apod. Díky diagnostice lze tato data vyčítat a vyhodnocovat. Využití může být také pro vyčítání a mazání poruchových kódů z chybové paměti automobilu. Při vývoji např. automatických převodovek je potřeba před dodáním softwaru zajistit, aby např. logika řazení automatické převodovky efektivně využívala co nejširší výkonové spektrum motoru. Analytická informace pro takové zefektivnění lze opět získat během jízdy diagnostikou pomocí vyčítání potřebných dat a následné kalibrace jednotlivých proměnných v elektronické řídicí jednotce automatické převodovky (TCU). V případě potřeby aktualizace softwaru v automobilu lze využít diagnostiku pro přenos velkého množství dat.

7 Praktické příklady

7.1 Praktická ukázka vyčítání diagnostické chyby z paměti

Jak již bylo zmíněno výše, simulační nástroj CANoe lze využít jak pro účely simulace, tak pro potřeby diagnostiky. Pro praktickou ukázku jsou vytvořeny dva simulační uzly (node). Simulační uzly jsou zobrazeny na Obr. 20: Nastavení simulace – uzel diagnostiky a uzel GearBoxECU.

Simulated GearboxECU uzel obstarává simulaci elektronické řídicí jednotky automatické převodovky. Jedná se pouze o demonstrativní příklad, kde je vytvořena jedna zpráva nesoucí informace týkající se teploty oleje automatické převodovky. Implementační detaily tohoto uzlu jsou popsány v 7.1.5 Uzel GearBox převodovky (Simulated GearBoxECU).

Diagnostic je druhým uzlem, který slouží pro simulaci diagnostického zařízení. Další informace o tomto uzlu jsou předmětem kapitoly 7.1.4 Diagnostický uzel (Diagnostic).

7.1.1 Definice CAN zprávy a jejich signálů

Všechny uzly jsou programované v CAPL skriptu. Automatické generování uzlu prostřednictvím interaktivního generátoru nebylo v tomto případě možné z důvodu nedostupnosti specifikace v datovém formátu ARXML, který je během vývoje reálného produktu typicky definován výrobcem automobilu a dále pak poskytován dodavatelům subsystémů a komponent. Pro účely této ukázky byl tedy využit alternativní a přímočarý způsob, a to definice ad-hoc CAN databáze. Tato definice byla realizována v nástroji Vector CANdb++ editor, který je součástí komplexní instalace nástroje CANoe. Takto vytvořená CAN databáze specifikuje CAN zprávy s konkrétními parametry vyhovujícími účelům této ukázky, jak naznačuje např. následující ukázka v Tab. 2: Definice zprávy v CANdb++ editoru.

Name	***ID***	***ID-format***	***DLC***	***Tx method***	***Cycle Time***
GearboxDATA	0x57D	CAN Standard	2	<n.a.>	0

Tab. 2: Definice zprávy v CANdb++ editoru

Příklad v Tab. 2: Definice zprávy v CANdb++ editoru definuje následující parametry CAN zprávy:

1. Významový název zprávy, který slouží pro snazší čitelnost a orientaci uživatele.

2. Identifikátor je pod označením ID a byla vybrána hodnota 0x57D. Jedná se o náhodně zvolenou hodnotu. Čím vyšší hodnota identifikátoru, tím nižší priorita zprávy na sběrnici.

3. Formát CAN zprávy byl vybrán jako CAN standard, protože pro účely poslání dvou Byte-ů zprávy je klasický CAN dostačující. Samozřejmostí je zvolení formátu CAN-FD v případě, že by byla potřeba délka zprávy více jak osm Byte-ů.

4. Cyklický čas byl zvolen nula, protože posílání zprávy je specifikováno a zajištěno CAPL skriptem

Po nadefinování zprávy je potřeba ve zprávě nadefinovat signály, se kterými se bude pracovat. Nadefinování je podobně jako u zprávy, rozdíl je v tom, že u signálu je potřeba nadefinovat více parametrů, jak ukazuje Tab. 3: Definice signálů pro zprávu GearboxDATA v CANdb++ editoru.

Name	***Startbit***	***Lenght [Bit]***	***Byte Order***	***Value Type***	***Initial Value***	***Minimum***	***Maximum***
TemperatureOfOIL	0	8	Intel	Unsigned	0	-128	127
AliveCounter	12	4	Intel	Unsigned	0	0	15

Tab. 3: Definice signálů pro zprávu GearboxDATA v CANdb++ editoru

Příklad v Tab. 3: Definice signálů pro zprávu GearboxDATA v CANdb++ editoru definuje následující parametry signálů TemperatureOfOIL a AliveCounter:

1. Každý signál má danou pevnou pozici ve zprávě. Pro signál s názvem TemperatureOfOIL, který obsahuje hodnotu teploty oleje automatické převodovky, je pozice ve zprávě nastavena na 0 (Starbit) a délkou signálu 8 bitů.

2. Datový typ je neznaménkový (unsigned) s inicializační hodnotou 0.

3. Minimální hodnota signálu je nastavena na -128 a maximální na 127.

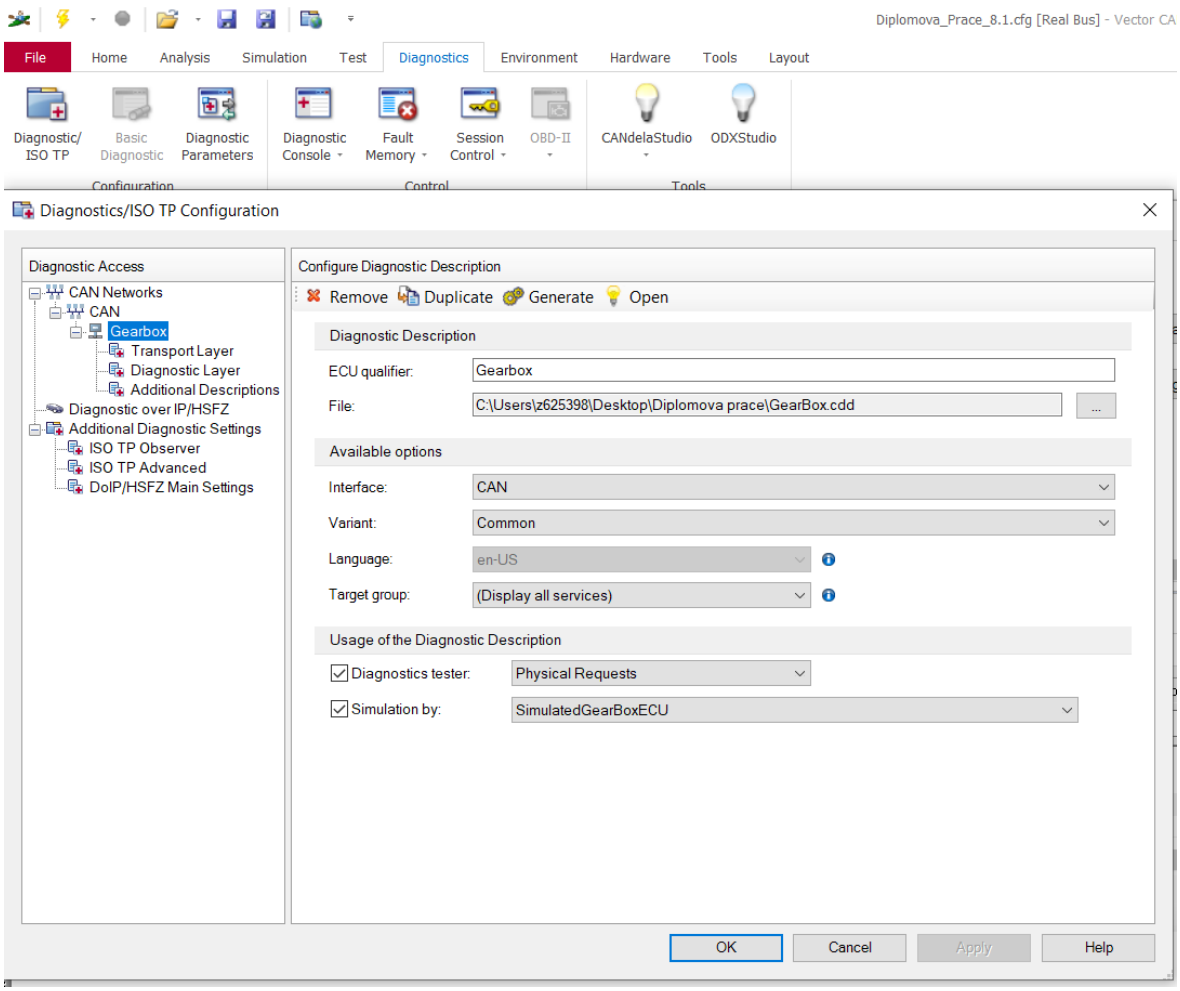
4. To samé platí pro druhý signál s názvem AliveCounter. Ten začíná na pozici 12 a má délku 4 bity. Počáteční hodnota je nastavena na 0 (minimální) a AliveCounter čítá do 15 (maximální). Po dosažení maximální hodnoty je AliveCounter vynulován a opět čítá od minimální po maximální hodnoty, tudíž časový průběh tohoto signálu je ve tvaru pily.

7.1.1.1 Pomocné signály CAN zprávy – čítač živosti (alive counter)

Alive counter se používá k detekci poruchového stavu zvaného „zmrazený rámec“ (neaktualizace přenášených dat) v komunikaci typu End-to-End (spojení mezi dvěma komunikačními uzly) např. na sběrnicích LIN a CAN. Hodnota alive counter-u se zvyšuje při každém požadavku na přenos na straně vysílače. Na straně přijímače se pak kontroluje, zda-li byla hodnota alive counter-u inkrementována (resp. zda-li se změnila) od předchozí přijaté hodnoty. Mechanismy vyhodnocení se liší dle požadavků na elektronickou řídicí jednotku, např. pokud se nekontroluje specifická hodnota inkrementu, ale vyhodnocuje se pouze změna (nekonstantnost) alive counter-u, pak je možné detekovat právě jen zmíněný stav „zmrazeného rámce“. [6]

7.1.1.2 Diagnostické popisy a parametry pro CAN zprávu a signály

Jakmile je definice zpráv a signálů hotová, je potřeba nakonfigurovat diagnostiku a její parametry. Nakonfigurování parametrů probíhá skrze okno Diagnostics > Diagnostic/ISO TP, jak naznačuje Obr. 17: Diagnostické konfigurační okno. Pro pracování na diagnostické vrstvě a používání diagnostických funkcí je potřeba přiřadit diagnostické popisy, které lze vygenerovat nebo naimportovat. Pro účely této práce byly diagnostické popisy importovány ze souboru s příponou *.cdd,



Obr. 17: Diagnostické konfigurační okno

K dispozici jsou čtyři druhy popisů:

- CDD – CANdelaStudio Diagnostic Description: .cdd soubory jsou vytvořeny v nástroji CANdelaStudio od společnosti Vector. Jsou podobné souborům .dbc pro nadefinování zpráv a signálů. Rozdíl je v tom, že díky. cdd souboru je možné získat symbolický přístup a reprezentaci diagnostických servisů a parametrů.

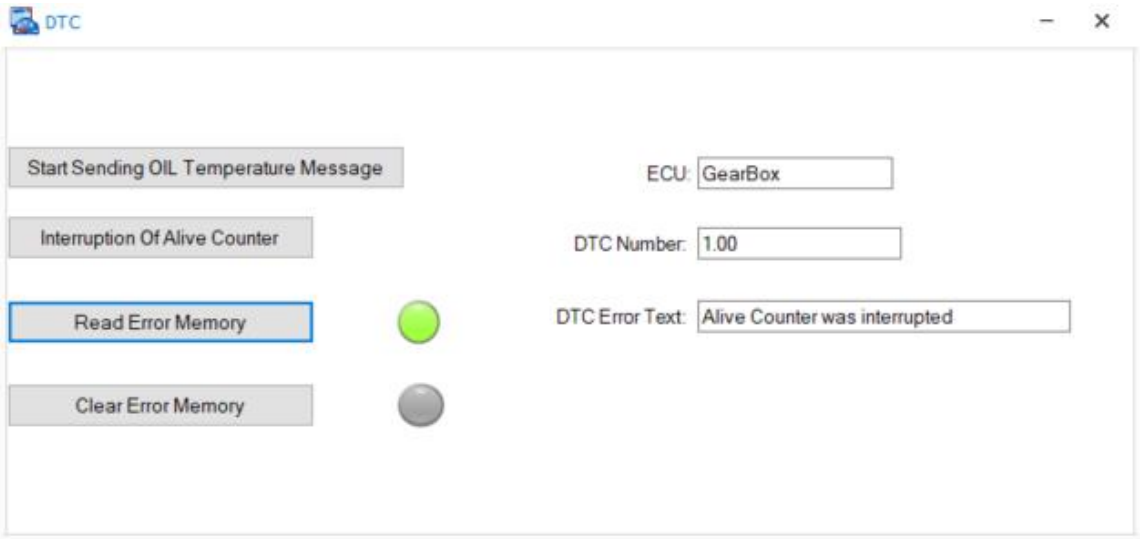
- Basic Diagnostic Description (UDS nebo KWP): základní diagnostický popis může být upraven uživatelem a lze ho vytvořit pomocí simulačního nástroje CANoe. Na rozdíl od ostatních formátů, je jeho funkcionalita omezena. Neobsahuje model pro vyčítání chybové paměti a model pro diagnostickou relaci. Diagnostická relace zajišťuje tok dat týkajících se diagnostických požadavků a odpovědí. Dohlíží a zaručuje správné načasování diagnostického protokolu a spravuje diagnostické stavy.

- ODX – Open Diagnostic Data Exchange – soubory ODX obsahují diagnostická data, které lze rozdělit do několika ODX souborů. Tyto soubory se ukládají do PDX souborů, tzv. ODX archívů. Použití těchto souborů je velice podobné jako použití .cdd souborů.

- MDX – Multiplex Diagnostic Data Exchange – Zákaznicky specifický formát MDX je velice podobný ODX.

7.1.2 CANoe – grafické prvky pro uživatelské ovládání a editaci

CANoe umožnuje vytvoření vlastních grafických prvků, které lze použít k úpravě proměnných, hodnot signálů a jejích zobrazení pomocí ukazovátek a posuvníků.



Obr. 18: Ovládací panel pro diagnostiku

Na Obr. 18: Ovládací panel pro diagnostiku je zobrazen vytvořený panel.

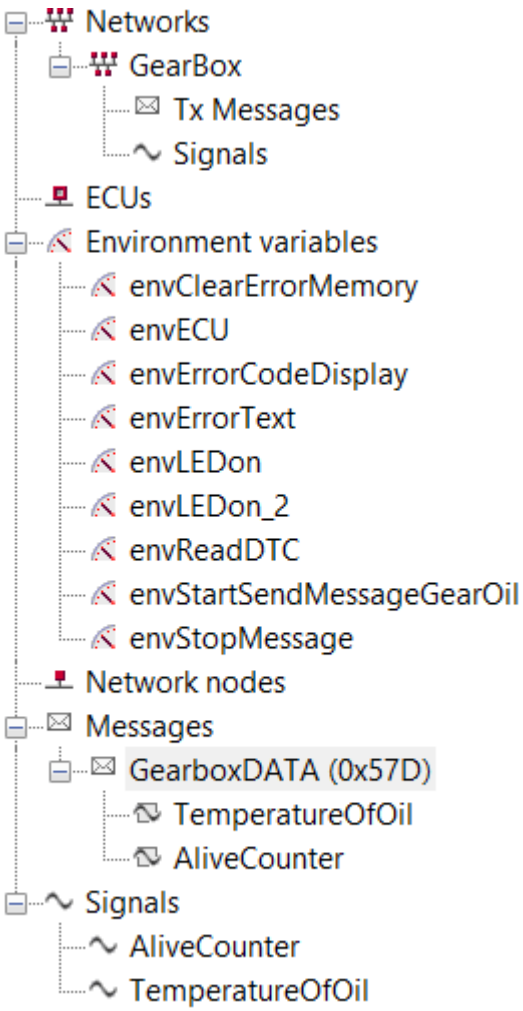
1. Posílání zprávy se zahájí stiskem tlačítka „Start sending OIL Temperature Message“.

2. Přerušení Alive Counter signálu ve zprávě je zajištěno druhým tlačítkem „Interruption Of Alive Counter“.

3. Tlačítko „Read Error Memory“ vyčte chybovou pamět pomocí diagnostického servisu „0x19 0x01“ a zobrazí text v datovém poli „DTC Error Text“ a následné zobrazení čísla chyby v datovém poli „DTC number“.

4. Tlačítko „Clear Error Memory“ vymaže paměť prostřednictvím diagnostického servisu „0x14 0xFF 0xFF 0xFF“.

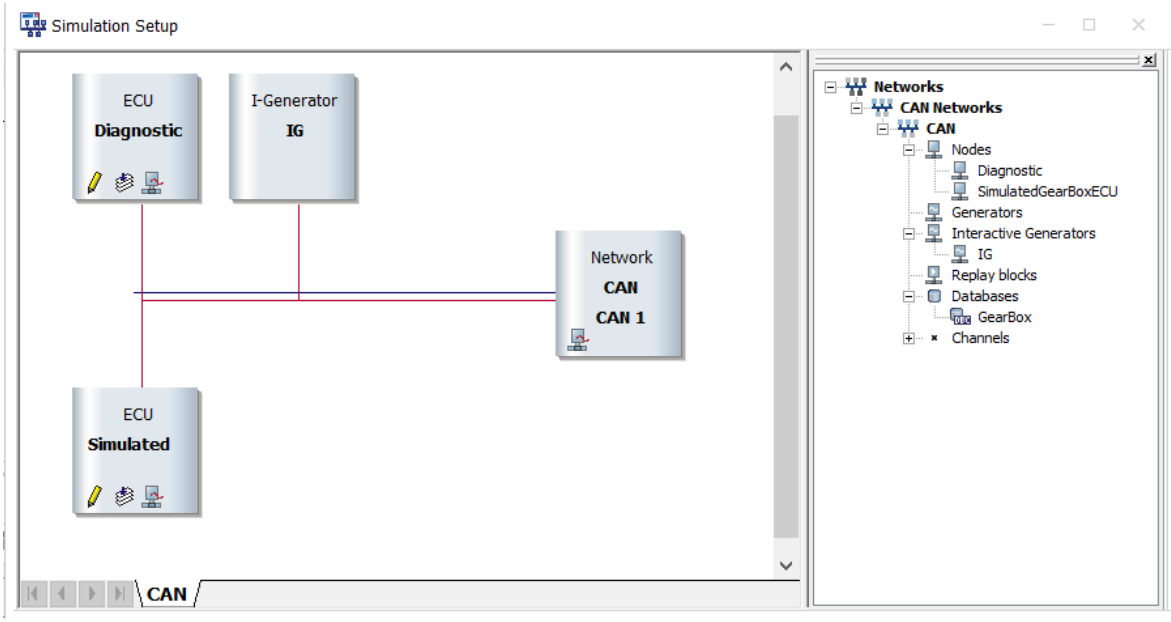
5. Grafický panel umožňuje interakci s CAPL skriptem prostřednictvím takzvaných „enviroment variable“, které jsou nadefinovány ve vytvořené CAN databázi, jak naznačuje Obr. 19: Definovaná CAN databáze – celkový náhled. Tyto proměnné mohou být definovány s různým datovým typem (Interger, String, Data, Float). Dále lze nastavit jejích inicializační hodnota, minimální a maximální hodnota.



Obr. 19: Definovaná CAN databáze – celkový náhled

7.1.3 CANoe – nastavení simulace a editace CAPL skriptů

Jakmile jsou všechny proměnné a zprávy nadefinované, mohou být využity pro zpracování kódem CAPL skriptu. Editace CAPL skriptu, který přísluší konkrétnímu komunikačnímu uzlu, je možná po kliknutí na žlutou tužku libovolného bloku, který odpovídá příslušnému uzlu, jak ukazuje Obr. 20: Nastavení simulace – uzel diagnostiky a uzel GearBoxECU.



Obr. 20: Nastavení simulace – uzel diagnostiky a uzel GearBoxECU

7.1.4 Diagnostický uzel (Diagnostic) – popis zdrojového kódu CAPL skriptu

Diagnostický uzel si lze představit např. jako zařízení v autoservisu připojené přes komunikační sběrnici (např. CAN) ke konkrétní elektronické řídicí jednotce automobilu. Diagnostika zahrnuje provádění rutin nebo služeb vzdáleně na elektronické řídicí jednotce v autě. Při provedení rutiny se odešle byte-ový řetězec jako požadavek na konkrétní elektronickou řídicí jednotku, která poté odpoví byte-ovým řetězcem zpět. V této práci slouží diagnostický uzel pro komunikaci se simulovanou řídicí jednotkou automatické převodovky.

7.1.4.1 Diagnostický uzel – deklarace proměnných

```
Variables
{
  msTimer LEDonTimer;
  msTimer LEDoffTimer;
  char ECU [50] = "GearBox";
}
```

Deklarace proměnných probíhá ve funkci {Variables}. Nadefinované proměnné v téhle funkci jsou globální pro celý skript. Proměnné jsou platné pouze pro lokální uzel. Nelze předávat proměnné mezi jednotlivými uzly.

Type	***Detail / Purpose***	***Name***	***Bit size***
Integers	Signed	int	16
		long	32
		int64	64
	Unsigned	byte	8
		word	16
		dword	32
		qword	64
Floating Point		float	64
		double	64
Single Charcter		char	8
Message Variable	for CAN	message	
Time Variables	for seconds	timer	
	for miliseconds	mstimer	

Tab. 4: Datové typy pro CAN

V Tab. 4: Datové typy pro CAN jsou zobrazeny všechny dostupné datové typy, které lze na deklarovat v poli ve funkci {Variables}. Pro účel diagnostického uzlu bylo využito datového typu „char“ a „mstimer“.

7.1.4.2 Diagnostický uzel – událost „on start“

```
on start
{
   putValue(envECU,ECU);
}
```

Událost {on start} je volána vždy na začátku pří spuštění simulace. V tomto případě je využita na vyplnění názvu do panelu vizualizace. Název je vyplněn hodnotou předchozí deklarace proměnné „GearBox“.

7.1.4.3 Diagnostický uzel – funkce „main“

```
void main()
{

}
```

Funkce {main} slouží jako hlavní tělo pro běh programu. V této práci tato funkce nebyla využita, protože celý běh programu je řešen skrze události a diagnostické funkce, které reagují na jednotlivé dotazy a odpovědi simulovaných uzlu.

7.1.4.4 Diagnostický uzel – událost „on envVar“ – proměnná „envClearErrorMemory“

```
on envVar envClearErrorMemory
{
  DiagRequest Gearbox.FaultMemory_Clear request;
  diagSendRequest(request);
}
```

Událost {on envVar} je volána vždy, když dojde k manipulaci „Environment“ proměnnou. Ve výše vloženém příkladu je událost vyvolána při stisknutí tlačítka na vymazání chybové paměti, jak naznačuje Obr. 18: Ovládací panel pro diagnostiku.

Třída „DiagRequest“ s odkazem na ECU „Gearbox“ a metodou „FaultMemory_Clear_request“ zajistí požadavek na smazání chybové paměti v simulovaném uzlu „Gearbox“. Jak již bylo zmíněné výše, všechny diagnostické objekty jsou nadefinované v GearBox.cdd souboru a k nim i příslušné diagnostické servisy. V souboru GearBox.cdd vytvořeném pro účely této práce má servis pro mazání chybové paměti specifikované následující parametry, jak ukazuje Tab. 5: Servis pro mazání chybové paměti. Konkrétní parametry např. servisu pro mazání chybové paměti se mohou lišit v závislosti na specifických požadavcích definovaných příslušným výrobcem automobilu (OEM).

SID	***SBF***	***Data Bytes***
0x14	0x00	0xFF 0xFF 0xFF

Tab. 5: Servis pro mazání chybové paměti

7.1.4.5 Diagnostický uzel – událost „on diagResponse“ – požadavek „Gearbox.FaultMemory_Clear“

```
on diagResponse Gearbox.FaultMemory_Clear
{
    if(diagIsPositiveResponse (this))
  {
    @envLEDon_2 = 1;
    setTimer(LEDoffTimer,5000);
    putValue(envErrorText,"");
    putValue(envErrorCodeDisplay,0);
  }
}
```

Odpovědi diagnostiky fungují na stejném principu jako požadavky. Zde se čeká na odpověď od simulovaného uzlu GearBox. Metoda „diagIsPositiveResponse“ vrací 0 v případě, že odpověď nebyla pozitivní. Metoda je využita při doručení pozitivní odpovědi k nastavení časovače a vložení konkrétních hodnot do proměnných prostředí. Za zmínku dále stojí odkaz metody na vlastní diagnostický objekt. To je zajištěno parametrem „this“.

7.1.4.6 Diagnostický uzel – událost „on envVar“ – proměnná „envReadDTC“

```
on envVar envReadDTC
{
  diagRequest Gearbox.FaultMemory_ReadNumber request;
  diagSendRequest(request);
}
```

V případě zavolání události {on envVar} skrze stisk tlačítka „Read Error Memory“, které ukazuje Obr. 18: Ovládací panel pro diagnostiku, dojde k odeslání diagnostického požadavku na vyčtení počtu chyb v chybové paměti. Metoda „diagSendRequest“ odkazující na diagnostický objekt „request“ pošle požadavek do ECU Gearbox. Požadavek na vyčtení z chybové paměti popisuje Tab. 6: Servis pro vyčtení chybové paměti.

SID	***SBF***	***Data Bytes***
0x19	0x01	0x00

Tab. 6: Servis pro vyčtení chybové paměti

7.1.4.7 Diagnostický uzel – událost „on diagResponse“ – požadavek „Gearbox.FaultMemory_ReadNumber“

```
on diagResponse Gearbox.FaultMemory_ReadNumber
{

  byte data[4096];
  long size;

  diagResponse * resp;
  size=this.GetPrimitiveSize();
  this.GetPrimitiveData(data, elcount(data));

  if (diagIsPositiveResponse(this))
  {
    @envLEDon = 1;
    setTimer(LEDoffTimer,5000);


    if((data.byte(0) ==0x59 )&&
       (data.byte(1) ==0x01 )&&
       (data.byte(2) ==0x01 )&&
       (data.byte(3) ==0x00 )&&
       (data.byte(4) ==0x00 )&&
       (data.byte(5) ==0x00 )&&
       (data.byte(6) ==0x00 )&&
       (data.byte(7) ==0x00))
       {
         write("DTC was stored correctly");
         @envErrorCodeDisplay = 0x01;
         putValue(envErrorText, "Alive Counter was interrupted");
       }

    else
       {
        write("DTC was not stored correctly");
        @envErrorCodeDisplay = 0x00;
        putValue(envErrorText, "There is no error in the error memory");
       }
     }
  }
```

Tato funkce zpracuje odpověď od jednotky GearBox a dojde k přepočítání odpovědí a následnému porovnání skrze rozhodovací podmínku. Aby bylo možné získat data z diagnostické odpovědi, je potřeba využít funkcí {GetPrimitiveSize}, která spočítá délku odpovědi a následně skrze funkci {GetPrimitiveData} zkopíruje aktuální odpověď do proměnné „Data“. Jakmile jsou data naplněny, dochází ke kontrole, zda byla obdržena pozitivní odpověď. V případě pozitivní odpovědi dojde k ověření, zda v pozitivní odpovědi byla chyba na pozici druhého byte-u, a poté dochází k potvrzení a ke kopírování aktuální hodnoty do proměnné pro vizualizaci.

7.1.5 Uzel GearBox převodovky (Simulated GearBoxECU)

Uzel Simulated GearBoxECU simuluje elektronickou řídicí jednotku automatické převodovky. Pro účely této praktické ukázky obecných mechanismů je pro tuto ECU definována právě jedna zpráva. Při požadavku na posílání skrze tlačítko „Start Sending OIL Temperature Message“, jak naznačuje Obr. 18: Ovládací panel pro diagnostiku, uzel nasimuluje zprávu a odešle jí skrze CAPL funkce na sběrnici. Uzel dále reaguje odpověďmi na diagnostické požadavky od uzlu Tester, který simuluje diagnostické zařízení.

7.1.5.1 Uzel GearBox – deklarace proměnných

```
variables

{
  message GearboxDATA myMessage;
  msTimer GearBoxDATATimer;
  int flag;
  int aliveCounterFlag;
}
```

Deklarace proměnných probíhá stejným způsobem jako u simulovaného uzlu diagnostiky. Pokud je potřeba deklarace zprávy, je potřeba zadat název zprávy z CAN databáze a poté jí přiřadit do vlastní proměnné. Další práce se zprávou probíhá v CAPL programu pod novým přiřazeným názvem. V deklaraci je využit časovač pod označením „msTimer“, kterým se definuje proměnná pro budoucí využití časovače k posílání zprávy na sběrnici. Proměnné „flag“ a „aliveCounterFlag“ slouží pouze pro indikaci čítání čítače.

7.1.5.2 Uzel GearBox – událost „on start“

Při začátku simulace dojde k resetování flagu.

```
on start
 {
   flag = 0;
 }
```

7.1.5.3 Uzel GearBox – událost „on envVar“ – proměnná „envStartSendMessageGearOil“

```
on envVar envStartSendMessageGearOil
{

  switch (getValue(this))
  {
   case 0: write("button released");
           break;

   case 1: write("button pressed");
            break;
  }

  flag = 1;
  setTimerCyclic(GearBoxDATATimer, 200);
  aliveCounterFlag = 0;

}
```

Stisknutí tlačítka pro posílání zprávy dojde k posílání zprávy. Funkce „setTimerCyclic“ nastaví časovač „GearBoxDATATimer“ na hodnotu 200 ms.

7.1.5.4 Uzel GearBox – událost „on timer“ – časovač „GearBoxDATATimer“

Po vypršení času přechází program do následující funkce „on timer GearBoxDataTimer“. Funkce „on timer“ jsou vždy volány po nastavení času „setTimer“.

```
on timer GearBoxDATATimer
 {
   int i;

      myMessage.byte(0) = 0x70

      for (i = 0; i <=15; i++)
      {
         myMessage.byte(1) = myMessage.byte(1) + 1;
         myMessage.AliveCounter = myMessage.AliveCounter + 1;
         output(myMessage);
      }
 }
```

Jakmile je funkce zavolána, dojde k naplnění na pozici 0 byte-u k hodnotě 0x70 hexadecimálně, což odpovídá hodnotě dekadické 112. Tato hodnota reprezentuje simulovanou teplotu oleje. V další části dochází k naplnění zprávy na 1. byte-u. Zde dochází přes cyklus „for“ k čítání alive counter od 0 do 15. Skrze „output(myMessage)“ dojde k odeslání zprávy na sběrnici s aktuální hodnotou teploty oleje a alive counteru.

7.1.5.5 Uzel GearBox – událost „on envVar“ – proměnná „envStopMessage“

```
on envVar envStopMessage
{
  cancelTimer(GearBoxDATATimer);
  aliveCounterFlag = 1;
}
```

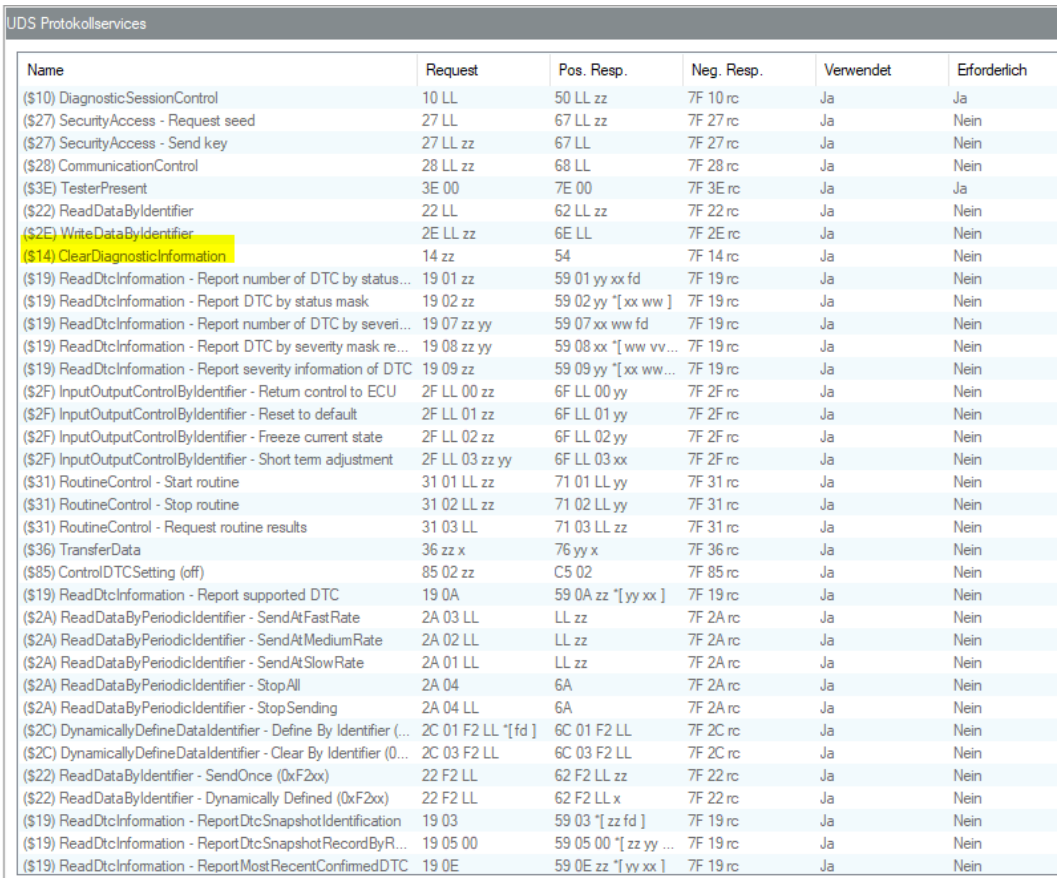
Při stisknutí tlačítka „envStopMessge“ dojde ke zrušení časovače skrze funkci „cancelTimer“ s parametrem časovače, který byl nastaven v předchozí funkci, a tím dojde k zastavení posílání zprávy.

7.1.5.6 Uzel GearBox – událost „on diagRequest“ – požadavek „Gearbox.FaultMemory_Clear“

```
on diagRequest Gearbox.FaultMemory_Clear
{
  DiagResponse this response;
  DiagSendResponse(response);

}
```

Při obdržení požadavku od diagnostického zařízení na smazání chybové paměti, jednotka odpovídá skrze funkci „DiagSendResponse“ pozitivní odpovědí. Odpověď je nadefinovaná v CANdela souboru Gearbox.cdd. Smazání probíhá servisem $14 a pozitivní odpověď $54 v tomto případě, jak naznačuje Obr. 21: CANdela soubor pro UDS diagnostické servisy.



UDS Protokollservices

Name	Request	Pos. Resp.	Neg. Resp.	Verwendet	Erforderlich
($10) DiagnosticSessionControl	10 LL	50 LL zz	7F 10 rc	Ja	Ja
($27) SecurityAccess - Request seed	27 LL	67 LL zz	7F 27 rc	Ja	Nein
($27) SecurityAccess - Send key	27 LL zz	67 LL	7F 27 rc	Ja	Nein
($28) CommunicationControl	28 LL zz	68 LL	7F 28 rc	Ja	Nein
($3E) TesterPresent	3E 00	7E 00	7F 3E rc	Ja	Ja
($22) ReadDataByIdentifier	22 LL	62 LL zz	7F 22 rc	Ja	Nein
($2E) WriteDataByIdentifier	2E LL zz	6E LL	7F 2E rc	Ja	Nein
($14) ClearDiagnosticInformation	14 zz	54	7F 14 rc	Ja	Nein
($19) ReadDtcInformation - Report number of DTC by status...	19 01 zz	59 01 yy xx fd	7F 19 rc	Ja	Nein
($19) ReadDtcInformation - Report DTC by status mask	19 02 zz	59 02 yy *[xx ww]	7F 19 rc	Ja	Nein
($19) ReadDtcInformation - Report number of DTC by severi...	19 07 zz yy	59 07 xx ww fd	7F 19 rc	Ja	Nein
($19) ReadDtcInformation - Report DTC by severity mask re...	19 08 zz yy	59 08 xx *[ww vv...	7F 19 rc	Ja	Nein
($19) ReadDtcInformation - Report severity information of DTC	19 09 zz	59 09 yy *[xx ww...	7F 19 rc	Ja	Nein
($2F) InputOutputControlByIdentifier - Return control to ECU	2F LL 00 zz	6F LL 00 yy	7F 2F rc	Ja	Nein
($2F) InputOutputControlByIdentifier - Reset to default	2F LL 01 zz	6F LL 01 yy	7F 2F rc	Ja	Nein
($2F) InputOutputControlByIdentifier - Freeze current state	2F LL 02 zz	6F LL 02 yy	7F 2F rc	Ja	Nein
($2F) InputOutputControlByIdentifier - Short term adjustment	2F LL 03 zz yy	6F LL 03 xx	7F 2F rc	Ja	Nein
($31) RoutineControl - Start routine	31 01 LL zz	71 01 LL yy	7F 31 rc	Ja	Nein
($31) RoutineControl - Stop routine	31 02 LL zz	71 02 LL yy	7F 31 rc	Ja	Nein
($31) RoutineControl - Request routine results	31 03 LL	71 03 LL zz	7F 31 rc	Ja	Nein
($36) TransferData	36 zz x	76 yy x	7F 36 rc	Ja	Nein
($85) ControlDTCSetting (off)	85 02 zz	C5 02	7F 85 rc	Ja	Nein
($19) ReadDtcInformation - Report supported DTC	19 0A	59 0A zz *[yy xx]	7F 19 rc	Ja	Nein
($2A) ReadDataByPeriodicIdentifier - SendAtFastRate	2A 03 LL	LL zz	7F 2A rc	Ja	Nein
($2A) ReadDataByPeriodicIdentifier - SendAtMediumRate	2A 02 LL	LL zz	7F 2A rc	Ja	Nein
($2A) ReadDataByPeriodicIdentifier - SendAtSlowRate	2A 01 LL	LL zz	7F 2A rc	Ja	Nein
($2A) ReadDataByPeriodicIdentifier - StopAll	2A 04	6A	7F 2A rc	Ja	Nein
($2A) ReadDataByPeriodicIdentifier - StopSending	2A 04 LL	6A	7F 2A rc	Ja	Nein
($2C) DynamicallyDefineDataIdentifier - Define By Identifier (...	2C 01 F2 LL *[fd]	6C 01 F2 LL	7F 2C rc	Ja	Nein
($2C) DynamicallyDefineDataIdentifier - Clear By Identifier (0...	2C 03 F2 LL	6C 03 F2 LL	7F 2C rc	Ja	Nein
($22) ReadDataByIdentifier - SendOnce (0xF2xx)	22 F2 LL	62 F2 LL zz	7F 22 rc	Ja	Nein
($22) ReadDataByIdentifier - Dynamically Defined (0xF2xx)	22 F2 LL	62 F2 LL x	7F 22 rc	Ja	Nein
($19) ReadDtcInformation - ReportDtcSnapshotIdentification	19 03	59 03 *[zz fd]	7F 19 rc	Ja	Nein
($19) ReadDtcInformation - ReportDtcSnapshotRecordByR...	19 05 00	59 05 00 *[zz yy ...	7F 19 rc	Ja	Nein
($19) ReadDtcInformation - ReportMostRecentConfirmedDTC	19 0E	59 0E zz *[yy xx]	7F 19 rc	Ja	Nein

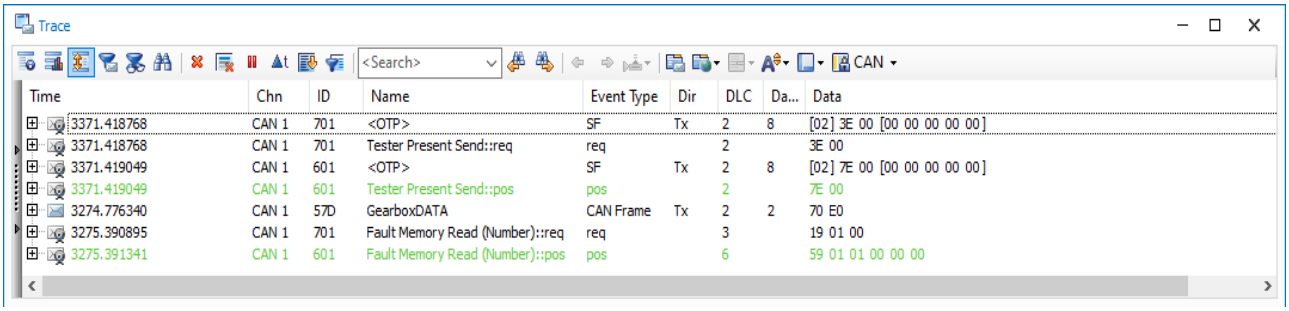
Obr. 21: CANdela soubor pro UDS diagnostické servisy

V CAPL skriptu a diagnostické konzoly lze poslat kterýkoliv servis nadefinovaný v tomto souboru.

```
on diagRequest Gearbox.FaultMemory_ReadNumber
{
  diagResponse Gearbox.FaultMemory_ReadNumber response;
  int i;
  if (aliveCounterFlag == 1 && flag == 1)
     {
       i = 1;
     }
  else
     {
      i = 0;
     }
  DiagSetPrimitiveByte(response,0,0x59);
  DiagSetPrimitiveByte(response,1,0x01);
  DiagSetPrimitiveByte(response,2,i);
  DiagSetPrimitiveByte(response,3,0x00);
  DiagSetPrimitiveByte(response,4,0x00);
  DiagSetPrimitiveByte(response,5,0x00);
  DiagSetPrimitiveByte(response,6,0x00);
  DiagSetPrimitiveByte(response,7,0x00);
  diagSendPositiveResponse(response);

}
```

Tato část kódu reaguje na požadavek vyčtení chybové paměti, který je odeslán diagnostickému uzlu. V případě, že dojde během přenosu na sběrnici k zarušení zprávy GearBoxDATA (tedy když aliveCounterFlag == 1), a zároveň je povolena diagnostika (detekce) takové chyby, pak je aktivní stav detekované chyby zaznamenám do pracovní proměnné (i = 1). Aktuální stav (aktivní vs. neaktivní chyba) je předán z pracovní proměnné do výstupní odpovědi na pozici byte 2. Vlastní odpověď je možno provést skrze funkci „DiagSetPrimitiveByte“. Výsledná zpráva se odešle na sběrnici, jak zobrazuje druhý výskyt zaznamenané zprávy s ID=601 na Obr. 22: Trace (záznam komunikace) po vyčtení chybové paměti s aktivní chybou.



Obr. 22: Trace (záznam komunikace) po vyčtení chybové paměti s aktivní chybou

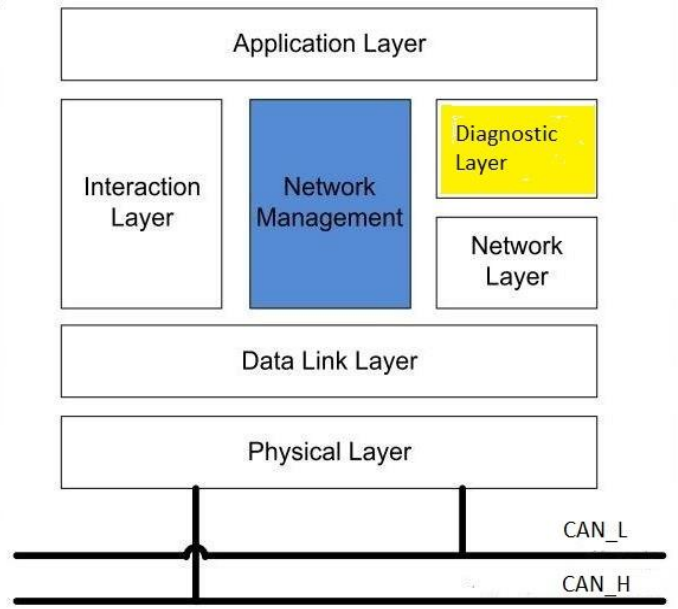
7.2 Praktická ukázka simulace network management (správy sítě)

7.2.1 Simulace Network managementu (správa sítě)

Network management (správa sítě) je nedílnou součástí každé elektronické řídicí jednotky v automobilu. Slouží ke správě a řízení stavu sítě, a tedy i stavů na síti připojených elektronických řídících jednotek.

Network management např. umožňuje efektivní řízení spotřeby elektrické energie. Elektronické řídicí jednotky musí minimalizovat svůj proudový odběr poté, co dojde k vypnutí zapalování v autě. Network management je aktivní také v opačném případě, tedy během inicializace sběrnicové komunikace, když po vložení klíčku do zapalování, si elektronické řídicí jednotky začnou vzájemně předávat informaci o svém probuzení a začínají následně mezi sebou komunikovat.

Network management je řešen přes architekturu AUTOSAR, která umožňuje jeho implementaci. Umístění části network management v rámci implementace komunikačního zásobníku zobrazuje Obr. 23: CAN – rozdělení vrstev dle OSI modelu.

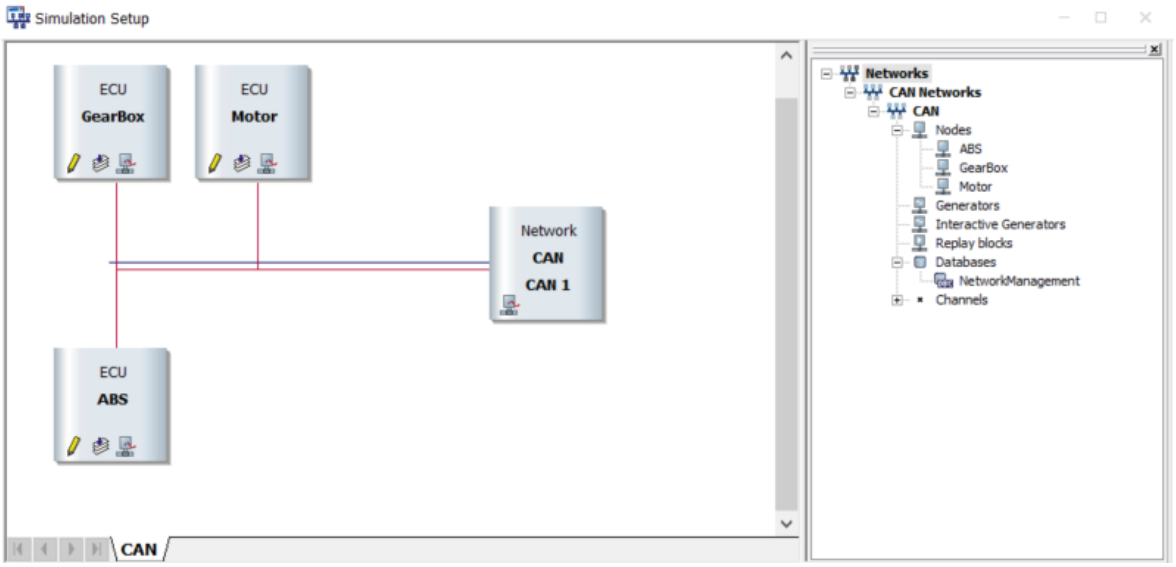


Obr. 23: CAN – rozdělení vrstev dle OSI modelu

Network management využívá algoritmus, který je založen na periodickém posílání zpráv (network management PDUs). Všechny uzly tyto zprávy přijímají a následně předávají dále. Přijetím network management zprávy získá elektronická řídicí jednotka informaci, že odesílající uzel této zprávy chce udržet síť v aktivním (probuzeném) stavu. V opačném případě, tedy chce-li uzel vstoupit do režimu spánku, přestane odesílat NM zprávy. Dokud ostatní uzly vysílají NM zprávy, nedojde k okamžitému vyvolání usnutí všech jednotek, ale vše se opozdí do té doby, než ustane vysílání všech NM zpráv od všech jednotek. Pokud z některých uzlů na sběrnici vyžaduje opět probuzení komunikace na sběrnici, je potřeba začít vysílat NM zprávy na sběrnici od všech uzlu.

7.2.2 Nastavení simulace pro network management

CANoe podporuje řadu funkcí pro network management. Pro praktický příklad byla využita knihovna ASRN33.DLL podporující funkce týkající se NM. Tato knihovna je distribuována společně se standardní instalací CANoe. Na Obr. 24: Simulované uzly pro network management jsou patrné simulace tří uzlu (elektronických řídicích jednotek): GearBox, Motor a ABS. Ke každému uzlu je přiřazena ASRN33.DLL knihovna k využití NM funkcionalit v CAPL prohlížeči. Knihovna lze přiřadit kliknutím pravým tlačítkem na vybraný uzel, kdeje k dispozici položka pro dodatečné DLL. Všechny dodatečné knihovny pro CANoe jsou dostupné ve zdrojovém adresáři instalace CANoe jako např: C:\Program Files (x86)\CANoe11\Exec32.



Obr. 24: Simulované uzly pro network management

7.2.3 Simulace network managementu – popis zdrojového kódu CAPL skriptu

Pro účel simulace byly vytvořeny tři uzly simulující opět elektronické řídicí jednotky v automobilu, které řídí automatickou převodovku, ABS a spalovací motor. Tyto jednotky mají vesměs podobný zdrojový kód, protože jejich reakce na změny stavu klíčku v zapalování je stejná. Následující kapitoly se tedy zabývají výhradně implementací CAPL skriptu pro elektronickou řídicí jednotku spalovacího motoru.

7.2.3.1 Uzel Motor – deklarace proměnných

```
Variables

{
  char gPanelName[50] = "Network Management";
  const cBusSleepState = 0;
  const cRepeatMessageState = 2;
  const cNormalOperationState = 3;
  const cReadySleepState = 1;
}
```

Network management využívá vícero stavů, a proto bylo třeba deklarace čtyř stavů jako konstanty. Hodnoty stavů jsou definované v manuálu pro ASRN33.DLL knihovnu. Proměnná „gPanelName“ je definice znaků, které vyjadřují název panelu. Toto je potřeba pro pozdější ovládání prvků v grafickém panelu. Pokud chceme ovládat nějaké prvky z grafického panelu, je potřeba znát název panelu a poté konkrétní kontrolní prvek.

Význam nadefinovaných stavů:

- ***Repeat Message State*** – U uzlů, které nejsou v pasivním režimu, musí být zajištěno, že přechody z „Bus Sleep“ nebo „Prepare Bus Sleep“ do stavu aktivní sítě budou viditelné pro všechny uzly v síti. Je důležité zajistit, aby jakýkoliv uzel zůstal aktivní po minimální dobu. Dále můžou sloužit pro detekci přítomných uzlu na sběrnici.

- ***Bus Sleep State*** – Přepnutí do spánkového režimu zajistí konkrétní komunikační řadič. Sníží se spotřeba celého systému na minimální definovanou hodnotu a aktivují se mechanismy pro probuzení

- ***Normal Operation State*** – Normální provozní stav je aktivní, dokud jsou network management zprávy posílány na sběrnici.

- ***Ready Sleep State*** – Tento stav čeká s přechodem do „Prepare Bus Sleep mode“, dokud jakýkoliv jiný uzel drží komunikaci při životě skrze network management zprávy.

Prepare Bus Sleep Mode – Všechny zprávy se přenášejí na sběrnici, dokud nedojde k vyprázdnění vyrovnávací pamětí pro zprávy. Poté dochází k uklidnění aktivitě na sběrnici.

7.2.3.2 Uzel Motor – událost „on prestart“

```
on preStart

{
  ILSetAutoStartParam(0);
  Nm_SetAutoStartParam(0);
  Nm_ConfigureILNotifications(0,1);
}
```

Tato událost je volána před začátkem měření a slouží pro inicializaci proměnných, ke čtení dat ze souboru a zobrazení zpráv v okně pro zápis. V tomto stavu nelze posílat zprávy na sběrnici.

Funkce {ILSetAutoStartParam} definuje chování interaktivní vrstvy na začátku měření a je nutné jí používat v události {on preStart}. Hodnota je nastavena do 0, aby interaktivní generátor neposílal na sběrnici žádné aplikační zprávy po zapnutí simulace.

Podobná funkcionalita je řešena funkcí {Nm_SetAutoStartParam}, která definuje chování network management zpráv na začátku simulace. Parametr je nastaven na nulovou hodnotu. Opět je důležité využívat funkci v události {on preStart} jinak nelze použít.

7.2.3.3 Uzel Motor – událost „on sysvar update“

```
on sysvar_update MyVariables::CarIgnition

{
  long status;
     switch (@this) {
           case 0:
                status = Nm_GetState();
                write("Nm State: %i", status);
                Nm_DisableCommunication();
                Nm_NetworkRelease();

                @sysvar::NMControl::CAN::Nodes::Motor::RequestBus = 0;

                enableControl(gPanelName, "NetworkState", 1);
                SetStateDisplayColor(255, 165, 0);

                  sysSetVariableString(sysvar::MyVariables::NetworkState,
                   "Car is in sleep mode - no key in the car !");
                break;

           case 1:

                status = Nm_GetState();
                write("Nm State: %i", status);

                if(Nm_GetState() == cNormalOperationState)
                {
                  Nm_RepeatMessageRequest();
                }


                @sysvar::NMControl::CAN::Nodes::Motor::RequestBus = 1;
                enableControl(gPanelName, "NetworkState", 1);
                SetStateDisplayColor(255, 255, 0);

                  sysSetVariableString(sysvar::MyVariables::NetworkState,
                  "Key is in ignition - Car is waking up");
                break;

           case 2:

              status = Nm_GetState();
              write("Nm State: %i", status);

              if(Nm_GetState() == cRepeatMessageState)
                 {
                 Nm_NetworkRequest();
                 enableControl(gPanelName, "NetworkState", 1);
                 SetStateDisplayColor(124,252,0);

                  sysSetVariableString(sysvar::MyVariables::NetworkState,
                  "Normal operation of car");

                 break;

                 }
      }
```

Procedury {sysVar update} slouží k reakci na globální systémové proměnné viditelné v celé simulaci. Reakce probíhá vždy při změně systémové proměnné a dojde k zavolání této procedury. Proměnná je navázaná na simulovaný klíč pro nastartování auta z Obr. 25: Panel pro řízení network managementu. Uvnitř těla procedury je nadefinován příkaz *switch* pro rozdělení jednotlivých network management stavů s reakcí na klíč.

- *Case 0*: Do proměnné status je získán aktuální status systému network managementu. Získání statusu je skrze funkci {Nm_GetState} z knihovny ASRN33.DLL. Aktuální stav je do informační tabule vypsán skrze funkci {write} – tato funkce funguje na stejném principu jako printf () v programovacím jazyce C. V další části příkazu je skrze funkci {Nm_DisableCommunication()} vypnuta komunikace uzlu Motor. V případě, že nemá být uzel aktivní na sběrnici, je potřeba povolení vstupu do Bus-Sleep mode skrze funkci {Nm_NetworkRelease}. S integrací network managementu do simulace přichází vygenerované proměnné, s kterými se dá pracovat. Proměnná „Request Bus“ pracuje skrze panel - Obr. 25: Panel pro řízení network managementu: Panel pro řízení network managementu s ovládacím modulem „CAN“, který je generován v prostředí návrhu grafických panelu. V tomto případě je zapsána do proměnné hodnota 0, aby bylo zajištěno opravdové vypnutí toho uzlu. Funkce {EnableControl} s třemi parametry umožnují ovládání některých položek z panelu přímo v CAPL skriptu. Je potřeba mít nadefinovaný název panelu a název kontrolního prvku.

- *Case 1*: Podobný proces probíhá v tomto příkazu. Opět se naplní hodnota do proměnné „Status“, která je naplněna skrze network management funkci {Nm_GetState}. Status se opět vypíše do funkce {write}, aby uživatel viděl hodnotu proměnné v případě debuggování. Rozhodovací cyklus zkontroluje, zda je NM status ve stavu „cNormalOperationState“ - v případě že ano, dojde k skrze funkci {Nm_RepeatMessageRequest() která zajistí network management opustit „Normal Operation state“ .

- *Case 2*: V posledním případě příkazu switch dojde k zapnutí klíčku do polohy 2, a tím by mělo být auto kompletně probuzené a jednotky začnou vysílat data na sběrnici. Princip je stejný jako v předešlých případech, dojde k získání aktuálního statusu. V případě, že status vrátí hodnotu „cRepeatMessageState“ uvedeme jednotku do probuzeného stavu pomocí funkce {Nm_NetworkRequest}. Poté dochází k předání informací uživateli skrze zobrazovací funkce.

7.2.3.4 Uzel Motor – funkce „SetStateDisplayColor“

```
void SetStateDisplayColor (int red, int green, int blue)
{
    SetControlForeColor(gPanelName, "NetworkState", MakeRGB(red, green,
    blue));
}
```

Funkce {setStateDisaplyColor} je nadefinována jako vlastní funkce s vlastními vstupními parametry. Vstupní parametry jsou čísla obsahující RGB hodnoty. Tyto parametry se předají do funkce {SetControlForeColor}, která zajišťuje nastavení kontrolního prvku „Network state“ v grafickém panelu „Network management“ na barvu danou vstupními parametry funkce {SetStateDisplayColor}. Funkce je využívána ve výše uvedených příkazech switch.

7.3 Panel pro řízení network managementu

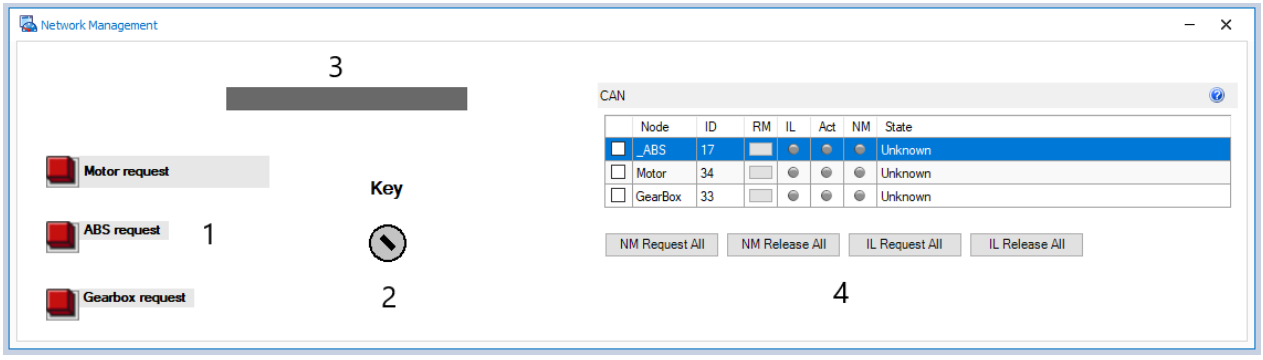
Na Obr. 25: Panel pro řízení network managementu je zobrazený grafický panel, který obsahuje ovládací prvky.

1. Zde jsou zobrazené simulované uzly, které indikují, zda jsou v režimu „Normal operation“ nebo ve „Sleep modu“. Dají se jednotlivě vypínat a zapínat. Pro přestání posílání aplikačních zpráv na sběrnici je potřeba mít všechny uzly ve „Sleep modu“

2. Ovládací prvek „Key“ reprezentuje klíč od zapalování v autě. Klíč má tři ovládací polohy. První poloha simuluje vytažení klíčku z automobilu. Druhá poloha je zaměřena na zastrčení klíčku od automobilu a zapnutí zapalování. Poslední třetí poloha simuluje nastartované auto.

3. Zobrazovací prvek, který zobrazuje aktuální stav network management sítě.

4. Ovládací prvek CAN je již vytvořením prvkem přímo v programu CANoe. Tento prvek umožňuje stejné věci jako předchozí prvky. Umí vypínat a zapínat network management zprávy, a to samé platí pro aplikační zprávy. Přes tlačítko „Nm Release All“ lze všechny jednotky poslat do režimu „Sleep Mode“. Opačným způsobem lze probudit jednotky ze „Sleep Mode“ do „Normal Operation“.



Obr. 25: Panel pro řízení network managementu

8 Závěr

Cílem této diplomové práce bylo seznámení se se základní funkcionalitou simulačního nástroje CANoe od německé společnosti Vector Informatik. Získané znalosti bylo třeba převést do praxe a vytvořit příklady, které demonstrují možnosti využití tohoto nástroje.

Úvod práce byl zaměřen na popis nejpoužívanějších sběrnic v automobilovém průmyslu, který simulační nástroj CANoe podporuje. Jedná se o sběrnice CAN, LIN a FlexRay. Doposud nejpoužívanější sběrnicí je díky rychlosti a ceně sběrnice CAN. V současné době dochází k přechodu na vylepšený formát protokolu CAN, a to na protokol CAN-FD, který disponuje vyšší přenosovou rychlostí a díky tomu nabízí i efektivnější využití protokolu. Pro praktické ukázky byla brána v potaz právě sběrnice CAN.

V další části byl kladen důraz na základnější podporované funkcionality simulačního nástroje. Do funkcionalit byl zahrnut také hardware, který se využívá pro získání a měření skutečných dat přímo z automobilu nebo z testované elektronické řídicí jednotky. V případě, že dochází pouze k simulaci jednotek, není hardware potřeba, protože uzly (elektronické řídicí jednotky) na sběrnici jsou virtuální, jejich chování je simulováno a všechna data na sběrnici jsou také generována funkční simulací.

Praktická část zahrnuje dva příklady. První z příkladů prezentuje využití CANoe jako diagnostického zařízení. Byl vytvořen grafický panel, který umožňuje ovládat chování diagnostiky. Skrze tento panel lze vyčítat i mazat chybovou paměť. Dále bylo potřeba vytvořit simulované zprávy a uzly, aby chování simulovaného prostředí korespondovalo s reálným chováním např. během testování v automobilovém servisu nebo při testování SW v řídicí elektronické jednotce během vývoje. Základní struktura tohoto příkladu umožňuje škálovatelnost a snadné rozšíření o simulaci dalších uzlů na sběrnici a dodefinování odpovídajícího datového provozu (zpráv a signálů).

Druhý z příkladů se zabýval simulací „Network management“. Tato simulace měla za úkol demonstrovat chování řídicích jednotek v autě při vytažení a vložení klíčku do zapalování. Konfigurace databáze zpráv a využití CAPL skriptu byla opět potřebná. Zprávy byly nakonfigurované pro tři uzly (motor, převodovka, ABS). Grafické zobrazení bylo vytvořeno a funkčně provázáno se zpracováním (řízením) pomocí CAPL skriptu. Funkční

demonstrace koresponduje s reálnými podmínkami a chování elektronických řídicích jednotek na datové sběrnici s úmyslným omezení implementované funkcionality na míru dostačující pro demonstrativní účely této práce. Všechny nasimulované jednotky jsou schopné po vytažení klíčku usnout, a následně se opětovně probudit po znovu zasunutí klíčku. Rozšířením pro tento příklad by bylo nasimulování celého auta a využití reálných elektronických řídících jednotek, které by byly připojeny na sběrnici mezi sebou. Pro tento účel by bylo potřeba využití i hardware od společnosti Vector Informatik pro měření reálných dat.

Jak naznačují odstavce výše, všechny cíle této diplomové práce bylo splněny v rozsahu bodů zadání. Výstupy (popis a funkční příklady) této práce mohou v následných praktických aplikacích posloužit jako základní funkční podklady rozšiřitelné o další funkcionality dle potřeb konkrétního projektu a produktu.

Seznam literatury a informačních zdrojů

[1] VECTOR. CANoe – Product Informations. [online] [cit. 2021-05-01]. Dostupné z: https://assets.vector.com/cms/content/products/canoe/canoe/docs/Product%20Informations/CANoe_ProductInformation_EN.pdf

[2] VECTOR. CANoe – CANdb++. [online] [cit. 2021-05-01]. Dostupné z: https://assets.vector.com/cms/content/products/candb/Docs/CANdb_Admin_FactSheet_EN.pdf

[3] VECTOR. CANdb++ - Managing Network. [online] [cit. 2021-05-01]. Dostupné z: https://www.vector.com/int/en/products/products-a-z/software/candb/

[4] VECTOR. License – New Licensing Options from Vector [online] [cit. 2021-05-01]. Dostupné z: https://assets.vector.com/cms/content/events/2019/VU/VectorGB_TS19_Tool_Licensing_at_Vector.pdf

[5] AUTOSAR. – Technical Overview. [online] [cit. 2021-05-01]. Dostupné z: https://www.autosar.org/fileadmin/user_upload/standards/classic/3-0/AUTOSAR_TechnicalOverview.pdf

[6] AUTOSAR. – E2E Protocol Specification. [online] [cit. 2021-05-01]. Dostupné z: https://www.autosar.org/fileadmin/user_upload/standards/foundation/1-2/AUTOSAR_PRS_E2EProtocol.pdf

[7] FLEXRAY. – Sběrnice Flexray. [online] [cit. 2021-05-01]. Dostupné z: https://automatizace.hw.cz/sbernice-komunikace-flexray-nejen-pro-automobily

[8] Network Management. – Detailed explanation of Autosar CAN network management. [online] [cit. 2021-05-01]. Dostupné z: https://www.programmersought.com/article/27737938527/

[9] UDS – Introduction to UDS protocol. [online] [cit. 2021-05-01]. Dostupné z: https://piembsystech.com/uds-protocol/

[10] AUTOSAR – Understanding AUTOSAR ARXML for Communication Networks. [online] [cit. 2021-05-01]. Dostupné z:

https://www.intrepidcs.net.cn/wp-content/uploads/2019/05/202._Understanding_ARXML_EEA_COM_TD_USA_2019.pdf

[11] LIN – Local Interconnect Network. [online] [cit. 2021-05-01]. Dostupné z:

https://automatizace.hw.cz/clanek/2005101501

[12] CAN – Practical tips for CAN - Bus. [online] [cit. 2021-05-01]. Dostupné z:

https://www.kmpdrivetrain.com/paddleshift/practical-tips-can-bus/

[13] VECTOR. – VN1600 Interface. [online] [cit. 2021-05-01]. Dostupné z: https://assets.vector.com/cms/content/products/VN16xx/docs/VN1600_Interface_Family_Manual_EN.pdf

[14] VECTOR. – VN89xx Interface. [online] [cit. 2021-05-01]. Dostupné z: https://assets.vector.com/cms/content/products/VN89xx/docs/VN8900_Manual_EN.pdf

[15] UDS – Introduction UDS. [online] [cit. 2021-05-01]. Dostupné z: https://embedclogic.com/uds-protocol/uds-introduction/

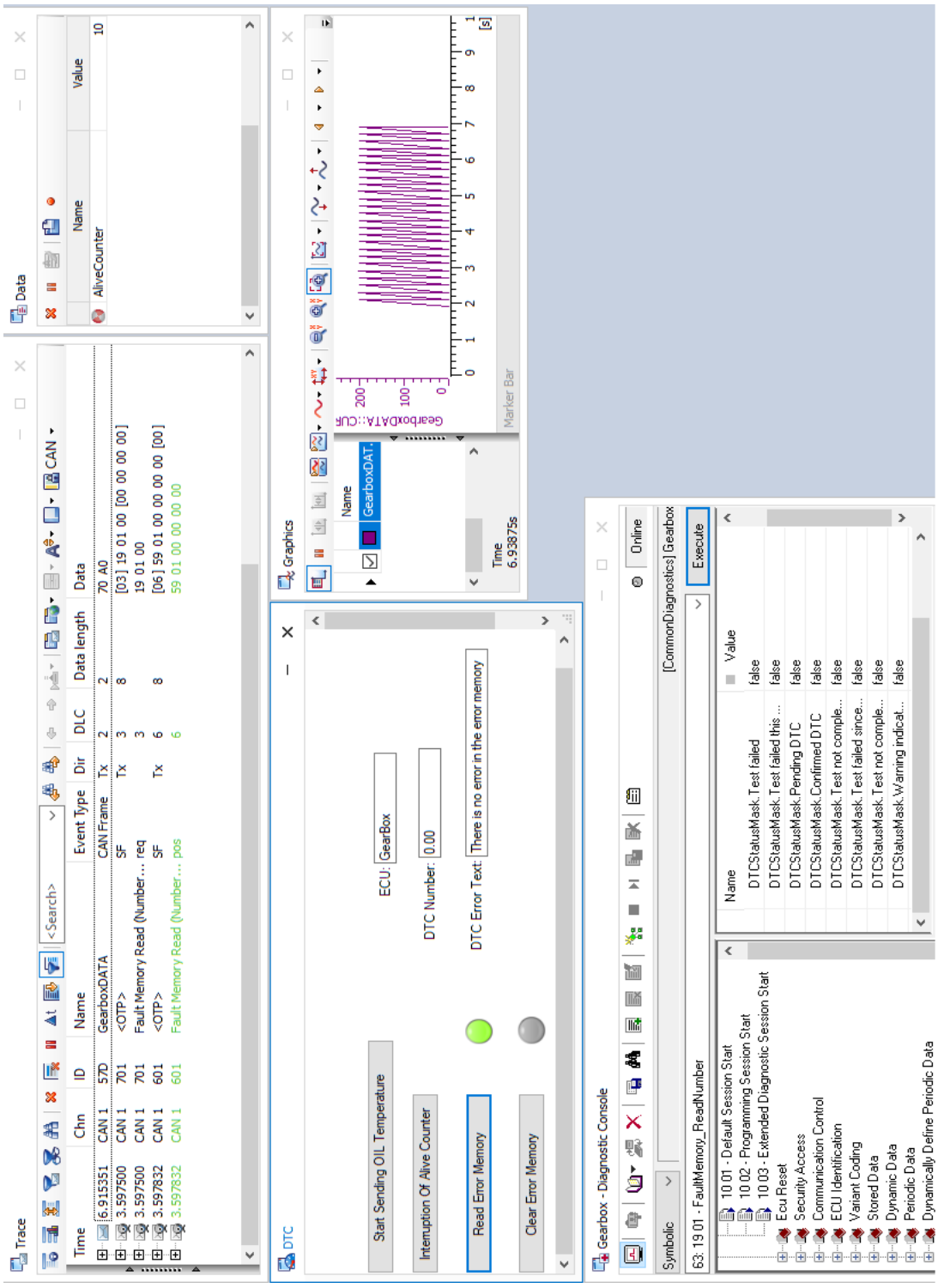
[16] Jendek, Jan *Návrh monitoru automobilových sběrnic*. Plzeň 2018. Bakalářská práce. Západočeská univerzita v Plzni. Fakulta Elektrotechnická

[17] Bursting – Maximizing Wireless Performance. [online] [cit. 2021-05-01]. Dostupné z: https://web.archive.org/web/20110721002833/http:/www.super-g.com/collateral/atheros_superg_whitepaper.pdf

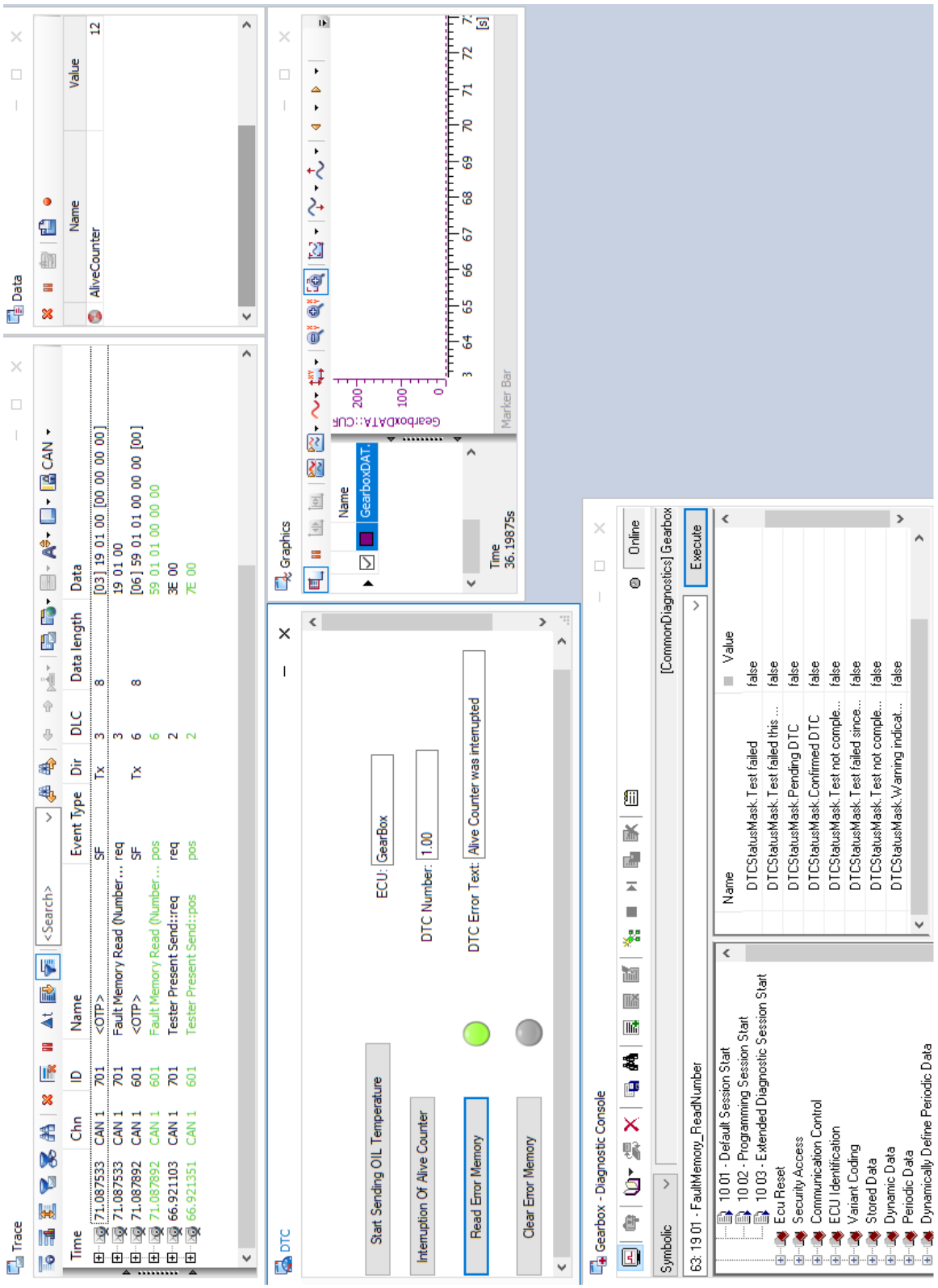
[18] CANoe – Simulation with CANoe. [online] [cit. 2021-05-01]. Dostupné z: https://www.vector.com/int/en/products/products-a-z/software/canoe/simulation/

9 PŘÍLOHY: Ukázka praktických příkladů

9.1 Praktická ukázka vyčítání diagnostické chyby z paměti

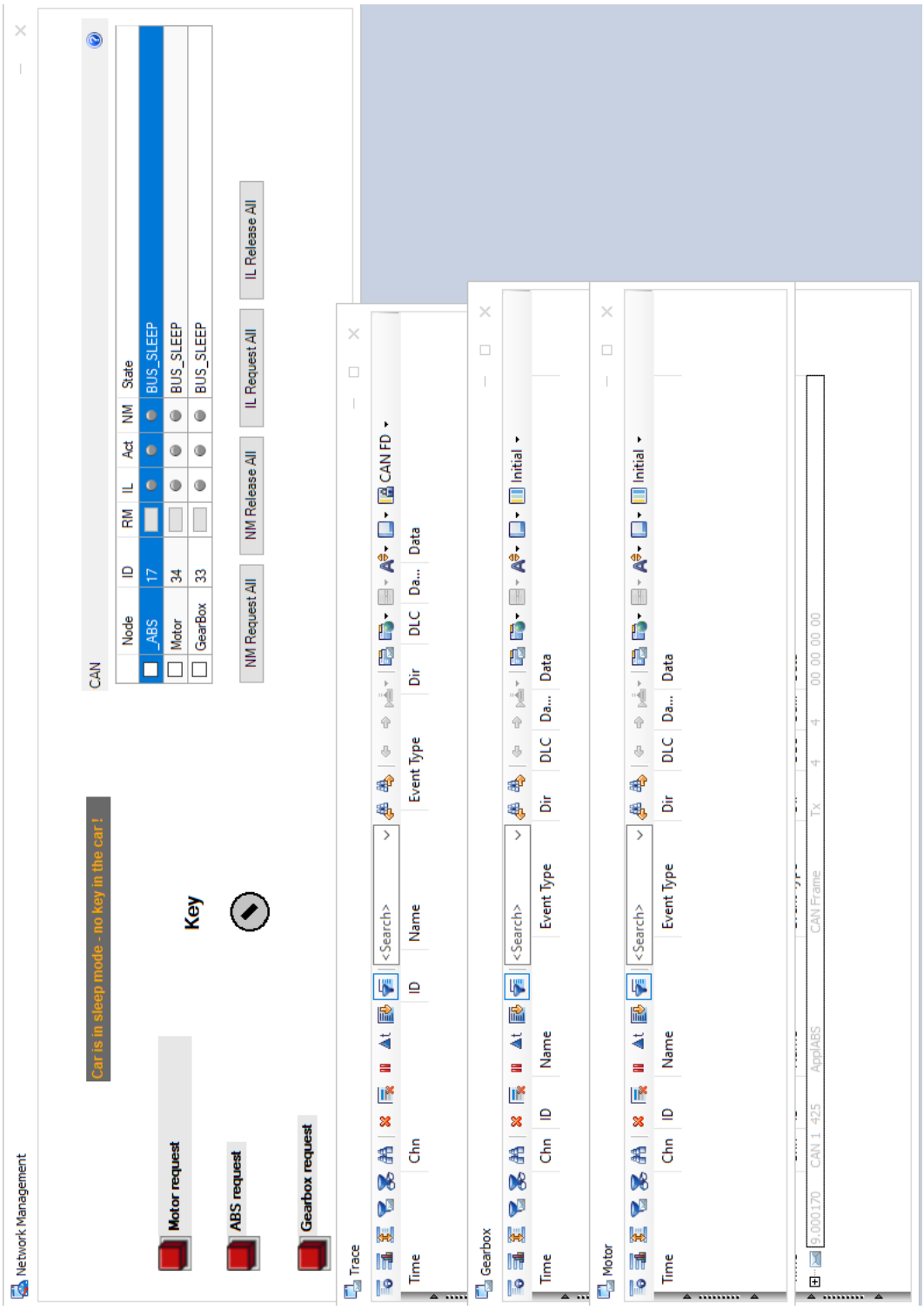


Obr. 26: Diagnostika – chyba neaktivní

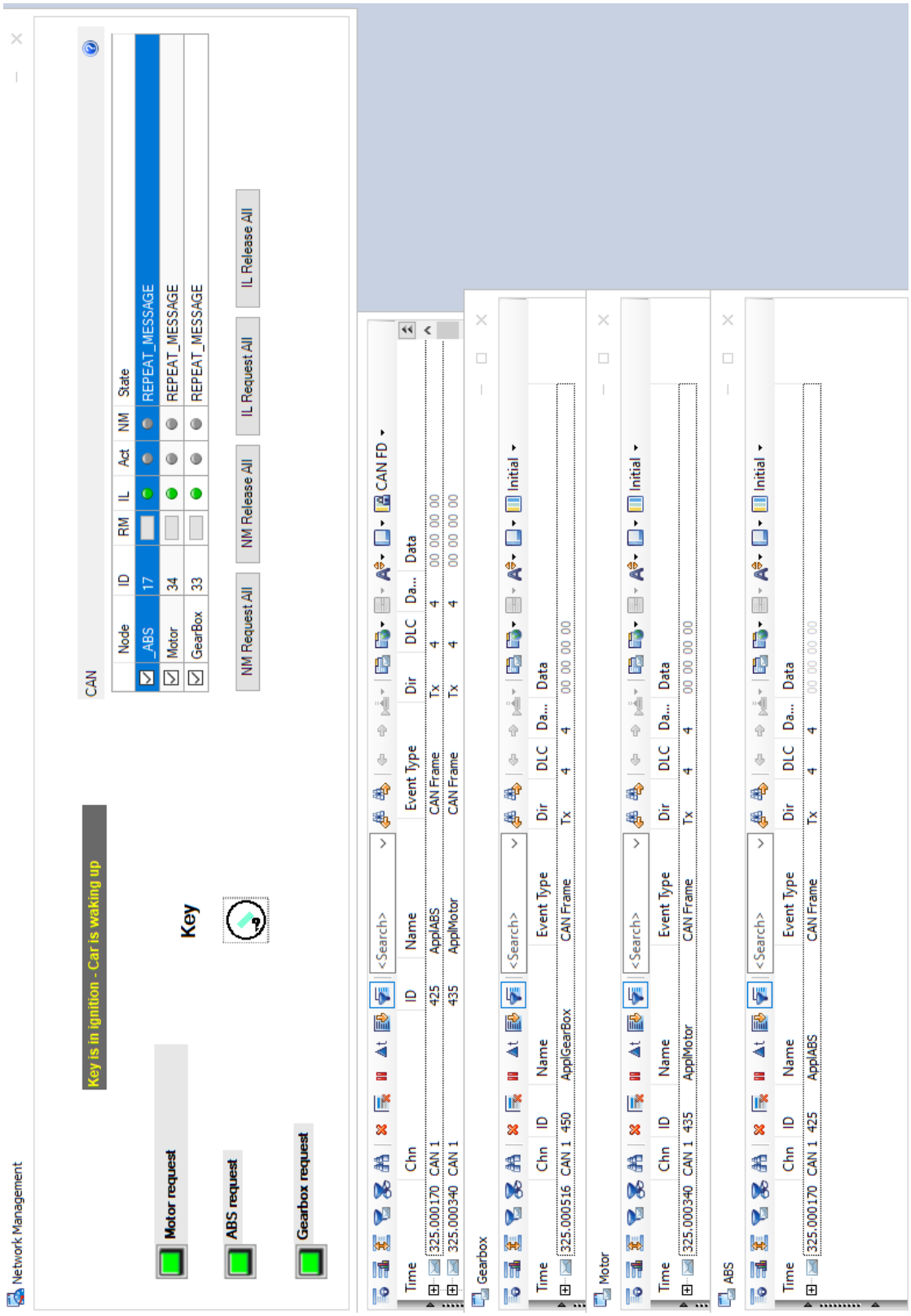


Obr. 27: Diagnostika – chyba aktivní

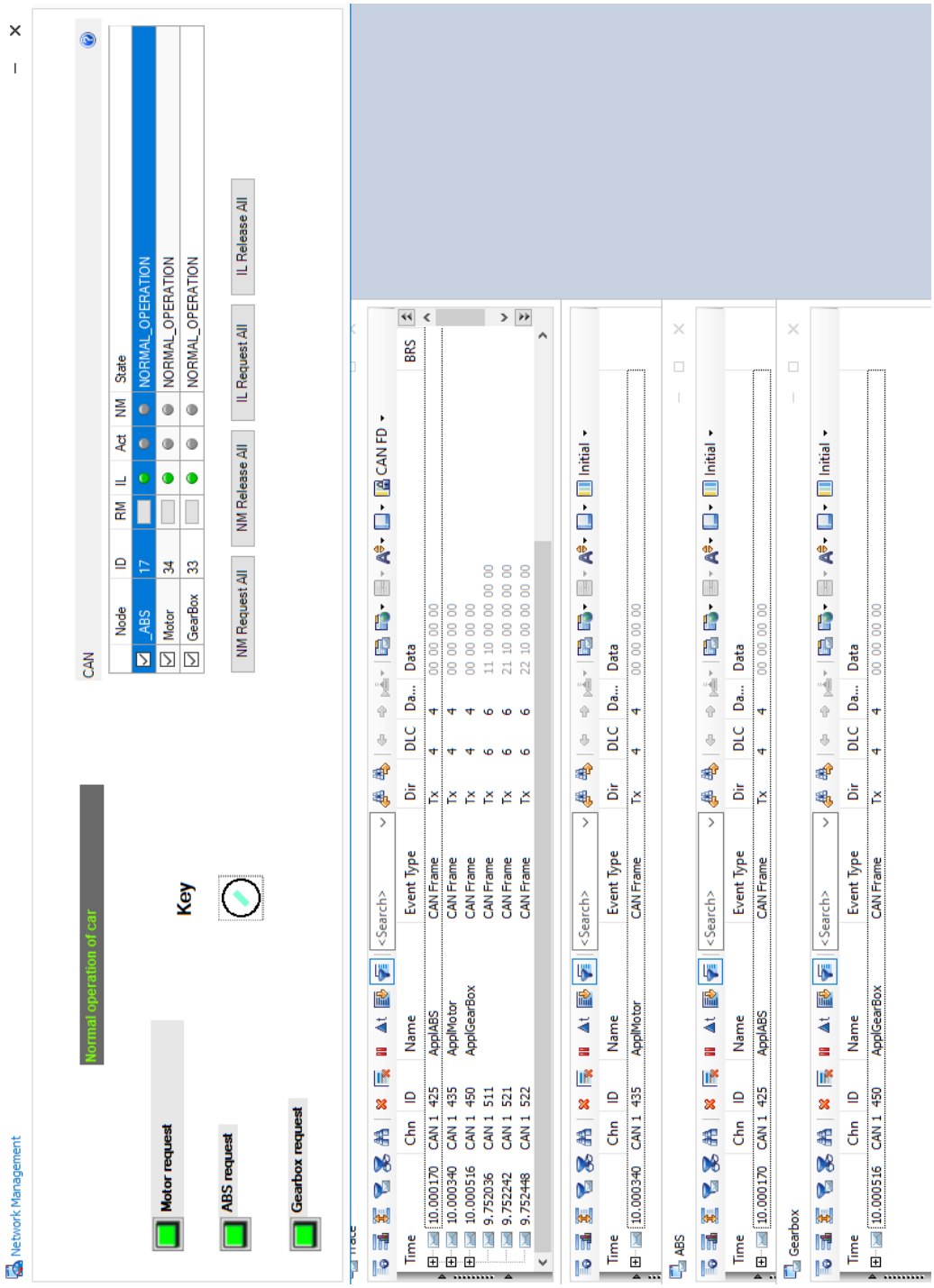
9.2 Simulace Network managementu (správa sítě)



Obr. 28: Network management stav 0



Obr. 29: Network management stav 1



Obr. 30: Network management stav 2