České vysoké učení technické v Praze
Fakulta elektrotechnická

DIPLOMOVÁ PRÁCE

Simulace částí přenosu výkonu u drážních vozidel

Praha, 2012 **Autor: Bc. Tomáš Diringer**

Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

V Praze dne 2.1.2012

podpis

Poděkování

V prvé řadě děkuji vedoucím práce – Ing. Richardu Šustovi Ph.D. a Dr. Ing. Ivo Myslivcovi za cenné rady, připomínky a ochotu. Svým rodičům děkuji za velkou podporu. Rovněž nejbližším přátelům děkuji za podporu během psaní této práce.

Abstrakt

Tato diplomová práce se zabývá modelováním a simulací komponent pohonu drážních vozidel. Ve své úvodní části popisuje důvody, které vedly k požadavku na simulaci zmíněného problému a stanovuje požadavky na model hnacího agregátu vozidla. Dále popisuje kolekci standardů datové komunikace, která je předepsaná jakožto rozhraní mezi modelem a řídicím systémem. V následující části je popsán vytvořený model hnacího agregátu vozidla, jeho výsledky a přínos. V další části práce jsou rozebrány další možnosti, jak daný problém modelovat a simulovat s ohledem na nutné podmínky. Rovněž je popsáno, jak je možné vytvořený model hnacího agregátu vozidla vylepšit.

Abstract

This diploma thesis deals with a simulation of rolling stock driveline. The thesis describes reasons for designing model of the problem in its introductory part. Model behaviour requirements are described in this part. Model of rolling stock driveline including its results and benefits is described in the next part. In following chapter, other ways of rolling stock driveline model designing are analyzed. Possible impovement of created model is analyzes too.

České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra řídicí techniky

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: **Bc. Tomáš Diringer**

Studijní program: Kybernetika a robotika
Obor: Systémy a řízení

Název tématu: **Simulace částí přenosu výkonu u drážních vozidel**

Pokyny pro vypracování:

1. Na zadaném existujícím hardwaru realizujte jednoduchý dynamický model hnacího agregátu zadaného železničního hnacího vozidla, tj. model jeho dieselagregátu a hydromechanické převodovky včetně retardéru a též model dynamického chování tohoto vozidla, resp. vlaku. Zařízení, jež je modelem řízené soustavy, bude s řídicím systémem komunikovat dle normy J1939 po rozhranní CAN. Zařízení musí umožňovat základní zobrazení komunikovaných parametrů a simulovaných veličin a nastavení základních parametrů vozidla a hnacího agregátu.
2. Prozkoumejte jiné možnosti návrhu nebo simulace modelu výše uvedeného dynamického systému.
3. Prozkoumejte možnosti vizualizace průběhu veličin modelu. Pokud bude k dispozici reálné vozidlo, proveďte měření a ověřte výsledky v praxi.

Seznam odborné literatury:

[1] Ivo Myslivec, Petr Vysoký - Systémy řízení dopravních prostředků - Vydavatelství ČVUT – Praha – 2004, SAE International - SAE J1939 Standard
[2] Jiří Danzer - Elektrická trakce. 1., Přehled problematiky - Plzeň: Západočeská univerzita, 2009
[3] Jiří Danzer - Elektrická trakce. 2., Stupňové řízení sériového motoru - Plzeň: Západočeská univerzita, 2009
[4] Jiří Danzer - Elektrická trakce. 3., Plynulá regulace cize buzeného motoru - Plzeň: Západočeská univerzita, 2009

Vedoucí: Ing. Richard Šusta, Ph.D.

Platnost zadání: do konce letního semestru 2011/2012

prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

prof. Ing. Boris Šimák, CSc.
děkan

V Praze dne 25. 1. 2011

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod do problému a specifikace cíle

Cílem této diplomové práce je podle specifikace vedoucího práce vytvořit model hnacího agregátu zadaného železničního vozidla na již existujícím, zadaném HW. Dalším úkolem je obecnější zmapování možností modelování hnacích agregátů železničních vozidel vzhledem k různým atributům, kterými mohou být: harwarová složitost, znovupoužitelnost, doba potřebná k vývoji nebo modifikaci, vizualizační možnosti modelu a přesnost modelu (ve smyslu odchylek modelu a systému).

Definice 1.1 (model HAV): Model HAV (hnacího agregátu vozidla) je fyzické zařízení simulující procesy v hnacím agregátu železničního vozidla do dostatečné míry (kapitola 1.1.2) a komunikující (výměna akčních a výstupních veličin) požadovaným způsobem s řídicím způsobem. (kapitola 1.1.3). ▶

1.1 Určení modelu a požadavky na něj

Myšlenka vytvořit model HAV přišla při řešení rekonstrukce (kapitola 1.2.2) motorového vozu řady 842 zhruba v polovině roku 2010. Naše firma – AŽD Praha, s.r.o. ve spolupráci 842
s MSV elektronika, s.r.o. v rámci této rekonstrukce dodává, mimo jiné, řídicí systém vozu stručně popsaný v kapitole 1.2.2. Při vývoji řídicího systému nebylo ze závažných důvodů možné ladit řídicí systém na opravdovém soustrojí a bylo rozhodnuto pro vývoj modelu HAV. Těmito důvody byly zejména absence zkušebního stavu pro motor v naší kanceláři o rozloze přibližně 30 m^2 a skutečnost, že výrobce motoru či převodovky zpravidla nemůže postrádat ve fázích vývoje ani jediný kus.

Jelikož se modelovaný problém nejvíce podobá typu *systém diskrétních událostí*, tak

se laděním nerozumí *tunning* parametrů regulátorů, ale spíše se jedná o ladění reakcí na poruchy (zde ve smyslu *failure – závada*), zotavení z výpadků komunikace a na diskrétní události obecně.

1.1.1 Výčet simulovaných komponent hnacího agregátu vozidla (HAV)

Komponenty HAV, jež se budou modelovat a simulovat jsou:

1. Dieselagregát,
2. hydromechanická převodovka,
3. retardér,
4. dynamika celého vozidla,
5. snímač otáček dvojkolí – tachogenerátor.

Komponenty 1–3 mohou být ve 4 stavech z hlediska poruchy (failure). Mohou být v bezporuchovém stavu, nebo ve 3 poruchových stavech dle závažnosti. Požaduje se, aby bylo možné v modelu HAV tyto poruchové stavy libovolně měnit, například tlačítkem. Dále je u těchto komponent třeba uvažovat různé teploty a tlaky. Není žádoucí, aby se tyto veličiny modelovaly dynamicky na základě jiných stavů komponenty, např. otáček motoru. Pro účely ladění ŘS pomocí modelu HAV je vhodnější, aby se tyto veličiny také zadávaly uživatelem.

1.1.2 Požadavky na model HAV z různých úhlů pohledu

Poznámka: V této kapitole jsem se snažil kvantifikovat míru věrnosti, či zjednodušení modelu z různých úhlů pohledu. Záměrně jsem použil takové číselné vyjádření, které se váže k modelům v jiném slova smyslu jako jejich měřítko. □

Model komunikace

1:1 Model HAV musí být z pohledu komunikace naprosto totožný (1:1) s reálnými řídicími jednotkami vyjmenovaných komponent agregátu. Požadované rozhraní mezi řídicím systémem (ŘS) a modelem HAV (případně reálným agregátem) popisuje kapitola 1.1.3.

Model systému diskrétních údálostí

Části modelující systém diskrétních událostí mohou být zjednodušené, míru zjednodušení určil vedoucí práce tak, aby byla dostatečná. Kvalitativní odhad zjednodušení modelu HAV z toho úhlu pohledu odhaduji na 1:4. 1:4

Model dynamického systému

Požadavky na dynamické chování modelu HAV (dynamika otáček motoru, dynamika celého vozidla) jsou velmi benevolentní. Pokles, resp. nárůst otáček motoru nemusí respektovat fyzikální podstatu, dynamika celého vozidla nemusí respektovat trakční charakteristiku daného vozidla, apod. Plně dostačující je kupříkladu nárůst a pokles otáček po rampě, místo ideální, či tabelizované trakční charakteristiky dostačuje konstantní tažná síla. Zjednodušení modelu HAV z toho úhlu pohledu odhaduji na 1:87. 1:87

Model elektrického stroje – tachogenerátoru

Během vývoje modelu rozhodl vedoucí práce o další potřebné komponentě a s ní spojeným rozhraním mezi řídicím systémem a modelem HAV. Mimo datového rozhraní (SAE J1939) vznikl požadavek na simulaci tachogenerátoru, který na vozidle slouží jako senzor otáček. Reálný tachogenerátor použitý na vozidle generuje 60 základních period sinusového signálu na jednu otáčku snímané nápravy. Modelovaný tachogenerátor může tento signál simulovat jako tvarově upravený, ale okamžiky průchodů nulovým napětím musí být stejné, jaké by byly u reálného tachogenerátoru. Zjednodušení odhaduji jako 1:2. 1:2

1.1.3 Požadavek na rozhraní model HAV—ŘS

Základním rozhraním pro výměnu akčních a výstupních veličin je datová komunikace modelu HAV s ŘS dle standardu SAE J1939, jíž je věnovaná kapitola 2. Tímto rozhraním SAE J1939
se přenášejí otáčky, kroutící momenty, teploty, tlaky, poruchy (failure), ...

V průběhu prací na modelu HAV rozhodl vedoucí práce o modelování a simulaci senzoru otáček, jehož výstup tvoří druhý typ rozhraní – napěťové rozhraní, kde měronosnou veličinou je frekvence signálu. Na tomto rozhraní se přenáší pouze jedna veličina modelu HAV – rychlost vozidla. rychlost

1.2 Popis motorového vozu řady 842 a jeho rekonstrukce

Čerpáno z [6]: Motorový vůz řady 842 vyráběla Moravskoslezská vagónka Studénka v letech 1988–1994. Provozní určení těchto vozů je vozba osobních a spěšných vlaků i lehké rychlíkové výkony.

Mechanická část

Citováno z [6]:

> Skříň vozu je lehké ocelové samonosné svařované konstrukce. Prototypové vozy mají čela zhotovena ze sklolaminátu. Otočnými čepy zabudovanými pevně ve dnu vozové skříně je skříň spojena se dvěma dvounápravovými podvozky, v nichž jsou dvojkolí vedena svislými vodicími čepy. Skříň je uložena na vzduchových pružinách sekundárního vypružení. Vnitřní dvojkolí každého podvozku jsou hnací a vnější běžná. Pod podlahou vozu je zavěšena trakční výzbroj vozu, skládající se ze dvou spalovacích motorů LIAZ a hydromechanické zahraniční převodovky Allison HTB 741 R s automatickým řazením. Spalovací motory jsou naftové rychloběžné přeplňované šestiválce s přímým vstřikem paliva, vodním chlazením a ventilovým rozvodem OHV. Tyto agregáty pohánějí trakční převodovku, z níž je točivý moment na nápravy přenášen kloubovými hřídeli. Vytápění zajišťuje naftovzdušný agregát V 35.00. Motorový vůz disponuje ruční brzdou, samočinnou tlakovou brzdou, přímočinnou brzdou a hydrodynamickou brzdou (retardérem). Samočinnou tlakovou brzdu ovládá brzdič DAKO BS-2, přímočinná brzda je řízena brzdiči DAKO BP. Všechna dvojkolí jsou brzděna třecí kotoučovou brzdou s kotouči na nápravách a přídavnou jednostrannou špalíkovou brzdou. Zásoba písku je 170 kg.

Další, případně detailnější informace je možné najít v [6] a [1].

1.2.1 Spolehlivost vozu

Z [1] vyplývá, že kilometrické proběhy vozů 842 před rekonstrukcí jsou velmi nízké, často jen 20 000 km, v extrémním případě 217 a 928 km. Poruchy vozu byly různorodé, nejčastěji bylo vozdidlo v poruše kvůli vodnímu hospodářství (17,76 %), spalovacímu motoru (16,98 %) a pojezdu (14,21 %). Konečně, byly hlášeny i „poruchy“ interiéru (9,10 %)

(a) Vůz 842–005

(b) Vůz 842–012 (vůz po rekonstrukci)

Obrázek 1.1: motorový vůz řady 842. Převzato z [6]. Autor druhé fotografie je uživatel „Sep“ z diskuzního fóra K–REPORT

a i tak nedpovídá interiér požadavkům jednadvacátého století. Nízká spolehlivost vozu a interiér neodpovídající současnému století je tedy část důvodů vedoucích k rozhodnutí o rekonstrukci celé této vozové řady.

Posledním důvodem rekonstrukce je úprava, aby tento motorový vůz mohl *plnohodnotně* jezdit ve spojením s řídicím vozem. Při tomto spojení je hnací agregát motorového vozu povelován nikoliv ze stanoviště strojvedoucího vlastního motorového vozu, ale ze stanoviště strojvedoucího řídicího vozu a povely jsou vysílany po vlakové lince na určitém standardu[1]. Provozní veličiny motorového vozu jsou dle tohoto standardu přenášeny na řídicí vůz a tam zobrazovány.

1.2.2 Rekonstrukce motorového vozu řady 842

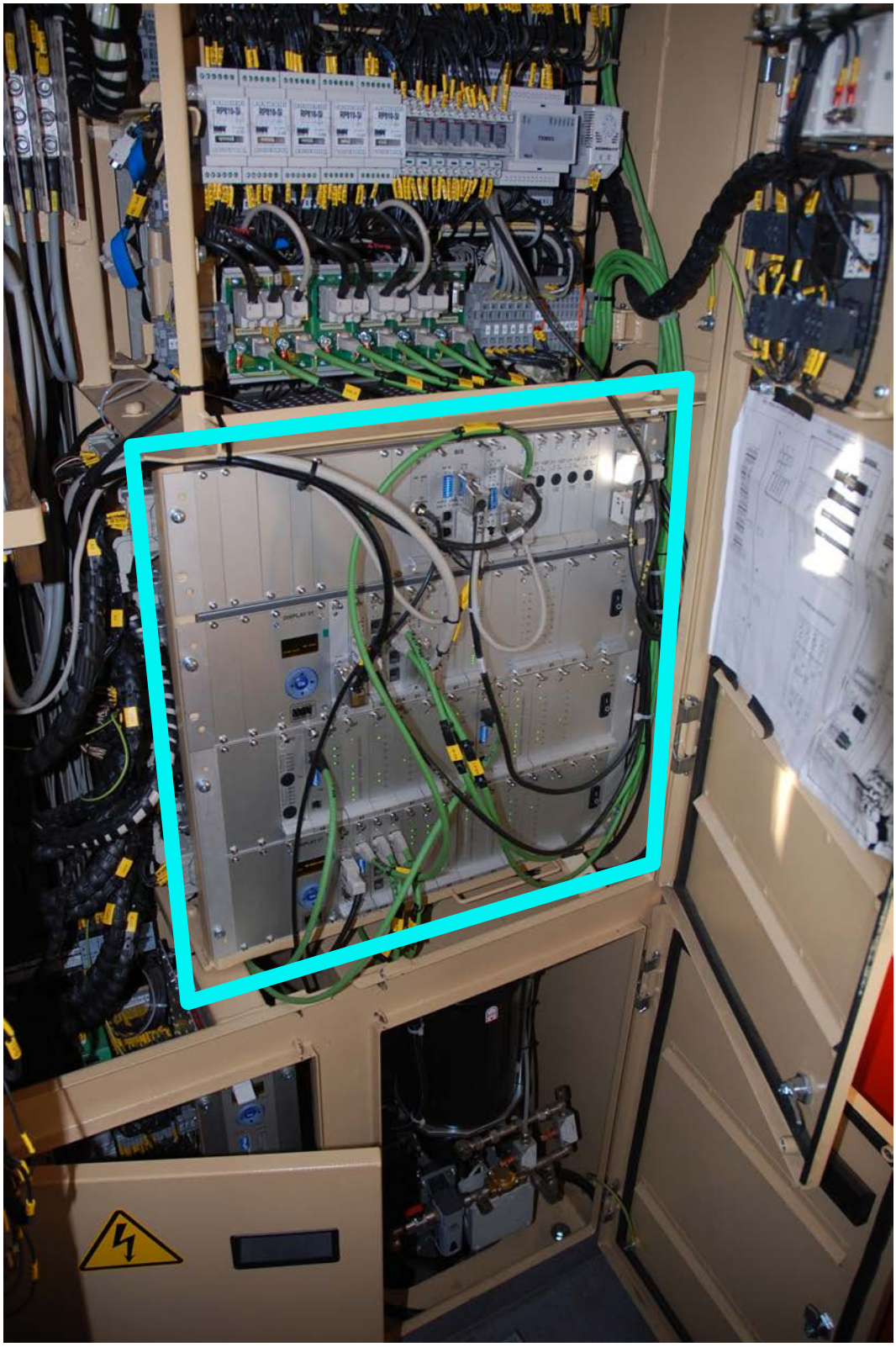
Rekonstrukci vozů řady 842 si vyžádala nízká provozní spolehlivost, popsaná v kapitole 1.2.1. Z toho titulu bylo nutné dosadit nové dieselagregáty, nové trakční i nápravové převodovky, přeprojektovat vodní a olejové hospodářství a provést generální opravy podvozků. Interiér vozu se konečně přiblížil jednadvacátému století a cestujícím je nyní k dispozici vakuové WC, informační systém a dobrá regulace teploty v oddílech. V následující kapitole je popsána změna řídicího systému vozidla a důvodu k ní vedoucí.

[1]NVL (národní vlaková linka)

Cílový stav rekonstrukce z hlediska ŘS

Nový řídicí systém (AŽD/MSV elektronika) oproti stávajícímu systému umožní:

- řízení motorového vozu z řídicího vozu na standardu NVL,
- lepší možnosti řízení motorového vozu co do kvality i stupně automatizace, vzestupně řazeno:
  1. řízení v nouzovém režimu,
  2. řízení v ručním režimu (volba poměrného tahu integračním způsobem pomocí hlavní jízdní páky (HJP) v rozsahu −100–100 %)
  3. automatická regulace rychlosti (ARR), zadávání klavesnicí,
  4. automatické vedení vlaku (AVV), ovládání klavesnicí a HJP
- šetrnější spolupráce obou hnacích agregátů vzhledem k vozové baterii,
- šetrnější spolupráce obou hnacích agregátů vzhledem ke spotřebě nafty,
- lepší vizualizace provozních i servisních údajů a diagnostiky.



Obrázek 1.2: elektrický rozvaděč vozu 842, modře označen ŘS vozidla, autor – Ing. S. Marek

Obrázek 1.3: pohled na displeje na stanovišti strojvedoucího, autor – Ing. S. Marek

Obrázek 1.4: displej motorového vozu se zobrazeným servisním obrázkem, autor – Ing. S. Marek

Kapitola 2

Specifikace SAE J1939

Pojmem SAE J1939 se rozumí kolekce standardů datové komunikace jednotlivých komponent na vozidle. Tyto standardy se prosadily a stále více prosazují zejména u drážních vozidel, nákladních vozidel a autobusů.

Poznámka: Problematika datové komunikace mezi komponentami hnacího soustrojí drážního vozidla je stěžejní částí této práce. □

2.1 Motivace

Ve výše zmíněném výčtu typů vozidel jsem schválně nezmínil osobní automobily. U osobních automobilů nedošlo ke standardizaci datové komunikace, protože u nich dostačuje standardizace na úrovni koncernu a není třeba širšího sjednocení tak, jak se u SAE J1939 nabízí. Tím je nastíněn jeden z rozdílů mezi těmito typy vozidel – u drážního vozidla je zpravidla hnací soustrojí tvořeno komponentami různých výrobců. V kokrétním případě rekonstrukce motorového vozu řady 842 se jedná o dieselagregáty TEDOM, převodovku s retardérem výrobce ZF a řídicí systém (MSV elektronika/AŽD). Je téměř nemyslitelné, aby např. převodovka ZF využívala jeden komunikační standard s motorem TEDOM a nějaký jiný, diametrálně odlišný standard s motorem MAN, MTU, CAT, ... Tímto jsem předložil významný argument pro širší standardizaci u drážních vozidel. Dalším argumentem je obrovská úspora kabeláže, protože, jak bude dále vysvětleno, SAE J1939 využívá sériovou komunikaci a počet veličin, nutných přenášet je několik desítek. Je možné namítnout, že si výrobci jednotlivých komponent mohou dohotnout jinou sériovou komunikaci. To určitě mohou, ale podobně jako na obrázku 2.1 by pak vymýšleli již vymyšlené,

protože stěžejní část této dohody – interpretace hodnoty fyzikální veličiny do datových bytů – je těžištěm normy SAE J1939.

Obrázek 2.1: ilustrace výrazu *reinventing the wheel*, rozhodně to však není případ normy SAE J1939, naopak. Převzato a upraveno z [5]

2.2 Stručný výtah ze specifikace

Celá rodina standardů SAE J1939 je rozepsána na webové stránce[4]. Názvy dílčích dokumentů jsou dobře vypovídající, případně lze ke každému dokumentu dohledat jeho abstrakt (označen jako *Scope*). Dále stručně popíšu dílčí standardy, důležité pro drážní vozidla, resp. tuto práci.

J1939–11 Fyzická vrstva je definována specifikací J1939–11. Ve zkratce se dá říci, že kopíruje specifikaci CAN 2.0B. Jako médium používá stíněný kroucený pár vodičů (STP). Komunikační rychlost je předepsána 250 kb s^{-1}. Druhou variantou fyzické vrstvy, je redukovaná varianta prvně zmíněné, narozdíl od ní využívá nestíněný kroucený pár (UTP). Definuje ji dokument J1939-15. Linková vrstva (J1939–21) také kopíruje specifikaci CAN 2.0B – prodloužená (29 bit) ID. Síťová vrstva (J1939–31) nenachází v našem kontextu uplatnění. Není třeba řešit problémy se směrováním zpráv nebo spojování odlišných technologií, neboť příslušné řídicí jednotky a řídicí systém jsou připojeny na jednu CAN sběrnici – každý ze dvou hnacích agregátů používá jednu CAN sběrnici.

J1939-15
J1939–21
J1939–31

Standardy, popisující aplikační vrstvu, se už samozřejmě neopírají o specifikaci CANu. Ta, zjednodušeně řečeno, „končí“ tak, že rámec obsahuje datové pole délky až 8 bytů. J1939–71 Standard J1939–71 je asi nejstěžejnějším dokumentem celé kolekce. Obsahuje předpis pro interpretaci různých veličin, parametrů, stavů a poruch do dvojkové soustavy. V případě otáček motoru se např. dočteme, že se interpretují do 2 bytů, rozlišení je 1/8 otáčky za

minutu na jeden bit. Offset je nulový. V jiné části tohoto dokumentu se dočteme, které byty datového pole rámce tato veličina obsazuje.

Zbývají dva důležité standardy aplikační vrstvy – diagnostika (J1939–73) a konfigurační zprávy (J1939–74). První zmíněný popisuje diagnostické zprávy, jež poskytuje daná řídicí jednotka. Ty mohou být krátké, tvořené jedním rámcem. Pak příslušná část standardu pouze popisuje interpretaci poruchy (failure) do dvojkové soustavy. Tento diagnostický rámec posílá řídicí jednotka zpravidla periodicky. V bezporuchovém režimu posílá kód „není žádná porucha“. Pokud řídicí jednotka detekuje na svém zařízení poruchu, vysílá její závažnost v tomto rámci (zelená, žlutá, nebo červená porucha). Při detekci poruchového stavu přijde zpravidla na řadu zaslání prodloužené, poruchové zprávy. Plnohodnotná informace o nastalé poruše se musí vysílat prodlouženou zprávou, neboť 8 datových bytů CAN rámce by na danou informaci často nestačilo. Princip vysílání prodloužených zpráv je vysvětlen dále, v kapitole 2.2.3. J1939–73

Konfigurační zprávy definuje dokument J1939–74. Konfigurační zprávy také často potřebují více než 8 bytů, které poskytuje CAN rámec, vysílají se proto jako prodloužené zprávy (kapitola 2.2.3). Konfiguračními zprávami může např. řídicí jednotka převodovky sdělit řídicímu systému svoje řadící poměry, motor může sdělit svojí momentovou charakteristiku apod. Příklad konfigurační zprávy retardéru je uveden v kapitole 2.2.4. J1939–74

2.2.1 Datový rámec

Z uživatelského hlediska je datový rámec tvořen jeho identifikací – ID o délce 29 bitů a datovým polem 8 bytů. Prostřední[1] 2 byty tvoří ve smyslu normy SAE J1939 číslo zvané PGN (**p**arametr **n**umber **g**roup. Datové pole u CANu obecně má délku 0–8 bytů, SAE J1939 používá pevnou délku 8 bytů. V těchto bytech jsou kódovány fyzikální veličiny, prouchy a různé parametry. Každá takováto veličina, porucha, parametr je označena SPN číslem – **s**uspect **p**arameter **n**umber. Hlavním přínosem celé normy je právě ta příloha[2], kde je předepsáno, jaké veličiny, parametry a poruchy se nachází v daném rámci označeném PGN číslem (tedy částí ID datového rámce) a daným parametrům, veličinám a poruchám předepisuje interpretaci do datového pole (dvojkové soustavy). ID PGN SPN

[1] Při zarovnání doprava, tedy doplnění 3 bitů do 32 zleva
[2] J1939-71

2.2.2 Příklad datového rámce převodovky a orientace v SAE J1939

ETC2

PGN

SPN

Mějme rámec vztahující se k převodovce, označovaný jako ETC2. Má ID $18F00503_H$, prostřední 4 byty jsou jeho PGN, tedy $F0005_H = 61443_D$. V příloze normy, která se týká PGN jsou seřazeny rámce vzestupně podle PGN, nalezneme tedy rámec s PGN 61443_D. Zjistíme například, že v tomto rámci jsou informace o požadovaném rychlostním stupni, aktuálním převodovém poměru a zařazeném rychlostním stupni. Zjistíme, kde jsou data v rámci umístěna. Dále ke každé veličině nalezneme její SPN číslo. V příloze, která obsahuje SPN řazená vzestupně, nalezneme námi požadované. Například v bytu 0 je obsažen požadovaný rychlostní stupeň, tento parametr má číslo SPN 524 a pro tento parametr vyčteme, že se toto číslo interpretuje do 1 B tak, že se přičte 125_D. Druhý rychlostní stupeň by tedy byl do dat interpretován jako 127_D. Dále se dočteme, že hodnota FA_H–FF_H znamená nekorektní hodnotu. Veličiny, které se rozhodneme nevysílat, nastavujeme na nekorektní hodnotu, obvykle FF_H.

frame-rate

Další informaci, kterou zjistíme, je perioda vysílání tohoto rámce, dále označovaná jako frame rate. Rámec ETC2 má frame rate 100 ms. Povelové rámce řídicího systému mají frame rate 200 ms, naopak pro rámce s teplotami nebo moto-hodinami stačí 5 s.

2.2.3 Prodloužené zprávy

BAM
DAT

Délka datového pole (8 B) v jednom CAN rámci někdy nestačí, proto je prodloužené zprávy potřeba vysílat jinak. Pro vysílání prodloužených zpráv se používají rámce speciálních typů – BAM a DAT. Vysílání začíná rámcem BAM, ve kterém odchází kód, že jde o multizprávu 20_H v bytu 0, na bytu 1–2 odchází celkový počet vysílaných datových bytů, v bytu 3 se vysílá počet vysílaných datových paketů (DAT rámců) a na bytech 5–7 odchzí ID zprávy. Po odvysílání takto naplněného BAM rámce odchází příslušný počet DAT rámců. Z 8 datových bytů se pro data používá 7 bytů, byte 0 se používá na sekvenční značku, která je u prvního DAT rámce rovna jedné a každý další DAT rámec má předchozí sekvenční značku zvýšenou o jedna.

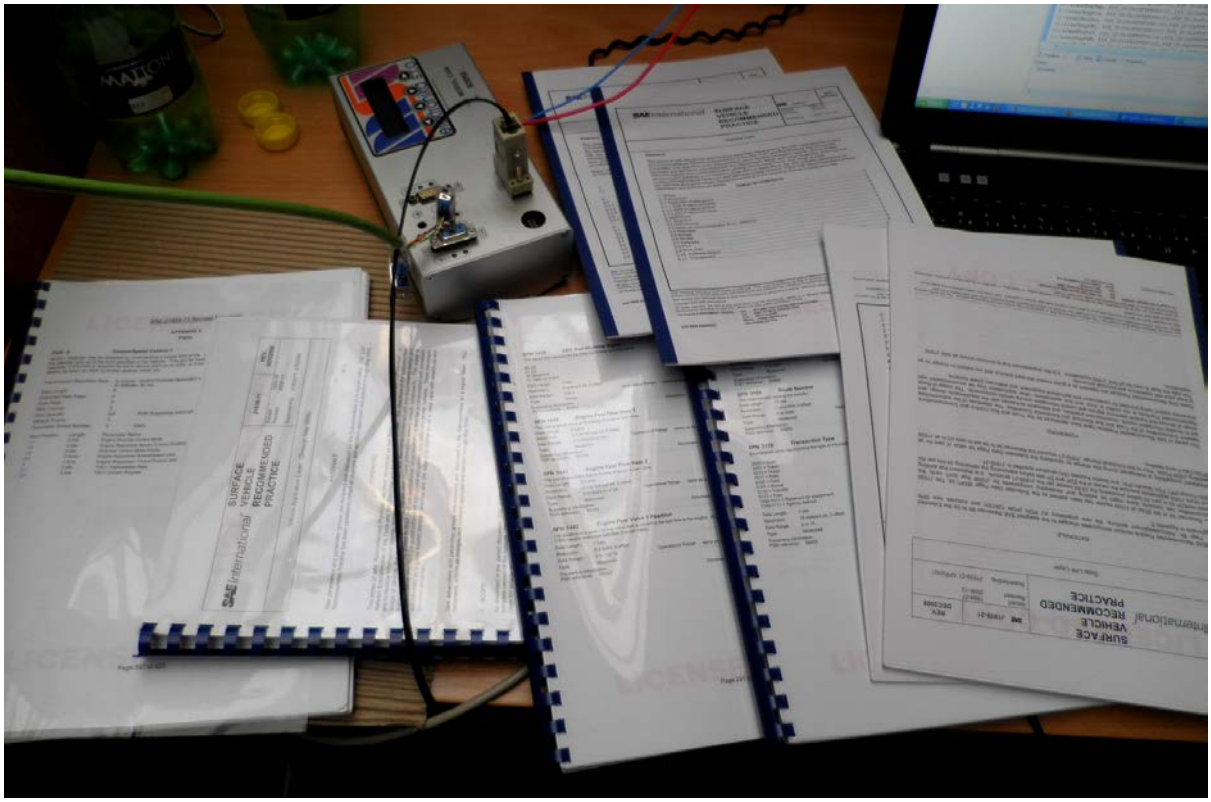
2.2.4 Příklad vysílání prodloužené zprávy – konfigurace retardéru

Řídicí systém požaduje, aby retardér vysílal svoji konfiguraci, pokud ji nevysílá, pak řídicí systém nepoužívá brzdění retardérem. Vysílání začíná naplněním datových bytů BAM rámce. Na bytu 0 se nastaví 20_{H} – multizpráva. Na bytu 1 se nastaví spodní byte počtu datových bytů ($18_{\mathrm{D}} = 12_{\mathrm{H}}$), na bytu 2 se nastaví horní byte – zde 0. Na bytu 3 se nachází počet datových paketů (DAT rámců)– zde 3. Byte 4 je nevyužit. Na bytech 5–7 se nachází ID multizprávy, nejvýznamnější byte ID je na bytu 7. ID konfigurační zprávy retardéru je $\mathrm{FEE1}_{\mathrm{H}}$. Takto připravený byte se odešle. Následují DAT rámce. Na bytu 0 DAT rámce se nachází sekvenční značka, první dat rámec má sekvenční značku 1, další dat rámce mají inkrementovanou minulou sekvenční značku. V každém DAT rámci zbývá 7 datových bytů. V prvním DAT rámci se na prvním datovém bytu (byte 0) nachází typ retardéru, na dalším je typ řízení retardéru. Tyto dvě informace nepotřebuje ŘS znát, mohou mít libovolnout hodnotu. Znát potřebuje referenční moment, vysílá se hodnota 2000_{D}. Spodní byte se odesílá na třetím DAT rámci v bytu 3, horní byte se vysílá v bytu 4 téhož rámce. Odvysílání třetího DAT rámce je konečnou fází, cyklus se opakuje každých 5 s (mezi BAM rámci).

2.3 Vlastní zkušenosti s implementací

Dle mého názoru norma velmi dobře plní svůj účel. Proces, jak se výrobce řídicího systému dohaduje s výrobcem motoru na *datovém rozhraní*, je zjednodušen tak, že si v tomto smyslu pouze domluví, které rámce definované v SAE J1939 si budou předávat. Mohou si ještě domluvit nějaká data, která nejsou specifikována v SAE J1939 (norma nezabrala pro sebe veškerá možná ID). Množství kabeláže je v důsledku sériové komunikace opravdu výrazně snížené, přesto je velmi vhodné důležité signály předávat jednak datově, tak i kontaktním či napěťovým rozhraním. Například u rekonstrukce motorového vozu řady 842 je povel ke stopu dieselagregátu realizován dvojí cestou – napětím i řídicím rámcem dle normy tak, že řídicí systém pošle typ požadavku: otáčky a požadované otáčky: 0 s^{-1}.

Množství nezbytné dokumentace ukazuje obrázek 2.2. V této dokumentaci citelně chybí rejstříkové vyhledávání, příloha k PGN je řazena vzestupně dle PGN čísel, ale uživatel by určitě uvítal rejsříkové hledání dle názvů rámců (EEC1, ETC2, ERC1, ...). Další nedokonalost spatřuji v nepropracovaném zarovnání slov v datovém poli, a to



Obrázek 2.2: množství nezbytně nutné dokumentace pro implementaci SAE J1939

i v místech, kde by zárovnání na 2 B vůbec nic nebránilo. Chybějícím zarovnáním myslím případ, kdy se 2bytová informace nachází na druhém a třetím bytu a tím je tedy znemožněno vyčíst tyto 2 byty jako jedno slovo, ale musí se vyčíst složitěji, což stojí drahocenný strojový čas.

Tabulka 2.1: klady a zápory specifikace SAE J1939

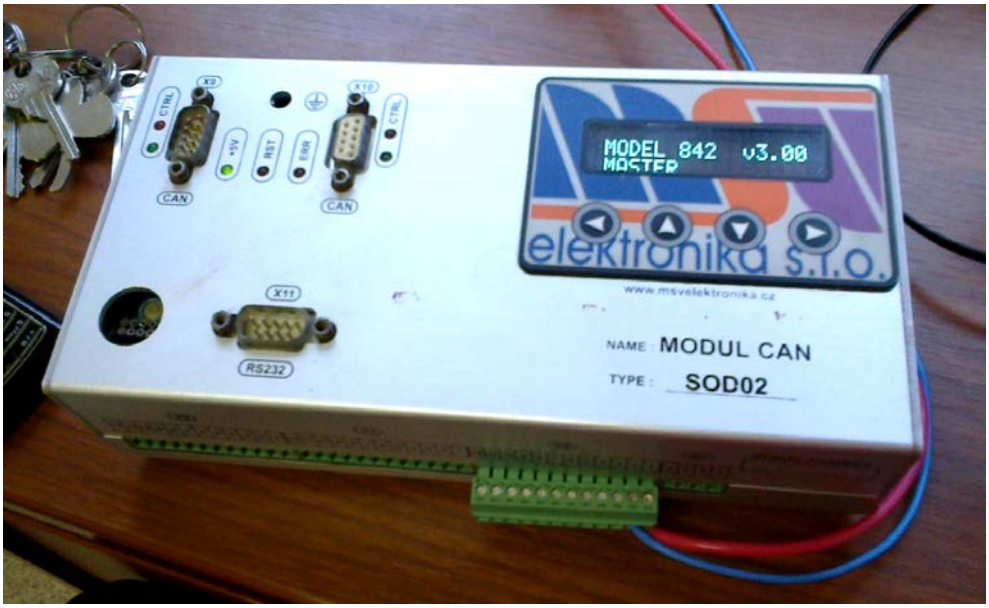
±	atribut
+	předpis pro interpretaci obrovského množstí různých veličin a parametrů
+	zjednodušení spolupráce kooperujících subjektů
−	nedomyšlené zarovnání vícebytových informacích do datového pole
−	absence rejstříkového vyhledávání dle názvu rámce
−	rozlišení přenášené veličiny je někdy nedostatečné, někdy naopak zbytečně jemné
−	SAE J1939 není volně dostupná

Kapitola 3

Model HAV realizovaný na HW MSV elektronika

3.1 Požadavky

Požadavky na tento konkrétní, realizovaný model HAV se skládají z obecných požadavků, jež jsou v zadání této práce a úvodní kapitole 1.1. Upřesňujícím požadavkem, jak už vyplývá z názvu této kapitoly, je požadavek na použití konrétního, již vytvořeného hardwaru. Ten je velmi stručně popsán v kapitole 3.2.1.



Obrázek 3.1: fotografie HW – MODUL CAN

3.2 Realizace

3.2.1 Hardware

Poznámka: Jedná se o komerčně využívaný HW – stavebnice *moduRail*, vyvinutý firmou MSV elektronika s.r.o. Z toho důvodu nemůžu k této práci přiložit dokumentaci zařízení, schémata ani popisovat HW do větších detailů. □

Použitý HW (prvek stavebnice moduRail) nese obchodní název *MODUL CAN*, používá se zejména na drážních vozidlech jako modul vzdálených vstupů a výstupů nebo jako řídicí jednotka dveří, vytápění a WC. Jedná se o lety prověřený, spolehlivý HW, procesor poskytuje dostatečný výkon a podporu pro CAN sběrnici.

MB90F543
16 bit

Pro účely simulace komponent pohonu se tento HW jevil po prvotním průzkumu možností jako dostatečný přesto, že procesor Fujitsu MB90F543 je pouze 16bitový. Protože má tento prosesor velmi slušnou podporu pro CAN, je předurčen pro automotive aplikace a i díky tomu je velmi dlouho vyráběn a distribuován. Pro práci s CAN sběrnicí má k dispozici 16 fyzických bank pro CAN rámce. Pro řadu aplikací je to dostatečný počet (počet can rámcu je menší nebo roven 16), avšak pokud se má vysílat a přijímat větší počet objektů, je třeba provádět multiplex. Pak je 16bitová architektura procesoru limitujícím faktorem, protože je třeba provádět více operací s 32bitovými čísly.

24 V ±30 %

Pro model byla použita verze modulu s displejem 2×16 znaků a čtyřmi tlačítky ovládajícími zobrazení na displeji. Obě CAN sběrnice jsou vyvedeny na čelní panel modulu na CANON 9 konektory. Na třetí CANON 9 konektor je vyvedeno rozhraní RS 232 (UART na procesoru a podpůrný obvod). Modul je napájen napětím 24 V ±30 %. Modul má vyvedeny 3 porty, které mohout být v různých konfiguracích (PT100 vstupy, PWM výstupy, digitální vstupy, digitální výstupy, analogové vstupy). Konkrétní konfigurace a další možnosti jsou uvedeny v katalogovém listu[3].

3.2.2 Software

C
assembler

SW, jež implementuje model HAV, je strojový kód, vykonávaný přímo procesorem. Software je vytvořen převážně v jazyce C a menší části kódu jsou napsány v jazyce symbolických adres (assembler). Jedná se například o rozšíření aritmetikých operací či úvodní inicializace procesoru. Tyto části SW byly vytvořeny v minulosti buď výrobcem procesoru nebo jsou dodávany se zařízením (MODUL CAN), anebo byly vytvořeny na našem pracovišti převážně vedoucím této práce – Dr. Ing. Myslivcem.

Struktura SW

Ideu, jak strukturovat SW pro řídicí aplikace, jsem převzal dle zvyků našeho pracoviště. SW tvoříme tak, aby se jednotlivé metody prováděly s definovanou posloupností a s definovanou pevnou periodou. Metody jsou prováděny buď v tzv. *rychlém přerušení*, nebo v tzv. *pomalém přerušení*. Zřídkakdy je využita i hlavní programová smyčka. Rychlé přerušení má vyšší prioritu než pomalé, často je použita frekvence přerušení 2 kHz. Pomalé přerušení má zpravidla periodu 50 Hz. Takováto struktura SW je spolehlivá a je léty ověřená. Různé dynamické problémy, jako např. deadlock, jsou touto strukturou eliminovány.

Časové rozlišení je dostačující – 500 µs u rychlého přerušení. Zde se vykonávají metody obsluhující CAN periferii (kapitola 3.2.2). Nastavená perioda rychlého přerušení 0,5 ms je vzhledem k nejnižší frame-rate řídicích CAN rámců (10 ms) zcela dostačující. CAN periferie je tedy obsluhována nikoliv v asynchronně se vyskytnoucích okamžicích (přerušení od vysílání či příjmu CANu), ale v pravidelném časovém intervalu. Dále se v rychlém přerušení vykonává simulace tachogenerátoru (kapitola 3.2.4), protože časové rozlišení u pomalého přerušení není dostačující.

V pomalém přerušení se vykonává zbytek metod softwaru. Až na konci pomalého přerušení se obsluhuje watchdog. Pokud by vykonávání metod mělo trvat déle, než je perioda přerušní, pak by nedošlo k obsluze watchdogu a po nějaké době by uvedl zařízení do resetu. Metody, které se provádějí v pomalém přerušení, uvádí následující výčet: watchdog

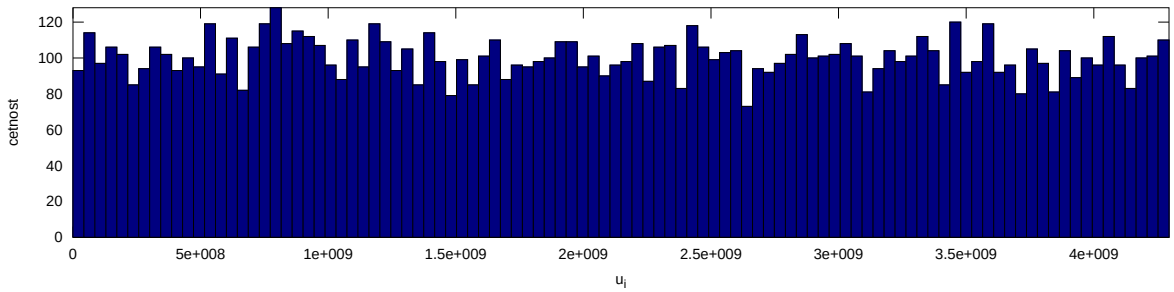
1. rekonfigurace can sběrnice, pokud je třeba. Pokud není potřeba, pak se provádějí body 2–8, jinak pouze tento,
2. kontrola, zda příchozí objekty mají správný frame-rate,
3. příjem/vysílání hnacího momentu od/do druhého modulu,
4. vyčítání příchozích CAN rámců,
5. **simulace komponent hnacího agregátu**,
6. plnění vysílaných CAN rámců nově spočtenými hodnotami veličin,
7. obslužná metoda pro prodloužené, konfigurační zprávy,
8. pomocné funkce pro komunikaci, statistika komunikace.

Generátor pseudonáhodných čísel

Generátor byl implementován dle materiálů[2], jedná se o generátor pseudonáhodných, rovnoměrně rozdělených čísel $u_i \sim U_{(0,\ 4\,294\,967\,296)}$. V jazyku C se s výhodou využije de-
seed finice *static* proměnné, jakožto prvního čísla zvaného *seed*. To se nastaví definicí makra na požadovanou hodnotu. Na řádku 4 využíváme toho, že nemusíme počítat zbytek po dělení číslem 2^{32}, protože pracujeme s 32bitovou proměnnou a jazyk C neřeší aritmetické přetečení.[1]. Algoritmus jsem ověřil – generované číslo posílal na CAN sběrnici a na PC tyto rámce odchytil, zpracoval a v prostředí GNU OCTAVE vykreslil histogram – obrázek 3.2.

Algoritmus 3.1: algoritmus pro generování pseudonáhodných čísel z rovnoměrného rozdělení na 32bitovém čísle

```
1 U32 gen_uniform_random_U32 ()
  {
          static U32 u = RAND_SEED_U32;
          u = 69069L * u + 1L;
          return u;
6 }
```



Obrázek 3.2: histogram náhodného výběru, $N = 10^4$. náhodný výběr je generován dle algoritmu 3.1

Původně bylo v plánu použít generátor čísel z normálního rozdělení o střední hodnotě 0. Algoritmy, pro generování normálně rozdělených čísel, vyložené v [2] jsou ale o několik řádů složitější[2], než generování rovnoměrně rozdělěných čísel. Právě vysoká spotřeba strojového času, který by ani nebyl k dispozici, rozhodla o použití jednodušší varianty.

Pro účely zašumění signálu simulovaného tachogenerátoru se použije algoritmus 3.1 následujícím způsobem: metoda vrací neznaménkové, 32bitové číslo $u_{i1} \sim U_{(0,\ 4\,294\,967\,296)}$.

[1] což je častou příčinou chyb, ale zde tuto vlastost jazyka s výhodou používám.
[2] ve smyslu spotřeby strojového času

Na číslo $u_{i2} \sim U_{(-2\,147\,483\,648,\ 2\,147\,483\,647)}$ se u_{i1} převede velmi jednoduše – přetypováním neznaménkového čísla na znaménkové. Další požadované zúžení intervalu se již provede dělením.

Pro účely stanovení náhodného času, po který bude CAN sběrnice v klidu po inicializaci, se opět využije algoritmus 3.1. Vrácené 32bitové číslo se odrotuje doprava o 26 bitů, tím se získá číslo $u_{i3} \sim U_{(0,\ 64)}$, použije se pro přičtení u_{i3}násobku desetin sekundy k minimálnímu časovému intervalu.

Rutiny pro práci s CAN sběrnicí

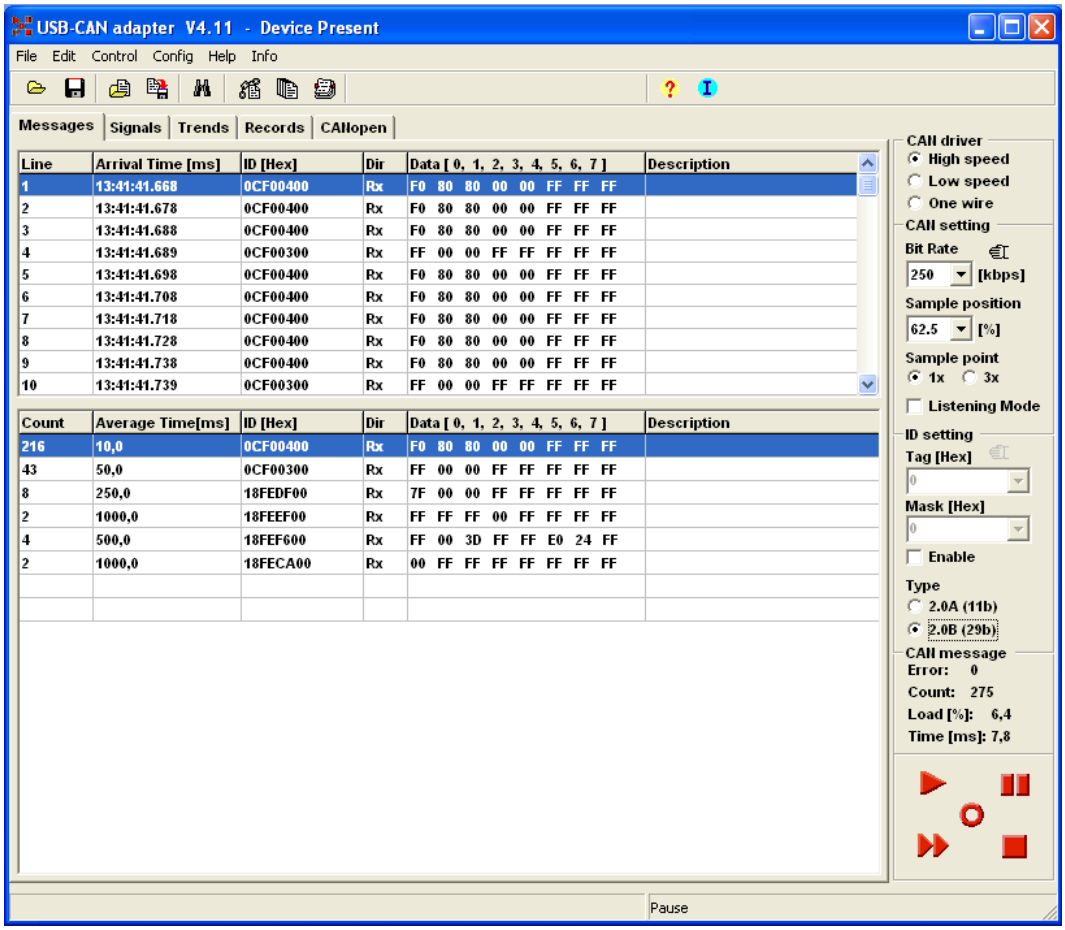
Pro účely této práce bylo nezbytně nutné přepracovat dosud užívané rutiny pro práci s CAN periferií procesoru. Dosud jsme neřešili případ, kdy by bylo potřebné *za běhu programu* změnit některé CAN objekty z příchozích na odchozí, případně opačně, nebo úplně zakázat jejich příjem či vysílání. Při tvorbě nových rutin, které budu dále nazývat jako CAN řadič, jsem v maximální možné míře využil již existující části kódu pro práci se sběrnicí, protože správnost tohoto kódu je ověřena bezproblémovou činností na desítkách drážních vozidel po dobu několika let. Dále jsem se rozhodl rozdělit řadič na 3 samostatné moduly:

CAN řadič

1. **transportní vrstvu**, která se stará o konfiguraci řadiče a vlastní vysílání a příjem,
2. **prezentační vrstvu**, která obsahuje všechny funkce pro vyčítání datových bytů CAN rámců na takové hodnoty, s nimiž pracuje simulace modelu HAV a naopak
3. a na **aplikační vrstvu**, která funkce z předchozích vrstev zastřešuje a volá ve správném pořadí. Uživatel v tomto souboru, případně v jeho hlavičkovém souboru, definuje použité CAN rámce a nastavuje chování řadiče.

V rámci této práce jsem si musel poradit i s tím, že během provádění programu bude umožněno, aby se např. rámce, patřící motoru, nikoliv vysílaly, ale přijímaly. Taková situace nakonec v reálu nastala, viz kapitola 3.3.1. Potom nastává nepříjemná situace, kdy je potřeba provádět multiplex přijímaných i odesílaných rámců. Musel jsem vytvořit strategii pro inicializační rutinu, která určí počty vysílaných a přijímaných rámců. Na základě těchto čísel určí, zda-li je třeba provádět nějaký multiplex, určí pozice daných rámců na bankách CAN periferie a umístí je. V hlavičkovém souboru aplikační vrstvy jsou pro účely nastavení parametrů strategie definovány makra. Těmi se například ovlivní, kolik ze šestnácti bank se využije pro příchozí rámce a kolik z toho se využije pro multiplex příchozích a kolik se umístí na banky přímo.

Velmi limitujícím faktorem obslužné rutiny CANu je fakt, že nesmí trvat 500 µs nebo více. Ideálně by rutina měla trvat zhruba do 350 µs. Proto jsem vynaložil maximální úsilí na to, aby rutina byla co možná nejrychlejší. Kontrolu frame-rate příchozích rámců jsem přesunul z rychlého přerušení do pomalého. Roztřídění objektů na nemultiplexované a mutliplexované, příchozí a odchozí se provádí při reinicializaci. Těmito postupy se mi podařilo zrychlit rutinu na 350–400 µs. Toto zrychlení CAN řadiče bylo téměř nejsložitější částí práce.



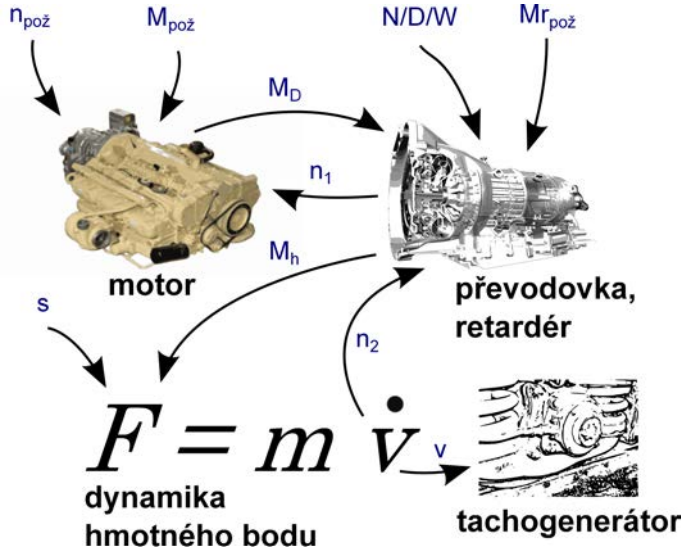
Obrázek 3.3: screenshot aplikace pro analýzu CAN sběrnice, spodní panel ukazuje bezchybný frame-rate vysílaných rámců z modelu HAV

3.2.3 Modelování a simulace fyzikálních dějů

Stěžejní[3] veličiny modelu HAV ukazuje obrázek 3.4, ukazuje 3 základní komponenty (motor, převodovka s retardérem a vlastní dynamika vozu) a vztahy mezi nimi. V obrázku je např. znázorněno, že otáčky motoru jsou do jisté míry (jízda se zařazeným, blokovaným stupněm) určeny rychlostí vozidla a ta je určena krouticím momentem z motoru přes převodovku. Implementace (zdrojový kód v jazyce C je uveden v příloze C.1).

Požadovanými veličinami jsou: požadované otáčky[4] ($n_{\text{pož}}$), požadovaný moment motoru ($M_{\text{pož}}$), požadovaný moment retardéru ($Mr_{\text{pož}}$) a povel pro řazení převodovky (N/-D/W). Dále je možné na modelu HAV nastavit sklon trati, po které simulované vozidlo „jede“ (s).

Výstupními veličinami jsou: rychlost, kterou simulovaný tachogenerátor převádí na frekvenční signál (v) a další různé, zakreslené i nezakreslené veličiny – otáčky (n), kroutící momenty (M), tlaky a teploty.



Obrázek 3.4: stěžejní veličiny modelu HAV

3.2.4 Simulace tachogenerátoru

Simulaci tachogenerátoru ukazuje algoritmus 3.2. Výstupní frekvenční signál se generuje na pinu TxD UARTu z toho důvodu, že je po příslušném nastavení možné tento TxD

[3] v obrázku nejsou z důvodu přehlednosti zakresleny včechny veličiny modelu HAV

[4] pouze pro start a stop motoru, nebo vytáčení motoru při neutrálu

ovládat přímo, jako jakýkoliv jiný výstupní pin. Signál je převeden na napěťové úrovně dle RS232, které již má napětové úrovně symetricky kolem nuly, typicky ±12 V. Takto je vytvářen požadovaný frekvenční signál, pouze tvar signálu se liší oproti reálnému tachogenerátoru, což stejně nebylo požadováno (kapitola 1.1.2 a 1.1.3).

Algoritmus 3.2: algoritmus simulace tachogenerátoru

```
void InitUart(B1 tachogen);


#define   PRUMER_KOLA      840
#define   HRAN_NA_OTACKU    60
#define   SIGNAL_PIN       PDR4_P45 // nutno na Tx UARTU (-12 - +12V)

static U32 vypocet_konstanty()
   {
   U32 tempU32;
   tempU32 = PRUMER_KOLA*314L*36L;
   tempU32 = tempU32 / 1000;
   tempU32 = tempU32 << 16;
   tempU32 = tempU32 / HRAN_NA_OTACKU;
   tempU32 = tempU32 * KMH;
   return tempU32;
   }

// generovani rychlosti
void signal_rychlost()
   {
   static U8 prubeh = 0;
   static U8 signal = 0;
   static U32 kum_rychlost = 0L;
   static U32 rychlost_prah = 0L;
   I32 randU;

   // inicializace uartu pri prvnim behu
   if (prubeh == 0)
      {
      InitUart(TRUE);
      rychlost_prah = vypocet_konstanty();
      prubeh++;
      }

   // integrace rychlosti
   kum_rychlost += *model.dynamika->rychlost_jemne;
   // nahodne, rovnomerne rozdelene cislo, znamenkove, stred kolem 0
   randU = gen_uniform_random_U32();
   randU /= 256;

   // prelezl jsem mez?
   if (kum_rychlost >= (rychlost_prah + randU))
      {
      signal = ~signal;
```

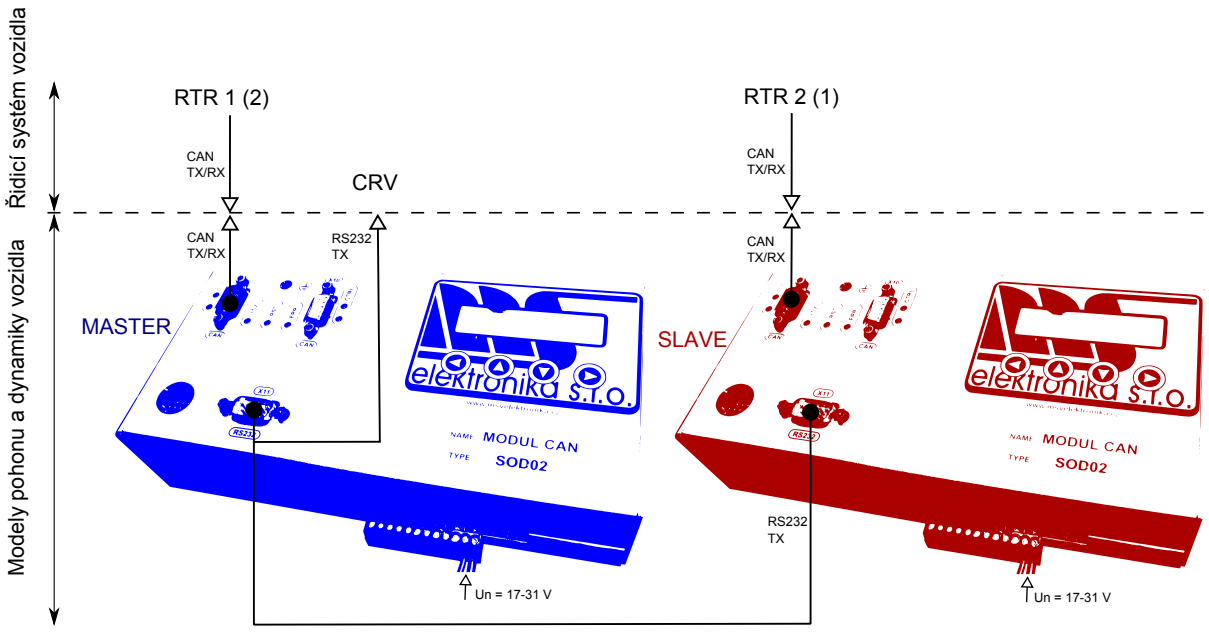
```
    kum_rychlost -= rychlost_prah + randU;
    }
  // prirazeni na pin
  SIGNAL_PIN = signal & 0x01;
  }
```

3.2.5 Připojení k ŘS

Jelikož předmětné drážní vozidlo obsahuje dvě hnací soustrojí, tak i řídicí systém obsahuje 2 bloky RTR (regulátor trakce). Model HAV je tedy nutno použít ve dvou kusech, každý z nich obsahuje jeden motor a jednu převodovku s retardérem. Pak se oba mezi sebou liší svojí konfigurací (kterou lze za běhu programu snadno změnit). Tzv. *slave* svůj vypočtený moment na hnané hřídeli převodovky (rozmezí −100–100 %)[5] vysílá po rozhraní RS 232 druhému modelu, který se nazývá *master*. Master přijímá moment od slave, připočte ke svému a tímto momentem modeluje rychlost vozidla. Rychlost vozidla vysílá tak, že jednoduchým způsobem simuluje tachogenerátor a na napěťovém rozhraní tuto rychlost poskytuje ŘS (bloku CRV). Oba modely jsou připojeny na „svůj“ blok RTR.

slave

master



Obrázek 3.5: připojení modelů HAV k řídicímu systému. RTR – regulátor trakce, CRV – centrální regulátor vozidla

[5] z $M_{\max} = 1600$ N m

3.2.6 Vizualizační možnosti a ovládání modelu HAV

Vizualizační možnosti modelu HAV jsou velmi omezené, HW obsahuje pouze displej 2×16 znaků a 4 tlačítka, použitá pro ovládání zobrazení. Displej vlastního modelu HAV je sice dostatečný pro vizualizaci a ovládání, nicméně velký komfort neposkytuje. Vizualizaci v daleko komfortnější podobě mohou poskytnout dva desetipalcové displeje připojené k ŘS. Displej se zobrazenou servisní stránkou regulátoru trakce ukazuje obrázek 1.4 na straně 9.

Pomocí zmíněných 4 tlačítek a displeje 2×16 znaků se provádí ovládání modelu HAV, je možné nastavovat:

- Poruchové stavy jednotlivých komponent modelu HAV,
- hodnoty jednotlivých tlaků, teplot a hladin,
- „zastavení vozidla“,
- sklon trati, po které vozidlo „jede“,
- směr komunikace jednotlivých komponent a k tomu příslušné vypnutí či zapnutí simulace (příjem rámců komponenty — simulace a vysílání rámců — ani vysílání, ani příjem)

3.3 Dosažené výsledky

Model HAV na zadaném HW se mi podařilo vytvořit, splňuje všechny požadavky. Navíc se podařilo splnit i ty požadavky, které vyvstaly během vývoje – simulace tachogenerátoru (kapitola 3.2.4). Zjistilo se např., že se ve vyhodnocení zrychlení na ŘS vyskytují jisté zázněje, nejspíš proto, že model HAV generuje a ŘS vyhodnocuje signál s takovými opakovacími frekvencemi, které jsou navzájem soudělné. Rozhodl jsem se proto výstupní signál v jistém smyslu zašumět. Byl implementován generátor pseudonáhodných čísel (kapitola 3.2.2), který slouží k zašumění signálu. Tím byl problém kompletně vyřešen. Během vývoje modelu HAV se samozřejmě vyvíjel i ŘS, od jisté verze vyžadoval ŘS zasílání konfigurace od retardéru, aby povolil brzdění retardérem. I na tento požadavek jsem zareagoval a implementoval zasílání prodloužených zpráv (kapitola 2.2.3).

Model HAV ve výsledku dobře posloužil během vývoje ŘS. S pomocí modelu HAV odladil a vyzkoušel vedoucí této práce své algoritmy. SW řídicího systému byl díky možnosti

laboratorního zkoušení velmi dobře připraven na reálné zkoušky. Model HAV velmi dobře posloužil při integračním testu ŘS—motor viz kapitola 3.3.1.

3.3.1 Integrační test ŘS—motor

Ve dnech 15. a 16. září 2010 jsme se s ŘS vozu účastnili integračního testu ŘS—motor ve firmě TEDOM v Jablonci nad Nisou. Automatická převodovka ZF v tu dobu ještě k dispozici nebyla. Hned z kraje druhého dne testů jsme řešili, proč motor nereaguje na požadavek zvýšených otáček. Tvůrce řídicí jednotky motoru odvětil, že pokud se otáčky na hřídeli motoru liší od otáček na hnací hřídeli převodovky, tak nepovoluje vyšší otáčky než volnoběžné (650 min^{-1}).

Protože software řídicí jednotky motoru je v jistém smyslu těžkopádný, bylo daleko elegantnějším řešením upravit SW mého modelu HAV a připojit ho na sběrnici. Udělal jsem takovou úpravu, kdy se rámce motoru nevysílaly, ale přijímaly a odpovídajícím způsobem se upravila i vlastní simulace – motor nebylo potřeba simulovat, ale veličiny motoru se vyčítaly z rámců, jež posílal opravdový motor. S těmito veličinami se odpovídajícím způsobem pracovalo, např. právě otáčky hnací hřídele převodovky se rovnaly otáčkám motoru. Tím jsme problém vyřešili během velmi krátké chvíle a bylo možné v testech pokračovat.

Dále byly vyzkoušeny možnosti, kdy do řízení motoru zasahuje převodovka. Přesto, že použitá převodovka umí řadit i pod plným kroutícím momentem motoru (1600 N m), vyzkoušeli jsme možnost zásílání řídícího rámce převodovky. Na stisk tlačítka na modelu HAV se půl sekundy vysílal požadavek na kroutící moment 10 %, podobným způsobem jsme zkusili požadavek na otáčky 1000 min^{-1}. Tím jsme firmě TEDOM pomohli ověřit, že jsou připraveni i na možnost, že by si převodovka musela omezovat kroutící moment motoru, případně snižovat či zvyšovat otáčky.

3.3.2 Omezující faktory

Největším omezujícím faktorem tohoto konkrétního modelu HAV je jeho vysoké vytížení procesoru, které v konfiguraci master dosahuje 90 %. Nejvíce vytěžují procesor rutiny pro obsluhu CANu, protože se volají často (2000 s^{-1}) a ve standardní konfiguraci se odesílá velký počet CAN rámců (24 odesílaných rámců s průměrným frame-rate 485 ms a 5 přijímaných rámců s průměrným frame-rate 520 ms).

Obsazení paměti programu i dat procesoru dosahuje v obou případech zhruba poloviny. Není to tedy prozatím limitujícím faktorem.

3.3.3 Znovupoužitelnost, doba vývoje

Doba vývoje tohoto modelu HAV byla poměrně dlouhá, přibližně 4 měsíce intenzivní práce. Nejvíce náročný byl vývoj nových rutin pro obsluhu CANu tak, aby respektovaly nové požadavky.

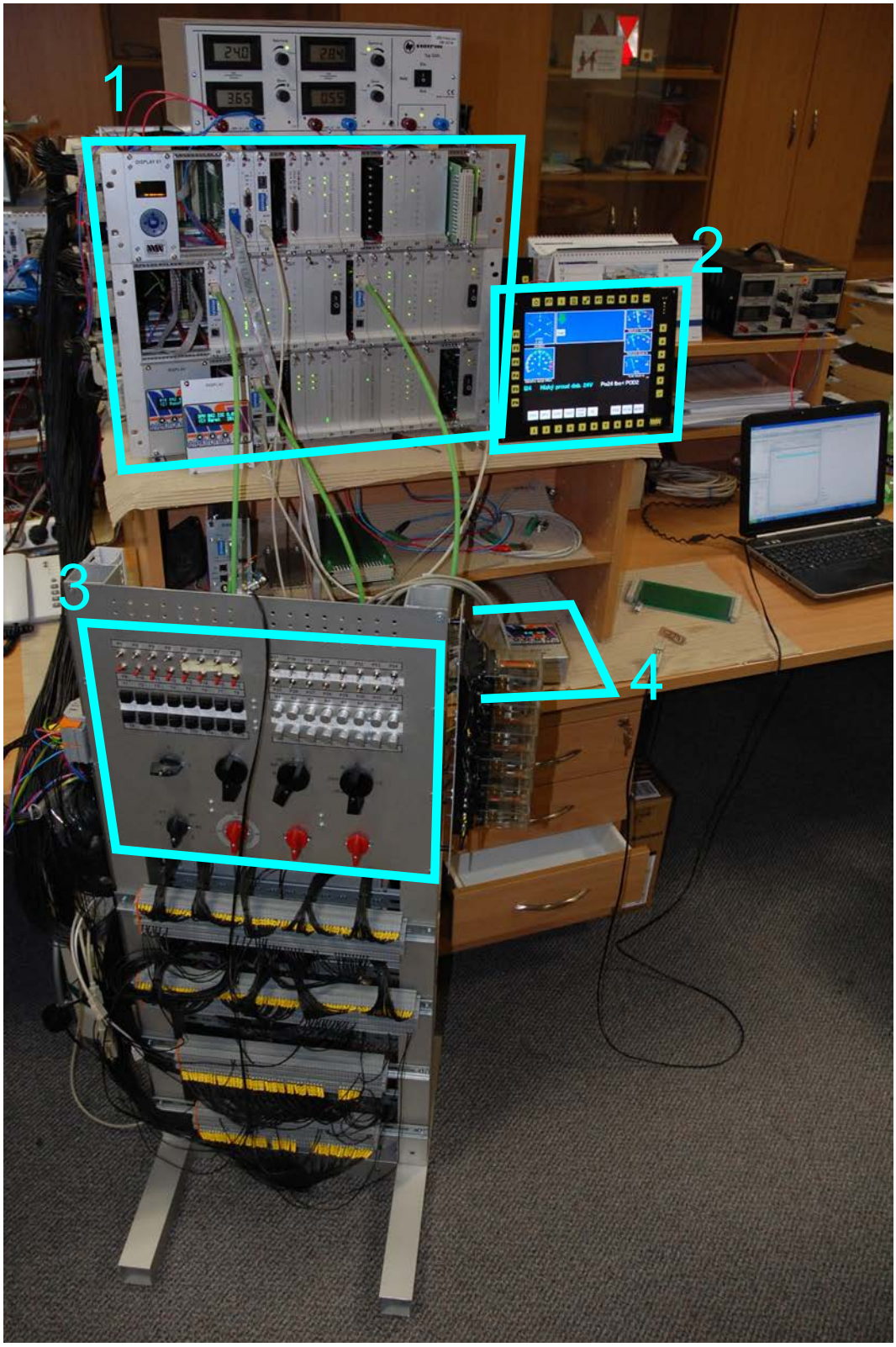
SW modelu HAV byl napsán tak, aby bylo možné provádět funkční změny co nejjednodušeji. Přidávání, ubírání a změna CAN objektů[6] je maximalně jednoduchá, bez potřebného zásahu do transportní vrstvy. Při potřebě změny v simulaci komponent hnacího agregátu vozidla by byla nutná větší revize v příslušném zdrojovém souboru. Ten stavající by posloužil jako zdroj inspirace.

3.3.4 Naměřené průběhy stěžejních veličin modelu HAV

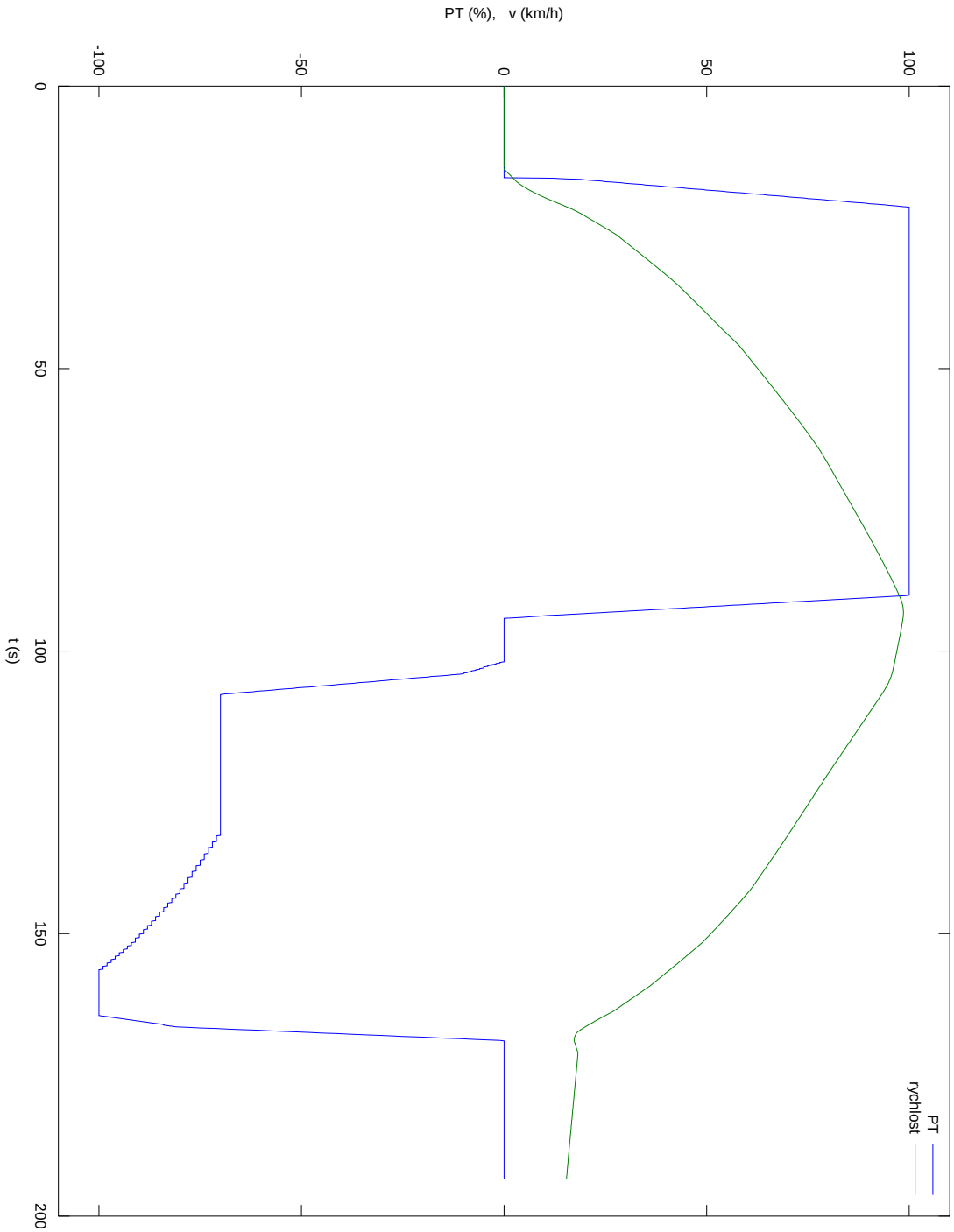
Všechny komponenty řídicího systému vozidla (CRV, RTR i DPV) umožňují záznam „svých“ veličin, parametrů, stavů. Pro ukázku jsem provedl záznam regulátoru trakce (RTR) a stěžejní průběhy vykreslil do grafů – obrázky 3.7–3.9.

Simulace byla následující: nejprve se stojící vozidlo nastartovalo, po startu jsem zadal plný poměrný tah (PT). Na obrázku 3.8 a 3.9 je dobře vidět řazení převodovky. Nejprve se vůz rozjíždí na první rychlostní stupeň s neblokovaným hydrodynamickým měničem, při rychlosti 20 $\mathrm{km\,h^{-1}}$ se měnič zablokuje, čímž poklesnou otáčky motoru a nyní je motor přímo spojen s planetovou převodovkou. Dále pokračuje řazení až do pátého rychlostního stupně (v obrázku 3.8 je dobře vidět zařazení čtvrtého rychlostního stupně, neboť je jeho převodový poměr 1:1). Před dosažením 100 $\mathrm{km\,h^{-1}}$ jsem zadal výběh ($PT = 0$ %) a převodovka vyřadila. Vozidlo jelo výběhem a zpomalovalo v důsledku odporových sil. Poté jsem zadal brzdění retardérem, plným brzdným účinkem. Převodovka zařadila nejvyšší možný rychlostní stupeň (5) a retardér, který je na hnací hřídeli převodovky, způsoboval brzdění vozidla. Dále převodovka podřazovala až do dosažení 18 $\mathrm{km\,h^{-1}}$, kdy je již efekt retardéru velmi malý, proto převodovka vyřadila a vozidlo opět jelo pouze výběhem. Při brzdění retardérem je vidět, jak RTR koriguje (zvyšuje) požadovaný moment retardéru s klesající rychlostí vozidla.

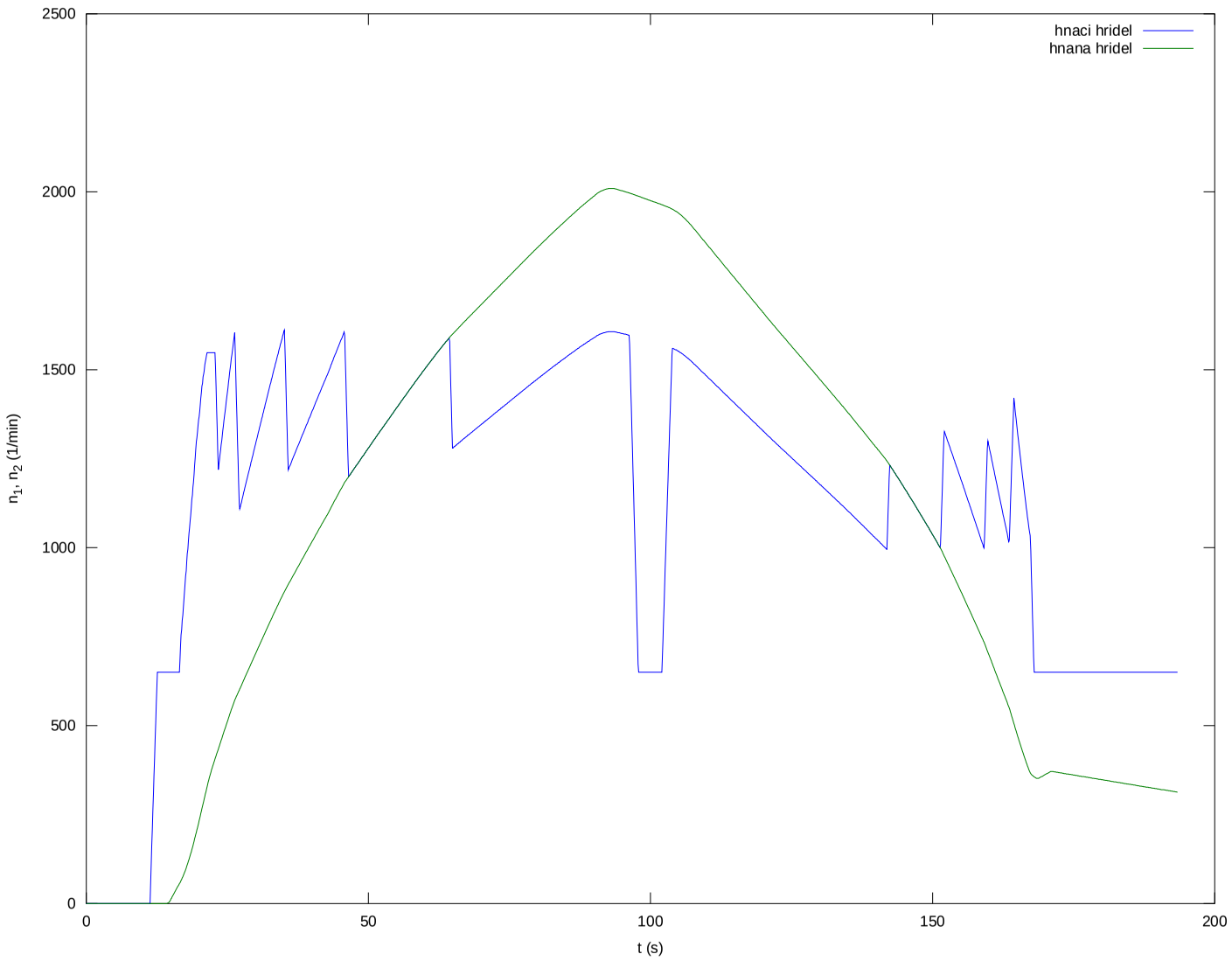
[6]definice CAN rámce, jeho periodicita, časová platnost, funkce z prezentační vrstvy (vyčítání, nebo plnění dat)



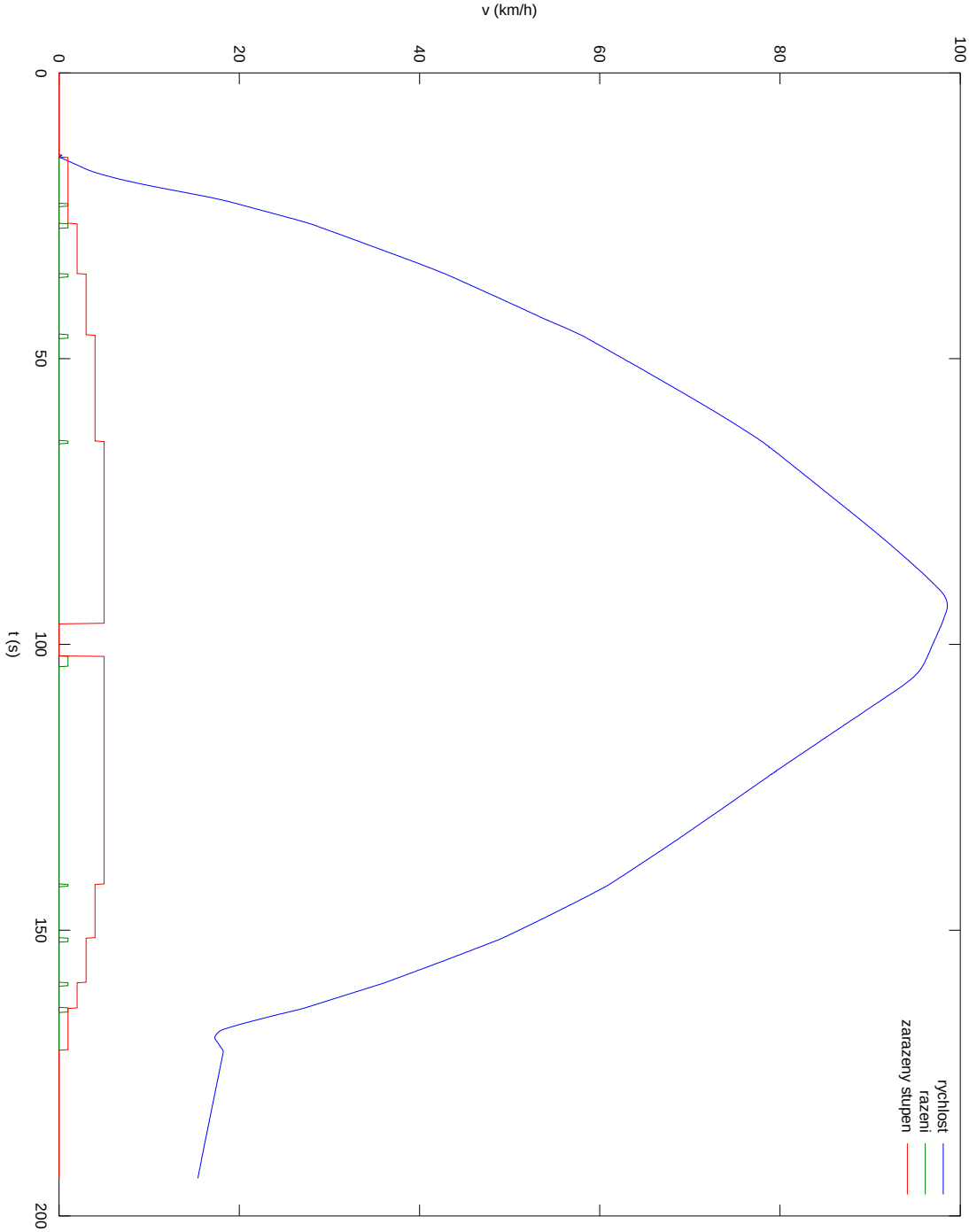
Obrázek 3.6: sestava ŘS, ovladače, modely HAV, 1 – ŘS, 2 – jeden z displejů, 3 – improvizovaný pult strojvedoucího, 4 – jeden z modelů HAV



Obrázek 3.7: průběh PT a rychlosti v čase



Obrázek 3.8: průběh otáček převodovky v čase



Obrázek 3.9: průběh rychlosti, řazení a zařazeného stupně převodovky

Kapitola 4

Možnosti rozšíření modelu HAV na HW MSV, jiná řešení, komerční výrobky

Cílem této kapitoly je rozebrat použitelnost již hotových řešení, postupně, podle míry použitelnosti pro modelování a simulaci komponent hnacího agregátu vozidla. Poslední sekce této kapitoly popisuje možné vylepšení vytvořeného modelu HAV, popsaného v kapitole 3

4.1 CAN analyzéry

CAN analyzér by obecně mohl být použit pro simulaci komponent HAV. Obvykle tyto analyzéry umožňují nastavit CAN rámce (ID, data, periodicita) a zasílat je na sběrnici. Takovýto simulátor by byl ale značně minimalistický, uživatel by musel data (simulavané veličiny) zadávat ručně, změny by se nedaly provádět rychle. Nejspíše by se dala vytvořit nadstavba k obslužné aplikaci analyzéru, avšak by šlo o stavění na nejistých základech (viz následující poznámka) a na operačním systému PC by se nedaly spolehlivě časovat intervaly nižší než půl sekundy, což by mohlo být limitujícím faktorem.

Poznámka: CAN analyzér od firmy IMFsoft s.r.o., kterým disponuje naše oddělení, byl použit při vývoji CAN driveru, sreenshot obslužné aplikace ukazuje obrázek 3.3 na straně 22. Tento analyzér sice posloužil při vývoji, avšak vykazuje obrovské a zásadní chyby funkčnosti, pravděpodobně v úrovni firmwaru i na úrovni obslužné aplikace. □

4.2 Statické přípravky

V drtivé většině komerčních výrobků s názvem nebo přívlastkem *J1939 simulátor* se jedná o statické modely. Umožňují pouze vizualizovat data z přijímaných CAN rámců, případně vysílat nastavené veličiny. Oproti obyčejným CAN analyzerům (kapitola 4.1) poskytují navíc pouze interpretaci veličiny z/do dat dle SAE J1939. Takovéto modely, by podobně jako CAN analyzéry, nebyly pro naše účely dostačující.

4.3 Dynamické přípravky

Velkou skupinou takovýchto zařízení jsou jednoúčelové testovací přípravky používané v průmyslu. Tyto přípravky jsou vyvíjeny za stejným účelem, jako model HAV. Často se jedná o nekomerční, interní výrobky dané společnosti, což vylučuje použití pro námi požadovaný model HAV.

Do této kategorie spadá i model HAV na HW MSV. Možnosti jeho rozšíření popisuje sekce 4.5. Existují i analogové dynamické modely určitých komponent hnacího agregátu. Vedoucí této práce – Dr. Ing. Ivo Myslivec – vytvořil analogový dynamický model šestiválcového vznětového motoru, který modeluje problém až do úrovně průběhu tlaků ve válcích. Jistě existují, nebo by šly vytvořit, i analogové modely ostatních komponent HAV, avšak by přibyla nutnost vyvinout převodník mezi napěťovým rozhraním a datovým rozhranním dle SAE J1939.

4.4 Simulinkové modely

Simulinkový model HAV by měl obrovskou výhodu v tom, že by jeho vývoj nebo modifikace trval pouze zlomek času oproti modelu HAV na HW MSV. Takový model by mohl být velmi detailní, propracovaný. Vizualizace průběhů modelovaných veličin by probíhala na PC. Při vývoji takovéhoto modelu by se daly použít nadstavbové knihovny pro simulink, přímo zaměřené na pohony. Takovéto na první pohled ideální řešení opět naráží na požadavek rozhraní modelu dle standardu SAE J1939. Jednodušším řešením by bylo použít kartu do PC, která poskytuje analogové vstupy a výstupy a vytvořit převodník mezi datovou komunikací dle SAE J1939 a analogovým rozhráním. Takové řešení by ale

nejspíš naráželo na omezený počet analogových vstupů a výstupů na zmíněné kartě, se kterou umí Matlab spolupracovat. Daleko sofistikovanějším řešením by byl vývoj karty do PC, která by se přímo připojila na CAN sběrnici a prostředí Matlab by poskytovala přímo hodnoty veličin. Pokud je mi známo, taková karta zatím neexistuje a její vývoj, společně se SW podporou a ovladači, by byl nejspíš velmi náročný.

4.5 Možnosti rozšíření modelu HAV na HW MSV

Vytvořený model HAV by se vzhledem k omezujícím faktorům (kapitola 3.3.2) dal rozšířit pouze o možnost ovládání (kapitola 3.2.6) simulace z PC (pomocí aplikace a USB–CAN převodníku). Ovládání simulace z PC by byla pohodlnější než pomocí displeje 2×16 znaků a 4 tlačítek. Simulaci některých pomalu se měnících veličin, např. teploty, by mohl provádět PC.

Kapitola 5

Závěrečné zhodnocení

Požadovaný model hnacího agregátu vozidla (HAV) na hardwaru CAN MODUL od MSV elektronika s.r.o. (kapitola 3) byl úspěšně realizován. Dosažené výsledky popisuje kapitola 3.3. Tento model HAV byl velmi užitečný při integračním testu motor—řídicí systém vozidla, kterého jsme se s naším řídicím systémem zúčastnili ve dnech 15. a 16. září roku 2010 (kapitola 3.3.1). Naměřené průběhy stěžejních veličin modelu HAV ukazují obrázky 3.7–3.9.

V kapitole 1.1.2 je uvedno, že velký důraz na model HAV je kladen z hlediska jeho komunikace s řídicím systémem. Při tomto úhlu pohledu se požaduje totožné chování jako reálné řídicí jednotky jednotlivých komponent pohonu. Proto jsem musel nastudovat kolekci standardů SAE J1939. Její stručné vysvětlení, příklady a zkušenosti s implementací jsou zpracovány v kapitole 2. Požadavek na chování komunikace byl splněn, řídicí systém nepozná rozdíl mezi modelem HAV a opravdovým HAV.

Kapitola 4 shrnuje, jaké jsou možnosti v modelování a simulaci komponent hnacího agregátu vozidla. Pevný požadavek na rozhraní modelu a řídicího systému poněkud komplikuje ty cesty, které by byly z hlediska modelování ideální (simulink). Pro takové použití by byl nutný velmi složitý vývoj s nejistým výsledkem. V závěru kapitoly je popsáno, jak by se dal vytvořený model HAV ještě vylepšit.

Literatura

[1] Elstner, M.: *Zvýšení provozní spolehlivosti motorových vozů ř. 842*. 2009, diplomová práce.

[2] Havlena, V.: *Odhadování, filtrace a detekce*. [online], [citováno 7.prosince 2011]. URL <https://moodle.dce.fel.cvut.cz/pluginfile.php/798/mod_page/content/11/OFD_slides.pdf>

[3] MSV elektronika s.r.o.: *CAN MODUL*. [online], [citováno 25.prosince 2011]. URL <http://test.msvelektronika.cz/externi%20soubory/CAN_150.pdf>

[4] SAE International: *SAE J1939 Standards Collection on the Web: Content*. [online], [citováno 31.prosince 2011].

[5] Neznámý: *A problem shared is a problem halved*. [online], [citováno 7.prosince 2011]. URL <http://cisolutions.co.uk/document_296>

[6] Švestka, D.: *Atlas lokomotiv*. [online], [citováno 10.prosince 2011]. URL <http://www.atlaslokomotiv.net>

Příloha A

Seznam použitých zkratek

ARR . . automatická regulace rychlosti – režim řízení vozidla
AVV. . . automatické vedení vlaku – režim řízení vozidla, anglicky ATO
AŽD . . . automatizace železniční dopravy
BAM . . broadcast announce message – typ CAN rámce dle SAE J1939
BP . . . . brzdič přímočinný – typ lokomotivního brzdiče
BSE . . . brzdič samočinný elektrický – typ vlakového brzdiče
CAN . . control area network
CRV . . . centrální regulátor vozidla – komponenta ŘS AŽD/MSV elektronika
EEC . . . electric engine control – typ rámce dle SAE J1939
RRC . . electric retarder control – typ rámce dle SAE J1939
ETC . . . electric transmission control – typ rámce dle SAE J1939
HAV . . . hnací agregát vozidla
HW . . . hardware
MSV . . moravskoslezská vagónka
NVL . . . národní vlaková linka – standard mezivozové komunikace
OHV . . over head valve – typ rozvodu motoru
PGN . . parametr number group
RTR . . . regulátor trakce – komponenta ŘS AŽD/MSV elektronika
ŘS . . . . řídicí systém
SAE . . . society of automotive engineers
SPN . . . suspect parameter number
STP . . . shielded twisted pair – stíněná kroucená dvojlinka
SW . . . . software
UTP . . unshielded twisted pair – nestíněná kroucená dvojlinka

Příloha B

Seznam použitých veličin

$n_{\text{pož}}$	otáčky požadované	(min^{-1})
$M_{\text{pož}}$	kroutící moment požadovaný	(% z $M_{\max} = 1600\ \text{N}\,\text{m}$)
n_1	otáčky hnací hřídele převodovky (a motoru)	(min^{-1})
n_2	otáčky hnané hřídele převodovky	(min^{-1})
$M_{r\text{pož}}$	kroutící moment retardéru, požadovaný	(% z $M_{\max} = 1600\ \text{N}\,\text{m}$)
M_D	kroutící moment motoru	(% z $M_{\max} = 1600\ \text{N}\,\text{m}$)
M_h	kroutící moment hnací	(% z $M_{\max} = 1600\ \text{N}\,\text{m}$)
v	rychlost vozidla	$(\text{m}\,\text{s}^{-1})$
s	sklon trati	(‰)

Příloha C

Ukázky implementace

C.1 Rutina vlastní simulace modelu HAV

Algoritmus C.1: model.c

```
#include "model.h"
extern B1 model_master;



////////////////////////////////////////////////////////////// PROMENNE A KONSTANTY
const I32 VmaxModel =  300L*KMH << 16;
const I16 hmot = 460;

I32        Vmodel =      0L;
U16        Vskut =       0;
I8         sklonTr =     0;
U16        Ndiesl =      0;
U8         stupen =      0;
I8         Rmax =        100;
U8         rezimR =      0x51;
e1939BOOL jizda_povP = ano;

I32 Amodel;
I16 Askut;
I8 MhnaciSlave;
I16 Mvysl, Mdiesel, MskutR, MdieselZtr, Mbrzd;
U16 Npoz, Nvyst, Nvst, Nprep, Nmenic;
U16 Qmenic, PomerS;
char rozsah;
e1939BOOL menic_blok, ready_for_break_releas,
   hridel_spojena, razeni, shift_inhibit_ind_activ;
e1939ACTIVE_SHIFT console_indi;

////////////////////////////////////////////////////////// INICIALIZACE MODELU
sDIESEL diesel = { &Ndiesl, &Npoz, &Mdiesel, &Mdiesel, &MskutR, &MdieselZtr,
```

```
                0, 81, 52, 0, 0, 21, 22, 0x00 };
sPREVOD prevodovka = { &hridel_spojena, &menic_blok, &razeni,
   &ready_for_break_releas, &console_indi, &jizda_povP,
   &shift_inhibit_ind_activ, &Nvyst, &Nvst, &Qmenic, &PomerS, &stupen, &stupen,
   &rozsah, &rozsah, tra_nic, 0, 63, 0, 100 };
sRET retarder = { &Mbrzd, &Mbrzd, &Mbrzd, &Rmax, &rezimR, 52, 0x00, 1, 2, 2000 };
sRIZENI rizeni = { disabled, ne, N, 0, 0, 0, 0, 0, 0, FALSE, 0, 0, 0 };
sDYNAMIKA dynamika = {&Vmodel, &Vskut,&Mvysl, &Mdiesel, &Mbrzd, &MhnaciSlave,
                &sklonTr, &Askut };
sPORUCHY poruchy = { por_cer_neni, por_cer_neni, por_cer_neni };
sMODEL model = {&diesel, &prevodovka, &retarder, &rizeni, &dynamika, &poruchy};

//////////////////////// TABULKY PRO NELINEARITY, RAZENI, POMERY PREVODOVKY, ATD
const I16 modelZtrat [] =
   {
   OBECNA+2,
    800*RPM,   2,
   1400*RPM,   5,
   2000*RPM,  10
   };
// tabulka menice moment->otacky
const I16 modelMeniceN [] =
   {
   OBECNA+4,
     2,     0*RPM,
    15,   650*RPM,
    35,   950*RPM,
    65,  1300*RPM,
   100,  1600*RPM
   };
// tabulka menice otacky->moment
const I16 modelMeniceM [] =
   {
   OBECNA+4,
        0*RPM,    2,
      650*RPM,   15,
      950*RPM,   35,
     1300*RPM,   65,
     1600*RPM,  100
   };
// prevodove pomery (x1000)
const U16 radiciPom [] = {0, 2810, 1840, 1360, 1000, 800};
// odpovidajici znaky, jez prevodovkaovka vysila
const char znakPrevStupen []  = "NDDDDD";
// razeni nahoru na:    blok.1, blok.2, blok.3, blok.4, blok.5
const U16 zmenH []  = { 20*KMH, 28*KMH, 43*KMH, 58*KMH, 78*KMH, 98*KMH };
// razeni dolu na:              menic.1,blok.1, blok.2, blok.3, blok.4
const U16 zmenD []  = {  0*KMH, 18*KMH, 27*KMH, 36*KMH, 49*KMH, 61*KMH };


static void mod_diesel (void);
static void mod_tra (void);
static void mod_momenty_agregatu (void);
```

```
static void mod_ret (void);
static void mod_hnaciMom (void);
static void mod_dynamika (void);
static void mod_ostatni_veliciny (void);

void mod_model (void)
{
        if (SLAVE)
                Vskut = model.rizeni->rychlost_ccvs;

        mod_diesel ();
        mod_tra ();
        mod_momenty_agregatu ();
        mod_ret ();
        mod_hnaciMom ();
        mod_dynamika ();
        mod_ostatni_veliciny ();
}

static void mod_diesel (void)
{
        // otacky pozadovane a jejich osetreni
        Npoz = model.rizeni->Npozad;
        if (model.rizeni->regulace == otacky)
        {
                // start jinak regulerni otacky v D-poloze
                if ((Npoz != 0) && (Npoz < 650*RPM))
                        Npoz = 650*RPM;
        }
        // nenacha potopit diesel pri normalni jizde
        else if (Ndiesl >= 400*RPM)
                Npoz = 650*RPM;
        // regulace na moment, nebo pochybne otacky.
        else
                Npoz = 0;

        // PREPOCET VSTUP. A VYST. OTACEK PREVODOVKY DLE RYCHLOSTI VOZIDLA
        // prepocet vystupnich otacek prevodovky podle rychlosti
        Nvyst = trojclenkaU (2040*RPM, Vskut, 100*KMH);
        // otacky dieselu prepoctene z vystupu a dle zarazeneho stupne
        Nprep = trojclenkaU (Nvyst, radiciPom[stupen], 1000);

        // pri blokovanem menici
        if (menic_blok == ano)
        {
                if (Npoz < Nprep)
                        Npoz = Nprep;
        }
        // regulace ma moment, menic neblokovan
        else if (model.rizeni->regulace == moment)
        {
                Npoz = nelinearita (modelMeniceN, model.rizeni->Mpozad);
                if (Npoz < 650*RPM)
```

```
                        Npoz = 650*RPM;
        }

        // NARUST OTACEK DIESELU PODLE POZADOVANYCH (dynamika dieselu)
        Ndiesl += rtr_narust (Ndiesl, Npoz, 500*RPM/SEK, 600*RPM/SEK);
}

static void mod_tra (void)
{
        static U16 tauNeni = 1*SEK;

        Nvst = Ndiesl;

        if (model.rizeni->stupen_pozad != D)
        {
                // nuceny neutral
                stupen  = 0;
                menic_blok = ne;
                Qmenic = 0xFB00;
        }

        // byl neutral (rozjezd, nebo opetovne zarazeni po vyrazeni),
        // ted se bude se radit odpovidajici stupen
        else if (stupen == 0)
        {
                Qmenic = 1000;
                // hledej nejvyssi mozny
                for (stupen = 5; stupen > 0; stupen--)
                {
                        if (Vskut > zmenD [stupen])
                        {
                                // nejaky blokovany stupen
                                menic_blok = ano;
                                break;
                        }
                }
                // rozjezd - menicova jednicka
                if (stupen == 0)
                {
                        stupen = 1;
                        menic_blok = ne;
                        Qmenic = 4000;
                }
        }
        // nebyl neutral, regulerni podrazovani, pokud je treba
        else if (Vskut < zmenD [stupen])
        {
                Qmenic = 1000;
                // podrazeni na blokovane 1..4
                if (stupen > 1)
                {
                        stupen--;
                        menic_blok = ano;
```

```
                }
                // podrazeni na menicovou jednicku
                else
                {
                        stupen = 1;
                        menic_blok = ne;
                        Qmenic = 4000;
                }
        }
        // nebyl neutral, regulerni razeni nahoru, pokud je treba
        else if (Vskut > zmenH [stupen])
        {
                Qmenic = 1000;
                if (stupen < 5)
                        stupen++;
                menic_blok = ano;
        }
        // menicova jednicka -> blokovana jednicka
        else if ((menic_blok == ne) && (Vskut > zmenH [0]))
        {
                Qmenic = 1000;
                menic_blok = ano;
        }

        rozsah = znakPrevStupen [stupen];
        PomerS = radiciPom [stupen];

        // zpetne signaly prevodovky
        if (stupen == 0)
        {
                razeni = ne;
                hridel_spojena = ne;
                tauNeni = 1*SEK;
        }
        else if (Ndiesl != Npoz && menic_blok == ano)
        {
                razeni = ano;
                hridel_spojena = ano;
        }
        else
        {
                razeni = ne;
                hridel_spojena = ano;
        }
    // signaly pro povoleni jizdy
        if (tauNeni > 0 && --tauNeni > 0)
                ready_for_break_releas = ne;
        else
                ready_for_break_releas = ano;
}

static void mod_momenty_agregatu (void)
{
```

```
        // ztratovy model dieselu
245     MdieselZtr = nelinearita (modelZtrat, Ndiesl);

        // momenty dieselu a moment za prevodovkou
        // stoply motor
        // neutral - za prevodovkou nic
250     if (stupen == 0)
        {
                MskutR = 0;
                if (Ndiesl == Npoz)             // rovnovazny stav
                        Mdiesel = MdieselZtr + 1;
255             else if (Ndiesl < Npoz)         // diesel dava moment
                        Mdiesel = 2 * MdieselZtr;
                else                            // diesel je roztacen,nez ze by daval vykon
                        Mdiesel = 0;
        }
260     // menicovy stupen. menic je schopen nasobit moment
        else if (menic_blok == ne)
        {
                Mdiesel = nelinearita (modelMeniceM, Ndiesl);
                if ((Ndiesl > 0) && (Ndiesl > Nprep)) // diesel bude davat moment
265                     MskutR = trojclenkaU(4*(Mdiesel-MdieselZtr), Ndiesl-Nprep, Ndiesl);
                else
                        MskutR = 0;
                if (MskutR < Mdiesel)
                        MskutR = Mdiesel;
270     }
        // blokovany stupen
        else
        {
                Mdiesel = model.rizeni->Mpozad;
275             MskutR = Mdiesel - MdieselZtr;
                if (MskutR < 0)
                        MskutR = MdieselZtr;
        }
}
280
static void mod_ret (void)
{
        if (model.rizeni->Rzap)
                if (model.rizeni->Rpozad < -Rmax)
285                     Mbrzd = -Rmax;
                else if (model.rizeni->Rpozad > 0)
                        Mbrzd = 0;
                else
                        Mbrzd = model.rizeni->Rpozad;
290     else
                Mbrzd = 0;
}

static void mod_hnaciMom ()
295 {
        // moment za prevodovkou
```

```
        Mvysl = trojclenkaI (MskutR + Mbrzd, PomerS, 1000);
        // vliv sklonu
        Mvysl -= sklonTr * 3;
        // druhy podvozek
        if (MASTER)
                Mvysl += MhnaciSlave;
        else
                if (Mvysl > 125)
                        MhnaciSlave = 100;
                else if (Mvysl < -100)
                        MhnaciSlave = -100;
                else
                        MhnaciSlave = (I8) Mvysl;
}

static void mod_dynamika (void)
{
        I16 tempV = Vskut / KMH / 16;
        // zrychleni zpusobovane hnacim momentem
        Amodel = Mvysl * 4;  // 100% ... 1600 Nm ... 12.3 kN ... 0,4 m/s2

        // vozidlovy odpor
        Amodel -= 70 + Vskut/256 + 2*tempV*tempV;

        // vliv hmotnosti
        Amodel *= (0x10000 / hmot);

        if (MASTER)
        {
                // kumulace rychlosti
                Vmodel += Amodel; // dv/dt = a * m
                // osetrene zaporne rychlost a osetreni maximalni rychl.
                if (Vmodel < 0L)
                {
                        Vmodel = 0;
                        Amodel = 0;
                }
                else if (Vmodel > VmaxModel)
                {
                        Vmodel = VmaxModel;
                        Amodel = 0;
                }
                Vskut = Vmodel >> 16;
        }
}

static void mod_ostatni_veliciny (void)
{
        I32 Atemp;
        model.diesel->pOlejM = (Ndiesl > 0) ? 200 : 0;
        model.diesel->MskutNm = trojclenkaU (1600, Mdiesel, 100);

        Atemp = Amodel*SEK;
```

```
        Atemp *= 10;
        Atemp /= 36;
        Askut = Atemp / 0x8000;
}

void mod_model_hard_stop ()
{
        Vskut = 0;
        Vmodel = 0L;
        stupen = 0;
}
```

C.2 Prezentační vrstva CANu

Algoritmus C.2: can_presentation.c

```
#include "can_presentation.h"

#define  JCELSIA                32
#define  JKPAolej               4
#define  JKPAtbo                2
#define  JMAX8                  0xFA
#define  JMAX16                 0xFAFF

#define  J_NA_J_PROC(h)         (125 + h)
#define  J_Z_J_PROC(h)          (h - 125)
#define  J_NA_J_RPM(o)          (o << 3)
#define  J_Z_J_RPM(o)           (o >> 3)
#define  J_NA_0_DOT_4(tmp,c)    (tmp = 0x0000 | c, tmp * 5, tmp / 2)
#define  J_NA_J_CELS8(t)        (40+t)
#define  J_NA_J_CELS16(t)       ((273 + t) * 32)
#define  J_NA_J_KPASC(p)        (p >> 2)


U16 tempU16;
U8 tempU8;


// PGN: 0; SPN: 695, 898, 518
void f_in_TSC1_FM(uCAN_DATA * data)
{
        model.rizeni->regulace = (e1939REGULACE) (data->byte[0] & 0x03);
        tempU16 = (data->byte[1]) + (data->byte[2] << 8);
        model.rizeni->Npozad = (tempU16 > JMAX16 ? 0 : J_Z_J_RPM(tempU16));
        tempU8 = data->byte[3];
        model.rizeni->Mpozad = (tempU8 > JMAX8 ? 0 : J_Z_J_PROC(tempU8));
}
// PGN: neni J1939
```

```
void f_in_TC1_FM(uCAN_DATA * data)
{
        model.rizeni->povelP1 = data->byte[0];
        model.rizeni->stupen_pozad = (eSTUPENp) data->byte[2];
        model.rizeni->rozsahP = data->byte[6];
}
// PGN: neni J1939
void f_in_TSC1_RC(uCAN_DATA * data)
{
        tempU8 = data->byte[0] & 0x33;
        model.rizeni->Rzap = (tempU8 == 0x22 ? TRUE : FALSE);
        tempU8 = data->byte[3];
        model.rizeni->Rpozad = (tempU8 > JMAX8 ? 0 : J_Z_J_PROC(tempU8));
}
// PGN: 65265; SPN: 70, 84, 595, 596, 597, 598, 86
void f_in_CCVS(uCAN_DATA * data)
{
        model.rizeni->park_brzda_spin = (e1939BOOL) (data->byte[0] & 0x0C);
        tempU16 = data->byte[1] + (data->byte[2] << 8);
        model.rizeni->rychlost_ccvs = (tempU16 > JMAX16 ? 0 : tempU16 >> 2);
        model.rizeni->povelP3 = data->byte[3];
        tempU8 = data->byte[5];
        model.rizeni->rychlost_pozadovana = (tempU8 > JMAX8 ? 0 : tempU8);
}
// PGN: 61444; SPN: 899, 512, 513, 190,
void f_out_EEC1(uCAN_DATA * data)
{
        data->byte[0] = model.diesel->rezimM | 0xF0;
        data->byte[1] = J_NA_J_PROC(*model.diesel->Mnast);
        data->byte[2] = J_NA_J_PROC(*model.diesel->MskutA);
        tempU16 = *model.diesel->Nskut << 3;
        data->byte[3] = tempU16 >> 0;
        data->byte[4] = tempU16 >> 8;
        data->byte[5] = 0xFF;
        data->word[3] = 0xFFFF;
}
// PGN: 61443; SPN: 91, 92
void f_out_EEC2(uCAN_DATA * data)
{
        data->word[0] = 0xFF;
        //data->byte[1] = 0xFF;//data->byte[1] = J_NA_0_DOT_4(tempU16, *model.diesel->Mpedal);
        data->byte[2] = *model.diesel->MskutR;
        data->byte[3] = 0xFF;
        data->dword[1] = 0xFFFFFFFF;
}
// PGN: 65247; SPN: 514, 515, 519
void f_out_EEC3(uCAN_DATA * data)
{
        data->byte[0] = J_NA_J_PROC(*model.diesel->Modpor);
        tempU16 = J_NA_J_RPM(*model.diesel->Nnast);
        data->byte[1] = (U8) (tempU16 >> 0);
        data->byte[2] = (U8) (tempU16 >> 8);
        data->byte[3] = 0xFF;//data->byte[3] = *model.diesel->Modber;
```

```
        data->dword[1] = 0xFFFFFFFF;
}
// PGN: 65262; SPN: 110, 175
void f_out_engTmp(uCAN_DATA * data)
{
        data->byte[0] = J_NA_J_CELS8(model.diesel->tVodyM);
        data->byte[1] = 0xFF;
        data->word[1] = J_NA_J_CELS16(model.diesel->tOlejM);
        data->dword[1] = 0xFFFFFFFF;
}
// PGN: 65263; SPN: 100
void f_out_engFlu(uCAN_DATA * data)
{
        data->word[0] = 0xFFFF;
        data->byte[2] = 0xFF;
        data->byte[3] = J_NA_J_KPASC(model.diesel->pOlejM);
        data->dword[1] = 0xFFFFFFFF;
}
// PGN: 65270; SPN: 102, 105, 173
void f_out_engTbo(uCAN_DATA * data)
{
        data->byte[0] = 0xFF;
        data->byte[1] = model.diesel->pTurbo >> 1;
        data->byte[2] = J_NA_J_CELS8(model.diesel->tTurbo);
        data->byte[3] = 0xFF;
        data->byte[4] = 0xFF;
        tempU16 = J_NA_J_CELS16(model.diesel->tVyfuk);
        data->byte[5] = tempU16 >> 0;
        data->byte[6] = tempU16 >> 8;
        data->byte[7] = 0xFF;
}
// PGN: 65226; SPN: priloha Diagnostic, DM1
void f_out_engDM1(uCAN_DATA * data)
{
        data->byte[0] = model.poruchy->diesel;
        data->byte[1] = 0xFF;
        data->word[1] = 0xFFFF;
        data->dword[1] = 0xFFFFFFFF;
}
// PGN: 61442; SPN: 560, 573, 574, 191, 161
void f_out_ETC1(uCAN_DATA * data)
{
        tempU8 = 0xC0;
        tempU8 |= (U8) (*model.prevodovka->hridel_spojena << 0);
        tempU8 |= (U8) (*model.prevodovka->menic_spojen << 2);
        tempU8 |= (U8) (*model.prevodovka->razeni_probiha << 4);
        data->byte[0] = tempU8;
        tempU16 = J_NA_J_RPM(*model.prevodovka->Nvystup);
        data->byte[1] = tempU16 >> 0;
        data->byte[2] = tempU16 >> 8;
        data->byte[3] = 0xFF;
        data->byte[4] = 0xFF;
        tempU16 = J_NA_J_RPM(*model.diesel->Nskut);
```

```
        data->byte[5] = tempU16 >> 0;
        data->byte[6] = tempU16 >> 8;
        data->byte[7] = 0xFF;
}
// PGN: 61443; SPN: 524, 526, 162
void f_out_ETC2(uCAN_DATA * data)
{
        data->byte[0] = J_NA_J_PROC(*model.prevodovka->stupenN);
        tempU16 = *model.prevodovka->pomerS;
        data->byte[1] = tempU16 >> 00;
        data->byte[2] = tempU16 >> 8;
        data->byte[3] = J_NA_J_PROC(*model.prevodovka->stupenS);
        tempU16 = ' ' << 8;
        tempU16 |= *model.prevodovka->rozsahN;
        data->word[2] = tempU16;
        tempU16 = ' ' << 8;
        tempU16 |= *model.prevodovka->rozsahS;
        data->word[3] = tempU16;
}
// PGN: 65098; SPN: 1850, 1849, 3086, 2945
void f_out_ETC7(uCAN_DATA * data)
{
        data->byte[0] = 0xFF;
        tempU8 = 0x00;
        tempU8 |= (U8) (*model.prevodovka->ready_for_break_release << 0);
        tempU8 |= (U8) (*model.prevodovka->console_ind << 2);
        tempU8 |= (U8) (*model.prevodovka->engine_crank_enable << 4);
        tempU8 |= (U8) (*model.prevodovka->shift_inhibit_ind_active << 6);
        data->byte[1] = tempU8;
        data->word[1] = 0xFFFF;
        data->dword[1] = 0xFFFFFFFF;
}
// PGN: 61452; SPN: 3030
void f_out_ETC8(uCAN_DATA * data)
{
        data->word[0] = *model.prevodovka->Qmenic;
        data->word[1] = 0xFFFF;
        data->dword[1] = 0xFFFFFFFF;
}
// PGN: 65272; SPN: 124, 177, 3027
void f_out_traFlu(uCAN_DATA * data)
{
        data->byte[0] = 0xFF;
        data->byte[1] = J_NA_0_DOT_4(tempU8, model.prevodovka->hladina_oleje);
        data->word[1] = 0xFFFF;
        data->word[2] = J_NA_J_CELS16(model.prevodovka->TolejP);
        data->byte[6] = 0;
        data->byte[7] = 0xFF;
}
// PGN: neni J1939
void f_out_TSC1_TC(uCAN_DATA * data)
{
        data->byte[0] = (U8) model.prevodovka->tsc_tc;
```

```
        tempU16 = J_NA_J_RPM(model.prevodovka->tsc_tc_otacky * 8);
        data->byte[1] = tempU16 >> 0;
        data->byte[2] = tempU16 >> 8;
        data->byte[3] = J_NA_J_PROC(model.prevodovka->tsc_tc_moment);
        data->dword[1] = 0xFFFFFFFF;
}
// PGN: 65226; SPN: priloha Diagnostic, DM1
void f_out_traDM1(uCAN_DATA * data)
{
        data->byte[0] = (U8) (model.poruchy->prevod);
        data->byte[1] = 0xFF;
        data->word[1] = 0xFFFF;
        data->dword[1] = 0xFFFFFFFF;
}
// PGN: 61440; SPN: 900, 520, 1085, 1480, 1717
void f_out_ERC1(uCAN_DATA * data)
{
        data->byte[0] = *model.retarder->rezimR;
        data->byte[1] = J_NA_J_PROC(*model.retarder->Rzpet);
        data->byte[2] = J_NA_J_PROC(model.rizeni->Rpozad);
        data->byte[3] = 0xFF;
        data->byte[4] = model.retarder->ovladR;
        data->byte[5] = J_NA_J_PROC(*model.retarder->Rzpet);
        data->byte[6] = 0xFF;
        data->byte[7] = J_NA_J_PROC(-(*model.retarder->Rmax));
}
// PGN: 65275; SPN: 120
void f_out_retFlu(uCAN_DATA * data)
{
        data->byte[0] = 0xFF;
        data->byte[1] = J_NA_J_CELS8(model.retarder->tOlejR);
        data->word[1] = 0xFFFF;
        data->dword[1] = 0xFFFFFFFF;
}
// PGN: 65226; SPN: priloha Diagnostic, DM1
void f_out_retDM1(uCAN_DATA * data)
{
        data->byte[0] = (U8) (model.poruchy->ret);
        data->byte[1] = 0xFF;
        data->word[1] = 0xFFFF;
        data->dword[1] = 0xFFFFFFFF;
}


// RETARDEROVA MULTIZPRAVA
void f_out_retMulti  (sCAN_MULTI_DATA * multiB)
{
        multiB->data[0] = model.retarder->konfTypR;
        multiB->data[1] = model.retarder->konfRizeniR;
        multiB->data[16] = model.retarder->konfRref;
        multiB->data[17] = model.retarder->konfRref >> 8;
        multiB->pocZn = 18;
}
```

C.3 Aplikační vrstva CANu – ukázka definice CAN rámců

Algoritmus C.3: can_aplication.c

```
sCAN_INT_DEF model842_defCAN0 [] =
{
// RAMCE OD RIZENI
{&can0TSC1_FM,PIS_ID(0x0C000027<<3),PIS_ID(0xFFFFFFF8),8,T_RIZ,
    0x0001, 1*SEK,          0,"TSC1",f_in_TSC1_FM, NULL },
{&can0TC1_FM, PIS_ID(0x0C010327<<3),PIS_ID(0xFFFFFFF8),8,T_RIZ,
    0x0002, 1*SEK,          0,"TC1 ",f_in_TC1_FM,  NULL },
{&can0TSC1_RC,PIS_ID(0x0C001027<<3),PIS_ID(0xFFFFFFF8),8,T_RIZ,
    0x0004, 1*SEK,          0,"rTSC",f_in_TSC1_RC, NULL },
{&can0EEC3_FM,PIS_ID(0x18FEDF27<<3),PIS_ID(0xFFFFFFF8),8,T_RIZ,
    0x0008, 5*SEK,          0,"EE3 ",NULL,         NULL },
{&can0CCVS,   PIS_ID(0x18FEF127<<3),PIS_ID(0xFFFFFFF8),8,T_RIZ,
    0x0010, 5*SEK,          0,"CCVS",f_in_CCVS,    NULL },
// RAMCE ENG
{&can0EEC1,   PIS_ID(0x0CF00400<<3),PIS_ID(0xFFFFFFF8),8,T_ENG,
    0x8000, SEK/10, 10*MILS,"EEC1",NULL,         f_out_EEC1 },
{&can0EEC3,   PIS_ID(0x18FEDF00<<3),PIS_ID(0xFFFFFFF8),8,T_ENG,
    0,        SEK, 250*MILS,"EEC3",NULL,         f_out_EEC3 },

const int pocCan0 = sizeof(model842_defCAN0) / sizeof(sCAN_INT_DEF);
```

C.4 Aplikační vrstva CANu – definice prodloužených zpráv

Algoritmus C.4: can_aplication.c

```
sCAN_INT_MULTI_DEF model842_multi_defCAN0 [] =
{
{&RetMultiB, &model842_defCAN0[28], &model842_defCAN0[27], 0x0000FEE1, 3, T_RET,
    0, 5*SEK, NULL, f_out_retMulti },
};
const int pocMulti = sizeof(model842_multi_defCAN0) / sizeof(sCAN_INT_MULTI_DEF);
```

Příloha D

Obsah přiloženého CD

■ CD	Kořenový adresář CD.
⊞ `model842`	Adresář se zdrojovým kódem SW modelu HAV na HW MSV
⊞ `obrazky`	Adresář s obrázky
⊞ `prace-latex`	Adresář se zdrojovými soubory tohoto textu