FAKULTA
APLIKOVANÝCH VĚD
ZÁPADOČESKÉ
UNIVERZITY
V PLZNI

Modelově orientovaný vývoj softwaru: řízení spojky automatické převodovky kamionů

Vedoucí: Ing. Martin Čech, Ph.D.
Konzultant: Ing. Jan Škach, Ph.D.
Vypracoval: Bc. Karel Kubíček - A17N0014P

Titulní list

Zadání

- **1.** Analyzujte a shrňte principy vývoje softwaru v automobilovém průmyslu včetně stručné historie a používaných bezpečnostních standardů
- **2.** Porovnejte výhody a nevýhody ručně psaného a automaticky generovaného kódu
- **3.** Popište regulátor spojky automaticky generované převodovky z pohledu dynamických systému a zpětnovazebního řízení
- **4.** Popište proces tvorby blokově orientovaného kódu řídicího systému a porovnejte jej s ručně psaným kódem
- **5.** Pro otestování řídicího systému vytvořte vhodné MIL, SIL a HIL struktury
- **6.** Analyzujte, případně zoptimalizujte parametry PID regulátorů, které jsou součástí řídicího systému

Assignment

- **1.** Analyze and summarize the principles of software development in the automotive industry, including history and safety standards
- **2.** Compare the advantages and disadvantages of hand-written code and automatically generated code
- **3.** Describe the controller of the clutch in the automatically generated gearbox system from the dynamic feedback control point of view
- **4.** Describe the creation process of a block oriented code of the control system and compare it with hand-written code
- **5.** Create a suitable MIL, SIL and HIL structures for control system testing
- **6.** Analyze and optimize, if possible, the parametrs of the PID controllers that are part of the control system

Čestné prohlášení

Předkládám tímto k posouzení a obhajobě diplomovou práci zpracovanou na závěr studia na Fakultě aplikovaných věd Západočeské univerzity v Plzni.

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím odborné literatury a pramenů, jejichž úplný seznam je její součástí.

V Plzni dne .....

Anotace

Cílem této práce je demonstrování postupu vývoje softwaru v automobilovém průmyslu při použití automatického generování kódu z modelu v Simulinku. V této práci bude nastíněn vývoj celého odvětví a rostoucí význam a objem softwaru v automobilového průmyslu, včetně uvedení nejdůležitějších norem a standardů. Dále jsou v práci popsány a vzájemně porovnány dva způsoby vývoje softwaru: klasický přístup ručně psaného kódu a modelově orientovaný přístup. Proces modelově orientovaného vývoje softwaru generování kódu bude demonstrován na modelu regulátoru spojky automatické převodovky a to včetně fyzikálního modelu. Práce popisuje, jak upravit model, a která pravidla dodržovat, aby bylo možné pomocí vhodných nástrojů automaticky vygenerovat kód z modelu. Dále se v diplomové práci pojednává o integraci vygenerovaného kódu do ručně psaného kódu. Závěr práce je věnován popisu testování, validace a vyhodnocení výsledků, kterých celý systém s integrovaným automaticky vygenerovaným kódem dosáhl.

Klíčová slova: automatická spojka, automobilový průmysl, dSPACE, generování kódu, HIL, Matlab, MBSD, MIL, SIL, Simulink, software, TargetLink

Abstract

The aim of this work is to demonstrate the development of software in the automotive industry. This software will be based on automated code generation from the model in Simulink. It will be describe how it differs from the classical programming approach and what benefits it has. This work will describe the development of the entire industry and the growing importance and volume of software in the automotive industry, including the introduction of the most important standards and norms, which are used. The code generation process will be demonstrated on the clutch controller of the automatic transmission model including the physical model. It will be describe how to modify the model and wich rules have to been followed to make it possible to use suitible tools for automatic code generation from the model. In the next step the generated code will be integrated into handwritten code. The conclusion of this work will deal with the test strategy, validation and verification of the results achieved by the entire system with the integrated automatically generated code.

Key words: automatic transsmision, automotiv industry, code generation, dSPACE, HIL, Matlab, MBSD, MIL, SIL, Simulink, software, TargetLink

Poděkování

Rád bych zde poděkoval vedoucímu mé diplomové práce, panu Ing. Martinovi Čechovi, Ph.D. za odborné rady, skvělý přístup, čas, ochotnou pomoc a možnost vyzkoušet si napsat vědecký článek. Dále bych chtěl poděkovat svému konzultantovi a kolegovi, panu Ing. Janu Škachovi, Ph.D. za výbornou spolupráci, za to jak mi s celou prací pomáhal, za věnovaný čas a za to, že mi předával své znalosti a zkušenosti z akademické půdy. Chtěl bych poděkovat i všem svým plzeňským kolegům, zejména pánům Ing. Janovi Heringovi, Ing. Martinovi Fajfrovi a vedoucímu naší sekce Ing. Liborovi Jelínkovi, díky kterým jsem mohl tuto práci z firemního prostředí napsat. Následně bych chtěl poděkovat svým německým kolegům, pánům Ing. Timo-Bastian Konkolovi, Ing. Wilhelm Moserovi a Ing. Manuel Schneiderovi, za odborné rady, materiály, vstřícný přístup a především věnovaný čas. Na závěr bych chtěl poděkovat své rodině, která mě při studiu vždy podporovala.

Obsah

1 Úvod **1**
1.1 Obecný úvod . . . 1
1.2 Motivace . . . 2

2 Vývoj SW v automobilovém průmyslu **4**
2.1 Historie automobilového průmyslu . . . 4
2.1.1 První automobil . . . 6
2.1.2 Průběh vývoje a důležité milníky v historii . . . 7
2.1.3 Budoucnost a elektromobilita . . . 9
2.2 Normy a bezpečnost v automobilovém průmyslu . . . 11
2.2.1 Automotiv SPICE . . . 12
2.2.2 ISO 26262 . . . 12
2.2.3 Interní normy a zásady v rámci ZF . . . 12
2.3 Principy vývoje SW . . . 13
2.3.1 Blokově orientovaný návrh systémů . . . 15
2.3.2 Model Based Design v rámci ZF . . . 16
2.3.3 Ručně psaný kód . . . 17
2.3.4 Automaticky generovaný kód . . . 18
2.4 Manuální a automatická převodovka . . . 19
2.4.1 Manuální převodovka . . . 20
2.4.2 Automatická převodovka . . . 21
2.4.3 Převodovka EL40 TraXon . . . 22
2.4.4 Převodovka EL57 OptiDrive Ecomid . . . 23

3 Systém spojky automatické převodovky **24**
3.1 Úvod . . . 24
3.2 Popis systému . . . 25
3.2.1 Mechanická část . . . 26
3.2.2 Pneumatická část . . . 27
3.3 Stavový popis . . . 32
3.4 Řízení spojky automatické převodovky . . . 33
3.4.1 Stav vypnuto . . . 34
3.4.2 Časový mód . . . 34
3.4.3 Mód řízení polohy . . . 34
3.4.3.1 Regulátor pro malé změny . . . 37
3.4.3.2 Regulátor pro velké změny . . . 38
3.4.3.3 Oscilace spojky - aktivní tlumení vibrací . . . 40
3.4.3.4 Řešení problémů při návrhu . . . 40

4 Proces tvorby automaticky generovaného kódu **41**
4.1 Úvod . . . 41
4.1.1 Používané programové nástroje . . . 41
4.1.2 TargetLink . . . 42
4.2 Transformace modelu regulátoru spojky pro automatické generování kódu . 44

4.2.1 Úprava modelu . . . 44
4.2.2 Požadavky úspěšného generování kódu . . . 47
4.2.3 Vygenerování autokódu . . . 48
4.2.4 Statická kontrola modelu . . . 48
4.3 Integrace generovaného kódu . . . 49
4.3.1 Označení částí ručně psaného kódu . . . 50
4.3.2 Nahrazení označených částí za automaticky generované . . . 50
4.4 Porovnání softwaru před a po transformaci . . . 50
4.4.1 Příklad generování kódu . . . 51
4.5 Zásady pro tvorbu efektivnějšího modelu . . . 52
4.5.1 Ošetření dělení nulou . . . 53
4.5.2 Vymazání nepoužitých signálů a bloků . . . 53
4.5.3 Správné nastavení datových typů . . . 53
4.5.4 Nastavení parametrů v databázi . . . 54
4.5.5 Definice a volání konstant a funkcí . . . 54
4.5.6 Upravení funkcionalit do nové podoby . . . 54
4.6 Continuous engineering . . . 54

5 Testování **58**
5.1 SIL testovaní . . . 59
5.2 MIL testovaní . . . 62
5.3 PIL testování . . . 63
5.4 HIL testovaní . . . 63
5.4.1 Manuální HIL testy . . . 63
5.4.2 Automatické HIL testy . . . 64
5.5 Back-to-back testování . . . 66
5.5.1 Noční automatické testy . . . 66
5.6 Testování na reálném zařízení . . . 67
5.7 Porovnání výpočetní náročnosti - CPU load . . . 68
5.8 Analýza a optimalizace regulátorů . . . 72
5.8.1 Další pokročilé metody regulace . . . 73
5.9 Aktuální chování regulátoru spojky . . . 75

6 Závěr - zhodnocení celé transformace **76**

Seznam obrázků **79**

Seznam tabulek **80**

Seznam literatury **81**

7 Přílohy **86**

1 Úvod

1.1 Obecný úvod

Hlavním tématem celé práce je vývoj softwaru (SW) pro regulátor spojky automatické převodovky pomocí metody automatického generování kódu. Celá tato práce vznikla ve spolupráci s firmou **ZF Engineering Plzeň, s.r.o**, která je pobočkou Německé firmy ZF Friedrichshafen AG (ZF) se sídlem ve Friedrichshafenu.

Obrázek 1: Logo ZF

Jedná se o firmu zabývající se výrobou a vývojem převodových skříní a automobilových komponent a to již od 20. srpna 1915, kdy byla založena Ferdinandem von Zeppelinem. Ten v té době stál za vývojem a výrobou vzducholodí. A právě do nich potřeboval převodovky, kterých neměl dostatek, a tak se rozhodl, že si je bude vyrábět sám. Od té doby ušla celá firma dlouhou cestu a nyní se pohybuje hlavně v oblasti automobilového průmyslu a průmyslovému využití převodových systémů. Patří také k jednomu z největších dodavatelů komponent pro výrobu automobilů na světě.

V první části této práce bude popsána motivace z jakého důvodu se objevilo a jaké benefity poskytuje automatické generování kódu.

Kapitola 2 je věnována vývoji SW v automobilovém průmyslu. Bude zde popsána historie vývoje SW a elektrotechniky na subjektivně vybraných milnících z historie automobilového průmyslu. Dále zde bude demonstrováno rostoucí množství a význam SW, rostoucí význam elektromobilů a zmíněny zde budou i autonomní automobily. Významnou roli v automobilovém průmyslu hrají standardy a normy [23, 24]. S tímto je výrazně spjata bezpečnost a nutnost splnění schválených norem [14]. Budou zde tedy stručně popsány ty nejdůležitější. Budou zde popsány rozdíly mezi klasickým vývojem SW jako ručně psaným kódem a pomocí **automatického generování kódu**. Jedna z podkapitol se také bude zabývat historií využívání „Model-Based Software Design“ (MBSD) metod v rámci ZF. Také zde bude popsán princip funkce převodovky v automobilu, budou zde uvedeny rozdíly, výhody a také nevýhody automatické, ale i manuální převodovky.

Kapitola 3 je věnována popisu komponenty regulátoru spojky. Bude zde popsán matematický model soustavy pomocí diferenciálních rovnic a také pomocí stavové reprezentace. Další část se bude věnovat popisu použitých principů při řízení celého systému. Tato část bude popsána pouze teoreticky a to z důvodu dodržení firemního tajemství.

Kapitola 4 je zaměřena na popis transformace modelu do podoby, kdy je z něj možné generovat kód. V této části bude také představen prostředek pro generování kódu a to TargetLink [3, 12, 35]. Následující podkapitola je pak věnována integraci získaného kódu do ručně psaného a kompilace výsledného SW. Jedna z podkapitola kapitola bude věnována optimalizaci modelu. Budou zde popsány body, které je vhodné při transformaci do autokódu dodržet. Při dodržení popsaných bodů bude výsledný vygenerovaný kód méně výpočetně náročný, bude tedy efektivnější. Bude zde také popsán rostoucí význam agilního přístupu a rostoucí význam tzv. „Continuous engineeringu“ [18, 33, 39, 46].

V kapitole 5 je přiblížena významná oblast vývoje softwaru - testování. Budou zde popsány nástroje používané pro testování v rámci ZF. Zaměřeno bude na Software in the loop (SIL), Model in the loop (MIL), Processor in the loop (PIL) a Hardware in the loop (HIL) testování [4, 13, 17, 22, 53, 56]. Je zde také obsažena zmínka o důležitosti testování SW v automobilu, tedy v reálném zařízení a provozu. Následující podkapitola se pak věnuje analýza výpočetní náročnosti mezi verzemi SW. Jedna verze SW je reprezentována ručně psaným kódem před transformací a druhá verze SW již obsahuje zaintegrovaný automaticky vygenerovaný kód. Budou zde také analyzovány a popsány jednotlivé parametry regulátoru a budou zde popsány výsledky spolupráce s Univerzitou v německém Rostocku.

Kapitola 6 shrnuje poznatky celé diplomové práce. Důraz je kladen na zhodnocení transformace modelu do podoby schopné automaticky generovat kód. Zároveň je v této kapitole stručně přiblížena budoucnost vývoje softwaru v automobilovém průmyslu.

1.2 Motivace

Hlavním tématem této práce je **generování, tedy získávání, kódu z modelu vytvořeného v Matlabu, Simulinku** a jeho použití [22, 38]. Kód bude generován z modelu regulátoru spojky automatické převodovky. Ten se skládá z několika částí: regulátoru, stavového automatu, ventilů a spojky. Jistě Vás napadá otázka, zda se vůbec z časového a finančního hlediska vyplatí, udržovat vedle sebe, jak ručně psaný kód, tak model v Simulinku. Dříve tomu bylo tak, že model sloužil jako reference a podle něj se kontrolovala funkčnost a zachování stejné funkcionality v ručně psaném kódu. To je však někdy zdlouhavé a ne vždy efektivní. Je důležité si uvědomit o jak složitý SW se jedná. Výsledný kód celého systému obsahuje desítky komponent a má miliony řádek kódu a je rozdělen do velkého strukturovaného celku.

Do procesu vývoje SW je zapojena řada lidí [2]. Je tedy potřeba, aby někdo nejprve specifikoval, jaká funkcionalita se bude vytvářet nebo předělávat. Dále popsal její funkce, co s čím bude komunikovat, a zasadil ji do celkového konceptu. V případě, že se jedná pouze o úpravu / update již vytvořené funkce, tak je nutno specifikovat její změnu. Tyto požadavky většinou přichází od zákazníka. Tento případ označujeme jako „Change request“. Na základě těchto informací vytvoří funkční vývojář model v Matlabu v Simulinku a nebo

jen upraví již existující model. Ten slouží dále jako podklad pro SW vývojáře, který na základě modelu v Matlabu v Simulinku realizuje implementaci do zdrojového souboru v C++ nebo C. Posledním krokem je testování [1]. Zde se musí zaručit, že vytvořený SW se bude chovat stejně jako model. Tedy, že funkcionalita zůstane zachována. Pohybujeme se v automobilovém průmyslu, kde hraje testování velice důležitou roli, protože v případě nehody jsou ohroženy lidské životy. Proto se dělá celá škála nezávislých testů a revizí. Jedná se například o SIL testy, MIL testy, HIL testy nebo validace v automobilu na testovacím polygonu a další [22, 43, 50, 53, 60]. Pokud SW projde celým procesem testování, je zařazen do sériové produkce a nahrán na hlavní větev verzovacího systému [28, 31].

Cílem generování kódu z modelu je celý tento proces výrazně zrychlit a tím ušetřit peníze a čas potřebný na vývoj nebo úpravu daných částí SW [58]. To znamená zjednodušení celého procesu, kdy funkční vývojář převezme danou specifikaci, na základě které sestaví nebo upraví model v Matlabu, Simulinku. Upraví ho do podoby vhodné pro generování kódu. Automaticky vygenerovaný kód následně zaintegruje do ručně psaného kódu. Dojdetedy k nahrazení celých funkcí voláním funkcí obsažených v automatiky generovaném kódu. [16, 22, 38]

Odpadá tedy nutnost znovu programovat v C++ to, co jsme již sestavili v Matlabu, Simulinku. Funkcionalita musí být zachována stejná. Psaní testů je také mnohem rychlejší, protože testy jsou psány tím samým vývojářem, který má díky tomuto postupu výrazně větší a hlubší znalosti o dané komponentě a částech SW. Pokud se při dalších testech objeví chyba, reportují se tyto nálezy opět stejnému vývojáří. Pro toho je díky tomuto přístupu mnohem snazší odhalit chybu a opravit ji, než když je do procesu zapojeno více lidí. Takto získaný SW musí splnit úplně stejné testy jako ten, který je psaný ručně. Pokud vše proběhne hladce a funkcionalita zůstala nezměněná, je výsledný SW zařazen do sériové produkce a je opět nahrán na hlavní větev verzovacího systému [28].

Jaké jsou tedy hlavní výhody automatického generování kódu?

- Ušetření peněz potřebných pro vývoj
- Zkrácení doby potřebné pro vývoj nové komponenty
- Ušetření lidských zdrojů
- Vývojář, který takto postupuje získává hlubší znalosti rychleji
- Celý proces je opakovatelný pro libovolnou část SW nebo modelu

První transformace do autokódu zabere nejvíce času, ale budoucí transformace budou již výrazně rychlejší, protože vývojář bude mít už celou řadu zkušeností s danou problematikou. Díky nabytým zkušenostem bude vědět, jak postupovat efektivněji.

2 Vývoj SW v automobilovém průmyslu

Při pohledu na vývoj automobilů a celého automobilového průmyslu je jasné, že SW a použitá elektronika hraje čím dál tím důležitější roli [31]. V posledních letech došlo k urychlení celého vývoje než tomu bylo v letech minulých a to zejména díky rozvoji mikroprocesorových počítačů a jejich rostoucí dostupnosti [2]. S tím také souvisí bezpečnost a požadavky na testování [14, 52]. Automobilový průmysl pracuje s lidskými životy a je nepřípustné, aby například ve 130 kilometrové rychlosti došlo k selhání brzdového nebo jiných systému. Proto jsou v automobilovém průmyslu zavedeny bezpečnostní normy a standardy, které jsou blíže popsány v podkapitole 2.2 [52]. Tématem této práce není analýza a podrobný popis norem, proto budou uvedeny pouze ty nejdůležitější, a to pouze informativně.

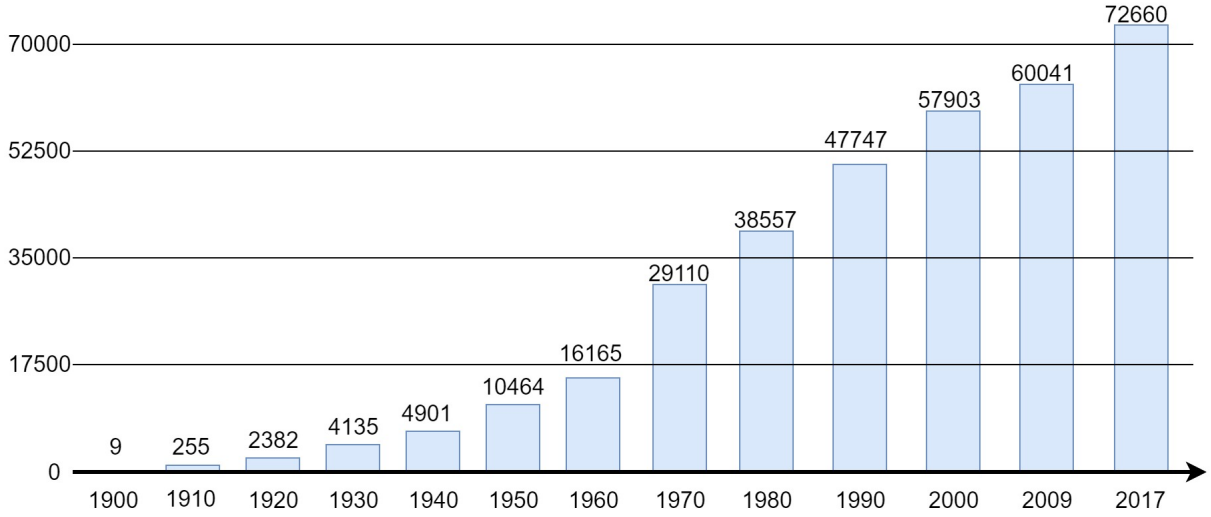
Nejprve v této kapitole bude uveden stručný vývoj automobilového průmyslu. Půjde pouze o stručný úvod a názornost, aby měl čtenář představu, kdy tento vývoj začal a kam směřuje dnes. Budou zde zmíněny důležité milníky ve vývoji elektrotechniky a SW použitého v automobilovém průmyslu. Tyto milníky byly vybrány subjektivně a jejich úkolem je pouze dokumentovat průběh vývoje. Dále zde bude uveden první automobil se spalovacím motorem a dán do kontrastu s novodobým automobilem. Budou zde popsány přístupy při vývoji SW a porovnán přístup s ručně psaným kódem s přístupem založeným na MBSD [16, 51]. V závěru je popsána manuální a automatická převodovka.

2.1 Historie automobilového průmyslu

V této kapitole bude demonstrován vývoj automobilů. Nepůjde o porovnávání trendů rok za rokem, ale pouze o názornou ukázku vývoje důležitých elektrotechnických součástí a SW, které dnes považujeme za zcela běžné. V dnešní době mají běžné automobily v průměru kolem padesáti řídicích jednotek. Luxusnější a vybavenější vozy i mnohem více. Tento trend je daný vývojem v elektrotechnice, kdy se stávají výkonnější čipy stále dostupnějšími a levnějšími. S tím také souvisí pokrok na poli materiálového inženýrství, kdy skoro každá nová generace automobilů používá nové a lepší materiály, například na výrobu karoserie, které zajišťují větší pevnost a tuhost karoserie. Dále materiály použité pro výrobu pneumatik, které zajišťují lepší jízdní vlastnosti na mokru, sněhu i suchu. Kvalitnější materiály použité při výrobě interiéru. Kvalitnější díly podvozku, které zaručují lepší pokrytí nerovností vozovky a další. V dnešní době se rozvíjí také tzv. adaptivní tlumiče, které jsou schopny „poznat“ a vyhodnotit, že automobil vjel do nerovnosti, a tak podle toho přitvrdit a nebo změkčit zadní tlumiče a tím odfiltrovat nerovnost. A podobných příkladů je celá řada [31].

V dnešní době hraje důležitou roli téma znečišťování ovzduší, tedy emise. S tím také souvisí problém spotřeby pohonných hmot, kdy se čím dál tím více přihlíží na spotřebu paliva-zvláště v nákladní dopravě. I zde hraje roli použitá převodovka. Například ZF převodovka EL40 TraXon je schopna ušetřit během cesty řádově jednotky procent na celkové spotřebě. V dnešní době je v platnosti v Evropě emisní norma Euro 6. Jedná se o velmi přísnou normu, která žene automobilky a pohonné jednotky na hranici fyzikálních limitů. S tímto tématem také souvisí nedávna aféra „Dieselgate“, kdy automobilky instalovali do svých vozidel SW, který byl schopný detekovat a rozeznat, že se automobil nachází

na měření emisí. Což mělo za následek úpravu palivové mapy a celkové nastavení řídicích jednotek tak, aby byly splněny požadované normy. Z inženýrského hlediska se jedná o naprosto brilantní řešení, avšak z pohledu zákazníka a právního hlediska se jednalo o nevhodnou cestu, za kterou automobilky byly nuceny zaplatit nemalé pokuty. V dnešní době se rozmáhá vývoj a produkce elektrických automobilů. U nich je však stále problém maximální dojezd a použité baterie pro akumulaci elektrické energie.



Obrázek 2: Počet vyrobených automobilů v jednotlivých obdobích v tisících

Na obrázku 2 je pro představu znázorněno, jak se během let ve světě vyvíjel počet vyrobených automobilů [15, 31]. Z něj je jasně patrné, o jak obrovskou část průmyslu jde. Z obrázku je patrné zpomalení rozvoje mezi lety 2000 a 2009. K tomuto jevu mohlo dojít vlivem nasycení trhu, tedy že lidé neměli potřebu si pořizovat nové automobily. Pro udržení trendu, růstu objemu produkce, tak bylo třeba nalézt a vytvořit nové inovační možnosti. V posledních letech se jedná zejména o hybridní automobily, elektromobily a v budoucnu o autonomní vozidla [7] V roce 2017 bylo již celosvětově vyrobeno téměř 73 milionů automobilů [15].

S vývojem SW také souvisí čím dál větší nasazení různých asistenčních a pomocných systémů. Jedná se například o asistent parkování, asistent jízdy v pruhu, asistent couvání nebo parkování, automatické brždění, asistent rozjezdů, hlídání mrtvých úhlů, adaptivní tempomat, adaptivní světlomety, které se natáčí ve směru zatáčení a hlídají oslnění protijedoucích aut a mnoho dalších. V dnešní době jsou také žhavým tématem samořiditelné (autonomní) automobily, které využívají různé senzory a kamery, díky nimž jsou schopné se orientovat a reagovat na situace v okolí [7, 8]. Něco takového bylo umožněno obrovským rozvojem na poli umělé inteligence a to zejména počítačového vidění, vyhodnocování odhadů a nasazení neuronových sítí. V dnešní době se o vývoj vlastního autonomního automobilu snaží mnoho velkých firem a SW gigantů. První prototypy se již testují ve skutečném provozu, ale pro nasazení ve větším počtu budou muset urazit ještě dlouhý kus cesty a bude potřeba vybudovat nutnou infrastrukturu, aby vše fungovalo správně. S tím také souvisí potřebná aktualizace právního systému, který na tuto problematiku ještě není připravený [8]. Mezi jedny z nejzásadnější otázek patří etická otázka. Například kolize auta s jiným na přechodu pro chodce za současného pohybu osob na přechodu. Samořiditelný

automobil by se tak musel „rozhodnout“, zda naboural například do druhého auta nebo do domu a nebo zda srazí lidi na přechodu. Je jasné, že lidé na přechodu budou mít daleko nižší šanci na přežití, než posádka automobilu. Majitel auta bude předpokládat, že když si takové auto pořídil, tak ho bude „chránit“. To je na jedné stranně logické, ale co je eticky a právně správně, stále není definováno a pro nasazení autonomních automobilů je nezbytné, aby se tyto právní věci vyřešily a jasně definovaly [8, 31].

2.1.1 První automobil

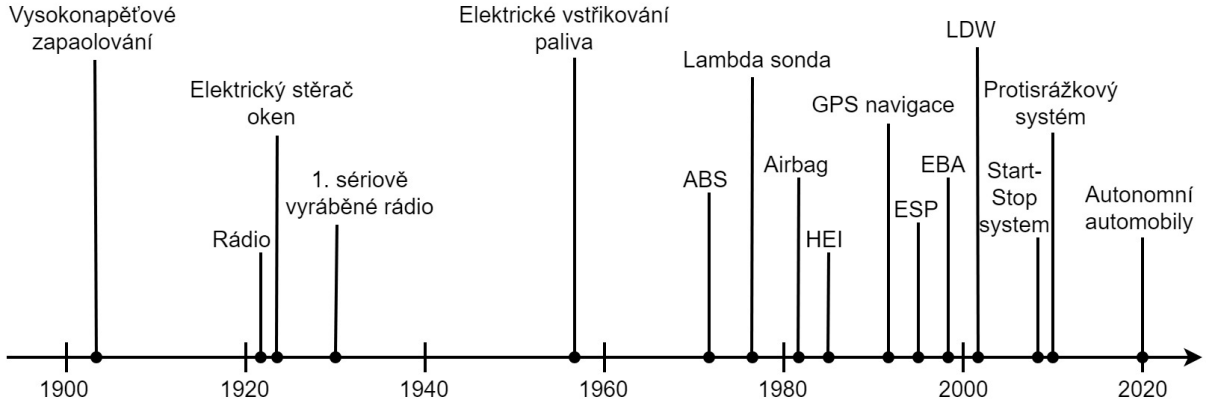
První automobil sestavený se spalovacím motorem jako pohonnou jednotkou má na svědomí automobilový konstruktér Karl Benz. Stalo se tak v roce 1885, kdy byl sestrojen první takovýto automobil. Byl pojmenován Benz Patent-Motorwagen No. 1, který je znázorněn na obrázku 3 v levé části. Jednalo se o dvousedadlový automobil, který byl poháněn jednoválcovým čtyřtaktním motorem o objemu 954 kubických centimetrů. Ten byl uložen horizontálně vzadu v trubkovém ocelovém rámu. Výkon byl na dvě drátěná kola přenášen pomocí diferenciálu. Výkon motoru byl 0,75 koně, což odpovídá 0,55 kW. Toto vozidlo obsahovalo automatický přívod paliva do spalovací komory, řízený výfukový ventil, vodní chlazení a o zapalování směsi se starala zapalovací svíčka, která dávala jiskru [10, 31].

Obrázek 3: Benz Patent-Motorwagen No. 1 a Mercedes třídy S rok 2018 [47, 63]

V roce 1888 absolvovala manželka Karla Benze Bertha Benzová s automobilem Benz Patent-Motorwagen No. 3 první dálkovou cestu. Jednalo se o zhruba 180 kilometrů dlouhou trasu, kterou Bertha úspěšně absolvovala bez svého manžela. Tato cesta jí zabrala tři dny. Benz Patent-Motorwagen No. 3 byl vylepšením původního typu. Měl silnější motor, který dosahoval výkonu 1,5 kW, což mu umožňovalo dosáhnout rychlosti až 16 kilometrů za hodinu [10]. Pokud tento automobil porovnáme na obrázku 3, pravou částí, na kterém je Mercedes třídy S z roku 2018. Ten je vybaven dvanáctiválcovým motorem o objemu 6 litrů a „přetéká“ různými asistenty a luxusní výbavou, tak je naprosto zřejmé, jaký obrovský pokrok automobilový průmysl udělal.

2.1.2 Průběh vývoje a důležité milníky v historii

Již bylo uvedeno, že první automobil se spalovacím motorem byl vyroben v roce 1885. Během let zaznamenal automobilový průmysl mnoho změn a inovací. Tento vývoj je znázorněn na obrázku 4 [31]. Zaznamenává na časové ose důležité SW a elektronické prvky, které se v současnosti naprosto běžně používají v automobilech [31]. Samozřejmě se jedná pouze o subjektivní a stručný výčet těch nejzásadnějších, aby si čtenář udělal představu o průběhu vývoje.



Obrázek 4: Důležité milníků automobilového vývoje na časové ose

První automobily a motory se startovaly pomocí kliky, kterou se roztáčela kliková hřídel a umožnila nastartování motoru. Pár let po roce 1900 bylo vynalezeno a použito **vysokonapěťové magnetické zapalování**. Magnetické zapalování bylo představeno Daimlerem Phönixem a používalo zapálení jiskrou pomocí zapalovacích svíček. Přibližně ve stejné době použil Robert Bosch podobný princip zapalování [31].

Po roce 1920 bylo poprvé v Kelly's motor v Austrálii instalováno **první rádio**. Rozvoj tohoto odvětví umožnil pokrok v používání elektronek. Po roce 1920 byl také poprvé použit elektrický stěrač oken, který byl v brzké době následován odstřikovačem stíraného okna [54].

V roce 1930 byl v American Galvin Manufacturing vytvořen **první sériově vyráběný rádio přijímač**. Jen pro představu, jeho cena tehdy činila 130 dolarů, což bylo opravdu hodně. Pro srovnání Ford Model A tehdy stál kolem 500 dolarů. Toto odvětví zažilo prudký rozvoj zejména v období druhé světové války [54].

V roce 1957 bylo vytvořeno firmou Bendix Corporation první **elektricky řízené vstřikování paliva**, které bylo nabízeno firmou American Motors Company (AMC). Díky tomu byla firma schopná nabídnout nový 5,4 litrový motor s výkonem 214,8 kW. Před tím se používaly výhradně mechanické rozvody paliva a to především pomocí karburátoru, který sloužil k přípravě směsi. Karburátor se však používá i v dnešní době a to především u menších strojů díky své poměrné jednoduchosti [45].

Anti-lock braking system nebo-li ABS je bezpečnostní systém, který slouží ke kontrole kol během brždění, aby nedošlo k jejich zablokování a znemožnění jakéhokoliv ovládání vozidla. První modernější a počítačově řízený, tříkanálový, čtyř senzorový systém byl

představen roku 1971 firmou Chrysler a Bendix Corporation. Tento systém fungoval již v té době velice spolehlivě [29].

Roku 1976 byla Robertem Boschem GmbH představena **lambda sonda**. Jedná se o senzor, který slouží k měření vypouštěného oxidu uhličitého ze spalovacích komor motoru. Tento senzor má za účel hlídat vypouštěné emise z motoru a podle toho zapalovat směs tak, aby docházelo k co nejčistšímu a nejefektivnějšímu spalování paliva [32].

Se zvyšující se důležitostí bezpečnosti se v průběhu let začaly používat **bezpečnostní pásy** a roku 1981 představil Mercedes ve své vlajkové lodi, třídě S, první přední-řidičův **airbag**. Při nárazu došlo ke stažení bezpečnostního pásu a vystřelení airbagu z volantu, který tak měl redukovat náraz [11].

V 80. letech 19. století se začalo využívat **vysokoenergetické zapalování** označované zkratkou HEI, které bylo vytvořeno Delco-Remy divizí General Motors. Od předchozího typu se lišilo tím, že nepoužívalo mechanický časovač odtrhu s kondenzátorem, typicky označované jako kladívko, ale bylo kontrolováno pomocí počítače. Někdy ještě v kombinaci s mechanickým časovačem [27].

Rozvoj satelitové navigace armády Spojených Státu v letech 1960-1982 umožnil vznik technologie zvané **Global Positioning System** (GPS). Tato technologie navigace byla poprvé využita automobilkou Toyota v roce 1991 a to v modelu Toyota Soarer [5].

Kolem roku 1990 byl hned několika automobilkami představen systém **Electronic Stability Program** (ESP). Tento systém byl poprvé použit v roce 1995 a to hned třemi automobilkami naráz. Jednalo se o Mercedes, který ho nasadil ve své třídě S, dále BMW v jeho třídě 7. Obě tyto automobilky používaly systém od Bosche. Třetí byla Toyota, která představila systém **Vehicle Stability Control** (VSC). Tento systém měl za úkol hlídat ztrátu trakce vozidla a pomocí přibržďování některých kol ji získat zpět pod kontrolu. Vozidlo díky tomu nedostalo smyk [48].

Roku 1996 Mercedes-Benz představil ve své třídě S systém **Emergency Brake Assist** (EBA). Dále v roce 1998 byl Mercedes první automobilkou, která tímto systémem vybavila své vozy. Ostatní automobilky tento trend rychle následovaly. Jedná se o systém, který byl schopen na základě tlaku kladeného na brzdový pedál poznat, zda se jedná o nouzovou kritickou situaci a podle toho elektronika začala tvrději brzdit až do momentu, než se spustil ABS system. Jednalo se vlastně o jednodušší verzi brzdícího asistentu, který známe z dnešních aut. Pokud Vám někdo vkročí do cesty, auto ho detekuje a v ideálním případě samo zabrzdí [31].

V roce 2001 nabízel Nissan Motors ve svém modelu Cima v Japonsku systém **Line Departure Warning** (LDW). Tento systém sloužil k varování řidiče, pokud vozidlo začalo vyjíždět ze svého pruhu. Jednalo se tak o předstupeň dnešních asistentů jízdy v pruhu. Tento systém byl již schopný sledovat dráhy na silnici a podle nich i s vozem zatáčet. Nejednalo se však o plnou automatizaci, zatím bylo nutné asi každých 15 vteřin s volantem nepatrně pohnout, aby systém poznal, že je řidič přítomen, jinak systém spustil zvukové varování. V dnešní době jsou vyššími vývojovými stupni tohoto systému vybavovány au-

tomobily stále častěji [61].

Kvůli snižování emisí a emisní normě Euro 5 byl zaveden **start-stop** systém, který má za úkol vypnout motor, pokud vozidlo stojí a má vyřazený rychlostní stupeň nebo sešlápnutou spojku. Typicky například při čekání v křižovatce. Mezi lety 2008-2010 byla již tímto systémem vybavena většina automobilů, i když vynalezen byl podstatně dříve a to kolem roku 1980, kdy tímto systémem byly vybaveny vozy Fiat Regata a Volkswagen Polo [31].

Kolem roku 2010 se začaly objevovat **systémy pro předcházení kolizím** (Collision avoidance), které slouží k zabránění srážce. Takovýto systém je vybaven radary, někdy lidary a kamerami, které slouží k detekci překážky před vozidlem a umožňují elektronice v automobilu na tuto skutečnost reagovat a zamezit tak srážce. Tento systém pracuje v součinnosti s ostatními systémy jako jsou například ABS, ESP, brake assisenty a další. Oblast asistentů se neustále prudce vyvíjí a je to nedílná součást pro autonomní automobily. [7, 34]

Z toho výčtu je jasně patrné, jak rychle se automobilový průmysl vyvíjí a jak se dříve prvky použité výhradně v luxusních vozech dostávají i do těch cenově dostupnějších. Tento trend se stále zrychluje [2]. Uvedený výčet historických milníků je zde uveden hlavně z toho důvodu, aby dokumentoval neustálý vývoj elektroniky a použitého SW v automobilech [31]. V dnešní době se prudce rozvíjí odvětví **autonomních automobilů**, kdy téměř všechny významné automobilové značky pracují na tomto konceptu [7]. Dále je významným trendem elektromobilita, které bude věnována následující podkapitola.

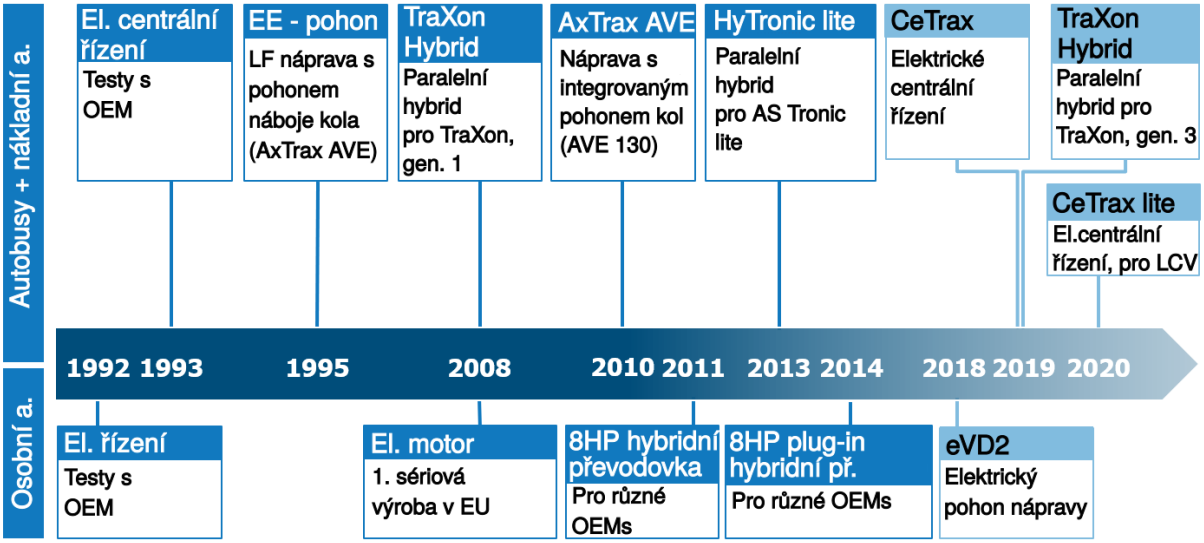
2.1.3 Budoucnost a elektromobilita

Každá významná automobilová společnost bere elektrifikaci automobilové dopravy velice vážně a na její vývoj vynakládá nemalé sumy. Situace je stejná i v rámci firmy ZF. Momentální brzdou většího rozvoje elektrických automobilů jsou především omezená kapacita baterií a s tím související dojezd. Auto, které na jedno nabití ujede 100 km asi nikoho příliš nezaujme. V dnešní době má například **Tesla** model X v nejnižší verzi 75D udávaný dojezd 383 km. Ve verzi 100D je to již 475 km. Číslo u verze udává kWh bateriového systému, tedy 100D znamená, že je vůz vybaven bateriemi o kapacitě 100 kWh. Spotřebu energie pak v kombinovaném režimu činí 24 Kwh/100 km [59].

Dalším problémem, který souvisí s bateriemi, je jejich nabíjení. Plné nabití takto velkých kapacit trvá poměrně dlouho. Řešením je snaha firem vyvinout nové baterie a samozřejmě také rychlonabíjecí stanice. Tyto stanice se nachází již i v České Republice, nicméně je jich stále velice málo. Domácí nabíjecí stanici s výkonem 2,3 kW zabere dobití baterií verze 100D celých 44 hodin. Dále existují nabíjecí stanice s výkonem 3,7 kW, 11 kW a 16,5 kW. Ty zvládnou nabít verzi 100D po řadě za 27,5 , 9,5 nebo 6,5 hodin [59].

Ani společnost **ZF** nebere elektromobilitu na lehkou váhu a již má v tomto odvětví více než 20 let zkušeností. Jedná se zejména o elektronické centrální řízení, hybridní diferenciály, elektromotory a hybridní převodovky, které jsou spřaženy s pomocným elektromotorem a jak v případě osobních, tak i nákladních automobilů. Na obrázku 5 je znázorněna his-

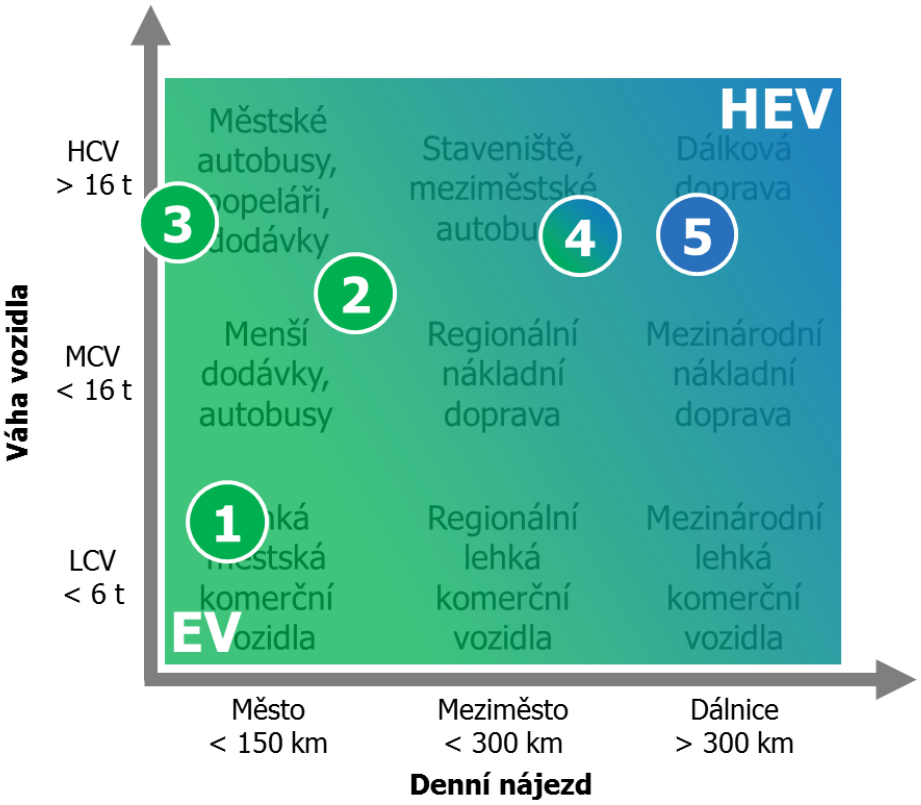
torie elektromobility v rámci ZF s vyznačenými významnými roky. Kde dolní část pod letopočty je věnována osobním automobilům a část nad letopočty je věnována autobusům a nákladním automobilům. Original Equipment Manufacturer (OEM) je obchodním termínem, který označuje výrobce zařízení, jehož výrobek je prodáván a propagován jinou obchodní značkou [4, 25].



Obrázek 5: Časová osa aktivity společnosti ZF v oblasti elektromobility

Mnoho lidí vnímá budoucnost v hybridních verzí automobilů, kde se spojují výhody jak spalovacích, tak elektrických motorů. Minimálně do té doby, než budou bateriové systémy natolik schopné, že na jedno nabití bude automobil schopný urazit tisíce kilometrů. Velice podobný názor zastává i společnost ZF. To si představuje, že do roku 2025 významně vzroste význam a podíl **hybridů**, dále uvádí, že plně elektrická auta budou hrát významnou roli především ve velkých městech. Celá strategie je rozdělena podle denní ujeté vzdálenosti a hmotnosti vozidel. Strategie pro komerční elektrická vozidla je znázorněna na obrázku 6, kde EV znamená „elektrická vozidla“ (electric vehicles) a HEV „hybridní elektrická vozidla“ (hybrid electric vehicles). Dále LCV znamená „lehká komerční vozidla“ (light commercial vehicle), MCV představuje „střední komerční vozidla“ (medium commercial vehicle) a HCV pak „těžká komerční vozidla“ (heavy commercial vehicle) [25].

Dále se na obrázku 6 nacházejí body s čísly, kdy každý bod reprezentuje určitý typ a zařazení vozidla. Bod číslo 1 představuje lehká komerční vozidla, jedná se například o osobní automobily, vany a menší dodávky. Bod číslo 2 představuje městské nákladní automobily a menší autobusy. Bod číslo 3 představuje městské autobusy, které známe z městské hromadné dopravy. Bod číslo 4 představuje meziměstské-dálkové autobusy. Číslo 5 je zastoupeno dlouhými nákladními vozy, které všichni známe z dálnic.



Obrázek 6: Znázornění elektromobility v závislosti na váze vozidla a denním nájezdu kilometrů

2.2 Normy a bezpečnost v automobilovém průmyslu

S rozvojem automobilového průmyslu a s postupným zvyšováním významu a množství SW, který je v automobilech použit, vznikla potřeba vytvoření jednotných standardů a norem pro jeho vývoj a správný provoz. Jak již bylo zmíněno dříve, tak v dnešním průměrném automobilu je přibližně 50 řídicích jednotek označovaných jako Electronic Control Unit (ECU). Každá řídicí jednotka řídí svůj daný systém/okruh a musí zajistit jeho správnou funkci. V dnešní době je toto extrémně důležité, protože se elektronika a SW nacházejí i v kritických systémech, jejichž selhání by mohlo vést až ke ztrátám na lidských životech. Je proto nutné všechny tyto řídící jednotky otestovat tak, aby byly splněny všechny požadavky kladené normami na tyto řídicí smyčky [1, 31]. A to bez rozdílů, zda byl SW vytvořen pomocí MBSD metod a generování kódu nebo je napsán ručně. V případě generování kódu musí být zabezpečena stejná bezpečnost a funkcionalita, jako při použití ručně psaného kódu [14, 52]. Účelem této práce není podrobný rozbor různých norem, a proto zde budou uvedeny pouze ty nejdůležitější vzhledem k SW. Zde je také na místě zmínit, že v každém státě platí rozdílné zákony a normy. Každý jistě ví, že například existují rozdíly mezi americkým, čínským a evropským trhem. Na základě zákonů a norem se vozidlům vystavuje takzvaná homologace. Tento pojem představuje ověření vlastností vozidla a opravňuje k jeho užití v silničním provozu. Pokud ověření vozidlo nezíská, nelze jej provozovat na silničních komunikacích.

2.2.1 Automotiv SPICE

Automotive SPICE je jeden z nejvýznamnějších standardů z hlediska řízení procesů a kvality vývoje SW. Od roku 2005 se označuje jako ISO/IEC 15504 celým názvem „international organization for standards“ a „international electrotechnical commision standards“. Tato norma byla odvozena od ISO/IEC 12207 a v v roce 2015 byla aktualizována na ISO/IEC 33001. Dalšími normami jsou například normy pro posouzení schopností procesu nebo různé měřící rámce. Celý popis normy lze nalézt na internetových stránkách www.automotivespice.com. Výsledky auditu a dodržování toho standardu bývají významným faktorem pro různé automobilky pří výběru svých partnerů. Význam tohoto standardu roste s rostoucím významem, objemem a použitím SW v automobilovém průmyslu. Proto je tento standard podporovaný řadou automobilek. Jedná se například o automobilky Volkswagen AG, Volvo Group, Audi AG, Daimler AG, BMW Group, Ford Motor Company a další. Tyto společnosti se sdružily do skupiny SPICE User Group. Tato skupina se stará o rozvoj a aktualizace tohoto standardu. Díky tomu je téměř každý rok tato norma aktualizována a doplněna o zkušenosti z vedení různých projektů, reaguje tak na aktuální témata této problematiky a udržuje se tak stále na aktuální úrovni.

2.2.2 ISO 26262

ISO 26262 je norma, která, volně přeloženo, nese název „Silniční vozidla-funkční bezpečnost“. Již z toho vyplývá, že obsahuje řadu norem, které se zabývají bezpečností. Tento standard byl publikován poprvé v roce 2009 a v roce 2011 byla provedena aktualizace. Jedná se o placenou normu a je rozdělena do deseti částí a každá část se dá koupit zvlášť. Významným tématem, kterým se tato norma zabývá, je analýza rizik pozorované funkce. Díky této analýze lze následně sestavit návrhy na zabezpečení provozu a různé bezpečností koncepty pro dané systémy. Při návrhu se zde uvažuje i určitý systém volnosti, který vede ke snížení nákladů na vytvoření bezpečnostního návrhu. Důležité je, že případná chyba některé systémové komponenty nesmí vést ke kritickému selhání celého systému. Na základě této analýzy lze poté rozhodnout, které komponenty jsou bezpečnostně relevantní, a stanoví se jakého bezpečnostního stupně se při jejich vývoji musí dosáhnout. Tato norma se používá se systémy, které obsahují jeden nebo více elektronických nebo elektrotechnických systémů. ISO 26262 se tedy zabývá rizikem selhání systému způsobených některou z použitých komponent. Nezabývá se nebezpečím požáru, kouře, hořlavostí, korozí a dalšími problémy. [4, 23, 24]

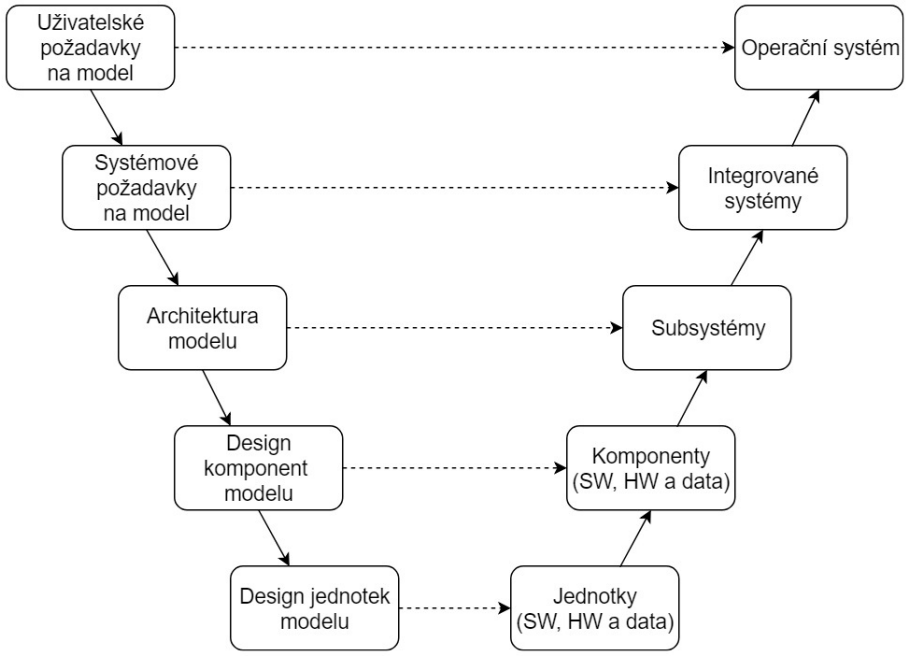
2.2.3 Interní normy a zásady v rámci ZF

Z důvodu zvýšené bezpečnosti si firmy zavádějí ještě interní normy, předpisy a postupy, kterých se drží, ať už během vývoje nebo při testování. Firma ZF není výjimkou, a proto i v ní platí nepřeberné množství předpisů, jak postupovat například při vytváření kódu, při vytváření modelu v Matlabu, Simulinku, při testování a mnoho dalších. Některé z těchto norem slouží především k tomu, aby například ručně psaný kód byl psán tak, aby byl maximálně optimalizovaný, přehledný, udržitelný a dostatečně zdokumentovaný. Tvorba modelu v Matlabu, Simulinku se řídí velice podobnými pravidly. Testování probíhá na několika úrovních a na každé se postupuje podle daných předpisů. Jedná se například o SIL a MIL testy, které si vývojář udělá u sebe na počítači. Některé z takto vytvořených testovacích scénářů se poté můžou začlenit do tzv. „nightly build“ testů. O nich je podrobněji

pojednáno v kapitole 6.1.5. Pokud proběhnou v pořádku, tak se přechází k integračním testům a HIL testování. Následně se přejde k testování a měření v automobilu na testovacím polygonu. To se někdy také označuje jako validace v automobilu. Je tedy jasné, že než se změna dostane do sériového provozu, tak musí projít přes celou řadu testů. Díky tomu je maximum možných chyb odhaleno již při testování a nebo během vývoje a nedostanou se tak do reálného provozu, kde by mohlo dojít k vážnému selhání. V průběhu testování se rozlišuje, zda byl použit ručně psaný kód nebo generovaný kód z modelu v Matlabu, Simulinku. Liší se to v typech použitých testů a to z toho důvodu, aby byly všechny normy a standardy otestovány a splněny. Nelze totiž vždy použít stejné testy pro oba přístupy [25].

2.3 Principy vývoje SW

Při vytváření nového SW se v automobilovém průmyslu často postupuje podle tzv. **V-modelu** [22]. Jedná se o schéma, které je na obrázku 7 a demonstruje postup vývoje. Jak již bylo zmíněno v kapitole 1.2 věnované motivaci, v úvodu je nejprve potřeba specifikovat, co bude předmětem vývoje, a to ať vytvoření zcela nového SW nebo jen úpravy stávajícího SW, typicky se jedná například o rozšíření nebo upravení některé funkcionality. Podle specifikace realizuje vývojář změny modelu v Matlabu, Simulinku. Následně tu samou funkcionalitu vytvořenou v modelu přepíše jiný vývojář do zdrojového souboru v C++, tomu tak model v Simulinku slouží jako reference. Nakonec následuje celá řada testů a pokud vše proběhne v pořádku, tak se tento SW zařadí do produkce a bude nahrán na hlavní větev verzovacího systému [16, 22, 28, 31, 58].

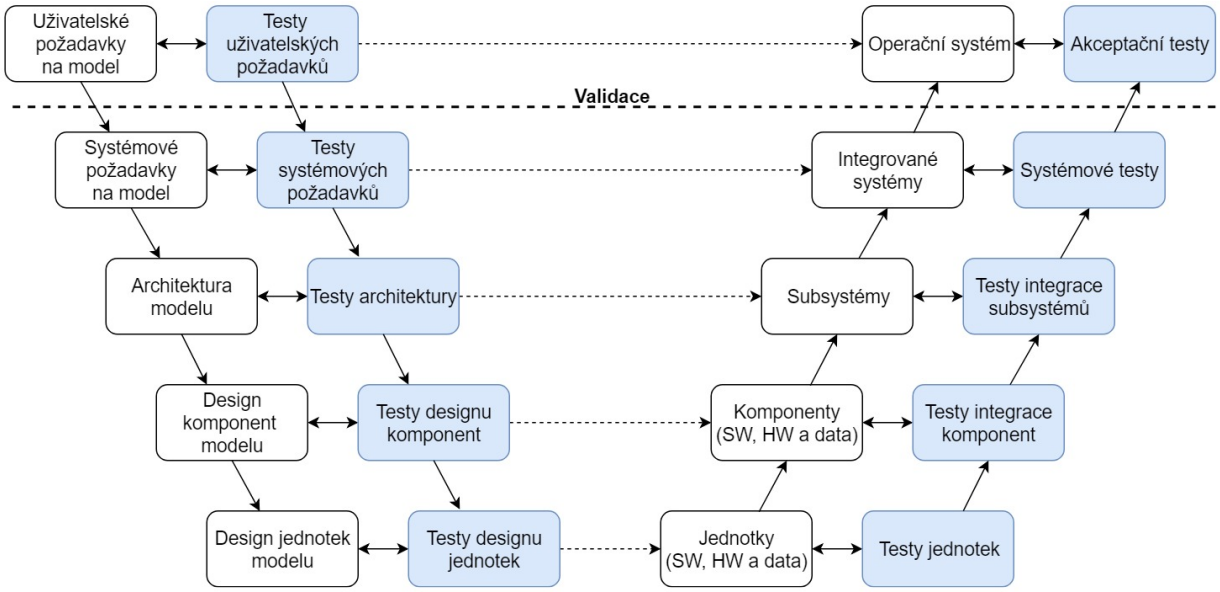


Obrázek 7: Struktura V-modelu jako procesu pro vývoj SW

V případě automaticky generovaného kódu dochází k jistému sloučení práce. Vývojář, který pracoval na vytvoření nebo úpravě modelu pro automatické generování kódu, udělá

zároveň jeho integraci do ručně psaného kódu v C++. Oba zmíněné přístupy jsou demonstrovány v následujících podkapitolách 2.3.3 a 2.3.4 a jsou znázorněny na obrázcích 11 a 12.

V-model popisuje spíše práci na kompletně novém projektu. V případě, že je ale SW již vytvořený a je potřeba upravit dílčí komponentu nebo dodělat některou funkci, jedná se spíše o tzv. **W-model** [55]. Dalo by se říct, že je to svým způsobem iterační proces, během kterého dochází k postupným úpravám jednotlivých funkcionalit. W-model představuje jinou variantu tradičního V-modelu. Pokud kdokoliv, kdo je zapojený do vývoje, objeví chybu, tak ji reportuje tomu, kdo je zodpovědný za předešlý krok. Ten udělá nutné změny k opravě a vrátí výsledek opět zpátky do procesu. Tento postup se opakuje tolikrát, dokud není zajištěna například požadovaná funkčnost SW nebo dokud nejsou odladěny všechny chyby objevené při testech. W-model se liší od normálního V-modelu hlavně v tom, že v případě V-modelu testování probíhá až v pravé větvi modelu. Kdežto v případě W-modelu probíhá testování na každé úrovni. To znamená, že si každý vývojář otestuje sám to, co navrhl. Je tak zajištěno, že daná úprava odpovídá daným specifikacím. Tento postup je mnohem bližší MBSD přístupu, než-li V-model. Pro jeho popis je přesnější [16, 31, 51, 55]. Schéma W-modelu je znázorněno na obrázku 8.

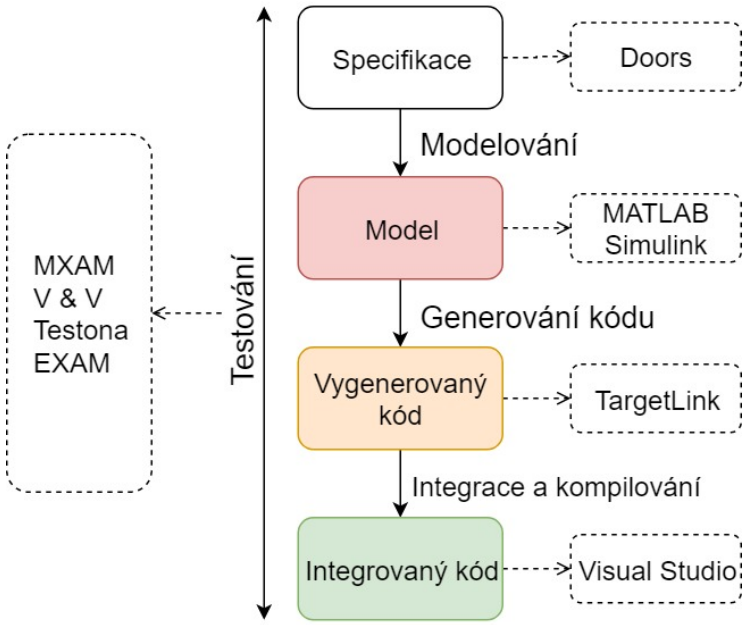


Obrázek 8: Struktura W-modelu jako procesu pro vývoj SW

Existuje i verze, která se označuje jako **trojitý V-model**. Ten se od W-modelu liší navíc tím, že je rozšířen o jednu větev. Tato větev se zabývá validací výsledků každých testů. Tedy, pro postup do další fáze vývoje je vždy potřeba validace a potvrzení dosažených výsledků [55]. Princip tohoto přístupu je znázorněn na obrázku 55 v přílohách, kde je umístěn kvůli zachování čitelnosti.

2.3.1 Blokově orientovaný návrh systémů

Blokově orientovaný návrh systémů neboli **MBSD** je metodou, která se používá pro rychlý a efektivní návrh systémů (například řídící, pro zpracování signálů a dalších) a SW řešení [31, 38, 51, 58]. Diplomová práce rozšiřuje tento přístup a automatické generování kódu z modelu.



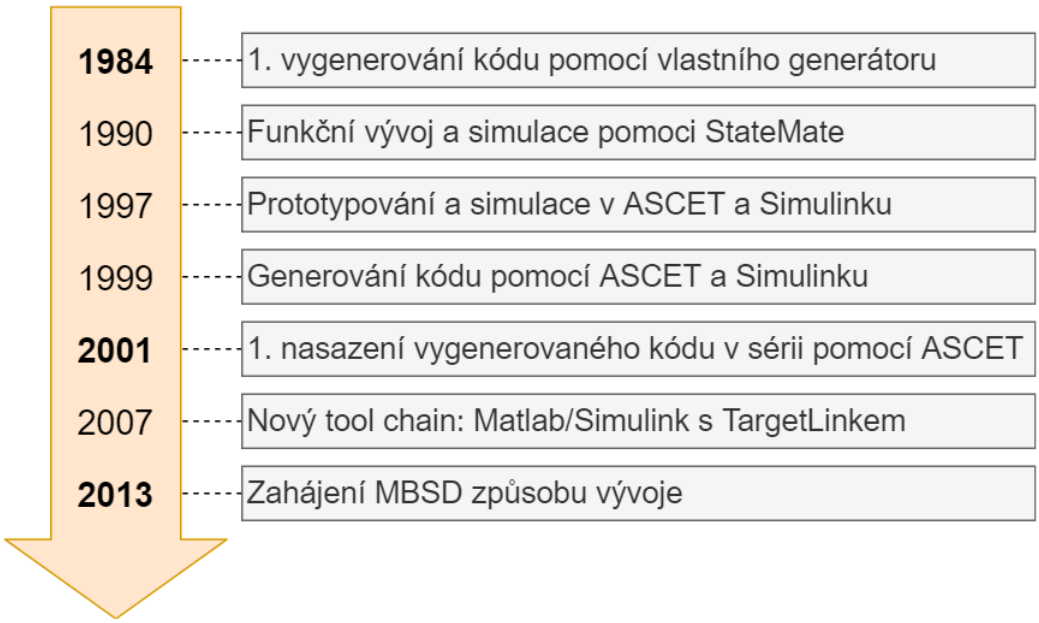
Obrázek 9: Ilustrace vývoje SW pomocí MBSD ve spojení s běžně používanými nástroji v automobilovém průmyslu

Přístup vývoje SW založeného na MBSD je ilustrován na obrázku 9. Podobně, jako v případě klasického vývoje SW, je vytváření modelu založené na specifikacích. Existují různé nástroje pro správu těchto požadavků. Jedná se například o DOORS, Enterprise Architect nebo JIRA. V poslední době je věnována zvýšená pozornost poslednímu zmíněnému nástroji a to zejména z důvodu rostoucího významu agilního vývoje SW [46]. Za nejrozšířenější blokově orientované prostředí pro modelování lze považovat Matlab Simulink. Existují i další nástroje jako například OpenModelica, Simulation X, LabVIEW, SCADE, ACED nebo Statemate. Při vytváření modelu je důležité dodržet předem definované postupy, aby byly splněny normy a standardy, které v automobilovém průmyslu platí. Jedná se například o ISO 26262, IEC 61508, AUTOSAR a další [52]. Pro testování všech zásad se například v Simulinku používá MAAB kontrola nebo kontrola pomocí MXAMu [14, 41]. Na straně SW se jedná například o MISRA kontrolu. Pro generování kódu lze použít širokou paletu nástrojů. Například v rámci Matlabu se může jednat o Matlab Coder, LabView poskytuje svůj C generátor, ale v prostředí automobilového průmyslu je nejvíce rozšířen TargetLink [12, 35]. Vygenerovaný kód je následně nutné zaintegrovat do ručně psaného kódu. Výsledný SW po integraci čeká testování [1]. Proces testování se nijak neliší od testování ručně psaného kódu, následují tedy SIL, MIL, HIL testy a validace v automobilu. Na obrázku 9 je tento proces ilustrován včetně nástrojů používaných při vývoji v rámci ZF. [2, 16, 17, 31, 43, 50]

2.3.2 Model Based Design v rámci ZF

Historie vývoje používání automatického generování kódu a MBSD postupů vývoje je znázorněna na obrázku 10. Modelově orientovaný vývoj SW se v rámci ZF začal používat již v roce **1984**, kdy byl poprvé vygenerovaný kód pomocí vlastního generátoru kódu. Označovat tento bod již jako MBSD není zcela přesné. Tento přístup se začal používat až v roce 1997, kdy se jednalo o kombinaci vývoje pomocí programů ASCET a Simulinku. U všech zde zmíněných nástrojů je velice důležité, že splňují bezpečnostní normy [14]. Jedná se například o normy ISO 26262, MISRA-C a další.

ETAS ASCET-DEVELOPER (také známý jako ASCET 7) představuje nástroj pro vývoj aplikačního SW pro embedded systémy (vložené systémy) za pomoci grafického prostředí a textových programovacích oken. Tento nástroj umožňoval a stále umožňuje generování vysoce bezpečného a účinného SW v jazyce C. Jedná se tak o konkurenci k programu TargetLink od firmy dSPACE [3].



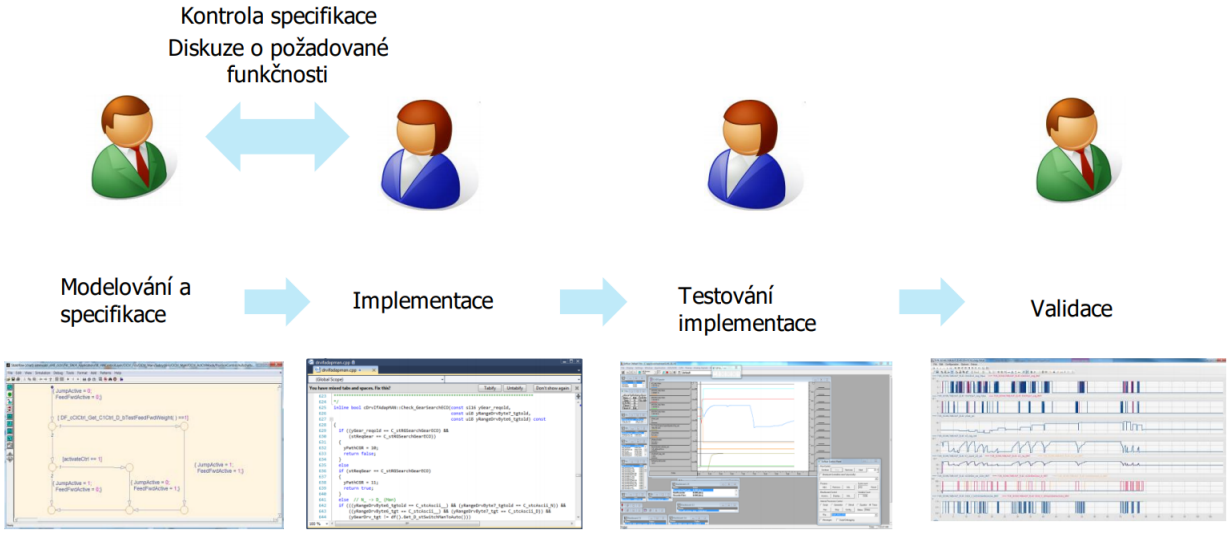
Obrázek 10: Historie používání automatického generování kódu v rámci ZF

I když se s generováním kódu poprvé začalo již v roce 1984, byl poprvé nasazen pro produkci sériového SW až v roce **2001**. To ilustruje fakt, že v té době se jednalo o zcela nový přístup a bylo nutno přepracovat přístup při vývoji SW a zajistit, aby byly splněny požadavky na bezpečnost při nasazení v reálném provozu. Dále se v roce 2007 změnily používané nástroje na Matlab a Simulink. Použité nástroje se často označují jako „tool chain“. Ke generování kódu se začal používat TargetLink od firmy dSPACE. Tato firma rovněž vyrábí celou řadu zařízení, například průmyslové počítače určené pro HIL testování. Tento „tool chain“ se používá i v současnosti, kdy postupně dochází k používání novějších a novějších verzí programů. Od roku **2013** se při vývoji aplikují metody MBSD. Aktuálně se v ZF používá Windows 7 a řeší se přechod na Windows 10. S tím také souvisí použití nové verze Matlabu se Simulinkem a také TargetLinku. Všechny nástroje jsou již zprovozněny a čeká se pouze na dodělání aktuálně rozpracovaných témat, aby následně došlo k hromadnému přechodu na novější verzi Windows a nástrojů.

2.3.3 Ručně psaný kód

Každý si asi dovede představit, co znamená ručně psaný kód. Na základě specifikace bude naprogramováno SW řešení [2]. V běžném automobilu se vyskytuje kolem 50 řídících jednotek a ve většině z nich se používá kód napsaný v jazyce **C** nebo **C++**. Samozřejmě existují i výjimky, ale těmi se nebudeme zabývat. V rámci ZF se zdrojové soubory píší v kombinaci jazyků **C** a **C++**. Veškeré parametry a hodnoty jsou uloženy ve zdrojových souborech, které se označují jako „datafieldy". Ty v sobě obsahují všechny potřebné informace o definovaných parametrech, konstantách, tabulkách a dalších a umožňují jejich volání a to jak v samotném SW tak v modelu v Simulinku.

Na obrázku 11 je znázorněn postup při vývoji ručně psaného kódu. Již bylo zmíněno, že na základě specifikace bude vytvořen jedním vývojářem model, který poslouží jako reference pro dalšího vývojáře SW. Poté následují testy a po jejich úspěšném absolvování následuje validace celého řešení v automobilu. V případě transformace modelu pro generování kódu se bere jako reference ručně psaný kód.



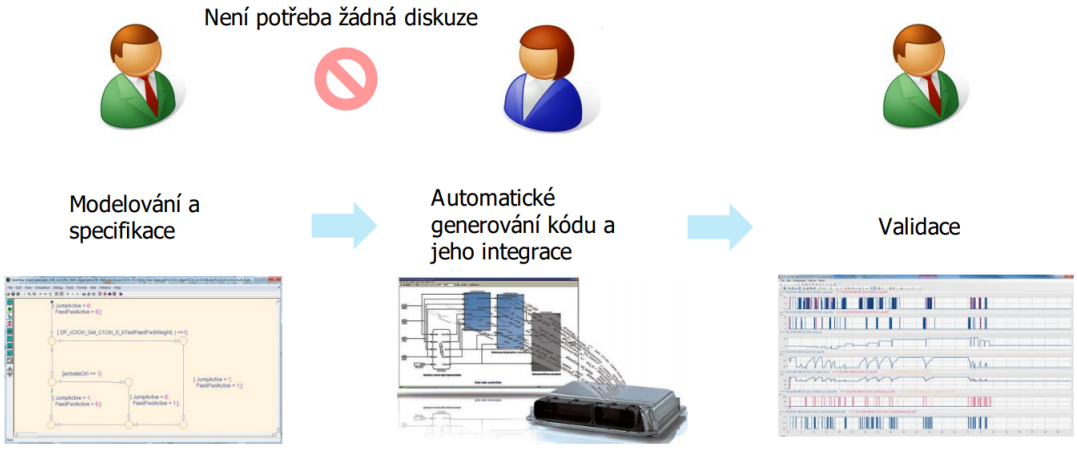
Obrázek 11: Klasický přístup vývoje SW při ručně psaném kódu

Jak je patrné z obrázku 11, tak v tomto procesu potřebujeme, jak vývojáře softwaru, tak modelu. Do tohoto procesu je tedy zapojeno více lidí, což se samozřejmě odráží na potřebném čase a prostředcích. Připomínám, že se pohybujeme v problematice automobilového průmyslu, kde každý dolar hraje roli. Řeknete si, že není problém koupit a osadit výkonnější čip, který bude o jeden dolar dražší. Ovšem v měřítku jednoho milionu automobilů to již dělá jeden milion dolarů nákladů navíc. Z toho vyplývá, že je zde velice důležitý požadavek na cenu a maximální možnou efektivitu a optimalizaci pro dané řešení [31]. Právě ze snahy ušetřit prostředky vzešla myšlenka na automatické generování kódu z modelu. Na druhou stranu u ručně psaného kódu realizuje vývojář danou funkcionalitu přesně tak, jak chce on a formuluje její podobu podle sebe. Není tak formou implementace SW svázán jako při generování kódu, což může být někdy výhoda.

2.3.4 Automaticky generovaný kód

Existuje veliké množství programů, které umožňují generování kódu z grafické reprezentace modelu. Jedná se například o programy, ASCET of firmy ETAS, Simulink coder zabudovaný v Matlabu, TargetLink od firmy dSPACE a mnoho dalších [3, 12, 22, 35]. V této práci se budeme zabývat pouze řešením od firmy **dSPACE-TargetLinkem** a to z toho důvodu, že právě toto řešení se momentálně v rámci ZF používá a neplánuje se do budoucna jeho výměna za jiný nástroj.

Z důvodu šetřit prostředky vzešla myšlenka udržovat pouze model, ve kterém se budou provádět funkční změny a z něj se bude následně generovat kód. Vygenerovaný kód se poté bude integrovat do ručně psaného kódu. Dojde tedy k postupnému nahrazení ručně psaných funkcí a tříd za vygenerované [16]. Tímto přístupem dojde k výraznému zjednodušení celého procesu, ušetří se čas a peníze potřebné na vývoj. Další nespornou výhodou je, že vývojář, který provede změny v modelu a vygeneruje kód, tak zároveň přesně ví, jaké změny udělal a jak by měl vygenerovaný kód fungovat. S tím souvisí také otestování provedených změn. Vývojář díky tomu přesně ví, jak danou funkcionalitu správně otestovat [43]. Tento přístup je znázorněn na obrázku 12.



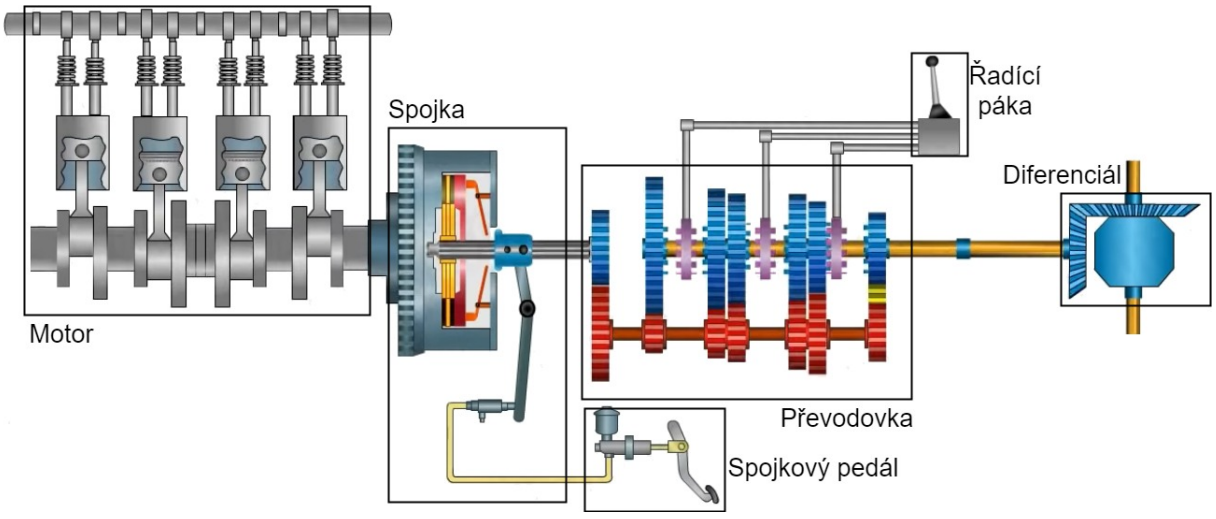
Obrázek 12: MBSD přístup vývoje SW při automaticky generovaném kódu

Při pohledu na obrázky 11 a 12 jsou rozdíly jasně patrné. Díky generování kódu jsme schopni získat i více optimalizovaný SW, který má nižší výpočetní náročnost než ručně psaný kód. Jedná se o náročnost zatížení procesoru, která bývá v angličtině označována jako „Central Processor Unit (CPU) load“. To platí jen za předpokladu, že je model vytvářen při respektování určitých zásad. Snižování výpočetní náročnosti je dobré především pro jeho životnost a další vývoj. Když se v automobilovém průmyslu nasadí nějaké řešení, očekává se od něj, že jeho série bude mít určitou životnost. Tedy, že na trhu nějakou dobu vydrží. S tím také souvisí zaplacení celého vývoje. Během let se SW neustále upravuje a vyvíjí, přidávají se nové funkcionality a opravují se objevené chyby. Proto je nesmírně důležitá optimalizace výsledného kódu, aby byla možnost se posouvat vpřed, držet krok s konkurencí a požadavky zákazníků. Problematiku, proč nepoužít výkonnější čip, jsem vysvětlil již v předchozích odstavcích.

2.4 Manuální a automatická převodovka

Dříve než začneme popisovat jednotlivé typy převodovek je důležité si uvést, čím to je, že je převodovka tak důležitým komponentem automobilu. Každý motor poskytuje jen omezené pásmo efektivních otáček a kroutícího momentu. Toto rozmezí se pohybuje například kolem 2000 až 6000 otáček. Záleží na typu motoru, kdy každý motor má toto rozmezí jiné. U dieselových motorů může být toto rozmezí ještě o něco nižší. Řeč je o běžných automobilech. Samozřejmě existují vysokootáčkové benzínové motory, ale ty pro nás momentálně nejsou důležité. Motory používané v nákladní automobilové dopravě jsou výhradně dieslové. Díky převodovce, která poskytuje několik převodových poměrů, ať už pomocí hřídelí v případě manuální převodovky nebo pomocí planetových převodovek v případě automatické převodovky, drží motor v efektivních otáčkách a umožňuje tak jeho co nejefektivnější využití. Dále se v této kapitole budeme věnovat popisu a rozdílům mezi manuální a automatickou převodovkou. V podkapitole 2.4.3 a 2.4.4 budou představeny převodovky vyráběné ZF. Jedná se o typy **EL40 TraXon** a **EL57 Ecomid**. Obě jsou určené pro nákladní automobily.

Převodová skříň je pomocí spojky spojena s motorem. Jak již bylo zmíněno, motor generuje točivý moment, který je pomocí spojky převáděn do převodové skříně a odtud za daného převodového poměru je převáděn na samotná kola automobilu. Někdy se na ose kol vyskytuje ještě diferenciál, který umožňuje připojení další nápravy. To je typické pro vozy s náhonem 4x4. Spojka je v případě manuální převodovky řízena pedálem, který umožňuje rozepnutí přenosu točivého momentu z motoru do převodové skříně. To umožňuje zařazení nového stupně a postupným puštěním spojkového pedálu dochází k zavírání spojky, tedy obnovení přenosu točivého momentu. Celý systém je schématicky znázorněn na obrázku 13.



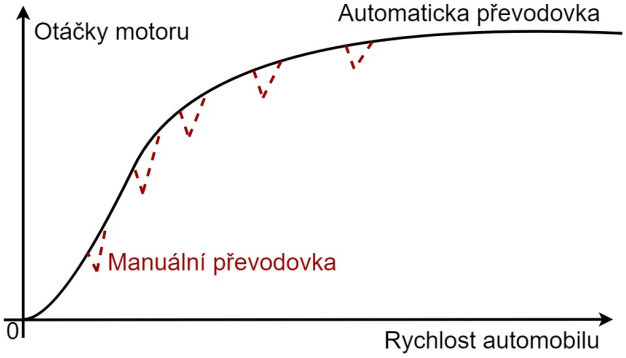
Obrázek 13: Schématické znázornění motoru, spojky, převodové skříně a hřídele

V případě automatické převodovky řídí otevírání a zavírání spojky elektronika a není tedy potřeba spojkový pedál. Princip přenosu točivého momentu zůstává stejný. Z manuální převodovky lze vytvořit automatickou přidáním systému pro automatické řazení. Tyto typy se označují jako robotizované převodovky. V následujících podkapitolách se budeme věnovat popisu manuální a automatické převodovky.

2.4.1 Manuální převodovka

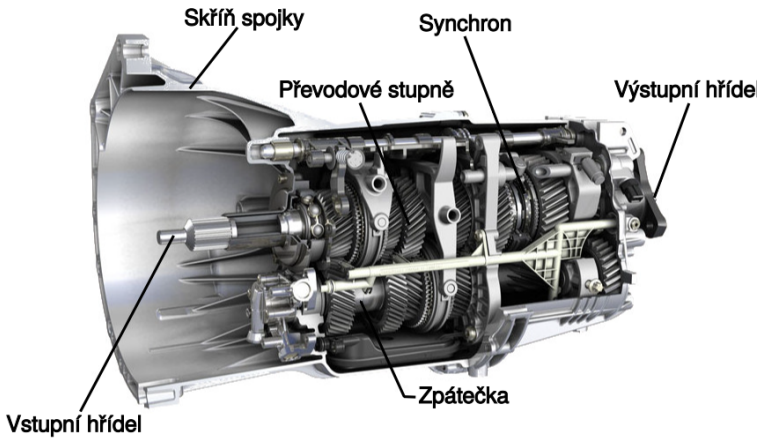
Princip funkce

Hlavní rozdíl mezi manuální a automatickou převodovkou je ten, že v případě manuální musí být proveden úkon **řazení** pomocí řadící páky a spojkového pedálu. Tímto úkonem je jasně definováno, jaký převodový stupeň bude zařazen. Většinou je schéma řazení znázorněno na hlavici řadící páky. Při sešlápnutí spojkového pedálu dochází k rozpojení motoru a může tak být zařazen nový převodový stupeň. S tím také souvisí ztráta výkonu. Otáčky motoru během řazení klesnou a při rychlém zavření spojky po zařazení nového převodového stupně, může celý automobil nepříjemně cuknout nebo může dojít až k zastavení motoru. Poté je nutné motor nastartovat znovu. S tím tedy souvisí i jisté nepohodlí během jízdy. Na obrázku 14 je výkonová křivka pro manuální a automatickou převodovku, na



Obrázek 14: Ilustrace porovnání výkonové křivky pro manuální a automatickou převodovku

které jsou jasně patrná místa propadu výkonu, ve kterých docházelo k řazení nového převodového stupně. Na obrázku 15 je řez manuální převodovkou. Jsou na něm jasně patrné hřídele, pomocí kterých dochází k přenosu točivého momentu, podle zařazeného převodového stupně. Zajišťují tak spojení v odpovídajícím převodovém poměru mezi vstupní hřídelí a výstupní hřídelí, která převádí výkon na kola automobilu.



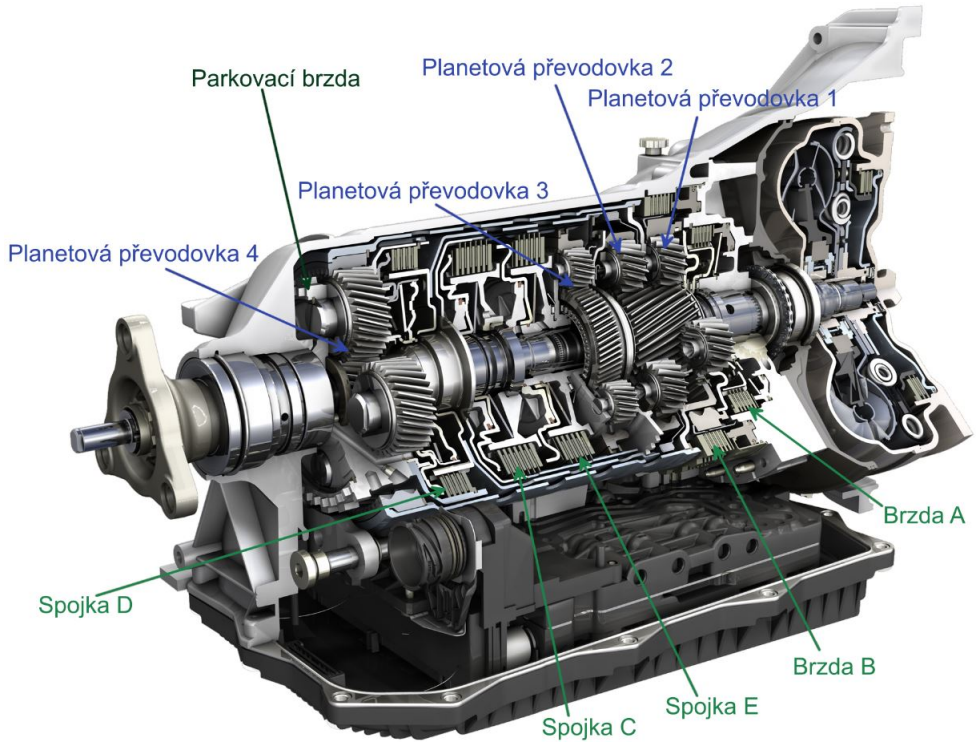
Obrázek 15: Řez manuální převodovkou používanou BMW [9]

Výhody a nevýhody

Mezi nespornou výhodu patří především **plná kontrola** nad převodovkou a vozidlem. S tím ale také souvisí nebezpečí zničení celé převodové skříně pokud zařadíme v nesprávnou dobu chybný převodový stupeň. Typickým příkladem může byt zařazení zpátečky nebo nízkého stupně během jízdy vpřed ve vysoké rychlosti. Mezi další výhodu patří menší servisní náročnost, kdy i při špatném zacházení manuální převodovka vydrží opravdu hodně a dlouho. Při rychlém a agresivním řazení může dojít k dotyku zubů a ozve se typický chrčivý zvuk. Další nevýhodou je, že během řazení dochází ke **ztrátám výkonu** motoru a při rychlém zavírání spojky může docházet k nepříjemným cuknutím. Další nevýhodou mohou být rozjezdy do kopce, kdy řidiči často nechávají spojku prokluzovat příliš dlouho, a dochází tak k jejímu nadměrnému opotřebení. To samé platí pro jízdu v koloně, opakované zařazování, popojíždění a opětovné vyřazování, kdy také dochází k nadměrnému opotřebovávání.

2.4.2 Automatická převodovka

V případě automatické převodovky se v dnešní době o řazení stará počítač a řidič tak musí pouze pomocí ovladače zadat příkaz, zda chce jet dopředu (D), dozadu (R), parkovat (P) a nebo jestli chce vyřadit a zůstat v neutrálu (N). Na obrázku 16 je znázorněn řez automatickou převodovkou. Jedná se o typ **ZF 8HP** s osmi rychlostmi a je použí-



Obrázek 16: Řez automatickou převodovkou ZF 8HP

vána v osobních automobilech. Pomocí vnitřního systému spojek a brzd, jak je patrné z obrázku 16, se zajišťuje přenášení točivého momentu mezi planetovými převodovkami v požadované míře. Důležité je uvést, že ne každá automatická převodovka má stejnou

konstrukci, i když zajišťuje stejnou funkci. Existuje mnoho typů a proto uvedu jen ty nejzákladnější. Jedná se například o převodovky hydrodynamické. Tato převodovka využívá hydrodynamický měnič jako spojku. To je zařízení umožňující přenos kroutícího momentu a při změně rychlosti dochází k jeho násobení. Dalším příkladem je typ postavený planetovými převodovkami. Dale se může jednat o dvouspojkové automaty, v dnešní době se objevují i trojspojkové automaty. Typickým příkladem je převodovka **Direct Shift Gearbox** (DSG) od Volkswagnu. Jedná se ve své podstatě o robotizovanou manuální převodovku, kdy řazení pomocí synchronů je řízeno počítačem. Princip této spojky spočívá v tom, že zatímco na jedné hřídeli je aktuálně využívaný převodový stupeň, tak na druhé už je připravený další. Díky tomu probíhá řazení velmi rychle a téměř bez ztrát výkonu. U trojspojkového automatu jsou nachystány dva další stupně namísto jednoho.

Princip funkce

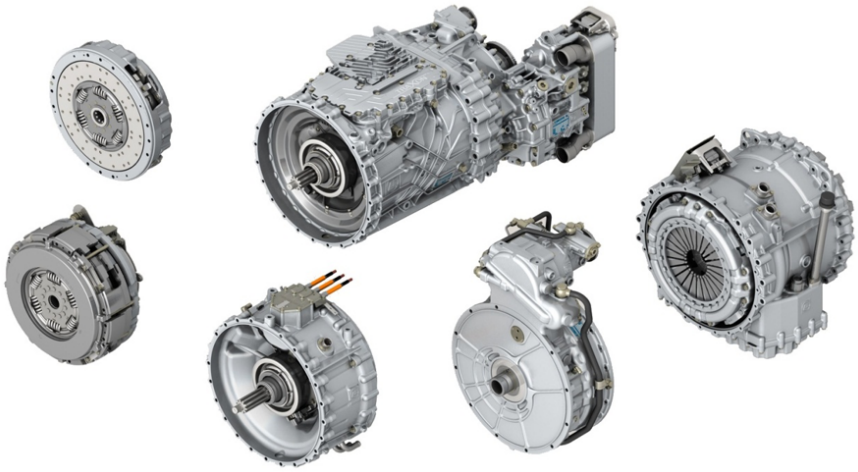
Jak už bylo nastíněno v předchozím odstavci, tak automatická převodovka na základě vstupních informací z motoru, senzorů a z ovládání směru od řidiče ECU vyhodnotí, jaký převodový stupeň se má zařadit. Automatická převodovka **řadí velice rychle** a nedochází tak ke zbytečným propadům výkonu, jako tomu je u manuální převodovky. Některé převodovky jsou vybaveny navíc sportovním režimem, který poskytuje ostřejší a ještě rychlejší odezvu. Motor se nechává vytočit výrazně více a i samotné řazení je více agresivnější. Dále existuje manuální režim, kdy je řidič schopen příkazy plus (+) a minus (-) řídit, jaký převodový stupeň se má zařadit. Ten je zařazen, pokud ho elektronika vyhodnotí jako možný. V opačném případě zůstane zařazený předchozí stupeň. Ovládání je někdy také realizováno ovladači na volantu. Toto ovládací prvek se někdy označuje jako tzv. „pádla“.

Výhody a nevýhody

Hlavní výhodou automatické převodovky je přenos výkonu, kdy při přeřazování převodových stupňů nedochází ke ztrátám výkonu. Celá jízda je tak výrazně **pohodlnější**. Starší automaty vykazovaly o něco vyšší spotřebu, ale rozdíl spotřeby mezi novějšími a modernějšími automaty a manuální převodovkou je již téměř zanedbatelný. Další výhodou jsou rozjezdy do kopce a popojíždění v koloně, kdy se automatika o vše postará a nedochází tak k nadměrnému opotřebení komponent. Nicméně **nákladnější údržba** platí stále. Především je důležité dodržovat předepsané termíny výměny oleje, jinak majitel riskuje kolaps mnohem dražších komponent ne-li celé převodové skříně. Dalším omezením je, že řidič není schopen kontrolovat celý proces řazení, jako je tomu v případě manuální převodovky.

2.4.3 Převodovka EL40 TraXon

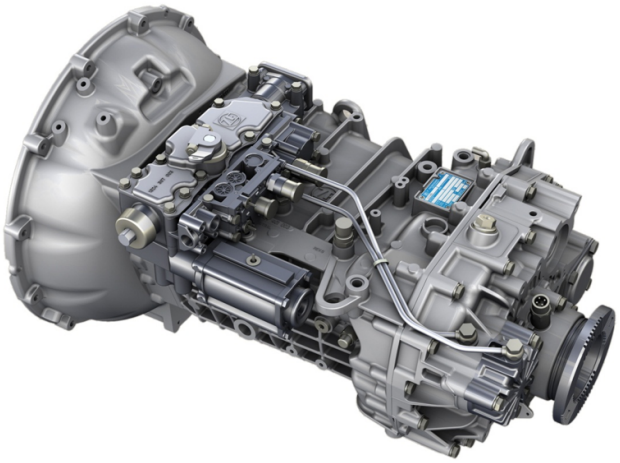
Jedná se o automatickou převodovku s variabilní konstrukcí, to znamená, že může být rozšířena o další funkční prvky. Jedná se například o elektromotor nebo pohon čerpadla a další. Má 12-16 dopředných rychlostí a 4 zpětné rychlosti. Do sériové produkce byla nasazena v roce 2015 a měla by nahradit ZF převodovku ASTronic. Jako Médium pro ovládání akčních členů převodovky je použit vzduch. Mezi zákazníky, kteří tuto převodovku používají, patří například: MAN, IVECO, DAF a další. Na jejím vývoji se podílí tým v Plzni a další týmy ve Friedrichshafenu.



Obrázek 17: Zobrazení těla převodovky EL40 TraXon s doplňkovými modulárními díly

2.4.4 Převodovka EL57 OptiDrive Ecomid

Jedná se o levnější variantu převodovky určenou pro neevropské trhy. Má 8 rychlostí vpřed a 1 rychlost zpět. Její konstrukce je založena na manuální ZF převodovce Ecomid. O automatizaci se postarala firma Westinghouse Air Brake Company (WABCO) a byla použita Andorra ECU. Jedná se tak o robotizovanou převodovku. Médiem je také vzduch. Vývoj SW pro tuto převodovku vznikl a je celý v režii plzeňského týmu.



Obrázek 18: ZF převodovka EL57 OptiDrive Ecomid

3 Systém spojky automatické převodovky

Celá tato kapitola se bude věnovat popisu systému spojky automatické převodovky a to jak z pohledu matematického popisu, tak z pohledu jednotlivých strategií řízení. Podkapitola 3.1 se bude zabývat teoretickým úvodem a popsáním základních principů. Podkapitola 3.2 se bude věnovat matematickému a fyzikálnímu popisu systému spojky. V podkapitole 3.3 bude uveden stavový popis systému. Podkapitola 3.4 bude popisovat použité strategie řízení automatické převodovky. Budou v ní popsány jednotlivé módy řízení spojky a použité regulátory. Celá kapitola byly vypracována s pomocí interních materiálů ZF [25].

3.1 Úvod

Jak již bylo nastíněno v předchozích kapitolách, spojku s převodovou skříní lze popsat jako systém ozubených kol, hřídelí, planetových převodovek a dalších komponent. Tento celý mechanizmus je uložen v převodové skříni. Na jednom konci je spojka, která je připojena k motoru. Její funkce je přenášení točivého momentu generovaného motorem. Na druhé straně se tento točivý moment přenáší přes převodové stupně na výstupní hřídel, která poté roztáčí kola. V dalších odstavcích bude popsána převodovka typu EL40 TraXon.

Řízení spojky automatické převodovky probíhá pomocí **čtveřice ventilů**, které pracují buď se vzduchem, anebo olejem. To záleží na typu a velikosti převodovky. Olejová náplň je dražší, ale díky nestlačitelnosti kapalin je následné řízení převodovky a pohybu spojky mnohem snáze realizovatelnější. Vzduchová náplň je levnější, ale následné řízení je mnohem komplikovanější. Pomocí těchto ventilů dochází k řízení pohybu spojky, k jejímu zavírání nebo otevírání. Jedná se o jeden velký ventil a jeden malý ventil pro otevírání spojky a opět jeden velký a malý ventil pro zavírání spojky. Tato konfigurace se často označuje jako **„split range control“**. Jedná se o regulaci s rozdělenou akční veličinou. Používá se v případě, kdy jsou použity dva akční členy. Obvykle jeden jemný s velkým rozlišením a druhý hrubý s menším rozlišením. Spojení takovýchto akčních členů do jednoho celku umožňuje zvýšit přesnost a rychlost regulace při malých hodnotách akční veličiny. Při řízení polohy spojky může dojít k několika situacím. Pokud se změní požadovaná poloha spojky jen nepatrně, tak se o regulaci polohy postará malý ventil. Pokud je změna větší, tak dojde k aktivaci velkého ventilu. A pokud je požadovaná změna ještě větší, tak může dojít k aktivování obou ventilů současně. To poskytuje maximální možnou změnu, aby přesun z aktuální polohy do té požadované proběhl v co nejkratším čase. To platí jak pro otevírání spojky, tak pro její zavírání. Rozdíl je pouze v tom, že jednou dochází k dodávání objemu média do systému a podruhé k jeho odebírání. Dochází ke změně hmotnostního toku. Ventily budou podrobněji popsány v následujících kapitolách.

Je důležité uvést, že SW pro řízení automatické spojky se neskládá pouze z jedné komponenty, ale z velikého množství dalších komponent. Jedná se například o části: základního SW, který se stará o bios a bootování, modul pro výběr finálního zařazeného stupně, který vymezuje rozsahy povolených rychlostí, modul pro plánování přidělování strojového času, který řídí periodu spouštění jednotlivých komponent, komponentu regulátoru spojky a mnoho dalších. Není tedy možné, aby většina výpočetního výkonu připadla pouze na jednu komponentu. Komponenta regulátoru spojky se spouští s periodou **5 ms**.

3.2 Popis systému

Symbol označení	Význam	Jednotka
F_k	rozpojovací síla	$[N]$
F_c	Coulombická třecí síla	$[N]$
c	tuhost pístu spojky	$[N/s]$
b	tlumení pístu spojky	$[Ns/m]$
t	čas	$[s]$
s	aktuální velikost rozpojení	$[m]$
v	rychlost rozpojení	$[m/s]$
a	zrychlení rozpojování	$[m/s^2]$
s_z	bod zavřené spojky	$[m]$
s_o	bod otevřené spojky	$[m]$
s_d	bod dotyku spojky	$[m]$
s_p	požadovaná poloha spojky	$[m]$
l	rozsah pohybu spojky	$[m]$
A	povrch pístu	$[m^2]$
A_j	průřezová plocha výstupního otvoru	$[m^2]$
V_0	zbytkový objem média	$[m^3]$
V	objem média	$[m^3]$
ν	specifický objem	$[\frac{m^3}{kg}]$
p_0	klidový tlak	$[Pa]$
p_1	vstupující tlak	$[Pa]$
p_2	odcházející tlak	$[Pa]$
p_s	vnitřní tlak válce	$[Pa]$
ϵ	strmost třecí charakteristiky	$[Pa]$
$\dot{p_s}$	časová derivace vnitřního tlaku válce	$[Pa/s]$
R_s	specifická plynová konstanta	$[\frac{J}{kgK}]$
T_s	absolutní teplota uvnitř spojky	$[K]$
T_1	absolutní teplota uvnitř spojky v klidu	$[K]$
T_1	absolutní teplota vstupujícího média	$[K]$
T_2	absolutní teplota odcházejícího média	$[K]$
m	hmotnost média	$[kg]$
$\dot{m}_1$	vstupující hmotnostní tok média	$[kg/s^2]$
$\dot{m}_2$	odcházející hmotností tok média	$[kg/s^2]$
ρ_j	hmotnostní hustota	$[\frac{kg}{m^3}]$
C	pneumatická vodivost podle ISO 6358	$[\frac{l}{s}]$
Φ	výtoková funkce	-
k	isentropický exponent	-
α_D	korekční faktor	-
τ_1, τ_2, τ_3	parametry systému	-
K	zesílení systému	-
T_t	koeficient dopravního zpoždění	-

Tabulka 1: Tabulka popisu fyzikálních veličin pro popis systému spojky

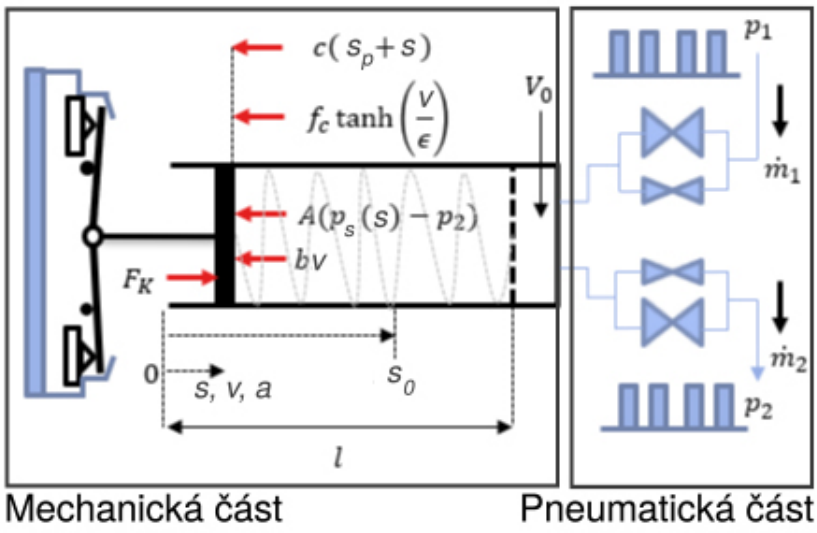
V této podkapitole budou popsány diferenciální rovnice našeho systému a na závěr bude odvozen stavový popis systému.

Soupis fyzikálních proměnných, které jsou použity při popisu systému spojky, je pro přehled uveden v tabule 1. Ke každé fyzikální veličině je uveden matematický symbol a fyzikální jednotka. Některé parametry jsou závislé na čase, ale pro přehlednost to není v tabulce uvedeno. Systém spojky automatické převodovky lze rozdělit na **mechanickou** a **pneumatickou** část. Mechanická část reprezentuje síly, které působí na mechanický píst spojky aumožňují tak pohyb spojky z polohy otevřeno do polohy zavřeno. Pneumatická část převodovky reprezentuje proudění vzduchu nebo kapaliny uvnitř spojky a ventily, přes které dochází ke změnám hmotnostního toku. Fyzikální rovnice popisující pneumatickou část jsou silně nelineární. Vstupní veličinou systému je hmotnostní tok a výstupem je výsledná pozice spojky. Pozice spojky se v případě převodovky EL40 TraXon pohybuje od 15 mm do 28 mm, kdy nižší hodnota odpovídá otevřené spojce a větší hodnota spojce zavřené.

Hodnoty jednotlivých parametrů byly změřeny na speciální zkušební platformě nebo experimentálně ve vozidle. Naměřené hodnoty byly poté zpětně identifikovány na základě experimentálního měření, aby se dosáhlo hladkých průběhů a odstranily se tak šumy.

3.2.1 Mechanická část

Mechanická část popisuje polohu samotné spojky. Spojka je tvořena přítlačným diskem, který umožňuje přenášení točivého momentu. Tento pohyb je zajišťován pístem, jehož pozice určuje pozici spojky. Zjednodušeně řečeno, zda bude spojka otevřená nebo uzavřená. Tato část systému je znázorněna na obrázku 19, dále je na něm zobrazena také pneumatická část systému.



Obrázek 19: Schéma mechanické a pneumatické části systému spojky

Na obrázku 19 jsou znázorněny matematické vztahy. Rozpojovací síla F_k označuje, jak velikou sílu je nutné vytvořit, aby došlo k pohybu spojky. Je patrné, že proti rozpojo-

vací síle F_k působí tlak, tedy tlaková síla dodávaná do systému pomocí ventilů společně s ostatními prvky systému. Člen rozdílu tlaků vynásobený plochou pístu A představuje tlakovou sílu. Člen s v vynásobený parametrem b představuje tření. Člen funkce tanh vynásobený Coulombickou třecí sílou F_c představuje model hystereze. Jedná se o symetrickou hysterezi. Člen součtu poloh s_p a s vynásobený parametrem c představuje tuhost pohybu pístu. Parametr V_0 na obrázku 19 představuje zbytkový objem média v systému spojky.

$$\begin{aligned} A \cdot (p_s - p_2) &= \text{tlaková síla} \\ b \cdot v &= \text{tření} \\ F_c \tanh\left(\frac{v}{\epsilon}\right) &= \text{model hystereze} \\ c \cdot (s_p + s) &= \text{tuhost pohybu pístu} \end{aligned} \tag{1}$$

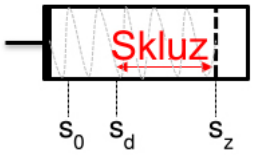
Rozdíl rozpojovací síly a hmotnosti média ve spojce m vynásobené zrychlením rozpojování a má tedy tvar

$$F_k - ma = A \cdot (p_s - p_2) + bv + f_c \tanh\left(\frac{v}{\epsilon}\right) + c \cdot (s_p + s). \tag{2}$$

Následně si vyjádříme zrychlení pro rozpojování spojky a získáme výsledný tvar rovnice pro mechanickou část systému

$$a = \frac{1}{m} \cdot \left[F_K - A \cdot (p_s - p_2) - bv - f_c \tanh\left(\frac{v}{\epsilon}\right) - c \cdot (s_p + s)\right]. \tag{3}$$

Cílem této části je tedy kompenzovat rozpojovací sílu F_k. V případě rovnosti působících sil nedochází k pohybu spojky, a tak setrvává v dané pozice. Spojka může během svého pohybu dosáhnout několika specifických bodů. Důležité body polohy spojky jsou znázorněny na obrázku 20. Poloha s_o značí polohu **otevřené spojky**, s_d značí **bod dotyku** někdy označovaný také jako „touch point“. Bod dotyku představuje takovou pozici spojky, kdy začíná docházet k přenášení točivého momentu mezi motorem a hřídelí kol. Bod s_z značí **bod uzavřené spojky**. Tyto tři pozice mají významnou roli i v řízení celého systému. Podrobněji je to rozebráno v kapitole 3.4.



Obrázek 20: Významné body polohy spojky

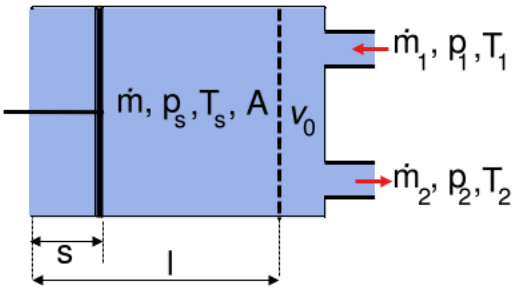
3.2.2 Pneumatická část

Pneumatická část popisuje chování **ventilů**, tedy derivaci hmotnostního toku $\dot{m}$ a derivaci vnitřního tlaku $\dot{p}_s$ v systému. V případě vyjádření $\dot{p}_s$ se vychází z derivace obecné rovnice plynu, někdy označované jako stavová rovnice. Na základě velikosti hmotnostního toku dochází k aktivaci ventilů, což vede k otevírání nebo zavírání spojky. V případě přivádění daného média dochází k otevírání spojky. Naopak při odebírání média bude docházet

k zavírání spojky. Před samotným odvozením budou nejprve uvedeny počáteční hodnoty některých parametrů, které jsou použity při odvozování. Jednotlivé parametry jsou blíže popsány v tabulce 1.

$$\begin{aligned} p_0 &= 1 \quad \text{bar} \\ T_0 &= 293 \quad \text{K} \\ R_s &= 286,9 \quad \frac{\text{J}}{\text{kgK}} \end{aligned}$$

Schéma pneumatické části systému je znázorněno na obrázku 21.



Obrázek 21: Schéma pneumatické části systému spojky

Tato část systému je zodpovědná za přivedení nebo odebrání daného množství vzduchu nebo kapaliny do systému spojky. Nyní bude provedeno odvození rovnic pro pneumatickou část systému. Při odvozování byl uvažován ideální plyn. Dále bylo uvažováno, že celý proces je isentropický, tedy že entropie zůstává zachována. Entropie představuje míru neurčitosti procesu. V praxi se však používá polytropická komprese, poněvadž lépe popisuje reaálné děje. **Stavová rovnice plynu** vyjadřuje vztah mezi tlakem p_S a objemem uvnitř válce V ve vztahu k hmotnosti daného plynu m, ke kterému se vztahuje měrná hustota plynu R_s, za dané teploty T_s.

$$p_s V = m R_s T_s \tag{4}$$

Dále se definuje specifický objem ν, který vyjadřuje vztah mezi objemem a hmotností.

$$\nu = \frac{V}{m} \tag{5}$$

Dosazením vztahu (5) do rovnice (4) se získá tvar

$$p_s \nu = R_s T_s. \tag{6}$$

Z rovnice (6) se vyjde při odvozování rovnice pro změnu isentropického stavu. Zde bude zaveden isentropický exponent k. Při výpočtech byla jeho hodnota zavedena velice blízka jedničce. Rozšířená rovnice má potom tvar

$$\begin{aligned} p_s \nu^k &= m R_s T_s, \\ p_s V^k &= m^k R_s T_s. \end{aligned} \tag{7}$$

Celkový objem uvnitř spojky lze definovat následujícím vztahem, kde parametr A představuje plochu pístu spojky, l značí celkovou vzdálenost, kterou může píst spojky urazit, a s značí aktuální polohu spojky. V_0 pak značí zbytkový objem uvnitř válce spojky.

$$V = A \cdot (l - s) + V_0 \tag{8}$$

Časová derivace celkového objemu $\dot{V}$ vychází z rovnice (8) jako

$$\dot{V} = -Av. \tag{9}$$

Nyní bude provedena časová derivace rovnice (7).

$$\dot{p}_s V^k + k p_s \dot{V} V^{(k-1)} = k\dot{m} m^{k-1} R_s T_s \tag{10}$$

Budou provedeny úpravy a dosazeni rovnice (9) do rovnice (10).

$$\dot{p}_s V^k = k p_s A v V^{(k-1)} + k\dot{m} m^{k-1} R_s T_s \tag{11}$$

Pro následující úpravy je nutné uvést vztahy, které jsou odvozeny na základě rovnic (4) a (7).

$$\begin{aligned} \frac{m^k}{V^k} &= \frac{p_s}{R_s T_s} \\ \frac{p_s}{m} &= \frac{R_S T_s}{V} \end{aligned} \tag{12}$$

Na základě vztahů (12) provedeme dosazení a úpravy rovnice (11).

$$\begin{aligned} \dot{p}_s &= \frac{k p_s A v V^{k-1}}{V^k} + \frac{k\dot{m} m^{k-1} R_s T_s}{V^k} \\ \dot{p}_s &= \frac{k p_s A}{V} v + k\dot{m} \frac{m^k}{V^k} \frac{1}{m} R_s T_s \\ \dot{p}_s &= \frac{k p_s A}{V} v + k\dot{m} \frac{p_s}{\cancel{R_s T_s}} \frac{1}{m} \cancel{R_s T_s} \\ \dot{p}_s &= \frac{k p_s A}{V} v + k\dot{m} \frac{p_s}{m} \\ \dot{p}_s &= \frac{k p_s A}{V} v + \frac{k R_s T_s}{V} \dot{m} \end{aligned} \tag{13}$$

Do výsledku odvození (13) bude dosazen tvar pro V z rovnice (8). Bude tak získán výsledný vztah popisující pneumatickou část systému. Dále je následně uvedena rovnice vyjadřující vztah pro $\dot{m}$. Dále budou odvozeny vztahy pro výpočet jednotlivých složek $\dot{m}$, tedy vztahy pro derivace hmotnostního toku $\dot{m}_1$ a $\dot{m}_2$. Pro přehlednost budou indexy vypuštěny a odvození bude nejprve provedeno obecně a po získání výsledného tvaru bude doplněn o jednotlivé indexy pro otevírání a zavírání spojky.

$$\begin{aligned} \dot{p}_s &= \frac{k p_s A}{A \cdot (l - s) + V_0} v + \frac{\cancel{R}_S T_s}{A \cdot (l - s) + V_0} \dot{m} \\ \dot{m} &= \dot{m}_1 - \dot{m}_2 \end{aligned} \tag{14}$$

Výpočet derivace hmotnostního toku $\dot{m}$ vychází z následující rovnice, kde A_j značí průřezovou plochu výstupního otvoru, ρ_j značí hmotnostní hustotu a v_j představuje rychlost. Index j značí jeden bod a index i značí další bod.

$$\dot{m} = A_j \rho_j v_j \tag{15}$$

Pro následující úpravy je nutné uvést vztahy pro v_j a podíl hustot pro $\frac{\rho_j}{\rho_i}$.

$$\begin{aligned} \frac{\rho_j}{\rho_i} &= \frac{p_j}{p_i}^{\frac{1}{k}} \\ v_j &= \sqrt{2\frac{p_i}{\rho_i} - \frac{k}{k-1}\Big(1 - \Big(\frac{p_j}{p_i}^{\frac{k-1}{k}}\Big)\Big)} \end{aligned} \tag{16}$$

Dále bude provedeno dosazení na základě rovnic (16) do rovnice (15).

$$\dot{m} = A_j \rho_i \Big(\frac{p_j}{p_i}\Big)^{\frac{1}{k}} \sqrt{2\frac{p_i}{\rho_i} - \frac{k}{k-1}\Big(1 - \Big(\frac{p_j}{p_i}^{\frac{k-1}{k}}\Big)\Big)} \tag{17}$$

Rovnici (17) upravíme do tvaru

$$\dot{m} = A_j \sqrt{2 p_i \rho_i} \cdot \sqrt{\frac{k}{k-1}\Big(\frac{p_j}{p_i}\Big)^{\frac{2}{k}} - \Big(\frac{p_j}{p_i}\Big)^{\frac{k+1}{k}}}. \tag{18}$$

Z rovnice (18) lze část označit jako $\Phi\Big(\frac{p_j}{p_i}\Big)$. Tato část má pak následující tvar

$$\Phi\Big(\frac{p_j}{p_i}\Big) = \sqrt{\frac{k}{k-1}\Big(\frac{p_j}{p_i}\Big)^{\frac{2}{k}} - \Big(\frac{p_j}{p_i}\Big)^{\frac{k+1}{k}}}. \tag{19}$$

Hmotnostní hustotu ρ_i lze vyjádřit jako

$$\rho_i = \frac{p_i}{R_s T_i}. \tag{20}$$

Dále bude provedeno dosazení vztahu pro ρ_i (20) a vztahu z rovnice (19) do rovnice (18).

$$\dot{m} = A_j p_i \sqrt{\frac{2}{R_S T_i}} \Phi\Big(\frac{p_j}{p_i}\Big) \tag{21}$$

Rovnici (21) lze přepsat do následujícího tvaru

$$\dot{m} = \alpha_D A_j \Phi_{max} p_i \sqrt{\frac{2}{R_S T_i}} = C p_i \rho_0 \sqrt{\frac{T_0}{T_1}}. \tag{22}$$

Kde parametr α_D značí korekční faktor. C představuje pneumatickou vodivost ventilů. Převod Φ na Φ_{max} značí eliptickou aproximaci. A parametr ρ_0 vyjadřuje hmotnostní hustotu za normálních podmínek. Jejich tvary jsou následující

$$\begin{aligned} \rho_0 &= \frac{p_0}{R_S T_0}, \\ c &= \frac{\alpha_d A_j \Phi_{max} \sqrt{2 R_s T_0}}{p_0}, \\ \Phi\Big(\frac{p_j}{p_i}\Big) &\approx \Phi_{max} \sqrt{\Big(1 - \Big(\frac{\frac{p_j}{p_i} - b}{1-b}\Big)^2\Big)}. \end{aligned} \tag{23}$$

Po dosazení vztahů uvedených ve (23) do rovnice (22) získáme výsledný vztah pro derivaci hmotnostního toku $\dot{m}$, který má následující tvar

$$\dot{m} = Cp_i\rho_0\sqrt{\frac{T_0}{T_1}} \cdot \begin{cases} \sqrt{\left(1 - \left(\frac{\frac{p_j}{p_i} - b}{1-b}\right)^2\right)}, & b \leq \frac{p_j}{p_i} \leq 1, \\ 1, & 0 \leq \frac{p_j}{p_i} \leq b. \end{cases} \tag{24}$$

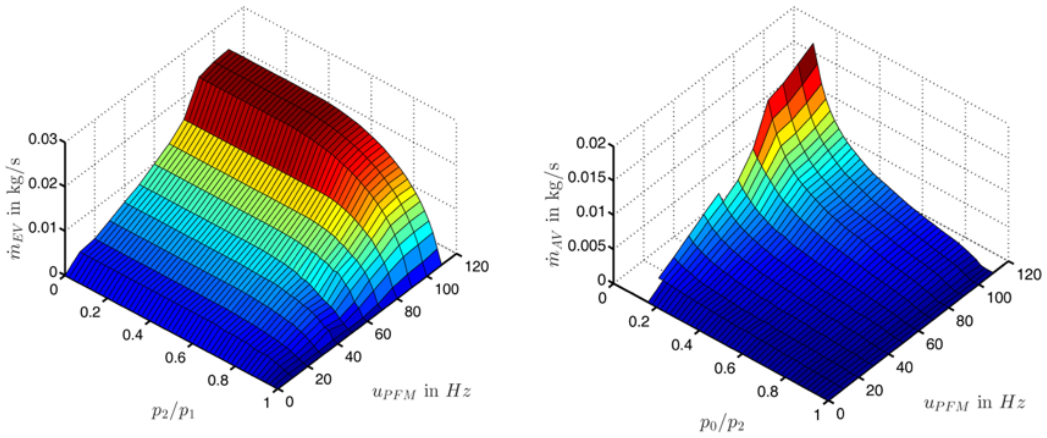
Rovnice (24) tedy udává vztah, který se používá pro výpočet derivace hmotnostního toku $\dot{m}$. Z rovnice je patrné, že mohou nastat dvě možnosti v závislosti na velikosti podílu tlaků p_j a p_i. Pokud je podíl tlaků větší nebo roven než konstanta tlumení b, ale menší nebo roven jedné, tak se pro výpočet použije tvar s odmocninou. Pokud je podíl tlaků větší nebo roven nule, ale menší nebo rovno než konstanta tlumení b, tak se použije pro výpočet tvar s jedničkou. Jak již bylo popsáno, tak v systému může docházet k otevírání nebo zavírání. Podle směru se změní i použité proměnné v rovnici (24). Nejprve bude uvedena rovnice $\dot{m}$ pro směr **otevírání spojky**, tedy přivádění media do systému

$$\begin{aligned} \dot{m} = \dot{m}_1 &= Cp_1\rho_0\sqrt{\frac{T_0}{T_1}} \cdot \sqrt{\left(1 - \left(\frac{\frac{p_s}{p_1} - b}{1-b}\right)^2\right)}, \quad \text{pro} \quad b \leq \frac{p_s}{p_1} \leq 1, \\ \dot{m} = \dot{m}_1 &= Cp_1\rho_0\sqrt{\frac{T_0}{T_1}} \cdot 1 \quad \text{, pro} \quad 0 \leq \frac{p_s}{p_1} \leq b. \end{aligned} \tag{25}$$

Dále rovnice $\dot{m}$ pro směr **zavírání spojky**, tedy odvádění media ze systému

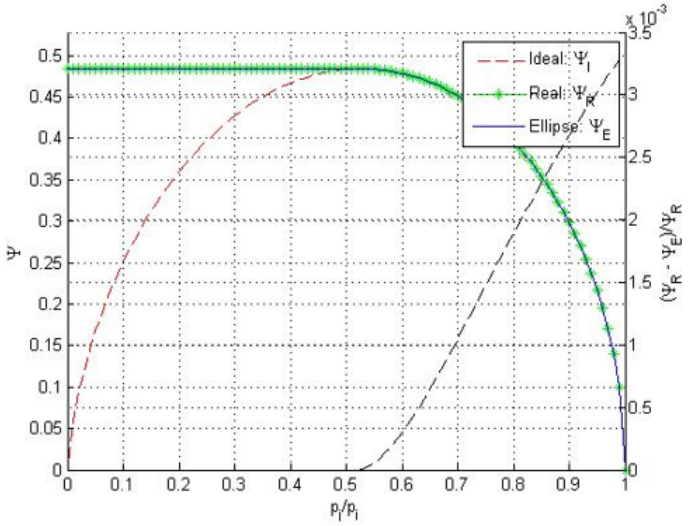
$$\begin{aligned} \dot{m} = -\dot{m}_2 &= -Cp_s\rho_0\sqrt{\frac{T_0}{T_s}} \cdot \sqrt{\left(1 - \left(\frac{\frac{p_2}{p_s} - b}{1-b}\right)^2\right)}, \quad \text{pro} \quad b \leq \frac{p_2}{p_s} \leq 1, \\ \dot{m} = -\dot{m}_2 &= -Cp_s\rho_0\sqrt{\frac{T_0}{T_s}} \cdot 1, \quad \text{pro} \quad 0 \leq \frac{p_2}{p_s} \leq b. \end{aligned} \tag{26}$$

Na základě rozdílu tlaků v systému a frekvenci pulzně šířkové modulace (PWM) spínání ventilů jsme získali výsledné hodnoty $\dot{m}$. Na obrázku 22 jsou uvedeny odměřené charakteristiky ventilů pro otevírání spojky (vlevo) a zavírání spojky (vpravo).



Obrázek 22: Charakteristiky ventilů pro derivaci hmotnostního toku $\dot{m}$, vlevo pro otevírání spojky, vpravo pro zavírání spojky [25]

Z grafů na obrázku 22 a rovnic (24) a (25) je patrné, že **charakteristika ventilů** má lineární část a nelineární část, kterou lze označit jako eliptickou část. Společnost ZF ve spolupráci s univerzitou v Německém Rostocku vytvořila algoritmus, který tyto informace převede do 2D charakteristiky, která se používá pro reprezentaci chování ventilů ve výsledném modelu a SW. Jedná se o firemní tajemství a patent ZF. Výsledná charakteristika je znázorněna na obrázku 23, kde Ψ značí hmotnostní tok v závislosti na rozdílu tlaků.



Obrázek 23: 2D charakteristika ventilů spojky pro hmotnostní tok [25]

3.3 Stavový popis

V této podkapitole bude uveden stavový popis systému spojky. Budou zde nastíněny vztahy, které jsou použité při výpočtu regulátoru pro malé i velké změny. Vektor stavu $\boldsymbol{x}$ je definován jako

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} s \\ v \\ p_s \end{bmatrix}. \tag{27}$$

Nechť $\dot{\boldsymbol{x}}$ je derivace stavu $\boldsymbol{x}$. Poté lze rovnici (27) přepsat do tvaru

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \end{bmatrix} = \begin{bmatrix} v \\ a \\ \dot{p_s} \end{bmatrix}, \quad kde \quad \dot{x_1} = x_2. \tag{28}$$

Po dosazení vznikne výsledný tvar stavového popisu

$$\begin{bmatrix} v \\ a \\ \dot{p_s} \end{bmatrix} = \begin{bmatrix} v \\ \frac{1}{m}[-ba - f_c \cdot tanh(\frac{a}{\epsilon}) - c(s_p - s) - A(p_s - p_2) + F_k] \\ \frac{k}{l - s + \frac{V_0}{A}} \cdot p_s v \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \frac{1}{A} \cdot \frac{kR_sT_s}{l - s + \frac{V_0}{A}} \end{bmatrix} \dot{m}, \tag{29}$$

$$y = s.$$

Výstupem systému je pozice spojky. Jedná se o nelineární stavový popis, který může být pro jednoduchost vyjádřen tvarem

$$\begin{aligned} \dot{\boldsymbol{x}} &= f(\boldsymbol{x}) + g(\boldsymbol{x}) \cdot u, \quad kde \quad u = \dot{m}, \\ y &= s. \end{aligned} \tag{30}$$

Kde $u = \dot{m}$ je vstupem systému a reprezentuje derivaci hmotnostního toku. A kde $f(\boldsymbol{x})$ a $g(\boldsymbol{x})$ mají stejný tvar jako v rovnici (29). Jeden konkrétní pracovní bod i je definován jako

$$\boldsymbol{x}_i = [s_i, v_i, p_{si}]. \tag{31}$$

Linearizace stavového popisu (29) pro jeden pracovní bod i se provede podle následujícího vztahu

$$\begin{aligned} \Delta\dot{\boldsymbol{x}} &= \frac{\delta f(\boldsymbol{x})}{\delta \boldsymbol{x}}\Delta\boldsymbol{x} + u\frac{\delta g(\boldsymbol{x})}{\delta \boldsymbol{x}}\Delta\boldsymbol{x} + \frac{\delta f(\boldsymbol{x})}{\delta u}\Delta u + g(\boldsymbol{x})\frac{\delta u}{\delta u}\Delta u, \\ \Delta\dot{\boldsymbol{x}} &= \Big[\frac{\delta f(\boldsymbol{x})}{\delta \boldsymbol{x}} + u\frac{\delta g(\boldsymbol{x})}{\delta \boldsymbol{x}}\Big]_i\Delta\boldsymbol{x} + g(\boldsymbol{x})\Big|_i\Delta u. \end{aligned} \tag{32}$$

Linearizované matice A a B pro jeden pracovní bod jsou potom ve tvaru

$$\begin{aligned} A &= \left[\begin{bmatrix} 0 & 1 & 0 \\ \frac{1}{m}(-c + \frac{\delta F_k}{\delta s}) & \frac{1}{m}[-b - \frac{f_c}{\epsilon}(1 + \tanh^2(\frac{v}{\epsilon}))] & \frac{-A}{m} \\ \frac{kp_s v}{(l - s + \frac{V_0}{A})^2} & \frac{\kappa}{l - s + \frac{V_0}{A}}p_s & \frac{k}{l - s + \frac{V_0}{A}}v \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ \frac{1}{A} \cdot \frac{kR_sT_s}{(l - s + \frac{V_0}{A})^2} & 0 & 0 \end{bmatrix} u\right], \\ B &= \begin{bmatrix} 0 \\ 0 \\ \frac{1}{A} \cdot \frac{kR_sT_s}{l - s + \frac{V_0}{A}} \end{bmatrix}. \end{aligned} \tag{33}$$

Z linearizovaného stavového popisu se vyjádří přenosová funkce, která je použita pro návrh regulátoru v daném pracovním bodě i. Výsledná přenosová funkce linearizovaného systému, včetně uvažování pásma necitlivosti, je

$$Y(s) = e^{-sT_t} \cdot \frac{K}{\tau_3 s^3 + \tau_2 s^2 + \tau_1 s}. \tag{34}$$

Parametry $\tau_1, \tau_2, \tau_3, T_t$ a K jsou určeny podle daného pracovního bodu. Při posunu po mechanické křivce dochází k jejich neustálému přepočítávání. Tento přenos je použit při syntéze regulátorů.

3.4 Řízení spojky automatické převodovky

V této kapitole bude nastíněn princip činnosti a funkce regulátoru spojky. Z důvodu zachování firemního tajemství zde však nebude popsána přesná implementace a odvození řízení, avšak dílčí podkapitoly se zaměří na obecný popis strategie řízení spojky. Systém řízení pohybu spojky funguje v několika režimech, o kterých rozhoduje stavový automat. Samotné řízení má lineární část, která se označuje jako **„regulátor pro malé změny“**. Dále se tu vyskytuje **„regulátor pro velké změny“**, který má za úkol při zvětšené chybě regulace e přejít co nejrychleji do dalšího pracovního bodu. Pracovní body jsou reprezentovány konečnou množinou bodů na křivce, kterou systém sleduje. Nový pracovní bod je definován cílovou pozicí spojky. Řízení takovéhoto systému je komplexní problém. V tabulce 2 se nachází popis jednotlivých veličin použitých pro popis regulace [25].

Symbol označení	Význam
K_p	proporcionální člen regulátoru
K_d	zesílení derivačního členu regulátoru
K_i	zesílení integračního členu regulátoru
T_i	integrační časová konstanta
T_d	derivační časová konstanta
T_v	časová konstanta čitatele PD regulátoru
T_p	časová konstanta jmenovatele PD regulátoru
e	regulační odchylka
$p(i)$	tlak v pracovním bodě i
P	tlaková síla
$s(i)$	poloha spojky v pracovním bodě i
v_p	požadovaná rychlost rozpojování spojky

Tabulka 2: Tabulka použitých veličin pro regulaci

Řízení spojky je rozděleno do **tří módů** řízení. Přechod mezi nimi řídí stavový automat. Tří stavový automat na základě vstupu určí, do kterého módu se přejde nebo zda se setrvá v aktuálním módu. Jedná se o módy:

- Vypnuto
- Časový mód
- Mód řízení polohy

3.4.1 Stav vypnuto

Do tohoto módu se přejde po zasunutí klíčku do zapalování automobilu, ještě před nastartováním motoru a nebo po vypnutí motoru. Nic se v něm neděje, neprobíhá žádný výpočet řízení. V momentě nastartování je tento mód opuštěn.

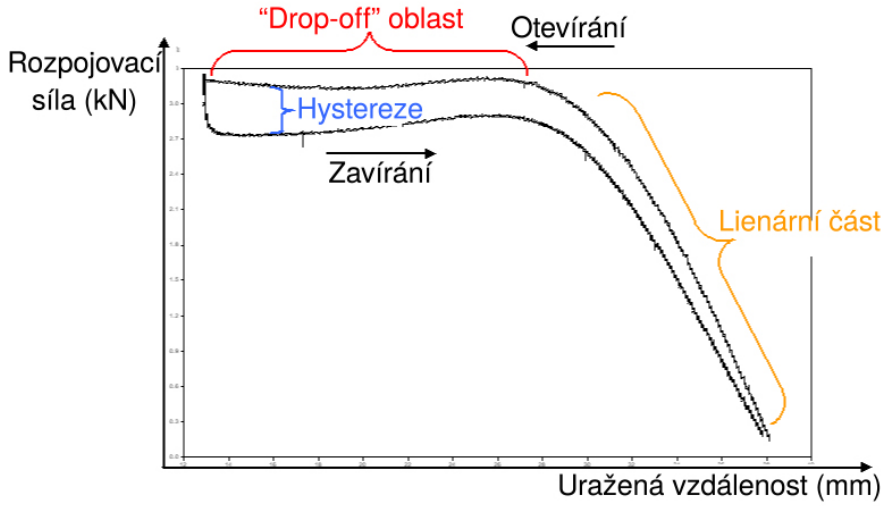
3.4.2 Časový mód

V časovém módu se jedná o řízení v otevřené smyčce. Tento mód se používá pro pohyb spojky v jakémkoliv směru, tedy jak pro otevírání, tak i pro zavírání. Délky pulzů jsou nastaveny parametrem a kódovány v PWM. Frekvence PWM určuje rychlost otevírání a nebo zavírání. V tomto stavu také dochází k takzvanému „rest-closing“. Jedná se o stav, kdy spojka již dosáhla bodu dotyku a nachází se směrem k uzavřené spojce a cílem je přechod do polohy zcela zavřené spojky. Pokud by zde došlo k příliš rychlému zavření, celý automobil by nepříjemně cuknul, což je nežádoucí vzhledem ke komfortu během jízdy.

3.4.3 Mód řízení polohy

V tomto stavu probíhá řízení pomocí uzavřené smyčky. Pro řízení ventilů spojky se používají dvě strategie. Jsou to již zmíněné regulátory pro velké změny a pro malé změny. Regulátor pro malé změny reguluje pozici spojky, pokud je regulační odchylka e menší než 1 mm. Pokud se regulační odchylka zvětší, dojde k přepnutí na regulátor pro velké změny. Ten má za úkol se co nejrychleji dostat k dalšímu pracovnímu bodu. Jakmile se k němu

dostatečně přiblíží, bude opět přepnuto na regulátor pro malé změny. Na obrázku 24 je znázorněna křivka, kterou systém sleduje při otevírání a zavírání spojky. Na základě pozice na křivce, ve které se systém nachází se přepočítávají hodnoty regulátoru K_p a K_d pro otevírací i zavírací směr. Díky tomuto neustálému přepočítávání parametrů nedochází při přepínání regulátorů k rázům. Tato metoda přepínání regulátorů se označuje jako **„gain scheduling“**. „Gain scheduling“ je metoda, která se používá pro řízení nelineárních systémů pomocí přepínaní mezi lineárními regulátory, které poskytují vhodné řízení v okolí daného pracovního bodu. Pro dosažení požadované polohy spojky je nutné kompenzovat rozpojovací sílu F_k, která na spojku působí.



Obrázek 24: Křivka pohybu spojky pro její otevírání a zavírání

Jak již bylo nastíněno v předešlých odstavcích, otevírání nebo zavírání spojky je umožněno čtyřmi ventily. Všechny ventily mají stejnou stavbu a liší se pouze ve velikosti jejich průměru, tedy průtoku. Ventily se řídí pomocí PWM. Pokud jsou pulzy dlouhé, pak se dají ventily popsat lineárně. Pokud ale pulzy PWM rychle kmitají, tak jsme nuceni použít aproximační řešení, které je založené na naměřených hodnotách. Toto řešení se používá zejména kvůli ušetření výpočetní náročnosti na CPU a problémům s vyjádřením teploty.

Pro každý typ regulátoru jsou použity jiné prostředky návrhu. Tvar obecného přenosu pro paralelní realizaci PID regulátoru je [6]

$$G(s) = K_p + \frac{K_i}{s} + K_d s = \frac{K_d s^2 + K_p s + K_i}{s}. \tag{35}$$

Lze také uvažovat tvar [6]

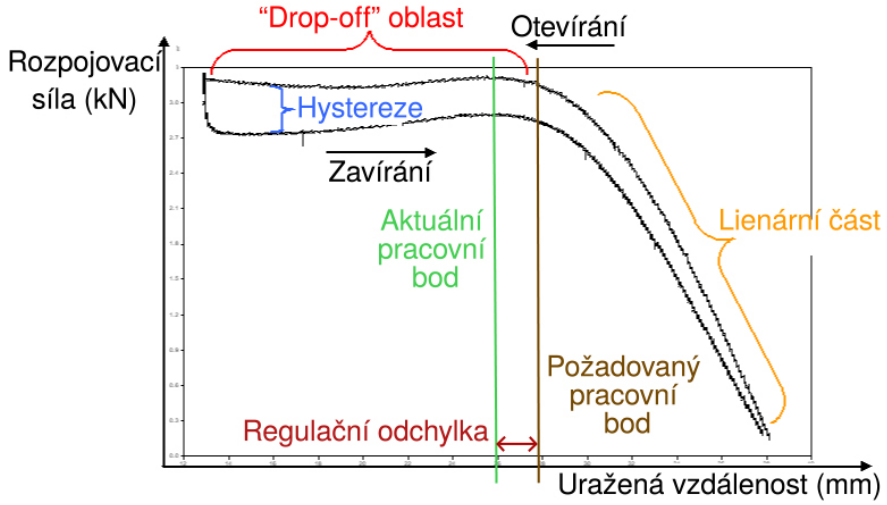
$$G(s) = K_p \cdot (1 + \frac{1}{sT_i} + sT_d). \tag{36}$$

V případě řízení spojky je použit PID regulátor, který je rozšířen o filtraci derivační složky a unášení integrační složky. Integrační složka regulátoru se aktivuje pouze ve chvíli, kdy v celém systému poklesne tlak, kdy tedy dojde k úniku média. Pro řízení systému pak stačí uvažovat pouze PD regulátor. Přenosová funkce získaná pomocí linearizace (33) je

použita pro výpočet parametrů PD regulátoru. Tento postup je využit u regulátoru pro malé změny. Je důležité uvažovat pásmo necitlivosti tzv. „dead time“ , aby se předešlo poškození akčních členů, proto je tento fakt zohledněn při návrhu. Přenosová funkce PD regulátoru $G(s)$, která se používá při návrhu je

$$G(s) = K_p(i) \cdot \frac{T_{v(i)}s + 1}{T_{p(i)}s + 1}. \tag{37}$$

Parametr i představuje jeden konkrétní **pracovní bod**. Systém spojky při svém pohybu sleduje křivku a tato křivka je rozdělena na konečný počet bodů, které reprezentují jednotlivé pracovní body. Pracovním bodem se rozumí aktuální poloha spojky na křivce popisující otevírání a zavírání spojky. V každém pracovním bodě dochází ke změnám parametrů systému a regulátoru. Na obrázku 25 je zeleně znázorněn aktuální pracovní bod. Hnědou barvou je znázorněn požadovaný pracovní bod, tedy s_p. Rozdíl jejich vzdálenosti představuje regulační odchylku e. Parametry T_v a T_p představují časové konstanty a K_p je zesílení. Část podílu s časovými konstantami představuje filtrovanou derivační složku.



Obrázek 25: Vyznačení regulační odchylky, aktuálního a požadovaného pracovního bodu na křivce po níž se spojka pohybuje

Zde se používá ještě předregulace, která se vztahuje k derivaci hmotnostního toku $\dot{m}$. $s_{(i)}$ označuje polohu v pracovním bodě. $p_{(i)}$ představuje tlak v pracovním bodě. v_p vyjadřuje požadovanou rychlost rozpojování. Vzorec použitý pro předregulaci je následující

$$\dot{m} = \dot{p}_{s(i)} \cdot \frac{A(l - s_{(i)}) + V_0}{\kappa R_s T_s} - \frac{p_{s(i)} A}{R_s T_s} v_p. \tag{38}$$

Ve své podstatě se vychází z rovnice pro pneumatickou část systému (14), ze které se vyjádří derivace hmotnostního toku $\dot{m}$. Následně bude vypočítán gradient požadované polohy spojky. Určená hodnota hmotnostního toku se dále převede na frekvenci spínání ventilů pomocí PWM modulace. Tato myšlenka je shodná pro obě verze regulátorů.

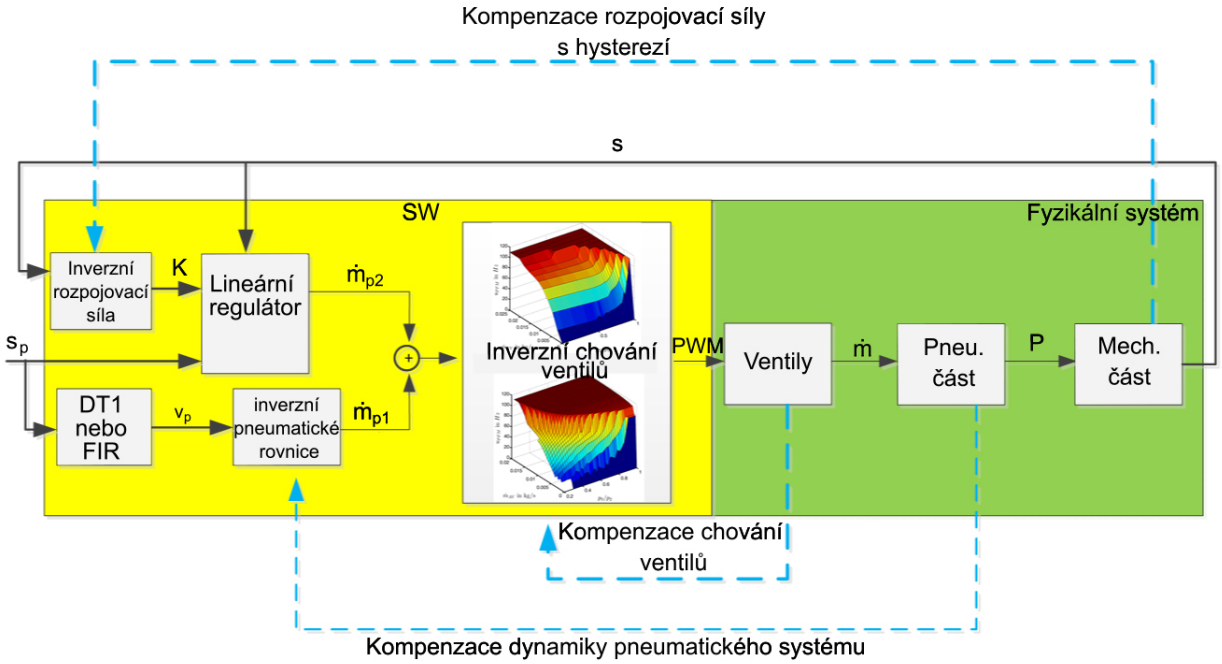
V případě regulátoru pro velké změny se k řízení používá nelineární P regulátor. Jeho předpis je

$$G(s) = \frac{Y(s)}{U(s)} = K_p, \qquad K_p = f(e, F_k). \tag{39}$$

Zesílení K_p závisí na regulační odchylce e a rozpojovací síle F_k. Na základě jejich hodnoty se načte příslušný parametr. Podrobnější popis obou typů řízení je rozepsán v následujících odstavcích.

3.4.3.1 Regulátor pro malé změny

Tato část řízení se aktivuje, pokud stavový automat přejde do módu řízení polohy spojky a regulační odchylka e **je menší než 1 mm**. Na obrázku 26 je znázorněna schématická struktura tohoto řízení.



Obrázek 26: Princip funkce regulátoru pro malé změny

Pro dosažení požadované polohy spojky se musí kompenzovat rozpojovací síla F_k. K tomu slouží inverzní rovnice, do kterých vstupuje aktuální poloha spojky s, což je výstup celého systému. Z fyzikálního modelu spojky sem vstupuje velikost rozpojovací síly společně s hysterezí. Na základě tohoto výpočtu získáme zesílení pro lineární regulátor K. Do regulátoru dále vstupuje požadovaná poloha spojky s_p. Regulátor je ve tvaru PID, ale většinu doby je aktivní pouze PD část. Derivační složka závisí na požadované poloze spojky a je počítána buď pomocí DT1, což odpovídá filtrované derivační složce nebo tato filtrace může být realizována pomocí filtru s konečnou impulzní odezvou (FIR) filtru. Derivace polohy spojky v_p slouží pro kompenzaci dynamiky pneumatického systému a vyjadřuje požadovanou rychlost rozpojování. Výstup z regulátoru $\dot{m}_{p2}$ se sečte s výstupem z inverzních pneumatických rovnic $\dot{m}_{p1}$. Tato hodnota reprezentuje změnu hmotnostního toku v závislosti na čase a slouží jako vstup pro inverzní rovnice ventilů. V nich proběhne výpočet frekvence, kterou se mají spínat ventily, aby bylo dosaženo požadované polohy. Jedná se o PWM modulaci. Signál PWM je použit při realizaci vypočítaných hodnot na základe rovnic (25) a (26). Na základě změny hmotnostního toku $\dot{m}$ dojde ke změně tlakové síly P a díky ní dojde ke změně aktuální polohy spojky s. Tato hodnota se pak použije v následujícím výpočetním kroku řízení.

Pro výpočet parametrů PID regulátoru se používá metoda „gain scheduling“. Na základě parametrů systému se vypočte v, pro výpočet se zde používá požadovaný gradient polohy, tedy trajektorie, kterou je potřeba sledovat. Celý výpočet parametrů regulátoru pro jednotlivé pracovní body je reprezentován tabulkou a to z důvodu náročnosti výpočtu. Všechny možnosti pracovních bodů, a tedy i jednotlivých parametrů, lze určit dopředu. Kombinace PD regulátoru s předregulací se označuje jako „Flatness based controller“ [37]. „Flatness based“ by se dal přeložit jako prostor plochosti/rovnosti. Flatness v teorii systémů reprezentuje takovou vlastnost systému, která rozšiřuje pojem ovladatelnosti z oblasti lineárních systémů na nelineární dynamické systémy. Systém, který má tuto vlastnost, se nazývá „flat“ systém (rovinný, plochý systém). Ploché systémy mají (fiktivní) plochý výstup, který lze použít k explicitnímu vyjádření všech stavů a vstupů z hlediska plochého výstupu a konečného počtu jeho derivací [37].

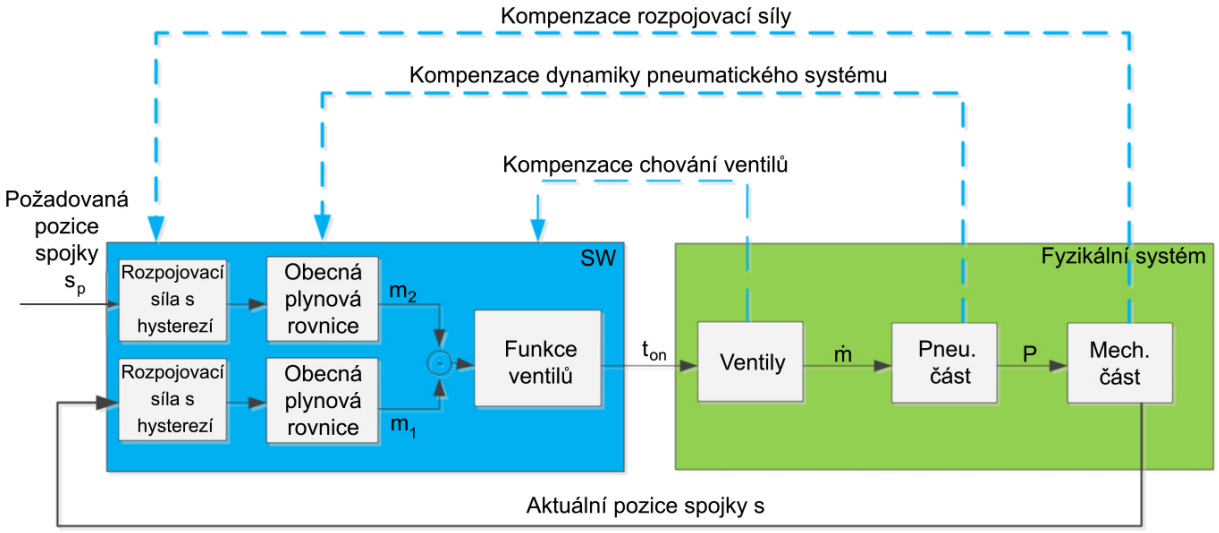
Parametry pro PD regulátor K_p a K_d jsou získávány pomocí Bodeho metody. Jedná se o frekvenční odezvu systému. Bodeho grafem je vyjádřena frekvenční odezva v decibelech a fáze, která popisuje fázový posun systému. Lze z něj určit bezpečnost v zesílení a ve fázi. Kvůli mechanickému tření regulátor zohledňuje nepřesnosti v parametrech systému pomocí pásma necitlivosti.

Dále se v regulátoru pro malé změny používá předregulace řízení pomocí dopředné vazby (36), které na základě požadované rychlosti v_p určí derivaci hmotnostního toku $\dot{m}$. Tento výpočet však v řídicí jednotce není implementován přes fyzikální rovnice, ale místo nich se používají předpočítané hodnoty, které jsou uložené v tabulce a to zejména z důvodu snížení výpočetní náročnosti. Hodnoty v tabulce byly získány na základě znalosti, jak veliký potřebujeme hmotnostní tok, aby došlo k posunu spojky z jednoho pracovního bodu do druhého. Tento požadavek je poté převeden na požadované PWM o dané frekvenci, tedy experimentálně na základě znalosti požadovaného výsledku. K tomuto výpočtu využijeme vodivost ventilů a rychlosti spínání ventilů, které jsou uložené rovněž v tabulce. Kalibrace dopředné vazby probíhá přímo na vozidle, kde vstupem je rychlost vozidla, ta je reprezentována signálem rampy s různým sklonem. Rychlost je tak jedním z parametrů tabulky. Druhým je v každém bodě odměřená vzdálenost k zavřenému bodu spojky s_z.

3.4.3.2 Regulátor pro velké změny

V této části je popsáno řízení v případě, že je regulační odchylka e **větší než 1 mm**. Tento regulátor má za úkol, co nejrychleji dostat spojku do okolí dalšího pracovního bodu. Po dostatečném přiblížení k novému pracovnímu bodu, kdy se regulační odchylka e dostane pod hodnotu 1 mm, se aktivuje opět regulátor pro malé změny. Na obrázku 27 je schéma tohoto řízení.

Vstupem do systému je požadovaná poloha spojky s_p, která vstupuje do subsystému jenž obsahuje rovnice pro kompenzaci rozpojovací síly F_k spolu s hysterezí. Velikost síly je vypočtena na základě fyzikálního modelu mechanické části spojky (3). Vypočtená hodnota slouží dále pro výpočet obecné rovnice plynu (4), která má za úkol kompenzovat dynamiku pneumatické části systému. Zde je vypočteno m_2, tedy jedna část hmotnostního toku. Výpočet druhé části hmotnostního toku m_1 se provádí následovně.



Obrázek 27: Princip funkce regulátoru pro velké změny

Aktuální poloha spojky s vstupuje také do subsystému pro kompenzaci rozpojovací síly F_k spolu s hysterezí. Dále dojde k odečtení hmotnostních toků m_1 a m_2. Obě hodnoty odečítáme proto, že jedna představuje vypočítaný hmotnostní tok pro aktuální pracovní bod a druhá představuje vypočítaný hmotnostní tok pro požadovaný pracovní bod. Na základě toho víme, zda má dojít k dodávání media do systému anebo jeho odebírání, tedy k otevírání nebo zavírání spojky. Tento výpočet poslouží ke kompenzaci chování ventilů, do kterého vstupuje informace z fyzikálního modelu ventilů. Potřebná změna hmotnostního toku $\dot{m}$ je přepočten na **požadovaný čas** t_{ON}, po který budou ventily aktivní a budou tak do systému dodávat a nebo odebírat médium. Vypočtený čas t_{ON} bude přepočten na potřebnou frekvenci spínaní ventilů pomocí PWM modulace. Následně projde výpočet přes fyzikální model ventilů, pneumatickou část systému a mechanickou část, kde dojde k výpočtu výsledné působící tlakové síly P na spojku. Vlivem působení tlakové síly dojde ke změně aktuální polohy spojky s. Výstupem systému je opět aktuální poloha spojky s, která se použije v následujícím časovém kroku řízení.

Při aktivaci regulátoru pro velké změny lze použít dva přístupy pro výpočet hmotnostního toku. V prvním se využívá model pro výpočet hmotnostního toku jako rozdíl dílčích výpočtů pro m_1 a m_2. Zde se pohybujeme již v silně nelineární oblasti. Regulace je řešena aproximačně použitím nelineárního P regulátoru (39). Z hlediska implementace na reálném zařízení je tento regulátor aproximován hodnotami uvedenými v tabulce. Jedná se o tabulku závislosti vzdálenosti do uzavřeného bodu s_z spojky a vzdálenosti k požadovanému pracovnímu bodu s_p. Požadovaný bod je nový pracovní bod-nová poloha na křivce, kterou spojka sleduje. Tento postup je jednou z možností řízení systému a je použita v převodovce typu **EL40 TraXon**.

Druhou možností je použití obecných rovnic plynu (4), ze kterých lze spočítat hmotnostní tok m. Pro různou rozpojovací sílu F_k spojky vyjdou různé požadované tlaky. Z rovnic pro hmotnostní tok dojde k vyjádření hmotnostního toku Δm. Z ní bude určena derivace hmotnostního toku $\dot{m}$. Výpočet $\dot{m}$ lze potom rozdělit na lineární a nelineární část.

Nelineární část má eliptický průběh. Pokud se $\dot{m}$ příliš nemění v čase, pak lze říct, že je lineární, neboli

$$\begin{aligned} \frac{\Delta m}{\Delta t} &= \dot{m} = \dot{m}_1 - \dot{m}_2, \\ \Delta t &= t_{ON}. \end{aligned} \tag{40}$$

Tento výpočet určuje hodnotu výsledného aktivačního času ventilů t_{ON}, který reprezentuje, jak dlouho musí být ventily aktivovány, aby bylo dosaženo požadované pozice spojky. Opět zde dochází k přepočtení času na potřebnou frekvenci spínaní ventilů pomocí PWM modulace. Tento přístup se využívá u modelu převodovky **EL57 Ecomid**.

3.4.3.3 Oscilace spojky - aktivní tlumení vibrací

Pro odstranění oscilací na spojce, označovaných jako tzv. „jitter“, se používá aktivní tlumení vibrací. Odstranění oscilací je docíleno pomocí redukovaného modelu celého vozidla. Jedná se o popis pomocí energie a momentů setrvačnosti jednotlivých částí vozidla. Mezi tyto části patří motor, spojka, převodovka, osa a kola automobilu. Mezi převodovkou a osu je vložen systém výstupní hřídele, která je popsána pomocí systému skládajícího se z tlumičů a pružin.

Dochází zde k tzv. „stick-slipping“ efektu, to lze přeložit jako efekt klouzání. Tento efekt se objevuje u každého zařazeného stupně a má frekvenci rovnou o jednu větší než je číslo zařazeného stupně. To znamená, že pokud máme například zařazený třetí převodový stupeň, tak frekvence tohoto efektu budou čtyři hertze. My ho detekujeme, určíme jeho frekvenci a kompenzujeme tento efekt opačnou frekvencí. Výsledek je hmotnostní tok přepočten na frekvenci, která je přičtena na výstup z regulátoru. Tím dojde k vyrušení tohoto efektu. Analýza a následné řešení tohoto problému proběhlo experimentálně na speciálních měřících a testovacích zařízeních.

3.4.3.4 Řešení problémů při návrhu

Během řešení návrhu modelu systému a řízení se přišlo na celou řadu problémů. Některé byly vyřešeny velmi dobře a dostatečně přesně, ale u některých se muselo zvolit kompromisní řešení, především z důvodu složitosti a výpočetní náročnosti. Jedním z největších problémů je tření a hystereze, protože nejsou konstantní. Realizovat model hystereze je obtížné. Kompenzace hystereze je také obtížná. V budoucnu bude nutné udělat přesnější model, než-li je tomu v případě symetrické hystereze a dále přesnější model pro kompenzaci mechanického tření. Pneumatický systém je silně nelineární. Celkově je model i systém řízení silně redukován a zjednodušen, a to zejména z důvodu použitého CPU, u kterého je kladen důraz na co nejnižší cenu při zachování vysoké kvality řízení. Zde je možné zlepšení v budoucnu a to zejména díky použití silnějšího hardwaru (HW), který by umožnil využití složitějších metod, a tím získání přesnějších výsledků, jak v popisu modelu, tak v kvalitě řízení. I když případ nasazení výkonnějšího HW bude spíše platit až pro novou generaci převodovky s modernější a výkonnější elektrotechnikou.

4 Proces tvorby automaticky generovaného kódu

V této kapitole budou popsány kroky, které je nutné splnit, aby bylo možné z modelu generovat kód. V úvodu 4.1 bude celý proces generování kódu popsán obecně. V kapitole 4.1.1 budou definovány použité programové nástroje během transformace a v kapitole 4.1.2 bude představen **TargetLink**. Kapitola 4.2 bude věnována popisu transformace modelu regulátoru spojky pro automatické generování kódu. Bude zde popsána úprava modelu a jednotlivé kroky, které bude potřeba během transformace dodržet. Kapitola 4.2.3 se bude věnovat samotnému vygenerování kódu. 4.2.4 bude popisovat statickou kontrolu modelu pomocí programu **MXAM** [41]. Kapitola 4.3 bude dokumentovat integraci vygenerovaného kódu do ručně psaného kódu. Kapitola 4.4 bude popisovat příklad generování kódu z modelu matematické operace a jeho porovnání. Kapitola 4.5 bude zaměřena na rozvíjející se budoucí způsob vývoje tzv. **„Continuous engineering“** [18, 33, 39, 57].

4.1 Úvod

Automatické generování kódu je proces, který umožňuje z modelu vytvořeného v blokově orientovaném prostředí, např. v Matlabu, Simulinku získat zdrojové soubory s kódem, například v jazyce C nebo C++ [35]. Existuje celá řada SW, které tento proces umožňují. V Matlabu je například již v základu implementován Simulink Coder, který umožňuje generovat kód v jazyce C a C++. Existují různé toolboxy, které přidávají například podporu pro cílovou platformu jako je Arduino nebo Raspberry a mnoho dalších. Jednotlivé toolboxy jsou různě profesionální, v automobilovém průmyslu je nutné dodržovat specifické normy a tyto normy musí splňovat i použité nástroje [25].

4.1.1 Používané programové nástroje

Pro vytvoření a úpravy modelu se používá program Matlab R2016b a jeho prostředí Simulink. Pro úpravy zdrojových textových souborů se aktuálně používá vývojové prostředí Microsoft Visual Studio 2013. V modelech je významně zastoupená stavová logika, která je reprezentována bloky Stateflow v Simulinku. Jelikož běžně dostupné komerční prostředky často neposkytují požadované možnosti, a tak jsou vytvářeny interní nástroje. Velký význam je přikládán definování datových typů, které jsou použity napříč celým modelem i SW a jsou realizovány a spravovány pomocí Datafield Factory (DFF). DFF se používá pro definování konstant, enumerací, parametrů, měřených veličin nebo třeba definici různých datových typů. DFF následně generuje data pro model a standardní ASAP2 soubor, který je propojen s výsledným kódem [62]. V případě modelu je vytvořena knihovna, která obsahuje specifické nastavení modelových bloků. DDF není jediný interní nástroj ZF, který slouží k vývoji SW. Dále se jedná například o TestView pro správu a vytváření automatických testů a SoftCar, neboli prostředí pro SIL simulace. Přehled dílčích SW nástrojů používaných pro vývoj automatické převodovky kamiónů v ZF je uveden v tabulkách 3 a 4, kde v tabulce 3 jsou popsány komerčně používané nástroje a v tabulce 4 interní nástroje [25].

Název	Určení
ASAP2 a HEX	generování specifických formátů souborů
Beyond Compare	porovnání verzí
CANape	analýza a nastavení parametrů ECU
CDM Studio	kalibrace parametrů
ClearCase	systém pro správu verzí souborů
ClearQuest	definování požadavku na změnu SW
DOORS	vedení a udržování specifikací
Enterprise architect	Systémová analýza
EXAM	statická kontrola modelu
imcFAMOS	analýza signálů
JIRA	agilní plánování
Matlab Simulink	vytváření skriptů a modelů
Medini Unite	analýza a porovnání změn mezi verzemi modelu
MS Visual Studio	vytváření a editace SW
MXAM	správa a spouštění HIL testů
Simulink V and V toolbox	analýza modelu
TargetLink	generování kódu

Tabulka 3: Tabulka použitých komerčních nástrojů

Název	Určení
Datafield Factory	centralizované úložiště informací
ModelCommander	rozšíření pro správu modelu v Simulinku
PASSxde	vytváření definovaných verzí SW a jeho testování
SoftCar	rozhraní pro SIL simulace
TestView	definování testů a spouštění SIL, MIL, SIL-MIL, MIL-MIL testů

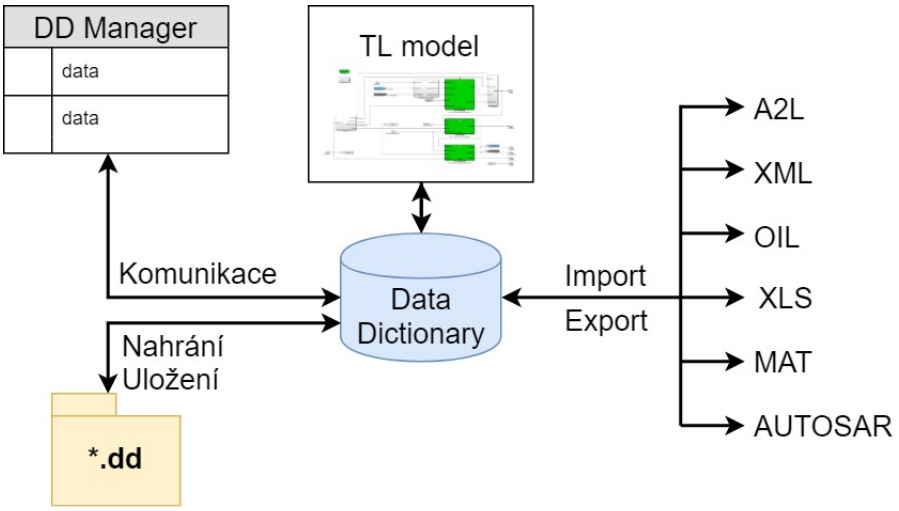
Tabulka 4: Tabulka použitých interních nástrojů

4.1.2 TargetLink

V této práci se budeme věnovat nástroji jménem TargetLink od firmy **dSPACE** [3]. TargetLink umožňuje generovat vysoce efektivní kód v jazyce **C** a **C++** a je určen do průmyslového prostředí. Důvodem rozšíření TargetLinku je splnění celé řady norem. Jedná se například o ISO 26262, ISO 25119 nebo AUTomotive Open System ARchitecture (AUTOSAR) [23, 24]. AUTOSAR je celosvětové vývojové partnerství automobilových firem založené v roce 2003, jejímž cílem je vytvořit otevřený standard pro vývoj SW architektury pro ECU. [12]

Jedná se o nástroj, který vznikl jako odpověď na poptávku po modelově orientovaném vývoji SW, tedy transformaci modelu na zdrojový kód, který může být nahrán na cílovou platformu. V našem případě se jedná o generování C a C++ kódu přímo z prostředí Matlabu, Simulinku i Stateflow. Tento nástroj umožňuje generování jednoduchého C kódu

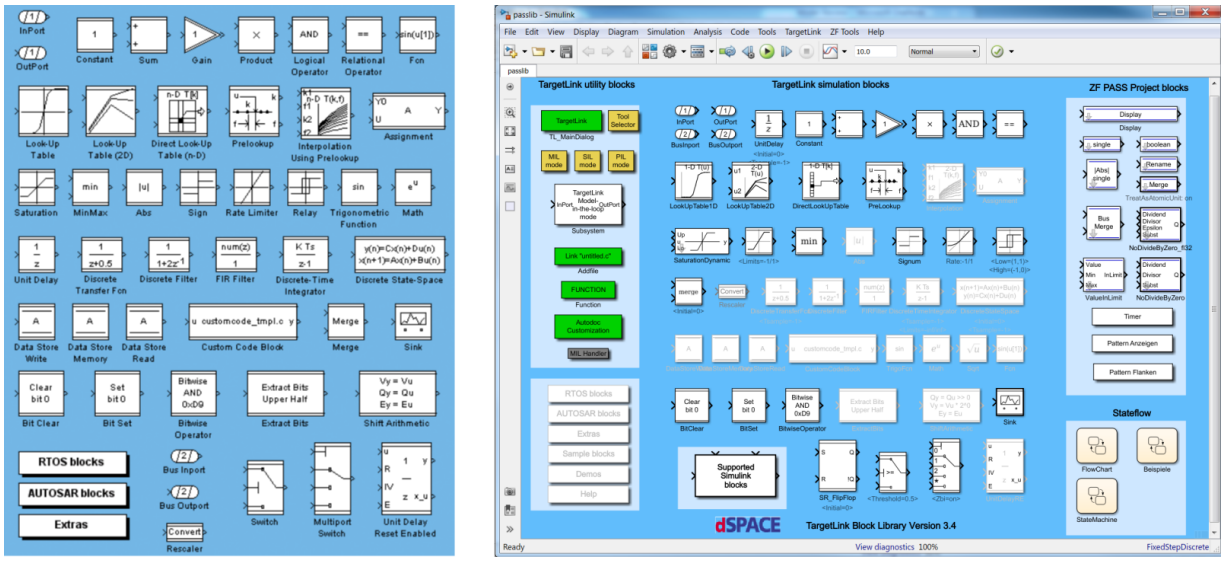
až po velice optimalizované C++ kódy s fixní nebo plovoucí přesností pro platformy AUTOSAR. Velmi všestranná možnost konfigurace způsobu generovaní kódu umožňuje respektování všech omezení procesu a cílového HW.



Obrázek 28: Ilustrace principu funkce a komunikace Data Dictionary

TargetLink zavádí do prostředí Matlabu, Simulinku vlastní knihovnu bloků, které jsou uzpůsobeny pro generování kódu. **TargetLink bloky** jsou vlastně nadstavbou standardních simulink bloků, nad nimiž vytvářejí masku. V té jsou uvedeny relevantní informace nutné pro generování kódu, jako definice datového typu umožňující použití vlastního datového typu, dále informace o škálování hodnot, extrémech, atd. Všechny tyto informace jsou uvedeny v centrální databázi zvané **„Data Dictionary“**. Komunikace Data Dictionary s okolním prostředím je zobrazean na obrázku číslo 28. Z něj je patrné, že tento nástroj umožňuje pracovat s celou řadou typů souborů. Mezi ně patří například soubory typu A2L, XML, OIL, XLS, MAT, AUTOSAR a další. Všechny zkratky jsou vysvětleny v seznamu všech použitých zkratek. Tato databáze je propojena s modelem, kterému poskytuje informace k jednotlivým blokům. Tyto informace mohou být upraveny pomocí prostředí „DDManager“. Veškerá data jsou uložena do specifických souborů s příponou $*.dd$. Data mohou být importována a exportována do mnohých formátů. Právě vzhledem ke specifickým požadavkům je data dictionary ve vývoji ZF nahrazena DFF.

Většinu standardních bloků simulinku lze převést na TargetLink bloky. V případě pokročilých knihovních bloků je zapotřebí nahrazení a realizace stejné funkcionality pomocí standardních bloků, aby bylo možné převod na TargetLink bloky dokončit. Právě z důvodu specifických bloků vznikla v rámci ZF vlastní knihovna, která se označuje jako **PASS-lib**. Tato knihovna obsahuje další množství upravených bloků tak, aby podporovaly generování kódu pomoci TargetLinku. vztahuje se ale pouze na konkrétní projekt. V rámci jiných projektů jsou definovány další interní knihovny. Na obrázku 29 je v levé části znázorněna TargetLink knihovna a v pravé části potom knihovna PASS-libu.

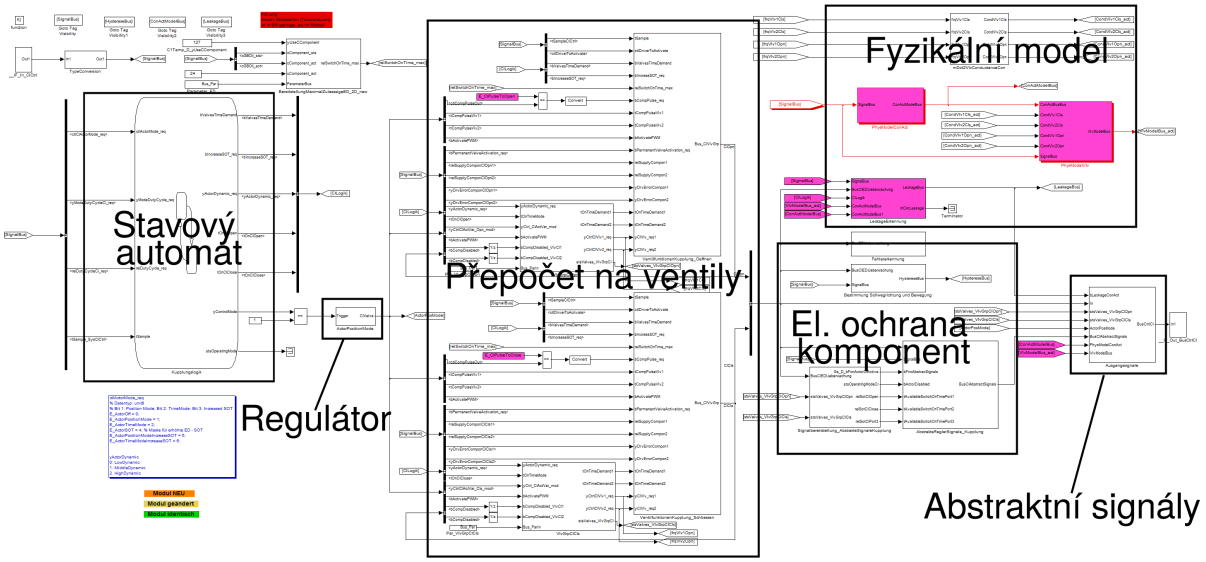


Obrázek 29: Ilustrace knihovny TargetLinku (vlevo) a PASS-libu (vpravo)

4.2 Transformace modelu regulátoru spojky pro automatické generování kódu

4.2.1 Úprava modelu

Na obrázku 30 je znázorněna podoba modelu regulátoru spojky před transformací. Tento obrázek je uveden zejména pro ilustraci podoby modelu před samotnou transformací a není příliš důležité, co který blok obsahuje, a jak se jednotlivé subsystémy jmenují. V principu se jedná o regulátor, který byl popsán v kapitole 3.

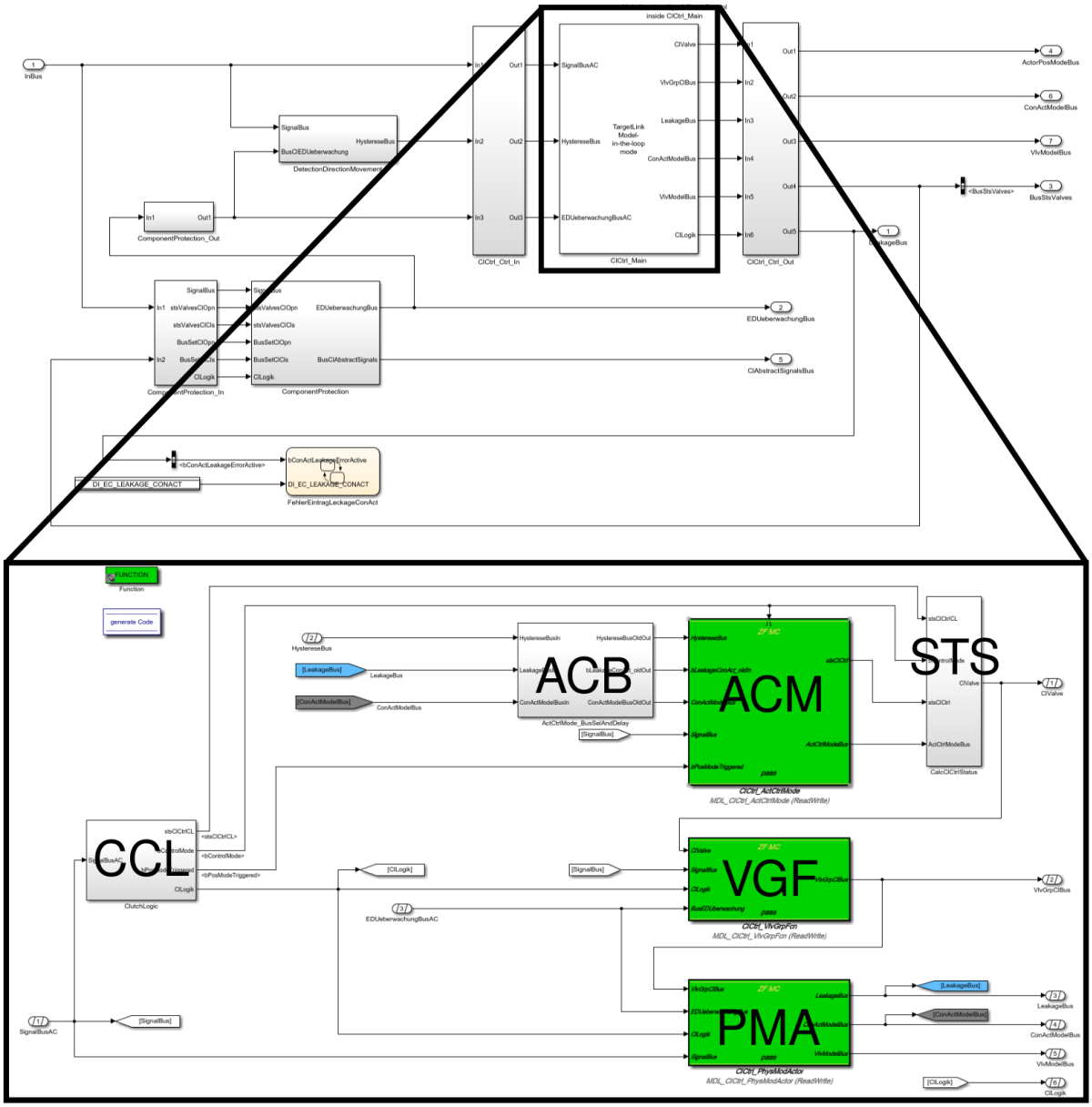


Obrázek 30: Ilustrace podoby modelu před transformací do autokódu

Prvním krokem je definování jednotlivých **komponent modelu**. Tedy celkové architektury, které části modelu se sloučí do jedné komponenty a naopak, které se rozdělí. Jedná

se o důležitý krok, který značně ovlivňuje následnou orientaci a další úpravy v modelu.

Druhým krokem je rozhodnutí, ze kterých částí modelu se bude generovat kód a ze kterých nikoliv. Tato skutečnost je ilustrována na následujícím obrázku 31.



Obrázek 31: Nová struktura regulátoru spojky pro generování kódu

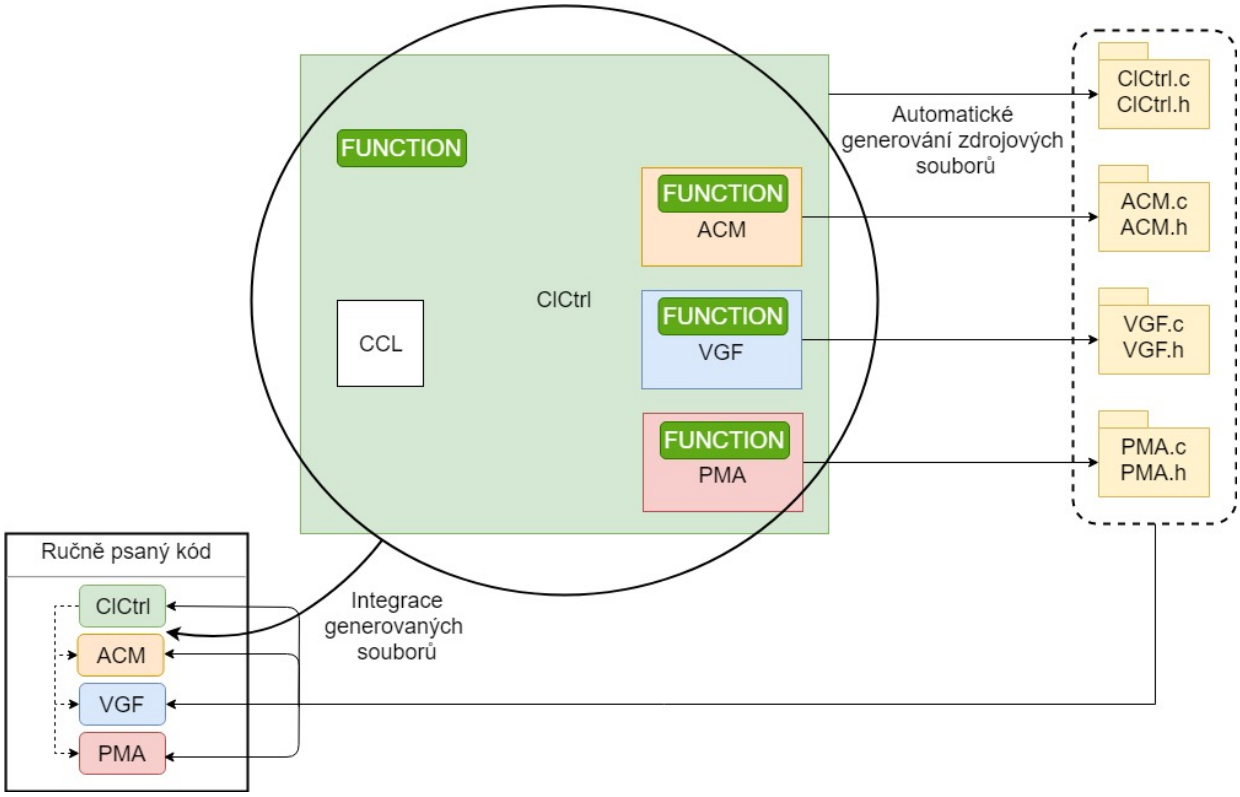
Dále je nutné vytvořit **TargetLink model in the loop (TL MIL)** subsystém, který v sobě obsahuje pouze bloky upravené pro podporu TargetLinku. Tento subsystém je znázorněn na obrázku 31 v prostřední horní části mezi vstupním a výstupním rozhraním (též označovaném jako interface), který v sobě obsahuje **ModelCommander (MC)** bloky. MC bloky slouží k uložení Simulink modelu do jednotlivých MDL zdrojových souborů. Tato úprava má velký význam pro správu a následné úpravy modelu. Pokud je potřeba realizovat změnu v dané částí modelu, která je uvnitř MC bloku, pak stačí upravit pouze

daný MDL soubor. V případě práce více vývojářů na jednom modelu umožňuje rozdělení modelu do MC bloků paralelizaci jejich práce. Pokud požadujeme, aby se z MC bloků generoval kód, musíme v nich definovat TargetLink **„Function“ blok**, který bude popsán v následujících odstavcích. Zbytek funkcionality, ze které se nebude generovat kód, musí zůstat mimo TL MIL subsystém. Takto přemodelovaný model je znázorněn v dolní části obrázku 31.

Pokud otevřeme TL MIL subsystém, nacházejí se v něm atomické části zapouzdřené v MC blocích. Pojem atomické představuje základní simulinkovské bloky, které jsou převedené do TargetLink bloků. Jedná se o zeleně zbarvené subsystémy, které jsou znázorněny na obrázku 31. Komponenta regulátoru spojky uvnitř TL MIL obsahuje celkem šest subsystémů. Z toho se jedná o tři klasické subsystémy a tři MC bloky.

- **ClCtrl_ ActCtrlMode (ACM)** - Jedná se o regulátor spojky pro mód řízení spojky.
- **ClCtrl_ VlvGrpFcn (VGF)** - Zde se provádí přepočet výstupu regulátoru na frekvenci a čas doby aktivace ventilů.
- **ClCtrl_ PhysModActor (PMA)** - Reprezentuje rovnice fyzikálního modelu systému spojky.
- **ClutchLogic (CCL)** - Obsahuje stavovou logiku pro výběr jednotlivých módů řízení spojky.
- **ActCtrlMode_ BusSelAndDelay (ACB)** - Obsahuje výběr potřebných signálů ze sběrnice, popřípadně jejich jednotkové zpoždění nebo datové přetypování.
- **CalcClCtrlStatus (STS)** - Zde je definována výstupní sběrnice pro výběr signálů nebo stavových proměnných.

Pro generování kódu je klíčový Function blok, který je na obrázku 32 zasazen do celého kontextu generování kódu. Celý obrázek 32 ilustruje vztahy mezi modelem, generováním kódu a jeho integrací do ručně psaného kódu. Jednotlivé označení bloků, funkcí a zdrojových souborů, jako je například ACM nebo VGF, odpovídá výše zmíněnému značení bloků modelu. Blok Function v sobě obsahuje informaci o generovaném souboru, definici subsystému pro generování kódu a název cílového textového zdrojového souboru. Blok Function je tedy umístěn uvnitř MC bloků pro ACM, VGF a PMA. Dále se také nachází uvnitř TL MIL subsystému, který je na obrázku číslo 32 znázorněn velkým blokem Model. Jednotlivé subsystémy budou přeloženy do zdrojových C nebo C++ souborů pomocí TargetLinku. Pro zjednodušení generování kódu lze vytvořit blok „Automatizace“, který je napojen na interní skripty, které zajistí automatické spouštění procesu generovaného kódu . „Clutch Control“ (ClCtrl) značí, že se jedná o regulátor spojky.



Obrázek 32: Ilustrace vztahů mezí MC bloky, FUNCTION bloky a generováním kódu

4.2.2 Požadavky úspěšného generování kódu

Aby proběhlo generování kódu z modelu úspěšně, je nutné dodržet několik kroků. Zde budou jednotlivé kroky popsány. Pokud se jimi bude vývojář řídit, mělo by generování kódu proběhnout úspěšně.

1) Nejprve musí být vytvořen samotný model systému. Pro úspěšné generování kódu je nutná jeho úspěšná inicializace, ale běžící simulace není podmínkou úspěšného generování kódu.

2) Dále je nutné definovat proměnné, parametry, konstanty, vlastní datové typy, škálování a další do databáze neboli do Data Dictionary, v ZF se jedná o DFF.

- **Proměnné** mohou být definovány jako vstupní, výstupní, nebo vnitřní příslušící určité komponentě. V databázi lze pak definovat jejich popis, datový typ, škálování, nastavení počáteční hodnoty, minimální a maximální možné hodnoty a další.
- **Parametry** a **konstanty** je potřeba také definovat. Uvádí se jejich popis, datový typ, škálování, hodnota (v případě parametru i počáteční) a další. Parametry lze kalibrovat, kdežto konstanty jsou neměnné.

3) Následně je potřeba převést Simulink bloky na TargetLink bloky, tzn. do podoby vhodné pro generování kódu.

4) Poté je nutno propojit Data Dictionary/DFF s TargetLink bloky použitými v modelu. Tento krok lze realizovat pomocí m-skriptů.

5) Rozšíření modeluo bloky Function, TargetLink MAIN Dialog a jejich nastavení. V TargetLink MAIN Dialognastavení je nutno nastavit parametry samotného generování kódu.

6) Pro správnou inicializace modelu, aby při ní nedocházelo k chybám, je potřeba opravit nastavení datových typů u použitých bloků nebo v Data Dictionary/DFF

7) Dále lze přistoupit ke generování kódu.

8) Spustit statickou kontrolu modelu pomocí testů MXAM (popsán v podkapitole 4.2.4) [41].

9) Zkontrolovat vygenerovaný kódu, zda se ve zdrojových souborech nevyskytují nežádoucí konstrukce nebo chybné části.

4.2.3 Vygenerování autokódu

To že jsme schopni vygenerovat zdrojový kód ještě neznamená, že je funkční. V modelu se může například vyskytovat **funkční nebo logická chyba**, která se projeví až při testování, kde zjistíme, že vygenerovaný kód nepracuje tak, jak by měl. Může se to projevit tak, že při testech nedojde například z hlediska spojky k otevření spojky, nebo třeba jen k velmi pomalému otevírání, zavírání nebo se dále mohou například objevit oscilace. Aby se předešlo některým chybám, tak je vhodné před generováním kódu model nejprve otestovat pomocí MXAMu.

4.2.4 Statická kontrola modelu

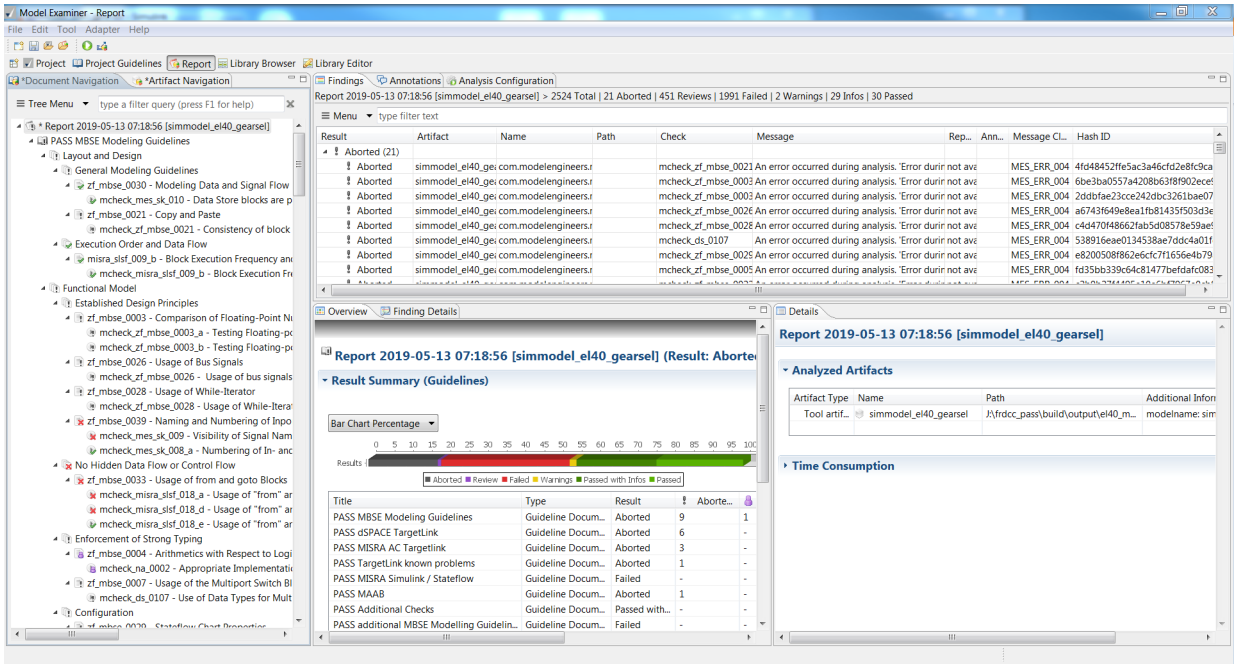
MOdel Engineering Solutions GmbH (MES) poskytuje několik dílčích řešení. Funkční bezpečnostní řešení MES (FSS) je určeno pro vývoj bezpečnostně relevantního standardního SW. Poskytuje nástroj zvaný MES M-XRAY® (MXRAY) pro analýzu struktury a složitosti modelu. Tento nástroj je také součástí řešení pro funkční bezpečnost. MES MISRA® Compliance Solution (MCS) je řešením, které poskytuje vybavení pro kontrolu vytvořených modelů při splnění požadavků MISRA®. Řešení MES ASCET Solution poskytuje postupy pro efektivní vytváření modelů v programu ASCET. ETAS ASCET-DEVELOPER zkráceně Ascet je nástroj od společnosti ETAS pro vývoj aplikačního SW pro embedded systémy za použití grafického a textového prostředí [42].

MES Model Examiner® zkráceně MXAM je SW pro automatickou kontrolu modelu. Tento nástroj je certifikovaný TÜV SÜD pro normy IEC 61508, ISO 25119 a ISO 26262. V kapitole 2.2 byly uvedeny některé používané normy v automobilovém průmyslu. Tyto normy by měly být dodrženy nejen při vývoji ručně psaného kódu, ale také při automatickém generování kódu z modelu. V praxi se tento test používá tak, že v rámci programu MXAM jsou definovány požadavky podle norem a ty by měly být v rámci modelu splněny. Tento seznam lze rozšířit o definici vlastních požadavků, které chceme, aby model splňoval. Jedná se například o níže specifikované požadavky, kdy některé jsou definovány významnými firmami [41, 42].

- **dSPACE TargetLink guidelines** - Tento seznam testů obsahuje zásady kontroly specifikované firmou dSPACE.

- **MAAB** - Zásady pro automobilový průmysl, které jsou definovány firmou MathWorks Automotive Advisory Board [14].
- **MISRA** - Obsahuje standardy pro vývoj SW v jazyce C, dále zásady kontroly pro Simulink Stateflow a autokód.
- **Problémy specifického projektu** - Zde je uveden seznam zásad, které jsou definované v závislosti na daném projektu. Tedy každá firma nebo oddělení si může definovat své vlastní.

Po spuštění testu MXAM a jeho proběhnutí se otevře výsledné okno, ve kterém jsou shrnuty výsledky. Každá nalezená chyba je označena, popsána a ve většině případů je tam na základě nahraných bodů testů i znázorněno jejich možné řešení. Cílem tedy je dostat model do správné podoby, aby splňoval požadavky testu a ten tak proběhl bez chyb. Po analýze výsledků je na vývojáři, aby veškeré chyby opravil. Na obrázku 33 je znázorněno prostředí programu MXAM s právě dokončenou analýzou. V jeho levé části jsou záložky pro nastavení a výběr jednotlivých testovacích pravidel, u kterých jsou značky, zda daný test prošel úspěšně nebo ne. V prostřední části se nachází grafické znázornění výsledků testu. Po rozkliknutí se zobrazí podrobnosti.



Obrázek 33: Prostředí programu MXAM se zobrazeným výsledkem testu

4.3 Integrace generovaného kódu

V této části již předpokládáme, že máme k dispozici vygenerovaný kód. Neřešíme zatím jeho správnou funkčnost, o tu je postaráno během testování. V následujících odstavcích se budeme věnovat integraci vygenerovaného kódu. Jedná se o začlenění automaticky generovaného kódu do zdrojového C++ kódu, který je napsán ručně. Tento proces lze rozdělit do několika částí, které jsme nuceni dodržet. Pokud bude vše provedeno správně,

mělo by být možno na konci integrace upravený C++ kód úspěšně zkompilovat, sestavit a spustit.

4.3.1 Označení částí ručně psaného kódu

Nejprve se budeme věnovat identifikaci míst, funkcí v ručně psaném kódu, které budou nahrazeny voláním funkcí z automaticky generovaného kódu. V případě komponenty regulátoru spojky byly označeny celé části kódu pomocí předem definovaných komentářů. Tento komentář obsahoval indikaci, že se jedná o funkci, která je obsažena v autokódu. Dále zde byla také uvedena cesta, pod kterou je možné v modelu tuto funkci najít. Při této identifikaci se postupovalo tak, že byl postupně prohledán celý model, ze kterého se generuje kód. Identifikované části modelu byly porovnány s funkcemi ručně psaného kódu. Mohou nastat případy, kdy dochází k odlišné implementaci funkcionality mezi modelem a původním ručně psaným kódem. Může se jednat o složité případy, kdy bez hlubší analýzy není na první pohled patrné, že se jedná o stejnou funkcionalitu. Cílem je identifikovat stejnou funkcionalitu mezi modelem a ručně psaným kódem a tyto části označit definovaným komentářem. V případě odlišné funkcionality se bere původní ručně psaný kód jako reference a model je nutné podle něj opravit. Tento postup lze vykonat již během transformace modelu do podoby podporující generování kódu. Díky tomu lze ušetřit čas, který bude potřeba na následnou integraci autokódu.

4.3.2 Nahrazení označených částí za automaticky generované

Pokud je nalezeno a označeno spojení funkcionality mezi modelem a původním ručně psaným kódem, lze přistoupit k samotné integraci. Ta v podstatě spočívá v nahrazení označených částí v ručně psaném kódu za automaticky vygenerovaný kód. Tedy nahrazení ručně psaných funkcí za funkce z vygenerovaného kódu. Důležité je správně definovat pořadí při volání jednotlivých funkcí. Cílem je dostat zdrojový soubor do spustitelné a kompilovatelné podoby. Během integrace se objeví množství chyb, které je potřeba vyřešit. Může se například jednat o rozdílné nastavení datových typů, změnu jména funkcí, vytvoření a volání nových funkcí atd... Pokud odstraníme veškeré chyby, tak bychom měli být schopni zdrojový soubor zkompilovat a sestavit. Pokud je k dispozici sestavený SW, tak může přijít na řadu testování integrovaného kódu.

4.4 Porovnání softwaru před a po transformaci

Po integraci automaticky generovaného kódu do ručně psaného kódu se tento SW musí podrobit **testování**. Jak již bylo zmíněno, funkcionalita musí zůstat zachována a nesmí se mezi starou referencí a novou verzí SW lišit. To v praxi znamená, že ze staré verze SW se vytvoří reference, ke které se poté vztahují veškeré výsledky testování. Někdy se během vývojového cyklu celého řídicího systému může stát, že během transformace do automaticky generovaného kódu probíhá vývoj i na dalších komponentách, a proto je důležité udržovat větev ve verzovacím systému synchronizovanou s aktuální hlavní větví. Pokud toto dodržíme, bude následná synchronizace s hlavní větví mnohem jednodušší a méně časově náročná. Při procesu zvaném „merge“ dochází ke spojení dvou zdrojových souborů. Triviální, neboli nekonfliktní změny, jsou vyřešeny automaticky, netriviální merge se musejí vyřešit ručně. V případě zdrojových souborů v textovém formátu se většinou

nevyskytuje problém. Ale v případě modelu, kde se operuje s binárními soubory, je většinou potřeba veškeré změny přenést ručně a vytvořit tak novou verzi souboru. Testování výsledného SW probíhá již klasickými MIL, SIL a HIL testy. Dále se provádí analýza zdrojového kódu týkající se chyb programování, podezřelých konstrukcí a syntaxe. Testování se podrobněji věnuje kapitola 5.

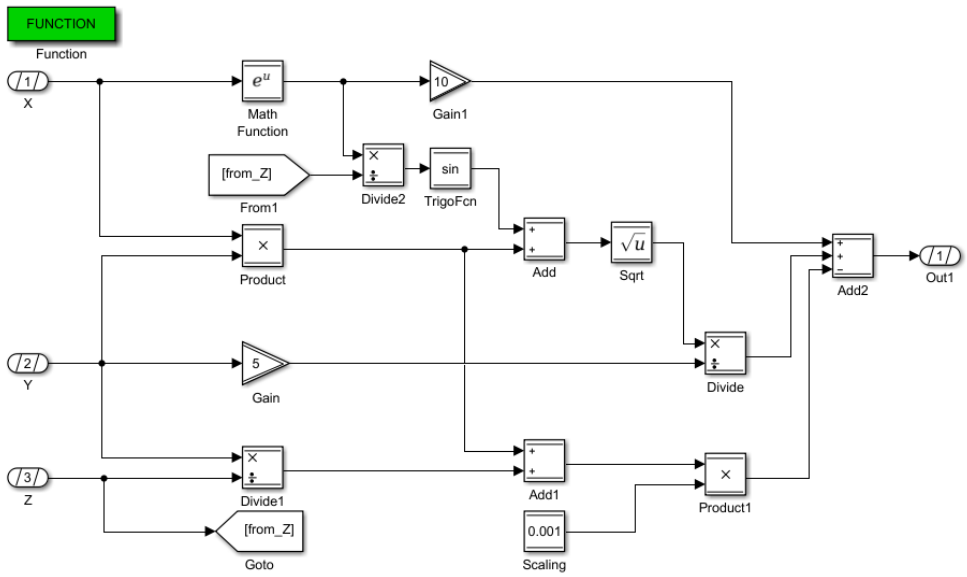
Vzhled generovaného kódu se oproti ručně psanému kódu liší především pojmenováním jednotlivých signálů a sběrnic, kdy TargetLink všem signálům přiřazuje předem danou posloupnost jmen na základě čísel portů a cest signálů. Dále se v něm také vyskytuje velké množství komentářů, které dokumentují, která část kódu patří které části v modelu, sběrnici nebo signálům. Někdy se v něm mohou vyskytnout několikanásobná přetypování datových typů. To často souvisí s nesprávným zacházením s datovými typy v modelu. Proto je nutné provést analýzu vygenerovaného kódu a tato několikanásobná nepotřebná přetypování odstranit.

4.4.1 Příklad generování kódu

V této podkapitole bude uveden příklad pro podobu vygenerovaného kódu pro definovaný matematický výraz. Tento příklad byl zvolen následující tvar matematického výrazu

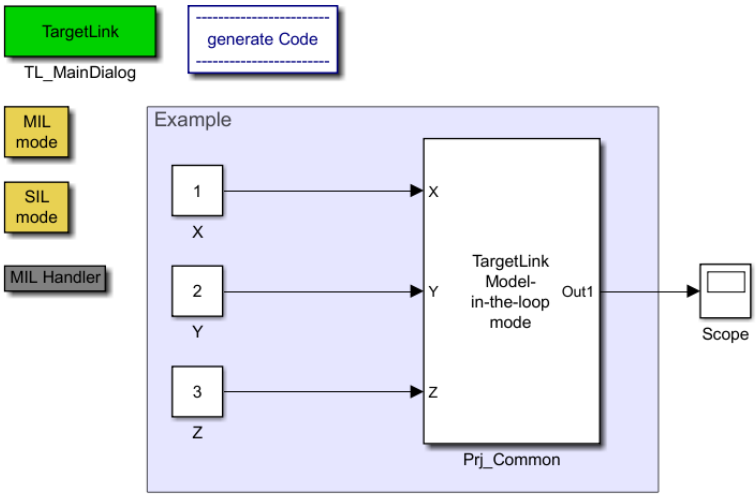
$$\text{výstup} = \frac{\sqrt{xy} + \sin(\frac{e^x}{z})}{5y} + 10e^x - \Big[\Big(\frac{y}{z} + xy\Big) \cdot 0,001\Big]. \tag{41}$$

Na základě rovnice (41) byl vytvořen model v Simulinku, znázorněný na obrázku 34, kdy jako vstupy jsou uvažovány proměnné x, y a z. Veškeré použité bloky byly přetvořeny na TargetLink bloky a byl přidán blok Function. Při transformaci se postupovalo stejně, jako bylo popsáno v předešlých kapitolách.



Obrázek 34: Příklad modelu matematického výrazu pro generování kódu vytvořeného pomocí TargetLink bloků

Model byl dále zapouzdřen do TL MIL subsystému, to je dokumentováno na obrázku 35.



Obrázek 35: Ukázka struktury nejvyšší úrovně modelu pro generování kódu pomocí nástroje TargetLink

Byly přivedeny vstupy pro proměnné a přidány jednotlivé bloky, aby bylo možno úspěšně vygenerovat kód. Jednalo se o bloky pro MIL mode, SIL mode a MIL handler pro umožnění MIL a následně SIL simulace. Dále byl přidán „TL_MainDialog blok“, který v sobě obsahuje informace o nastavení TargetLinku a výsledných vygenerovaných souborech. Poslední přidaný blok „generate Code“ je navázán na skripty, které umožňují po kliknutí spustit generování kódu. Podoba výsledného vygenerovaného kódu z takto upraveného modelu je k dispozici **v příloze** této diplomové práce. Přiložený zdrojový kód byl zkrácen o některé zakomentované řádky, aby vynikla podoba vygenerované funkce.

Při pohledu na vygenerovaný kód lze usuzovat, že generátor provede automaticky vložení všech příslušných headrů. Jedná se například o „\# include <math.h>“. Mezivýsledky jednotlivých operací jsou ukládány do nových proměnných, které nesou jméno podle příslušného bloku. Jde tak například o „Sa1_ Add“, „Sa1_ Divide“ atd... Dále každá provedená operace je okomentována, je tak na první pohled zřejmé, co daný kus kódu představuje. Celý výpočet je proveden v datovém typu float32, to se odráží i ve vygenerovaném kódu. Typické je to pro ošetření dělení nulou, kde se hodnoty porovnávají s hodnotou 0,F. Pokud by k dělení nulou došlo, je výsledku přiřazena hodnota „Sa1_ Divide2 = 3,402823466e-38F“.

4.5 Zásady pro tvorbu efektivnějšího modelu

Během vytváření nebo přepracování modelu v Simulinku je vhodné se řídit jistými zásadami a pravidly vytváření modelu tak, aby byl co **nejoptimalizovanější, nejpřehlednější a nejkvalitnější**. Samozřejmě pokud je pro nás důležitá pouze výsledná funkcionalita a nejsme limitováni výpočetní ani paměťovou náročností, tak pro nás optimalizace nepředstavují podstatnou část. Na druhou stranu, pokud pro nás výpočetní náročnost generovaného SW představuje velice důležitý parametr, jako v našem případě, jsou optimalizace nezbytné. Optimalizace se týkají zejména přístupů k modelování a ošetřování okrajových stavů. To souvisí zejména s celkovou udržitelností celého modelu. V této kapi-

tole budou popsány některé optimalizační kroky, které je vhodné dodržet. Určitou formu optimalizace zajišťují již **MXAM** testy, které podchytí značné množství chyb. Jedná se ovšem spíše o syntaktické optimalizace než-li funkční [41].

V bodech níže jsou popsány některé kroky, které je vhodné dodržet pro získání optimalizovanější formy modelu a vygenerovaného kódu. Přestože některé zmíněné body mají při jednotlivých výskytech pouze zanedbatelný dopad na celkovou náročnost generovaného kódu, v případě většího množství výskytu může jejich respektováním dojít k výraznému snížení výpočetních nároků.

4.5.1 Ošetření dělení nulou

V některých případech se může stát, že SW nebo model přejde do takového stavu, kde může dojít k dělení nulou a to i v případech, kde by k tomu docházet nemělo. To v lepších případech skončí chybovou hláškou a pádem SW nebo modelu, v horších případech se chyba objeví až na reálném zařízení. Je proto důležité správně nastavit inicializaci daných proměnných a případná dělení nulou ošetřit. Výsledkem by mělo být eliminování takových událostí. Dále je důležité hlídat si datové přetypování signálů. Pokud máme například signál typu float64 o hodnotě 0.123 a z nějakého důvodu signál přetypujeme do datového typu integer, vznikne tak signál s hodnotou 0. Nejen že dojde ke ztrátě informace, ale může následně dojít i k dělení nulou, čemuž se chceme vyhnout.

4.5.2 Vymazání nepoužitých signálů a bloků

Veškeré vstupy, výstupy a bloky jsou při generování kódu překládány do zdrojových C++ souborů. Každý signál představuje instanci, která je uložena do paměti. Nepoužité signály, které například jen projdou skrz model do výstupní sběrnice a nevyužívají se k výpočtům, tak zbytečně zabírají místo v kódu i v paměti. V případě přejmenování signálu dochází také k vytvoření nové instance proměnné stejného typu, ale pouze pod novým jménem. Jedinou výjimkou je, pokud signál, část bloků nebo subsystém bude terminován, tedy uzemněn-ukončen. To znamená, že jimi poskytnutá informace není propagována do dalších částí modelu. Dalším případem, kdy dané bloky nejsou převedeny do automaticky generovaného kódu, je zakomentovaní bloků v modelu. Dále se může jednat o případ bloku pro switch, kdy jako rozhodovací signál je přivedena konstanta, která má trvale definovanou hodnotu, a povoluje tak pouze jednu z možností. To vede k tomu, že nevybraná možnost není při generování kódu vůbec uvažována.

4.5.3 Správné nastavení datových typů

Správné nastavení datových typů je obzvláště důležité u všech matematických výpočtů. Je nutné datové typy nastavit správně, aby nedocházelo k chybným výpočtům a ke ztrátám informace. Dále veškeré signály použité v té části modelu, ze které se bude generovat kód, musí být nejvýše typu single. Každé datové přetypování je výpočetně náročné, což vede ke zvýšené výpočetní náročnosti celého výsledného SW. Navíc může vlivem špatně nastavených datových typů dojít k nepřesnostem v reprezentaci hodnot a tím pádem k rozdílnému chováním mezi MIL simulací a SIL simulací. I nepatrné počáteční odchylky se mohou šířením v systému znatelně zvětšit a celé chování systémů tak může být zásadně rozdílné.

4.5.4 Nastavení parametrů v databázi

Veškeré proměnné, parametry a konstanty jsou definovány v databázích. Zde je velice důležité je mít správně nastavené. To znamená zejména počáteční hodnotu, datový typ a případně rozlišení daného parametru. Jakákoliv chyba nastavení se může projevit během simulace nebo při testování, kdy dochází k nepřesnostem ve výpočtech nebo například přetečením dané proměnné. Jakékoliv přetečení hodnoty je nepřípustné.

4.5.5 Definice a volání konstant a funkcí

V modelu se někdy můžeme setkat se situací, kdy při složitějším výpočtu je několikrát použit ten samý parametr z vlastní definované knihovny nebo konstanta. Ovšem každé takové volání z hlediska generovaného kódu, kde spojení může být realizováno voláním několika funkcí, znamená čtení z paměti, které stojí výpočetní čas. Roste tak celková výpočetní náročnost SW. Je tedy vhodné všechny tyto výskyty nahradit pouze jedním voláním a načtením do dané proměnné a poté šířit tento signál do všech potřebných míst. V případě StateFlow se nabízí uložení hodnoty do lokální proměnné uvnitř StateFlow a s tou dále pracovat.

4.5.6 Upravení funkcionalit do nové podoby

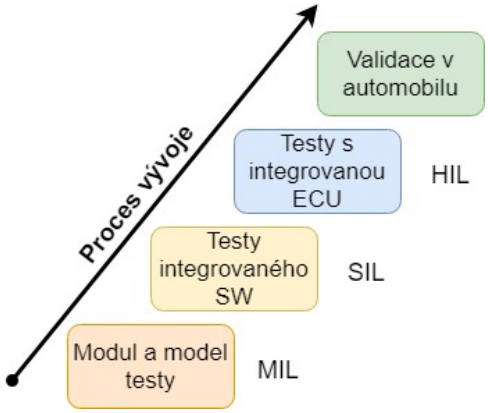
Můžeme se setkat se situací, kdy vývojář vytvoří danou část modelu podle specifikace a zajímá ho pouze funkcionalita. To se může také například stát, pokud je hodnota zmíněné proměnné vypočtena na úplně jiném místě modelu a vývojář nemá potřebné znalosti, aby si požadovaný signál z jiné části modelu pouze vyvedl na místo, kde ho potřebuje. Typicky tak dochází k situacím, kdy je stejný výpočet jedné proměnné realizován několikrát. Jednou to může být například pouze výpočet a podruhé v rámci nějakého většího celku, kde se se zmíněnou proměnnou provádějí další operace. Nabízí se tak možnost nahradit část výpočtu pouze odkazem na předtím vypočtený signál. Stejná je situace i v případě StateFlow.

4.6 Continuous engineering

V posledních letech se v oblasti vývoje SW objevují snahy o maximální možnou efektivitu a využití času vývojářů. To vedlo ke vzniku mnohých nových přístupů vývoje SW. Jedním z nich je tzv. agilní přístup k vývoji, který je charakteristický zejména iterativními a inkrementálními postupy při vývoji [46]. Agilní metody umožňují rychlejší vývoj SW, iterativní získání SW produktu v kratších krocích a dokáží výborně reagovat na náhlé změny požadavků ze strany zákazníka. Dalším novým trendem je zautomatizování procesu vývoje. **Continuous engeneering** je široký pojem, který v sobě skrývá veliké množství všemožných přístupů, které se snaží zefektivnit, zautomatizovat a zoptimalizovat již zavedené trendy a postupy [18, 57].

V případě generování kódu je potřeba vygenerovaný kód ručně zaintegrovat do ručně psaného kódu. Objevují se snahy o automatizaci celého tohoto procesu. Po provedení změny v modelu a nahrání jeho aktuální verze na verzovací systém by se spustili určité úkoly automaticky. Jednalo by se o sestavení modelu, kontrolu modelu, vygenerování kódu a jeho následná integrace do předem připravených souborů s ručně psaným kódem. Tento

proces se často označuje jako **„continuous integration“** [33, 36, 39]. Výsledkem uplatnění tohoto přístupu bude snížení potřebného času a zdrojů na integraci kódu. Podobnou automatizaci lze použít i v případě testování. Jakékoliv testování vyžaduje velké množství času a zdrojů a v případě automobilového průmyslu je celé testování ještě náročnější. Objevují se tak různá řešení, která si dávají za cíl výrazně zkrátit celkový čas potřebný na testování a redukování čekání na zpětnou vazbu pro vývojáře. Na základě této zpětné vazby provedou vývojáři potřebné změny. Celý tento proces automaticky spouštěných testů se označuje jako **„continuous testing“** [18].



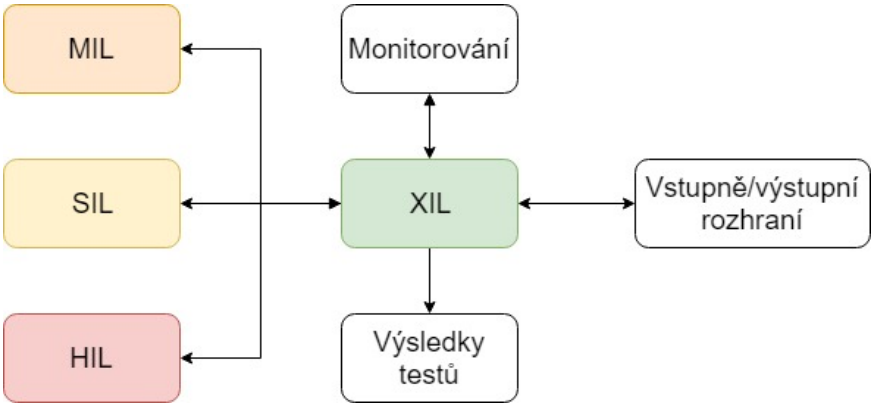
Obrázek 36: Ilustrace různých simulací založená na běžném testovacím přístupu

Testování je celkově velice obtížně popsatelný proces. Neexistuje totiž jednotně platná metodika, která by se dala použít vždy a všude. Některé komponenty nebo části SW mohou byt dodávány třetí stranou, která při vývoji uplatňuje své postupy a používá své nástroje a rozhraní. S tím souvisí rostoucí náročnosti SW, HW a rostoucí požadavky na bezpečnost. S tím také souvisí uplatňování nových a přísnějších norem, nebo například neustále se zpřísňující emisní normy. Obecně lze však říct, že testování SW probíhá v daných krocích a podle platformy lze vykonávat MIL, SIL, PIL nebo HIL testy [4, 50, 56]. PIL testy se implicitně v rámci společnosti ZF nepoužívají. Veškeré popsané druhy testů jsou běžně užívány pří jakémkoliv vývoji embedded SW, liší se ovšem ve své implementaci. Každá firma používá různé nástroje, programy a rozhraní. Tato myšlenka je znázorněna na obrázku 36.

V případě **continuous testing** se nabízí jako řešení použití **X in the lopp (XIL)** standardu. Jedná se o přístup při testování SW, který poskytuje možnost výměny simulačního modelu vytvořeného v daném programu mezi různými nástroji zprostředkovávající testování. XIL standard může být použit pro definování jednotného souboru, který může být použit pro SIL, MIL a HIL testování. Podstata XIL standardu je znázorněna na obrázku 37. XIL tedy kromě možnosti testování na libovolné podporované platformě nabízí jednotné vstupně výstupní rozhraní, které poskytuje jednotný přístup k monitorování průběhu simulace a jeho zaznamenávání [19, 26, 49, 60].

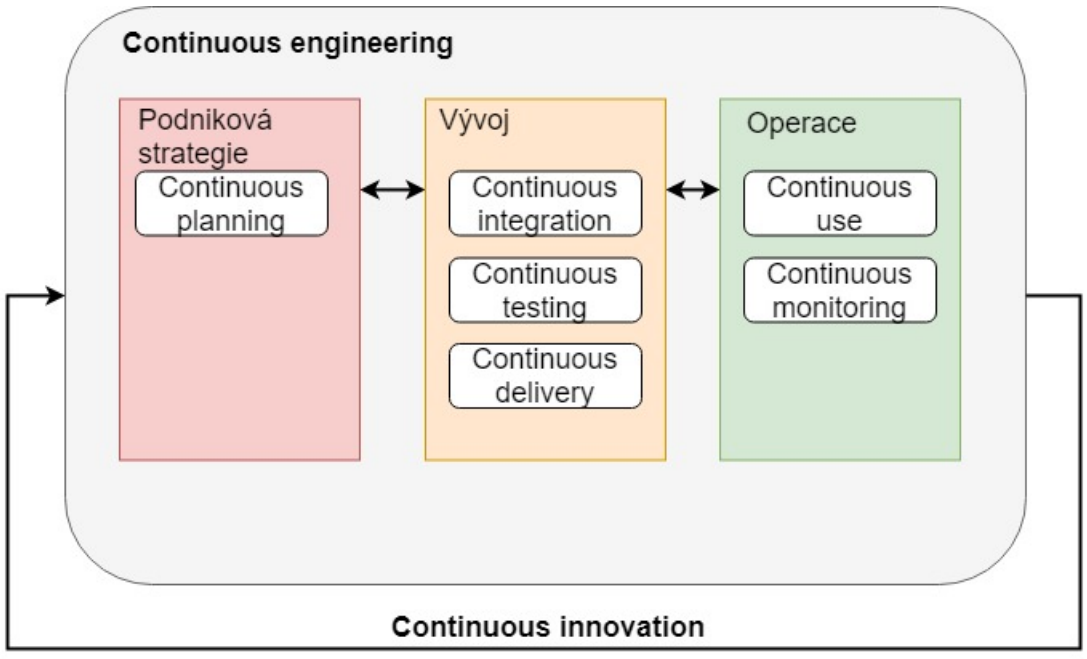
Aby bylo možno použít jednotný soubor, tak tento soubor musí být vytvořen ve specifickém formátu schopném uchovávat a pracovat s danými daty. Stejně jako jednotný soubor, tak musí být použito i jednotné rozhraní, které zajistí komunikaci mezi různými prostředími. Jako jedno z použitelných řešení se nabízí **Functional Mock-up Interface (FMI)** [44]. Toto rozhraní bylo vytvořeno firmou Daimler v roce 2010 a v roce 2014 byla

zveřejněna jeho druhá verze. Toto rozhraní poskytuje formáty souborů tak zvané **Funcional Mock-up Unit (FMU)**. Jsou k dispozici dva typy FMU. Jedním je FMU pro



Obrázek 37: Princip struktury XIL standardu

Model Exchange a druhé je FMU pro Co-Simulation. Liší se zejména v tom, že Model Exchange v sobě nenese vlastní solver. Ten tak musí poskytnout prostředí, ve kterém chceme FMU spouštět. Solver je tzv. „řešič“, který definuje jakou numerickou metodou bude simulace vypočítána. V Matlabu se jedná například o ODE. Naproti tomu verze pro Co-Simulation v sobě solver obsahuje, tedy cílové prostředí v sobě nemusí tento solver poskytovat. S tím také souvisí přesnost simulace FMU Co-Simulation, která se nastavuje nezávisle na prostředí, ve kterém se spouští [13]. Podrobněji bylo toto téma popsáno v autorově bakalářské práci [30].



Obrázek 38: Ilustrace myšlenky „continuous engineeringu“

Dalším místem kde se může využít automatizace je generování SW pro cílovou platformu. To se často označuje jako **„continuous delivery“**. Cílem je produkování výsledného SW v krátkých cyklech, které tak zajišťují že výsledný SW bude k dispozici takřka kdykoliv.

Všechny výše uvedené metody spadají do přístupu označovaného jako **continuous engineering**. Hlavní myšlenka continuous engeneeringu je ilustrována na obrázku 38. Jeho hlavním cílem je zefektivnění celého procesu vývoje od samotného plánování, přes realizaci po testování a monitorování. Tento přístup může být tedy použit také v obchodní strategii jako **„continuous planning“** nebo v případě obsluhy činností jako **„continuous use“** nebo **„continuous monitoring“**. Jednotlivé provedené iterativní kroky v rámci continuous engineeringu lze popsat jako **„continuous inovation“**. Všechny popsané metody se velice rychle vyvíjí, snaží se reagovat na nejnovější trendy, a nacházejí uplatnění na stále více místech, ne jenom v případě vývoje [18, 33].

5 Testování

Jak již bylo nastíněno v předešlých kapitolách, tak vývoj SW je dlouhý a náročný proces a jelikož se pohybujeme v automobilovém průmyslu, kde může dojít k ohrožení lidských životů, hraje testování velice důležitou roli. Musí tak být splněna celá řada norem a standardů. Pří vývoji SW v automobilovém průmyslu se obvykle postupuje podle V diagramu, popřípadně jeho modifikacích, a jeho nedílnou součástí je i samotné testování. Testy je možné rozdělit do dvou kategorií. První se věnují testování dané funkcionality a patří sem například **SIL, MIL, HIL testy a validace v automobilu** [4, 17, 22, 31, 38, 43, 50, 56, 60]. PIL testy se v rámci ZF nepoužívají, a proto budou uvedeny pouze informativně. Na druhé straně máme syntaktické testy, které kontrolují, že je kód napsán podle daných pravidel nebo splňuje definované zásady [4]. V ZF se používá pro tento typ testování **Lint**. To samé platí i v případě modelu, kde se pro tuto kontrolu používá **MXAM**.

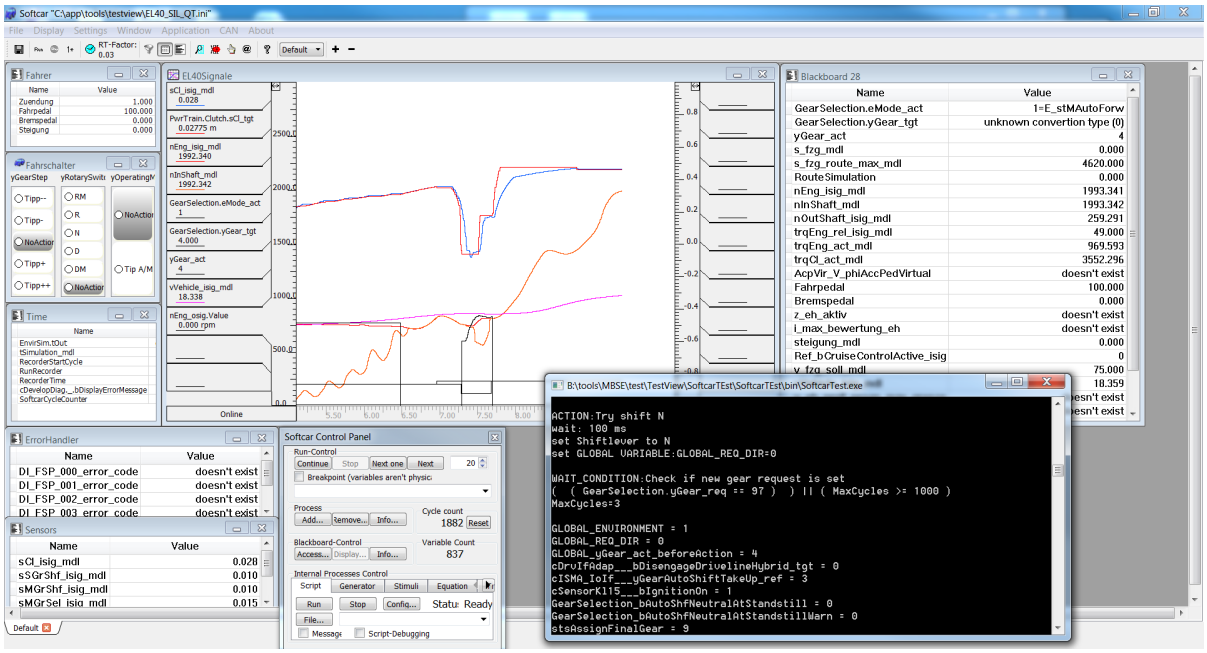
Zvolená testovací strategie se liší v závislosti pro kterou komponentu je vytvořena. Pro celý výsledný SW se používají klasické metody. Komponentou se rozumí část modelu s charakteristickou funkcionalitou a jedná se například o zmíněnou komponentu regulátoru spojky. Dalším důležitým faktorem při testování je pokrytí, neboli otestování všech možností, které mohou nastat. Zde mohou nastat ještě dvě možnosti. Některé podmínky při větvení mohou mít více částí oddělených logickými spojkami. První z nich požaduje pouze průchod všech podmínek a ne všech možností pro danou podmínku, které mohou nastat. Druhá požaduje otestování všech možných kombinací průchodů, tím pádem je také mnohem náročnější na definování a realizaci daných testů [25]. Dále je důležité uvést, že se běžně používá jiná kalibrace parametrů pro SIL, MIL a HIL testy než kalibrace použitá v automobilu. Získané výsledky se tak nepatrně liší. To ale v případě regulátoru spojky nevadí, protože důležitá je funkcionalita a ta musí zůstat nezměněna.

Například některá komponenta v sobě může skrývat velmi rozsáhlou logiku realizovanou stavovým automatem. Zde lze teoreticky dosáhnout 100% pokrytí všech možností. Naproti tomu komponenta regulátoru spojky v sobě nese řídicí systém pro pozici spojky a nelineární model celého systému. Je tedy jasné, že tato komponenta nikdy neposkytne stejné výsledky a proto se u ní kontrolují obecné stavy jako například směr pohybu spojky, dosažení bodu dotyku a podobně.

V úvodu 5. kapitoly byl nastíněn postup při testování. V další části se budeme zabývat pouze funkčními testy, nikoliv syntaktickými. Bude zde popsáno jak se mohou testy pro jednotlivé typy prostředí vytvářet, jaké se pro jejich realizaci používají programy a jaké další nástroje se používají pro zpracování a analýzu výsledků. V podkapitole 5.1 bude popsáno SIL testování s použitými nástroji. Podkapitola 5.2 se bude věnovat MIL testování. Podkapitola 5.3 bude popisovat prostředky pro HIL testování a to jak pro manuální testy, tak i automatické testy. Podkapitola 5.4 bude ilustrovat možné postupy při testování dvou verzí modelu, modelu a SW a nebo dvou verzí SW. Podkapitola 5.5 se bude zabývat nočními automatickými testy a na závěr podkapitola 5.6 se bude věnovat testování na reálném zařízení.

5.1 SIL testovaní

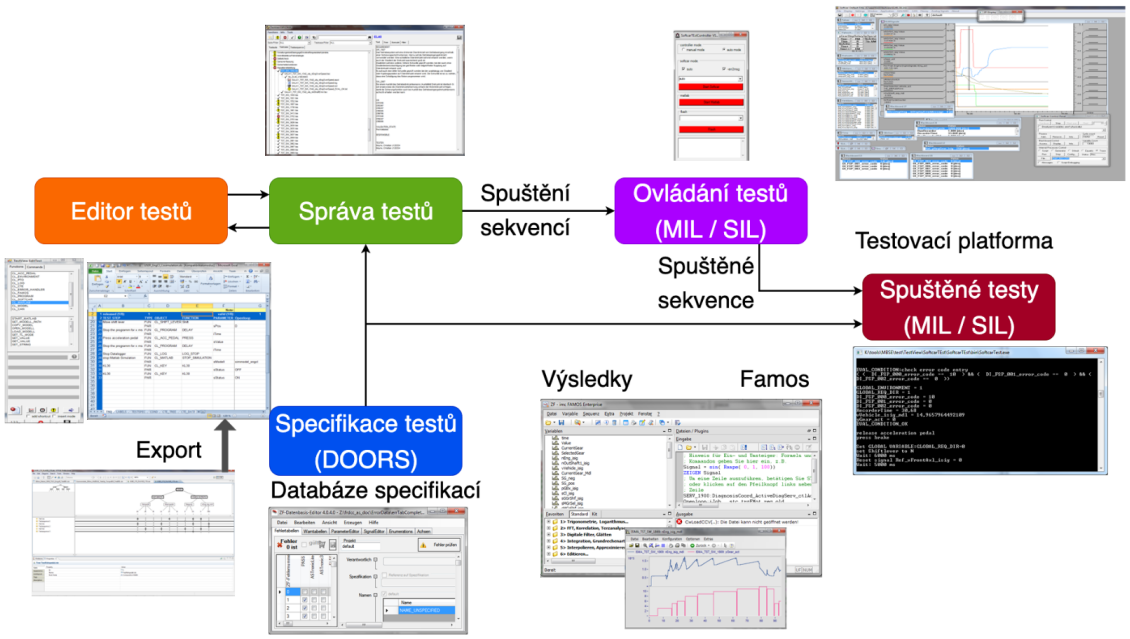
SoftCar je interní nástroj pro SIL testování SW v celém okolí prostředí vozidla. Program nabízí k dispozici grafické uživatelské prostředí, ze kterého je možné zadávat pokyny řízení , například řadit, sešlápnout plynový pedál, ale i nastavit sklon vozovky. SoftCar také poskytuje kontrolní výpisy proměnných a jejich vizualizaci. Simulaci v programu lze vykonat ručně nebo spuštěním automatických testů. Grafické uživatelské prostředí SoftCaru je ilustrováno na obrázku 39.



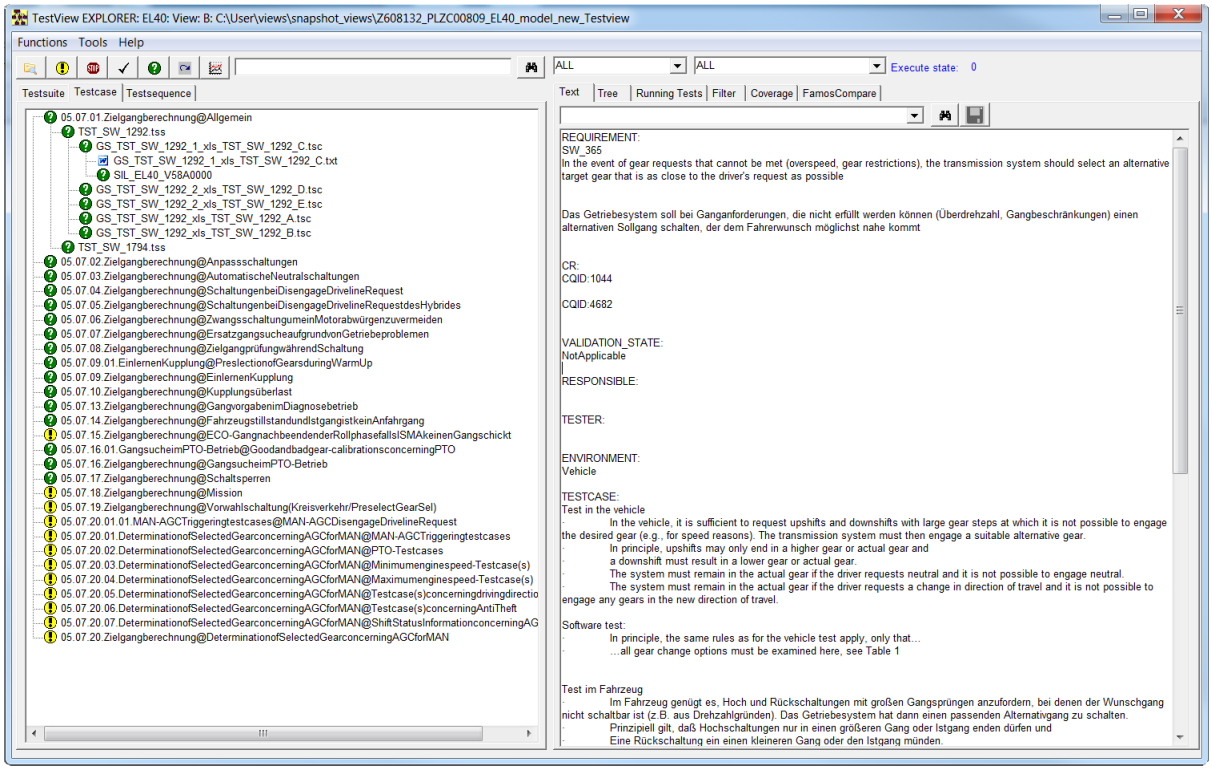
Obrázek 39: Prostředí programu SoftCar

Hlavní nástroj pro správu SIL testů je interní nástroj nazývaný **TestView**. TestView je program napsaný v basicu, který umožňuje vytváření, správu, spouštění a analýzu výsledků jednotlivých testů. Na obrázku 40 je znázorněna interakce a komunikace TestView mezi vícero nástroji.

TestView dále umožňuje při startu testů vybrání verze SW, kterou chceme testovat a umožňuje také testovat SW pro různé varianty zákazníků. Mezi zákazníky jsou rozdíly například při definování pinů určených pro čtení nebo zápis, rozdílná HW řešení nebo například kalibrace. Prostředí tohoto nástroje je znázorněno na obrázku 41.



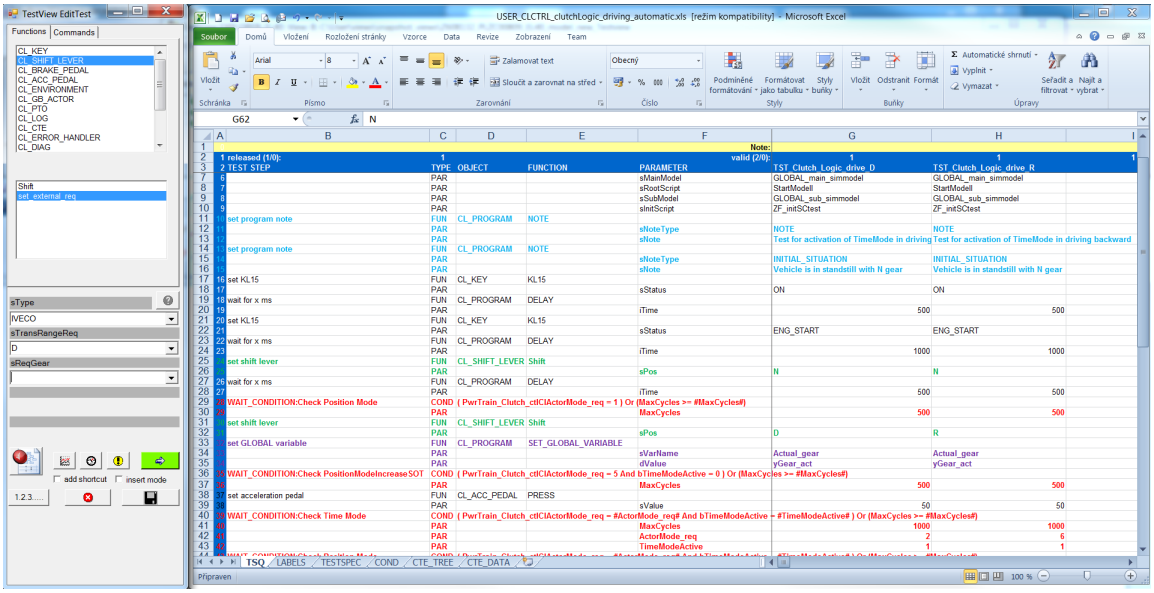
Obrázek 40: Princip funkce a interakce programu TestView s dalšími nástroji



Obrázek 41: Prostředí programu TestView

V levé části obrázku 41 je seznam testovacích skupin, které v sobě obsahují jednotlivé testovací scénáře. Po rozkliknutí testu se zobrazí jeho specifikace, to je znázorněno v pravé části obrázku. Po spuštění se k testu uloží výsledky včetně průběhů jednotlivých signálů. To umožňuje následnou analýzu testu při jeho selhání. Může se však někdy stát, že je

daný testovací scénář sestaven chybně a kvůli tomu nemusí chybu odhalit. Pokud je takový test objeven, je nutné ho přepracovat nebo odstranit ze seznamu testů. TestView je se SoftCarem spárováno a umožňuje spouštění a simulaci předem definovaných testovacích sekvencí. Tento fakt je dokumentován na obrázku 41. Testové sekvence se na základe specifikací píší v TestView pseudokódem, kdy jsou jednotlivé příkazy ukládány po řádcích do **excelovské tabulky**. Jednotlivé řádky jsou očíslovány a jejich vykonávání probíhá od prvního do posledního řádku. Na obrázku 42 je znázorněno prostředí s konzolí pro psaní takovýchto testů. Takto vytvořené testy lze použít i v případě MIL testování.

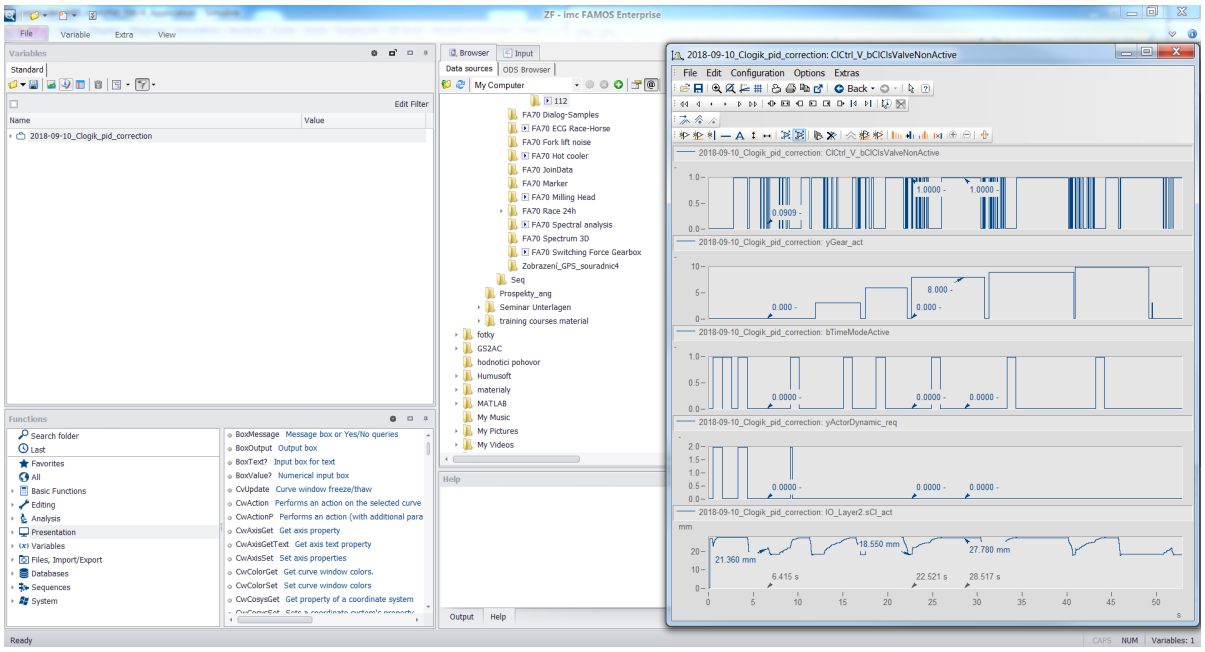


Obrázek 42: Příklad realizace testu v TestView

Definují se zde také proměnné, které budou zaznamenávány pro offline analýzu. To znamená, že po skončení testu se uloží jejich průběh do souboru a budou tak k dispozici k následné analýze průběhu trajektorií. Při spouštění vybraných testů se vybírá, na jaké verzi SW se má test spustit. Vybraný test proběhne a jeho průběh je zapsán do výpisu, kde je vidět, pokud nějaká podmínka v testu nebyla splněna nebo došlo během testu k chybě. Pokud test proběhne v pořádku, tak je v seznamu TestView znázorněn zelenou fajfkou. Pokud ne, tak se u něj objeví ikonka červené stopky. Usnadňuje to orientaci a je tak na první pohled patrné, kde nastala chyba.

Pro analýzu výsledků je možné použít program **imc FAMOS**, celým jménem „Fast analysis and monitoring of signals“. Jedná se o grafický program pro analýzy dat, vyhodnocení a vizuální zobrazení výsledků měření. V něm se provádí analýza průběhů signálů a detekce chyb. Prostředí tohoto programu je znázorněno na obrázku 43.

V levé části se nachází seznam načtených signálů pro analýzu, pod touto částí se nachází panel s nástroji. V pravé části se nachází průzkumník pro procházení složek na pevném disku a příkazové okno, které umožňuje psát vlastní kód pro analýzu a zpracování signálů.



Obrázek 43: Prostředí programu imc FAMOS

Pro otestování byly vytvořeny v programu TestView testovací sekvence, pro ověření správného přepínání módu řízení, pro ověření procesu správného nastavení významných bodů spojky (bod dotyku, bod otevření a bod zavření, pro otestování otevírání a zavírání spojky během řazení a kontrola stavových signálů. Ty samé testy byly aplikovány i v případě MIL testování.

5.2 MIL testovaní

Tyto testy slouží k testování modelu vytvořeného například v Matlabu, Simulinku. V případě blokově orientovaného vývoje SW je MIL testování jednou z prvních možností, jak ověřit správnou funkcionalitu podle požadavků. Vývojové nástroje jako Matlab/Simulink nabízejí možnost simulace a zobrazení výsledků. Testování modelu systému může prvotně probíhat v otevřené smyčce, například za pomoci vstupních signálů definovaných pomocí bloků konstant nebo periodických signálů. Další možností testování v otevřené smyčce je použití vstupních signálů z naměřených dat reálného prostředí. V další fázi testování může být model systému vložen do modelu svého prostředí a simulace proběhne v uzavřené smyčce. Testy napsané v programu **TestView** umožňují být spouštěny jak pro SIL testování, tak i MIL testování. Ovšem, aby bylo možné MIL test správně spustit v otevřené smyčce, musíme nejprve získat vstupní data na základě SIL simulace. Při spuštění SIL testu se nejprve vygenerují soubory, které budou obsahovat hodnoty vstupů. Tyto soubory budou použity pro průběh MIL simulace, tedy jako vstupy pro model v Simulinku. Proto bylo SIL testování v tomto výčtu uvedeno jako první, i když v praxi lze MIL testování svým způsobem aplikovat již při tvorbě modelu. Výsledky simulace budou stejně jako v případě SIL k dispozici pro analýzu v prostředí TestView a umožňují také následnou analýzu průběhu testu. K analýze signálů lze take použít program imc FAMOS.

5.3 PIL testování

PIL je metoda testování určená pro testování SW přímo na cílovém HW. PIL testy jsou navrženy tak, aby odhalili případné problémy s exekucí SW na cílovém HW. Typicky se může jednat například o to, zda se exekuce výsledného SW vejde do přiřazeného času. Toto platí i v případě automaticky generovaného kódu. Lze tak testovat vytvořený model přímo na cílovém HW. V rámci testování v ZF došlo k jistému sloučení PIL a HIL testování. Samotný PIL se nepoužívá a provádějí se pouze HIL testy [56].

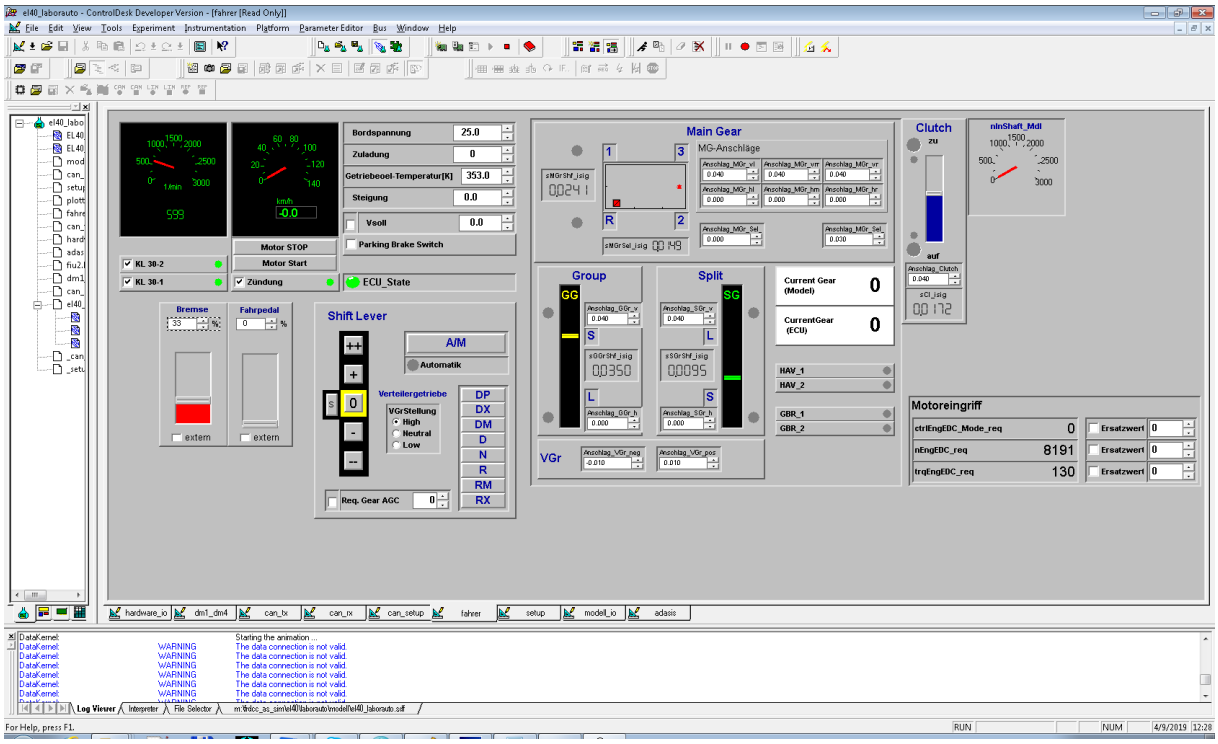
5.4 HIL testovaní

Abychom mohli realizovat HIL testování, musíme mít funkční verzi sestaveného SW, ze které vygenerujeme soubor ve formátu HEX. Formát HEX představuje zápis informace pomocí ASCII znaků po řádcích, které odpovídají formátu souboru Intel **HEX**. Každý řádek obsahuje jeden záznam v hexadecimální soustavě. Jednotlivé záznamy jsou tvořeny hexadecimálními čísly, která představují kód strojového jazyka. Ten nahrajeme do ECU, která je nainstalovaná na HW zařízení, tzv. HILu. Zde je velice důležitý použitý soubor s kalibračními parametry. Pokud chceme opravdu zajistit stejné výsledky, které byly získány pomocí SIL a MIL testů, tak musíme použít stejnou kalibraci. To samé platí pro následující testy v automobilu. Pokud je vše vytvořeno a nastaveno správně, měl by se HEX soubor v pořádku nahrát do HILu. Zde se používají dva typy testů: manuální a automatické. K HILu se dá připojit jako ke vzdálenému počítači a ovládat ho takto, ovšem někdy je nezbytné být přímo u HILu a například ho restartovat, pokud dojde k chybové situaci.

5.4.1 Manuální HIL testy

V případě manuálních testů je situace podobná jako v případě SoftCaru s ovládacími prvky uživatelského rozhraní pro SIL simulaci pouze s tím rozdílem, že SW je simulován na cílovém HW. Uživatelské rozhraní může poskytnout program **Control Desk od společnosti dSPACE**. ControlDesk je experimentální SW pro bezproblémový vývoj ECU. Provádí všechny nezbytné úkoly a poskytuje jediné pracovní prostředí od začátku experimentování až do konce. Podoba tohoto prostředí je znázorněna na obrázku 44.

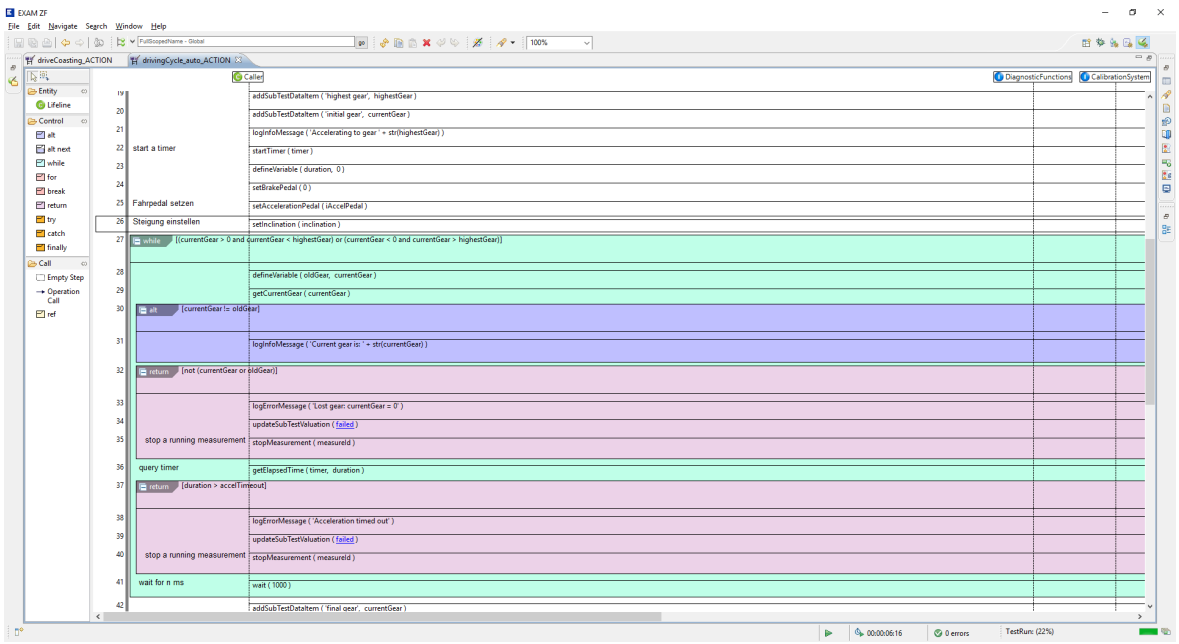
V levé části obrázku 44 se nachází adresářová struktura, ve které je možné přistoupit na další prvky HILu a změnit například jejich nastavení. Spodní část zabírá konzole, ve které se vypisují informace k průběhu simulace. Největší část zabírá uživatelské rozhraní, které poskytuje informace například o rychlosti, otáčkách, zařazeném převodovém stupni, nastavení sklonu vozovky, procentuálním stisknutí brzdového a plynového pedálu a mnoho dalších údajů. Tyto grafické prvky umožňují kromě vizualizace také ovládání celého HW, tedy možnost provádění manuálních testů. Tento přístup se nazývá manuální proto, že veškeré instrukce jsou provedeny nebo zadány ručně. To se používá zejména v případech, kdy není možné realizovat daný test pomocí automatického testu nebo pro případ rychlého ověření funkcionality. Někdy mohou být k HILu připojeny dodatečné HW ovládací prvky. Může se jednat například o pedály, volant nebo řadící páku. Způsob testování pak zůstává stejný, pouze řídicí signály nejsou simulovány, ale přichází od reálného HW zařízení.



Obrázek 44: Prostředí programu ControlDesk

5.4.2 Automatické HIL testy

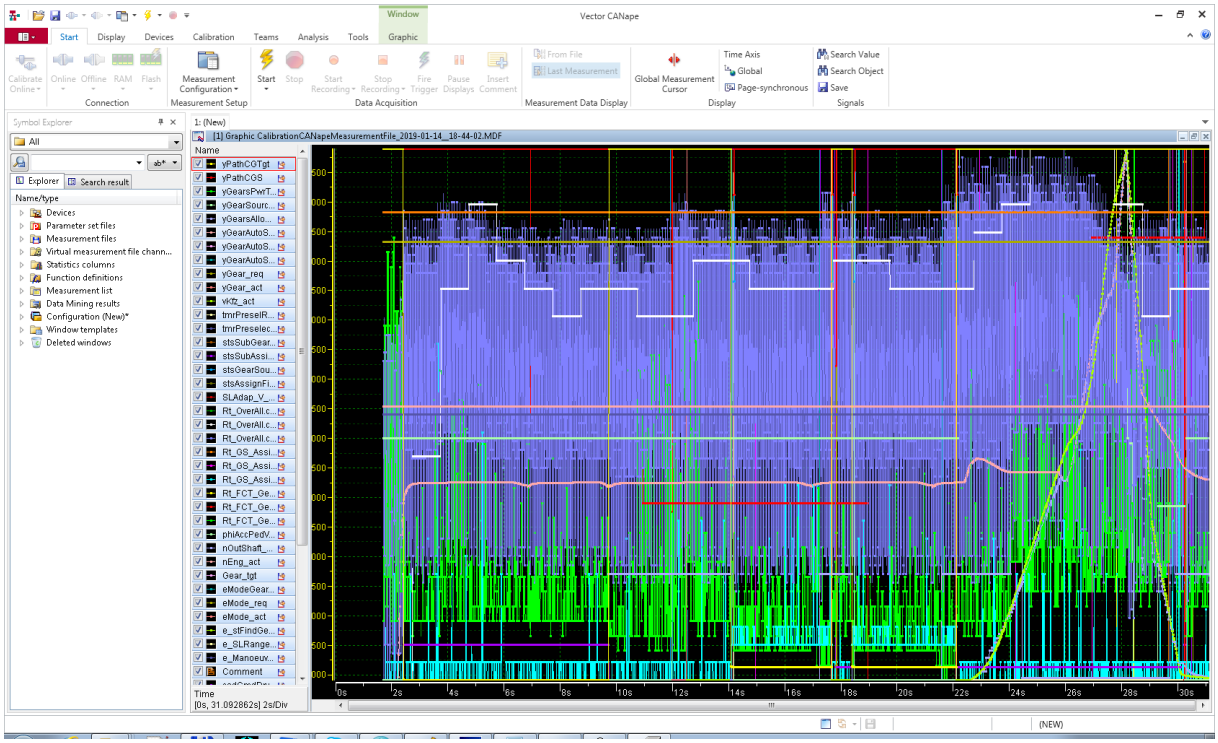
V případě automatických testů jsou k dispozici definované testovací scénáře. Ke správě a vytváření takovýchto testů lze použít program **EXAM** [40]. Program byl vyvinut v roce 2006 firmou MicroNova AG pro Volkswagen AG pro vývoj a vytváření testovacích scénářů. Podoba tohoto prostředí je znázorněna na obrázku 45.



Obrázek 45: Prostředí programu EXAM

V levé části se nachází prohlížeč adresářové struktury, ve které můžeme najít skupinu testů nebo danou testovací sekvenci, kterou je možné otevřít k úpravám. Většinu prostoru okna programu vyplňuje právě otevřený test. V horní bílé části jsou nadefinovány a nastaveny parametry, dále například zelená část představuje while cyklus, ve kterém se simulovaný systém pohybuje tak dlouho, dokud je splněna jeho podmínka. Modré části značí, že nastala nějaká událost a informace o ní bude uložena do protokolu průběhu testu a červené části zastavují měření daného signálu. Veškeré příkazy se vykonávají postupně a to od shora dolů. Z tohoto popisu je patrná podobnost s nástrojem TestView, ve kterém jsou testovací sekvence ukládány v excelovských tabulkách. Jednou vytvořený test je opakovatelný a lze ho modifikovat. Testy je možné slučovat do balíku testů a poté celý takto definovaný balík spustit. Popsané testování je poměrně časově náročné, a tak se často definují větší balíky testů, které se nechávají běžet například přes noc anebo přes víkend, kdy jsou testovací stanice méně vytíženy. Výsledky testů se ukládají do souborů pro jejich následnou analýzu, například pomocí imc FAMOS. Offline analýza ale není jedinou možnou kontrolou. Pro online analýzu se používá program **CANape**. CANape je SW nástroj od společnosti Vector Informatik určený k analýze a kalibraci algoritmů běžících na ECU v reálném čase. Prostředí tohoto programu je znázorněno na obrázku 46.

Pomocí programu CANape lze za běhu sledovat průběhy jednotlivých signálů a to nejen v jaké podobě prošly přes sběrnici CAN, ale i v jaké podobě se uložily v dané adrese paměti. Dále za běhu lze měnit nastavení kalibračních parametrů anebo přímo zasahovat do uložených dat v paměti. Definované signály lze graficky znázornit v grafu, jako je to na obrázku 46. Tento nástroj pro online analýzu lze použít pro automatické testy, tak i pro manuální testy.



Obrázek 46: Prostředí programu CANape

5.5 Back-to-back testování

V případě blokově orientovaného vývoje SW a automatického generování produkčního kódu je zásadní, aby chování vygenerovaného kódu přesně odpovídalo chování modelu. Z toho důvodu se využívá tzv. „back-to-back“ testů [4]. **Tyto testy v zásadě ověřují chování SW v různých fázích jeho vývoje**. Jedním z prvních ověření je srovnání modelu před transformací pro automaticky generovaný kód a modelu po transformaci v rámci tzv. „MIL-MIL“ testování. Další možností je testování modelu vůči generovanému kódu v rámci tzv. „MIL-SIL“ testování. V neposlední řadě lze vykonat srovnání s chováním na reálném HW v rámci tzv. „MIL-HIL“ a „SIL-HIL“ testů.

Při tomto způsobu testování mohou být k dispozici naměřená data z vozidla. Naměřená data jsou získány na základě specifikací testů, které budou provedeny v automobilu. Takto získaná data lze následně použít jako vstupní data při testování našeho SW v prostředí MIL, SIL a HIL. Velice důležitá je v tomto případě kalibrace parametrů systému. Pro zajištění správnosti výsledků musí být stejná. Pokud při vývoji, testování nebo kalibraci nedošlo k chybě, musí se tímto testem získat identické výsledky. To znamená, že systém na předem daný vstup reaguje daným výstupem. Jedná se tedy o testování v otevřené smyčce. V případě transformace komponenty regulátoru spojky do podoby schopné generovat kód došlo k odchylkám mezi SW a modelem, ale díky těmto testům se tyto nedostatky odhalily. Bylo tak potřeba opravit model, aby plně odpovídal SW před transformací do automaticky generovaného kódu.

5.5.1 Noční automatické testy

Automatické spouštění testů v nočních hodinách, tzv. **„Nightly build“** testování, se typicky používá na místech, kde je výsledný kód značně rozsáhlý a kde na něm navzájem pracuje větší množství lidí. Většina SW firem používá při vývoji verzovací systém, ve kterém se ukládají veškeré změny, které vývojáři udělali a nahráli je zpátky na verzovací systém. Jedná se například o programy jako je Git, Concurrent Versions System, AccuRev, IBM Rational ClearCase a mnoho dalších. V praxi jde o to, že SW je rozdělen do mnoha částí, tedy jednotlivých zdrojových souborů, na kterých může naráz pracovat velké množství lidí. Veškeré soubory se při kompilaci překládají a na jejich základě je vytvořen výsledný SW. Noční automatické testy probíhají na serverech v pravidelných cyklech, například každou noc. Vždy vezmou aktuální soubory z hlavní větve verzovacího systému, pravě ty ve kterých došlo přes den ke změnám. Ze souborů se sestaví celkový SW, který se následně otestuje v MIL, SIL nebo HIL simulacích pomocí předem definovaných testovacích sekvencí. Díky tomuto přístupu je zajištěno, že aktuální kód na hlavní větvi verzovacího systému se neustále udržuje v **kompilovatelné a funkční podobě**. Usnadňuje to značně správu celého SW, protože v případě, že došlo při nočních automatických testech k chybě, bude to zaznamenáno ve výsledné zprávě. Výsledná zpráva je dostupná SW vývojářům a testerům. Ti na ni mohou okamžitě reagovat a výslednou chybu detekovat a opravit. Pokud „nightly build“ test v předchozích dnech proběhl v pořádku a nyní nastala chyba, je velmi pravděpodobné, že za tuto chybu mohou změny provedené během předchozího dne. Usnadňuje se tak a urychluje zpětné detekování a oprava chyb v SW.

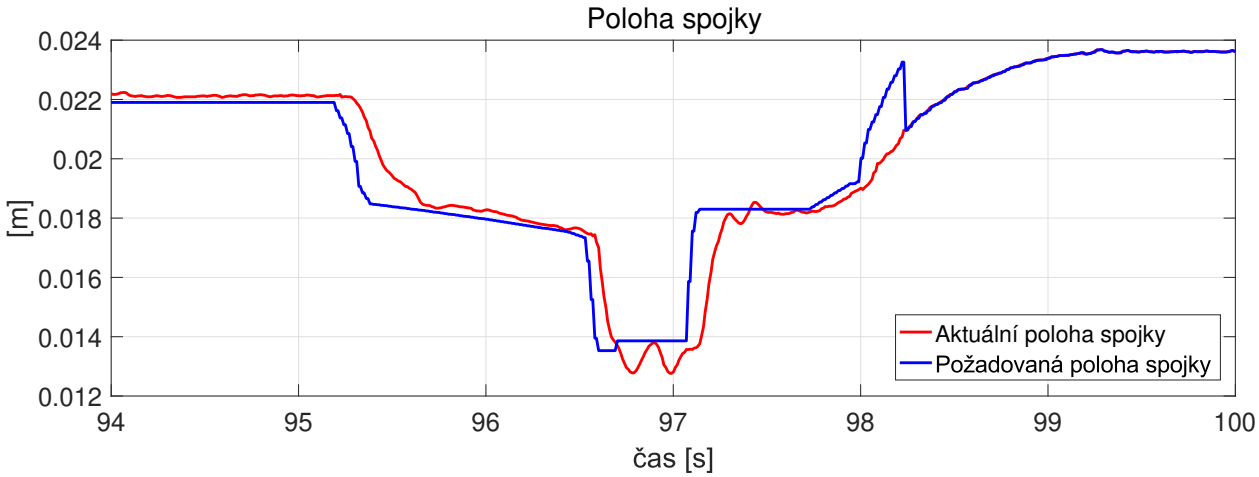
5.6 Testování na reálném zařízení

Validace SW na reálném zařízení je jedním z posledních kroků, který musí SW splnit než bude nasazen do sériové produkce. V případě automatické převodovky TraXon se validace ve vozidle provádí aktuálně na testovacích polygonech ve Fridriechshafenu v Německu a v Egeru v Maďarsku. Validace ve vozidle probíhá tak, že v kabině je vyvedena komunikační sběrnice CANu, na kterou se připojí počítač, na kterém běží speciální prostředí, které umožňuje monitorovat sběrnici CAN. Definují se zde signály, které chceme sledovat. Toto prostředí je velice podobné SoftCaru, který byl popsán v předchozích kapitolách, pouze s tím rozdílem, že zde nejsou ovládací grafické prvky. Jedná se tedy pouze o grafickou vizualizaci signálů a jejich průběhů. Dále toto prostředí umožňuje ukládání průběhů všech signálů na pevný disk počítače. Získaná data následně slouží k závěrečné analýze, zda se veškeré signály chovají přesně tak, jak by měly.

Někdy se může stát, že pro danou SIL simulaci je nemožné dosáhnout počátečních podmínek nebo testovaná funkcionalita je spouštěná pouze v krátkém okně, které je těžko detekovatelné. V takovém případě se může provést testování v automobilu. Pokud ani zde nebude možné splnit všechny podmínky testu, tak se tato skutečnost poznamená a následně musí být rozhodnuto, zda je test i nadále relevantní. Pokud ano, tak musí být přepracován, jinak bude odstraněn.

Dalším příkladem testování v automobilu jsou tak zvané **„Summer“ a „Winter“** testy. Jak již název napovídá jedná se o sekvence testů, které se testují a měří buď v létě, nebo v zimě. Tyto testy probíhají každý rok hlavně z důvodu otestování chování SW a HW v rozdílném okolním prostředí. V předchozích kapitolách bylo popsáno z čeho se systém převodovky skládá. Mechanická konstrukce, ve které je buď olejová, nebo vzduchová náplň. Jistě si každý dovede představit, že mechanické a fyzikální vlastnosti materiálů budou rozdílné v -20 a +20 stupních Celsia. Například se může jednat o roztažnost kovových dílů nebo viskozitu oleje. Tyto testy jsou důležité z toho důvodu, že převodovky se používají po celém světe a je tedy potřeba zajistit, že v různě variabilním prostředí budou fungovat správně. Díky nim se tak například mohou odhalit chyby v konstrukci nebo SW, které vedou k poklesu tlaku v převodové skříni a znemožnění dalšího řazení. Výsledky těchto testů jsou shrnuty v závěrečných reportech, které jsou poté předány vývojářům a na jejich základě bude rozhodnuto o dalším postupu. To samé platí i pro další komponenty vozidla.

Ukázkový průběh měření aktuální a požadované polohy spojky v automobilu je znázorněno na obrázku 47. Z obrázku je patrné, že systém na požadovanou změnu polohy spojky reaguje se zpožděním a v případě držení stejné hodnoty dochází k nepatrným oscilacím kolem této hodnoty. Výkyv požadované polohy spojky v 98 vteřině je dán výpočtem polohy spojky, kdy dochází k přepnutí mezi módem pro řízení polohy a časovým módem.



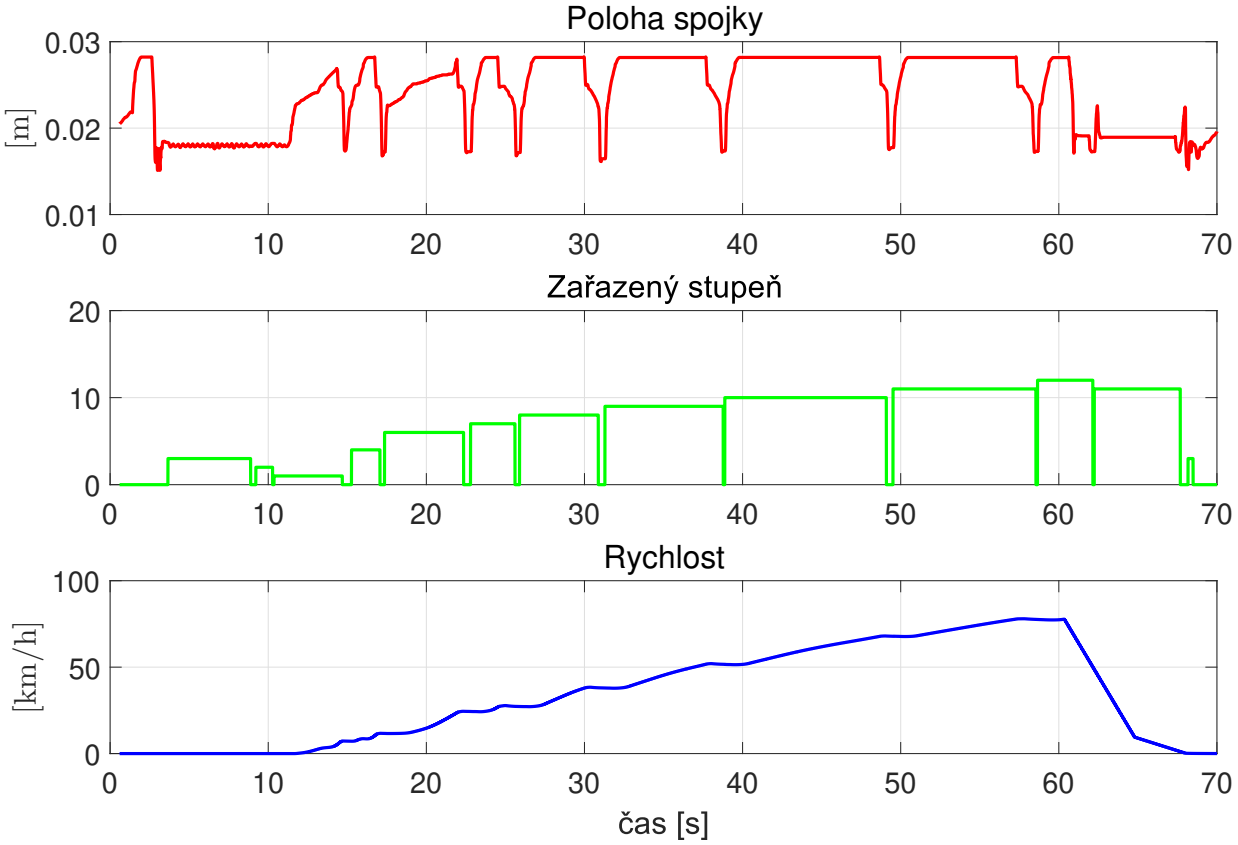
Obrázek 47: Srovnání průběhu signálů pro požadovanou a skutečnou polohy spojky z reálného zařízení

5.7 Porovnání výpočetní náročnosti - CPU load

Při porovnávání výpočetní náročnosti na reálném procesoru jsme srovnávali dvě verze SW. Prvním byl **referenční SW před transformací**. Jednalo se tedy o kód, ve kterém byl regulátor spojky napsán ručně. Ve druhém SW byl již regulátor spojky nahrazen **automaticky generovaným kódem**, který byl získán z upraveného modelu. Předtím, než jsme mohli přistoupit k této fázi testování, museli jsme zajistit, že vygenerovaný kód poskytuje stejnou funkcionalitu jako ručně psaný kód. Pro představu o velikosti modelu lze uvést, že se komponenta modelu regulátoru spojky, ze které byl generován kód, skládá z 5612 jednotlivých bloků a 35 bloků StateFlow.

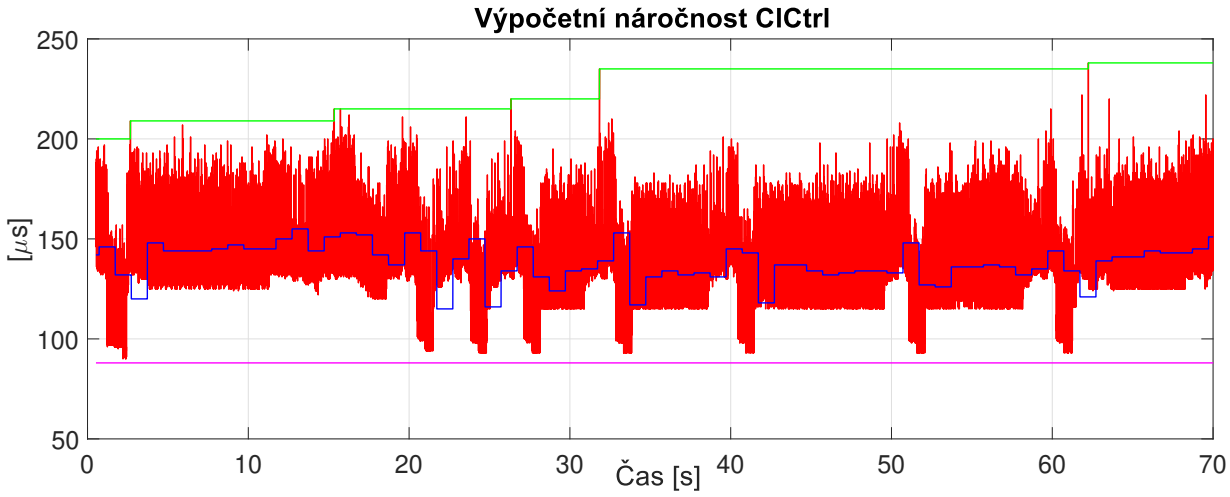
Pro měření náročnosti na CPU bylo vybráno **sedm testovacích scénářů**, kde každý definoval jiný test, který se se spojkou a převodovkou provedl. Jednalo se o testy na určení důležitých bodů spojky jako je například bod dotyku, jízdu vpřed a vzad, automatické a manuální řazení a jízdu do kopce a z kopce. Jednotlivé testy byly definovány v programu MicroNova EXAM a byly spouštěny na HILu, který byl postaven z komponentů od firmy dSPACE [40]. Porovnání bylo provedeno pomocí 200 Monte Carlo simulací pro každý testovací scénář. Monte Carlo je stochastická metoda, která využívá pseudonáhodná čísla.

Celkem bylo tedy spuštěno 1400 testovacích sekvencí pro jednu verzi SW. Při porovnávání SW před a po transformaci bylo tedy celkem spuštěno **2800 sekvencí**. V závislosti na charakteru jednotlivých testových scénářů potřeboval každý SW pro vykonání všech příkazů rozdílnou dobu. Výsledné hodnoty byly počítány jako aritmetický průměr pro všechny scénáře. Dále byla zaznamenána maximální a minimální hodnota výpočetní náročnosti. Pro názornost zde bude uveden příklad průběhu scénáře pro automatickou jízdu vpřed. Na obrázku 48 je znázorněn průběh některých důležitých signálů při tomto scénáři. Jedná se o signály pro polohy spojky, zařazený převodový stupeň a rychlost.



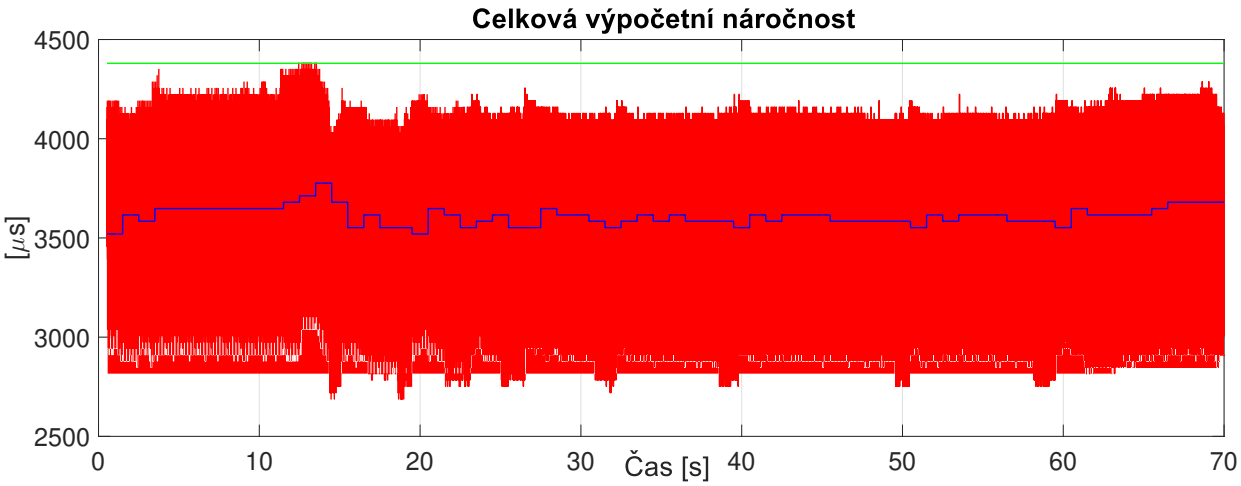
Obrázek 48: Průběh některých signálů pro testovací scénář automatické jízdy vpřed

Při každém scénáři byla také měřena výpočetní náročnost a celkový spotřebovaný čas CPU. Na obrázku 49 je znázorněn průběh signálů jednotlivých signálů pro **komponentu regulátoru spojky** (ClCtrl). Zelená křivka značí maximální dosaženou hodnotu zatížení a růžová minimální. Červená křivka značí vytížení CPU v jednotlivých časových okamžicích. Modrá je pak střední hodnota zatížení CPU v čase. Na ose y je definován výpočetní čas v mikrosekundách $[\mu s]$, to platí i pro další grafy.



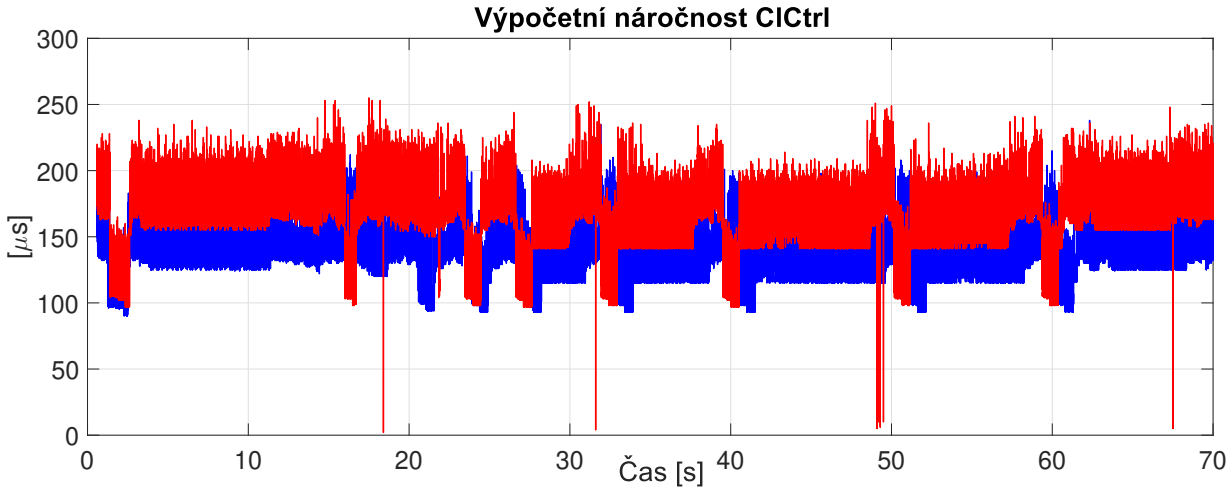
Obrázek 49: Výpočetní náročnost komponenty regulátoru spojky

Na obrázku 50 jsou pak znázorněny trajektorie jednotlivých signálů v kontextu **celého SW**. Zelená křivka značí maximální dosaženou hodnotu zatížení, takto posunutá je z důvodu prvního kroku spuštění programu, kde se zatížení CPU na jednu periodu prudce zvýší. V případě růžové linky, která značí minimální dosaženou hodnotu, došlo po prudkém nárůstu zatížení CPU na začátku běhu programu k jeho prudkému poklesu a z důvodu zachování přehlednosti grafu není tato trajektorie znázorněna. Červená křivka značí opět vytížení CPU v jednotlivých časových krocích. Modrá křivka představuje opět střední hodnotu zatížení CPU v čase.



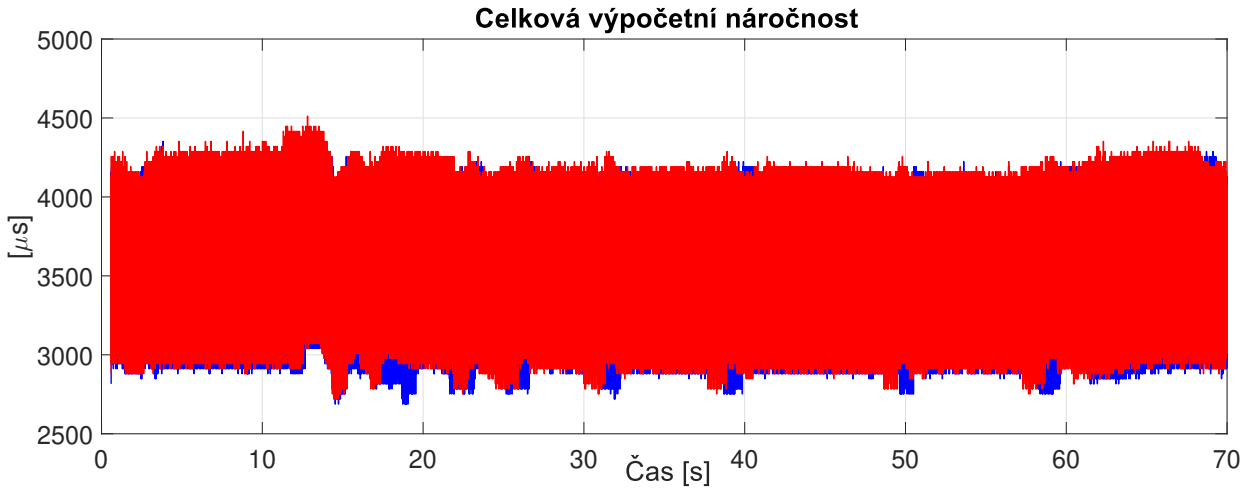
Obrázek 50: Výpočetní náročnost celého SW s automaticky vygenerovaným kódem

Porovnání náročnosti komponenty regulátoru spojky před a po transformaci je znázorněno na obrázku 51. Červená křivka značí zatížení CPU v jednotlivých časových krocích pro referenční SW. Modrá křivka znázorňuje zatížení CPU pro SW s automaticky generovaným kódem. Z grafu číslo 51 je tedy patrné, že došlo k výraznému poklesu náročnosti na zatížení CPU mezi referenčním SW a automaticky generovaným kódem.



Obrázek 51: Porovnání výpočetní náročnosti pro referenční SW a nový SW s integrovaným automaticky generovaným kódem

Porovnání náročnosti jednotlivých verzí výsledného SW je znázorněna na obrázku 52. Červená křivka značí zatížení CPU v jednotlivých časových krocích pro referenční SW. Modrá křivka značí zatížení CPU pro SW s automaticky generovaným kódem. Z grafu je patrné, že tedy v kontextu celého SW došlo pouze k poměrně malému zlepšení.



Obrázek 52: Porovnání výpočetní náročnosti pro obě verze výsledného SW

Při analýze výsledků všech 7 simulací pro jednotlivé verze SW bylo zjištěno, že průměrná hodnota **výpočetní náročnosti se snížila o 16,98 %** celkového výpočetního času ze všech testů. Maximální hodnota snížení výpočetní náročnosti, ke které v testech došlo, byla o 18,52 %. Minimální naměřená hodnota snížení výpočetní náročnosti byla o 15,99 %. Dále byla měřena náročnost SW na paměť. Jednalo se o paměť typu Random-Access-Memory (RAM). Jedná se o operační paměť typu Static Random Access Memory (SRAM) a velikosti 256 kB, ve které jsou uloženy všechny běžící programy a jejich data. Dále byla měřena náročnost na paměť typu Read-Only Memory (ROM). Velikost této paměti je 3,75 MB. Je určena pouze ke čtení a je na ní uložen firmware. Firmware je mikroprogramové vybavení, které slouží pro řízení celého CPU. Zde došlo k mírnému zvýšení náročnosti na paměť a to zejména z důvodu definování nových, dříve instancovaných parametrů, které je potřeba uložit do paměti jednotlivě pro všechny možné případy. V případě RAM došlo k nárůstu o 0.16 % a v případě ROM o 0,39%. Veškeré výsledky jsou shrnuty v tabulce 5.

Náročnost na	Generovaný kód
Výpočetní čas - max	87,01%
Výpočetní čas - min	81,48%
Výpočetní čas - průměr	83,02%
RAM	100,161%
ROM	100,38%

Tabulka 5: Porovnání náročnosti mezi ručně psaným SW a autokódem

Z výsledků shrnutých v tabulce 5 je jasně patrné, že při použití regulátoru spojky reprezentovaného pomocí autokódu, klesne průměrná hodnota výpočetního času o téměř 17%

a to pouze za nepatrného nárůstu náročnosti na paměť. Díky transformaci do autokódu došlo k **ušetření výpočetního času**, který tak může byt použit pro budoucí přidané funkcionality a růst celého SW. Došlo tak k prodloužení životnosti aktuálního řešení, což byl jeden z hlavních důvodů, proč se k transformaci do autokódu přistoupilo. Získané výsledky tedy můžeme shrnout tak, že transformace regulátoru spojky do autokódu naprosto splnila očekávání. Nárůst v náročnosti paměti byl v našem případě přijatelný z důvodu její velké dostupné velikosti.

5.8 Analýza a optimalizace regulátorů

Limitujícím faktorem pro použitou strategii řízení je především výkon ECU. Aktuální verze ECU pracuje na frekvenci **200 MHz**. Původní hodnota byla 150 MHz, avšak náročnost celého SW stoupla během let probíhajícího vývoje natolik, že bylo potřeba čip přetaktovat téměř na hranici bezpečné frekvence, která byla konzultována s výrobcem čipu. Naše úloha řízení spojky je spouštěna s periodou 5 ms. Proto aktuálně použitá strategie řízení je svým způsobem **kompromisem mezí kvalitou řízení a výpočetní náročností**. V teoretické rovině bylo vypracováno a vyzkoušeno mnoho různých návrhů, které ale zatím nejsou kvůli výpočetní náročnosti nasazeny v sérii, více v kapitole 6.3.1. Samotná struktura a princip regulátorů byly popsány v kapitole 3.2. Víme tedy, že řízení je rozděleno podle regulační chyby na regulátor pro velké a malé změny. Parametry obou regulátorů byly získány na speciálním zkušebním zařízení ve spolupráci s univerzitou v **Německém Rostocku** [25]. Na obrázku 53 je znázorněno zmiňované zkušební zařízení.

Obrázek 53: Testovací zařízení pro nastavení regulátorů

Na tomto zařízení probíhalo množství experimentů ať už pro identifikaci modelu spojky, nalezení optimální strategie řízení, následné nastavení parametrů regulátorů, tak i následné ověření výsledků. Ověření výsledků samozřejmě probíhalo také na HILech a při

validování ve zkušebním vozidle. Všechny vyzkoušené strategie jsou popsány v následující kapitole 6.4. Z důvodu firemního tajemství není možné uvést přesné hodnoty pro jednotlivé parametry regulátorů, a proto zde budou popsány pouze principiálně.

Regulátor pro velké změny má za cíl se co nejrychleji dostat z aktuální polohy spojky do cílové. Aktivuje se při větší regulační odchylce než 1 mm. Nese v sobě jednoduchý P regulátor. Nastavená hodnota zesílení byla zjištěna **experimentálně** a byla nastavena na maximální možnou hodnotu pro každý bod definovaný na křivce, kterou spojka sleduje. Díky tomu je zajištěna co nejrychlejší odezva, ale zároveň není způsoben nežádoucí překmit, případně oscilace, v přechodové charakteristice.

Regulátor pro malé změny má za cíl držet spojku v dané poloze nebo sledoval malé změny kolem aktuálního pracovního bodu, při nichž je regulační odchylka menší než 1 mm. Ovládání pozice spojky řídí PID regulátor. **Veškeré parametry regulátorů byly opět získány experimentálně** při vyzkoušení různých metod návrhu regulátoru a byly sestaveny funkce, které umožňují přepočítávání parametrů regulátoru v závislosti na tom, kde se nacházíme na mechanické křivce spojky. Proporcionální složka byla nastavena na maximální možnou hodnotu, která poskytuje co nejrychlejší odezvu, ale zároveň nezpůsobí nežádoucí překmit, popřípadně oscilace v přechodové charakteristice. Derivační složka ovlivňuje rychlost reakce na změnu v požadované hodnotě. Zde však při vysokých hodnotách dochází i k zesílení šumu, což může vést až k nestabilitě systému, tedy až k poškození celého zařízení. Cílem je maximální možné potlačení šumů při maximální možné hodnotě. Proto se zde používá regulátor v kombinaci s filtrem, například s FIR filtrem, jehož řád se nastavoval tak, aby bylo vyhověno všem požadavkům na kvalitu regulace. Derivační složka je nastavena na maximální možnou hodnotu při dodržení výše zmíněných kriterií. To platí pro všechny pracovní body na sledované křivce. Integrační složka regulátoru se v běžném provozu nepoužívá a slouží zde pouze k ošetření stavů, kdy dochází k poklesu tlaku v převodové skříni, tedy při úniku média. Pokud si s poklesem tlaku regulátor nedokáže poradit sám, tak dochází ke spuštění sekvence, která má za následek zvýšení otáček motoru. To vede k ohřevu celého ústrojí, což by mělo mít za následek díky teplotní roztažnosti materiálů, utěsnění všech průniků. Pokud ani to nepomůže, tak dochází k signalizaci na přístrojové desce a z bezpečnostních důvodu je znemožněno pokračování v jízdě.

5.8.1 Další pokročilé metody regulace

Společně s univerzitou v Rostocku bylo v ZF vyzkoušeno mnoho různých strategií řízení a návrhů regulátorů. Téměř všechny dosahovaly uspokojivých výsledků, tzn. že v autě nebyl pozorován a ani cítit žádný rozdíl. Při testování a v Matlabu se při analýze výsledků ukázalo, že rozdíly kvality regulace jsou minimální. Jednotlivé realizace se lišily použitou metodou řízení a výpočetními nároky. Pro názornost zde budou uvedeny některé vyzkoušené strategie řízení. Nejlepší výsledky dává Ricattiho regulátor, to je ale vykoupeno značnou výpočetní náročností, kvůli které tento typ není zatím nasazen v sériové výrobě [25].

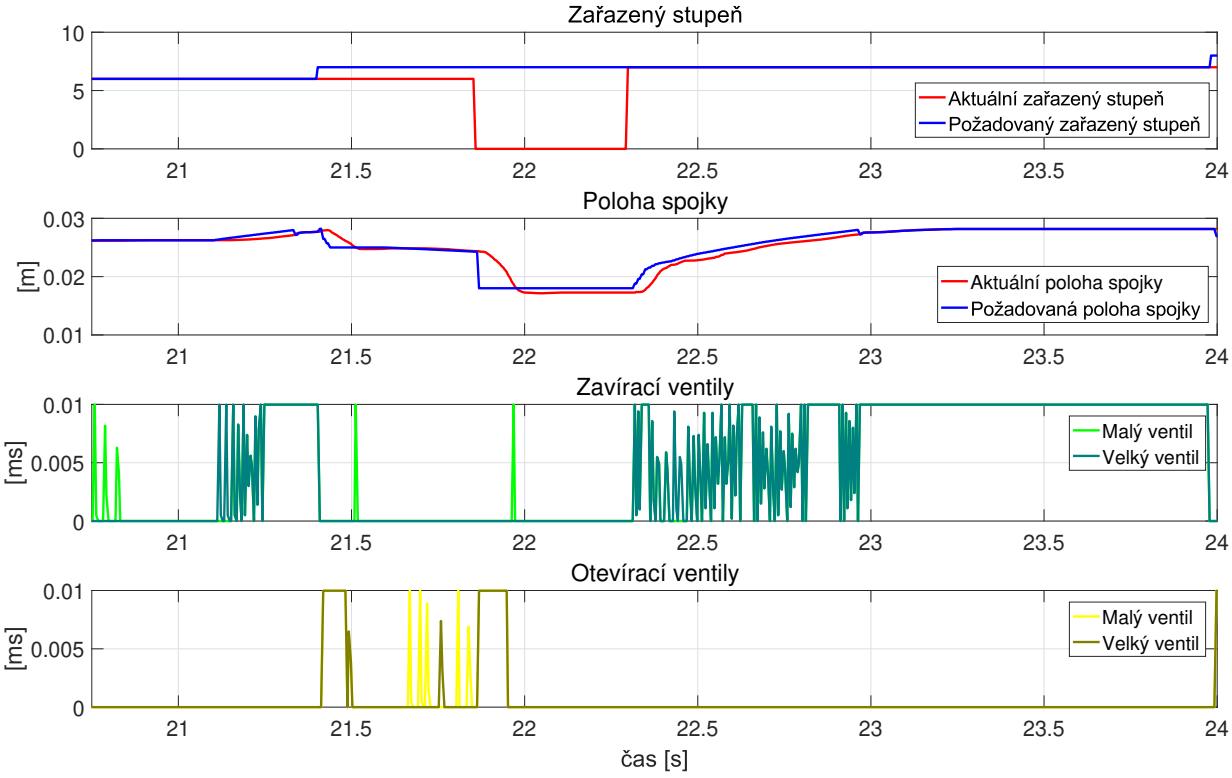
- **Adaptivní Ackrmanův regulátor** - Jedná se o regulaci pomocí stavového popisu, variabilního zesílení pro jednotlivé stavy a linearizaci v jednotlivých pracovních bodech[25].

- **Quasi lineární regulátor** - Oproti stávajícímu regulátoru je navíc rozšířen o kompenzaci třecích sil, které se v systému vyskytují [25].
- **Ricattiho adaptivní kaskádní regulátor** - Při realizaci tohoto regulátoru byl použit dvou krokový horizont řízení [20].
- **Adaptiv cascading backstepping controller** - Využívá Lypunovovu funkci, používá se kvadratické kritérium [25].
- **Smithův prediktor** - Byl využit pro kompenzaci pásma necitlivosti. Zkoušel se použít u všech typů regulátorů, ale jak se ukázalo, tak pro řešení tohoto problému není vhodný. Zejména z důvodu kdy se regulační odchylka příliš zvětší, tak prediktor začne generovat nesmyslné výsledky, což není žádoucí [21].
- **Optimalizovaný nelineární regulátor** - Ten je reprezentován pouze P regulátorem, tedy pouze parametrem K_p. Takovýto regulátor kompenzuje pouze mechanickou část systému a pneumatická část systému je kompenzována v dopředné vazbě [25].

Aktuálně probíhá testování poslední zmíněné verze - Optimalizovaný nelineární regulátor. A pokud všechny testy dopadnou dobře a všechna očekávání budou naplněna, tak pravděpodobně dojde k jeho použití, tedy nasazení v sériové výrobě.

5.9 Aktuální chování regulátoru spojky

Typické trajektorie pro aktuální zařazený stupeň, požadovaný stupeň, požadované polohy spojky, aktuální polohy spojky a aktivace ventilů při změně převodového stupně pro SIL simulace jsou zobrazeny na obrázku 54. Z obrázku je patrný vliv aktivace ventilů ventilů na aktuální polohu. Dále lze říct, že přeřazení na nový převodový stupeň proběhne za méně než jednu sekundu. Jde o přeřazení ze stupně pět na stupeň šest.



Obrázek 54: Aktuální chování regulátoru spojky a znázornění aktivace jednotlivých ventilů

6 Závěr - zhodnocení celé transformace

Tato práce se zaměřuje na vývoj SW v automobilovém průmyslu. Nejprve je uvedena historie rozvoje automobilového průmyslu. Následně je pojednáno o samotném vývoji SW a jeho možných způsobech implementace. Jednou možností je tradiční psaní zdrojového kódu ručně. Druhou možnost představuje nový směr tvorby SW, tzv. blokově orientované vytváření kódu. Práce popisuje srovnání těchto dvou způsobů tvorby SW. Jelikož se autor sám podílí na vývoji SW v jedné z největších firem v automobilovém průmyslu, diplomová práce se tak úzce zaměřuje na specifickou automatickou převodovku pro kamiony a další těžká průmyslová vozidla, kde „centrální mozek“ převodovky tvoří CPU, na kterém se spouští celý výsledný SW složený z mnoha komponent. Jednou z takovýchto komponent je spojka a právě systém regulátoru spojky je dále přiblížen a popsán. Dále na modelu regulátoru spojky je demonstrována a popsána transformace do podoby, kdy je model schopný automatického generování kódu. V této částí je také popsána tvorba blokově orientovaného kódu a následná integrace vygenerovaného kódu do ručně psaného kódu. Po těchto krocích následuje testování a validace získaného SW. Je zde uveden popis všech použitých testovacích možností jako je MIL, SIL, HIL, ale i jiné prostředky testování, které se v automobilovém průmyslu používají. Pro validaci výsledků celé transformace je provedeno porovnání výpočetní náročnosti mezi referenčním ručně psaným SW a SW založeným na automaticky generovaném kódu. Na závěr je provedena analýza aktuálně použitých regulačních technik a jsou zde uvedeny další možné způsoby regulace.

Vzhledem k neustále se zvyšujícím nárokům na kvalitu výrobků v souvislosti s omezením zdrojů se společnosti snaží o maximální efektivnost během vývoje. To má za následek optimalizaci stávajících metodik a investic do automatizace celého procesu. Tento rozvoj na poli automatizace umožňuje používání stále sofistikovanějších a efektivnějších metod vývoje. Ať už se jedná o agilní přístup nebo metody Continuous engineeringu, jejichž význam se rozšiřuje napříč celým průmyslem a není tak zaměřeno pouze na vývoj SW. Vývoj založený na blokově orientovaném programování rozšířeném o automatické generování kódu vytváří vhodný základ pro efektivní vývoj SW. Schopnosti moderních generátorů kódů umožňují dokonce vytvářet efektivnější kód, než bychom dosáhli pomocí ručně psaného kódu, nicméně nelze snížit nároky na paměť a výpočetní nároky pro libovolnou komponentu. V případě komponenty regulátoru spojky došlo k výrazně lepším výsledkům než za použití klasického postupu při vývoji. Výpočetní náročnost nového SW celé komponenty výrazně klesla. Díky snížení náročnosti je možno na stejném cílovém HW spustit složitější výsledný kód. Ten tak může obsahovat více/složitější funkcionality. Došlo tak k prodloužení životnosti celého řešení. Nicméně tyto výsledky nelze označit za všeobecně platné. Záleží na typu a stavbě jednotlivých komponent. Zde je tedy nesmírně důležitá zkušenost s celou problematikou a i jistý cit vývojářů. Na druhou stranu lze tvrdit, že převod ručně psaného kódu do blokově orientovaného kódu zvyšuje přehlednost a udržitelnost. Další vývoj a rozšiřovaní metod Continuous engineeringu, agilního přístupu a generování kódu závisí na vývoji specifických postupů pro běžně komerčně používaná pracovní prostředí a je úzce spjato s cenou licencí za jednotlivé nástroje.

Na základě poznatků uvedených v této diplomové práci vznikl vědecký článek, který byl přijat k prezentaci na mezinárodní odbornou konferenci „The 24^{th} IEEE Conference on Emerging Technologies and Factory Automation“ (ETFA), konanou v září ve Španělsku.

Seznam Všech použitých zkratek

- **AMC**-American Motors Company
- **AUTOSAR**-AUTomotive Open System ARchitecture
- **A2L**-ECU description file
- **ClCtrl**-Clutch Control
- **CPU**-Central Processing Unit
- **D**-Drive
- **DFF**-Datafield Factory
- **EBA**-Emergency Brake Assist
- **ECU**-Electronic Control Unit
- **ESP**-Electronic Stability Program
- **ETFA**-Emerging Technologies and Factory Automation
- **EXAM**-EXtended Automation Method
- **FIR**-Finite Impulse Response
- **FMI**-Functional Mock-up Interface
- **FMU**-Functional Mock-up Unit
- **FSS**-MXAM Functional Safety Solution
- **GPS**-Global Positioning System
- **GUI**-Graphical User Interface
- **HW**-Hardware
- **HIL**-Hardware In the Loop
- **LDW**-Line Departure Warning
- **MAT**-MAT-File Level 5 File Format (v5, v6, v7)
- **MBSD**-Model Based Software Development
- **MC**-Model Commander
- **MCS**-MXAM MISRA Compliance Solution
- **MES**-MOdel Engineering Solutions GmbH
- **MIL**-Model In the Loop

- **MXRAY**-MES M-XRAY
- **N**-Neutral
- **OEM**-Original Equipment Manufacturer
- **OIL**-Open Image Library format
- **P**-Parking
- **PD**-Proporcionální derivační regulátor
- **PI**-Proporcionálně integrační regulátor
- **PID**-Proporcionálně integračně derivační regulátor
- **PIL**-Processor in the loop
- **PWM**-Pulse Width Modulation
- **R**-Reverse
- **RAM**-Random Access Memory
- **ROM**-Read Only Memory
- **SIL**-Software In the Loop
- **SRAM**-Static Random Access Memory
- **SW**-Software
- **TL MIL**-TargetLink model in the loop
- **VSC**-Vehicle stability Control
- **WABCO**-Westinghouse Air Brake Company
- **XIL**-X In the Loop
- **XLS**-Microsoft Excel formát
- **XML**-Extensible Markup Language
- **ZF**-ZF Friedrichshafen AG

Seznam obrázků

1 Logo ZF 1
2 Počet vyrobených automobilů v jednotlivých obdobích v tisících 5
3 Benz Patent-Motorwagen No. 1 a Mercedes třídy S rok 2018 [47, 63] 6
4 Důležité milníků automobilového vývoje na časové ose 7
5 Časová osa aktivity společnosti ZF v oblasti elektromobility 10
6 Znázornění elektromobility v závislosti na váze vozidla a denním nájezdu kilometrů 11
7 Struktura V-modelu jako procesu pro vývoj SW 13
8 Struktura W-modelu jako procesu pro vývoj SW 14
9 Ilustrace vývoje SW pomocí MBSD ve spojení s běžně používanými nástroji v automobilovém průmyslu 15
10 Historie používání automatického generování kódu v rámci ZF 16
11 Klasický přístup vývoje SW při ručně psaném kódu 17
12 MBSD přístup vývoje SW při automaticky generovaném kódu 18
13 Schématické znázornění motoru, spojky, převodové skříně a hřídele 19
14 Ilustrace porovnání výkonové křivky pro manuální a automatickou převodovku 20
15 Řez manuální převodovkou používanou BMW [9] 20
16 Řez automatickou převodovkou ZF 8HP 21
17 Zobrazení těla převodovky EL40 TraXon s doplňkovými modulárními díly 23
18 ZF převodovka EL57 OptiDrive Ecomid 23
19 Schéma mechanické a pneumatické části systému spojky 26
20 Významné body polohy spojky 27
21 Schéma pneumatické části systému spojky 28
22 Charakteristiky ventilů pro derivaci hmotnostního toku $\dot{m}$, vlevo pro otevírání spojky, vpravo pro zavírání spojky [25] 31
23 2D charakteristika ventilů spojky pro hmotnostní tok [25] 32
24 Křivka pohybu spojky pro její otevírání a zavírání 35
25 Vyznačení regulační odchylky, aktuálního a požadovaného pracovního bodu na křivce po níž se spojka pohybuje 36
26 Princip funkce regulátoru pro malé změny 37
27 Princip funkce regulátoru pro velké změny 39
28 Ilustrace principu funkce a komunikace Data Dictionary 43
29 Ilustrace knihovny TargetLinku (vlevo) a PASS-libu (vpravo) 44
30 Ilustrace podoby modelu před transformací do autokódu 44
31 Nová struktura regulátoru spojky pro generování kódu 45
32 Ilustrace vztahů mezí MC bloky, FUNCTION bloky a generováním kódu . 47
33 Prostředí programu MXAM se zobrazeným výsledkem testu 49
34 Příklad modelu matematického výrazu pro generování kódu vytvořeného pomocí TargetLink bloků 51
35 Ukázka struktury nejvyšší úrovně modelu pro generování kódu pomocí nástroje TargetLink 52
36 Ilustrace různých simulací založená na běžném testovacím přístupu 55
37 Princip struktury XIL standardu 56
38 Ilustrace myšlenky „continuous engineeringu“ 56

39 Prostředí programu SoftCar 59
40 Princip funkce a interakce programu TestView s dalšími nástroji 60
41 Prostředí programu TestView 60
42 Příklad realizace testu v TestView 61
43 Prostředí programu imc FAMOS 62
44 Prostředí programu ControlDesk 64
45 Prostředí programu EXAM 64
46 Prostředí programu CANape 65
47 Srovnání průběhu signálů pro požadovanou a skutečnou polohy spojky z reálného zařízení 68
48 Průběh některých signálů pro testovací scénář automatické jízdy vpřed . . 69
49 Výpočetní náročnost komponenty regulátoru spojky 69
50 Výpočetní náročnost celého SW s automaticky vygenerovaným kódem . . . 70
51 Porovnání výpočetní náročnosti pro referenční SW a nový SW s integrovaným automaticky generovaným kódem 70
52 Porovnání výpočetní náročnosti pro obě verze výsledného SW 71
53 Testovací zařízení pro nastavení regulátorů 72
54 Aktuální chování regulátoru spojky a znázornění aktivace jednotlivých ventilů 75
55 Trojité V - model pro vývoj SW 86

Seznam tabulek

1 Tabulka popisu fyzikálních veličin pro popis systému spojky 25
2 Tabulka použitých veličin pro regulaci 34
3 Tabulka použitých komerčních nástrojů 42
4 Tabulka použitých interních nástrojů 42
5 Porovnání náročnosti mezi ručně psaným SW a autokódem 71

Seznam literatury

Reference

[1] F. Abbors, D. Truscan "Approaching Performance Testing From a Model-Based Testing Perspective", *2010 Second International Conference on Advances in System Testing and Validation Lifecycle*, Nice, France, August 2010.

[2] H. Aerts, H.Schamiée, Z. Lattmann, M van Genuchten, L. Hatton, "How Software Is Changig the Automotive Landscape", *IEEE SOFTWARE Published by the IEEE computer society*, pp. 7-12, November/December 2017.

[3] N. Ajwad, "Evaluation of Automatic Code Generation Tools", April 2007.

[4] C. Andrici, A. Ipatiov, "Automotive multi-project architecture with model based development: An inside look at multi project handling within steering projects", *18th International Conference on System Theory, Control and Computing*, Sinaia, Romania, October 2014.

[5] Autoportal Team, Know everything about Satellite Navigation in cars, "`https://autoportal.com/articles/know-everything-about-satellite-navigation-in-cars-2867.html`", January 2015, Accessed: 2019-05-19.

[6] K. J. Ästrom, T. Hägglund, "PID Controllers: Theory, Design, and Tuning", *2nd Edition, Research Triangle Park : ISA-The Instrumentation, Systems and Automation Society*, 1995.

[7] K. Bimbraw, "Autonomous cars: Past, present and future a review of the developments in the last century, the present scenario and the expected future of autonomous vehicle technology", *2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, San Francisco, CA, USA, August 2017.

[8] M. Bisht, J. Abbott, A. Gaffar, "Social Dilemma of Autonomous Cars A Critical Analysis", *2015 12th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, Colmar, France, July 2015.

[9] H. Becker, The workings of a manual gearbox, "`https://www.bmwblog.com/2015/04/20/the-workings-of-a-manual-gearbox/`", April 2015, Accessed: 2019-05-19.

[10] Daimler, Company History, Benz Patent Motor Car: The first automobile(1885–1886),"`https://www.daimler.com/company/tradition/company-history/1885-1886.html`", February 2006, Accessed: 2019-05-19.

[11] Daimler, Airbag and belt tensioner – world premiere in 1981, "`https://media.daimler.com/marsMediaSite/en/instance/ko/`

Airbag-and-belt-tensioner--world-premiere-in-1981.xhtml?oid=9913288", Accessed: 2019-05-19.

[12] dSPACE, TargetLink, "https://www.dspace.com/en/inc/home/products/sw/pcgs/targetli.cfm" ,Accessed: 2019-05-19.

[13] M. Čech, J. Königsmarková, J. Reitinger, P. Balda, "Novel tools for model-based control system design based on FMI/FMU standard with application in energetics", *Proceedings of the 2017 21st International Conference on Process Control*, Štrbské Pleso, Slovakia, June 2017, 7976250, pp. 416-421

[14] T. Erkkinen, M. Conrad, "Safety-Critical Software Development Using Automatic Production Code Generation", *SAE Technical Paper 2007-01-1493*, 2007.

[15] European Automobile Manufacturers' Association, "https://www.statista.com/statistics/262747/worldwide-automobile-production-since-2000/" ,Accessed: 2019-05-19.

[16] M. Chunyang, S. Lining, D. Zhijiang, C. Yanchun, "Method Based on OSEK/VDX Platform Using Model-based and Autocode Technology for Diesel ECU SW Development", *31st Annual International Computer Software and Applications Conference*, Beijing, China, July 2007.

[17] Z. Fei, Z. Xiaolin, Z. Junjun and H. Jingsheng, "Hardware-in-the-loop Simulation, Modeling and Close-Loop Testing for Three-level Photovoltaic Grid-connected Inverter Based on RT-LAB", *2014 International Conference on Power System Technology (POWERCON 2014)* , Chengdu, China, October 2014.

[18] B. Fitzgerald and K.-J. Stol, "Continuous Software Engineering and Beyond: Trends and Challenges", *1St International Workshop on Rapid Continuous Software Engineering*, At Hyderabad, India, June 2014.

[19] H. Gao, T. Zhang, H. Chen, Z. Zhao, K. Song, "Application of the X-in-the-Loop Testing Method in the FCV Hybrid Degree Test", *Energies*, vol. 11, February 2018.

[20] S.E. Hamamci, I. Kaya and D.P. Atherton, "SMITH PREDICTOR DESIGN BY CDM", *2001 European Control Conference (ECC)*, Porto, Portugal, September 2001.

[21] R. Herzallah, "Non Standard Ricatti Solution and Linear Quadratic Pinning Control", *19th Mediterranean Conference on Control and Automation*, Corfu, Greece, June 2011.

[22] M. Hu, Y. Huang, Ch. Zhao, X. Di, B. Liu, H. Li, "Model-based development and Automatic Code Generation of Powertrain Control System", *ITEC Asia-Pacific 2014*, Beijing, China, September 2014.

[23] International Organization for Standardization, "ISO 26262 (2011): Road vehicles - Functional safety", Geneva, Switzerland, 2011.

[24] International Organization for Standardization, "ISO 26262-1:2018, Road vehicles - Functional safety", December 2018.

[25] Interní materiály společnosti ZF.

[26] T. Iwagaya and T. Yamaguchi, "Speed improvements for xIL Simulation based on Symbolic-Algebraic method", *SICE Annual Conference 2013*, September 2013.

[27] B. Kimbrough, Chevrolet HEI Distributor Casting Number Reference, "`https://www.enginelabs.com/news/chevrolet-hei-distributor-casting-number-reference/`" , April 2014, Accessed: 2019-05-19.

[28] A. Koc and A. Uz Tansel, "A Survey of Version Control Systems", January 2015.

[29] J. Koscs, Anti-lock Brakes: Who Was Really First?, "`https://www.hagerty.com/articles-videos/articles/2013/04/09/antilock-brakes`", April 2013, Accessed: 2019-05-19

[30] K. Kubíček, "Vývoj metod pro pokročilou integraci simulačních nástrojů na bázi standardu "FMI 2.0 for Co-Simulation"a její validace na modelu vybraného energetického zařízení", *bakalářská práce*, 2017.

[31] S. Kugele, "Model-Based Development of Software-intensive Automotive Systems", 2012.

[32] D. Kuhlgatz, Bosch History Blog, Sensing the adequate mixture – the Bosch Lambda Sensor, "`https://blog.bosch.com/history/en/2016/07/20/939/`" , July 2016, Accessed: 2019-05-19.

[33] E. Laukkanen and M. V. Mäntylä, "Build Waiting Time in Continuous Integration – An Initial Interdisciplinary Literature Review", *in Proceedings of the IEEE/ACM 2nd International Workshop on Rapid Contínuous Software Engineering*, Florence, Italy, May 2015, pp. 1-6.

[34] K. Lee, H. Peng, "Evaluation of automotive forward collision warning and collision avoidance algorithms", *Article in Vehicle System Dynamics 43(10):735-751*, November 2005.

[35] Y. Liu, Y. Li, R.Zhuang, "The Application of Automatic Code Generation Technology in the Development of the Automotive Electronics Software", *Applied Mechanics and Materials*, vol. 321-324, pp. 1574-1577, 2013.

[36] D. Marijan, A. Gotlieb1, M. Liaaen, "A learning algorithm for optimizing continuous integration development and testing practice", *Special Issue: Software Engineering in Practice*, vol. 49, pp. 192-213, February 2019.

[37] Ph. Martin, R. M. Murray, P. Rouchon "Flatness based design", *Notes for EOLESS*, January 2002.

[38] J. J. A. Mendes Jr., J. H. Z. Neme, M. M. D. Santos, S. L. Stevan Jr., "Model and Software in the Loop with Automatic Code Generation for Indicator Lights Warnings", *Journal of Applied Instrumentation and Control*, vol. 5, pp. 21-27, August 2017.

[39] M. Meyer, "Continuous Integration and Its Tools", *IEEE SOFTWARE Published by the IEEE computer society*, pp. 14-16, May/June 2014.

[40] MicroNova Software and System, EXtended Automation Method (EXAM), "`https://www.exam-ta.de/en/about-exam/exam-provides.html`", Accessed: 2019-05-19.

[41] Model Engineering Solutions (MES), MES Model Examiner® (MXAM), "`https://model-engineers.com/en/quality-tools/mxam/`", Accessed: 2019-05-19.

[42] Model Engineering Solutions (MES), Quality Tools, "`https://model-engineers.com/de/quality-tools/`", Accessed: 2019-05-19.

[43] M. Muresan, D. Pitica, "Software in the Loop Environment Reliability for Testing Embedded Code", *: 2012 IEEE 18th International Symposium for Design and Technology in Electronic Packaging (SIITME)*, October, 2012.

[44] H. Neema, J. Gohl, Z. Lattmann, J. Sztipanovits, G. Karsai, S. Neema, T. Bapty, J. Batteh, H. Tummescheit, Ch. Sureshkumar, "Model -Based Integration Platform for FMI Co -Simulation and Heterogeneous Simulations of Cyber-Physical Systems", *10th International ModelicaConference*, Lund, Sweden, March 2014.

[45] P. Niedermeyer, Automotive History: 1957 Chevrolet Fuel-Injected 283 V8 – Ahead Of Its Time And The Competition, "`http://www.curbsideclassic.com/blog/the-1957-chevrolet-fuel-injected-283-v8-gms-greatest-hit-12/`", June 2016, Accessed: 2019-05-19.

[46] P. Patanakul and R. Rufo-McCarron, "Transitioning to agile software development: Lessons learned from a government-contracted program",*Journal of High Technology Management Research*, vol. 29, pp. 181-192, October 2018.

[47] PNG MART, "`http://www.pngmart.com/image/tag/mercedes-benz`", Accessed: 2019-05-19.

[48] J. Sajdl, ESP (Electronic Stability Programme), "`http://www.autolexicon.net/cs/articles/esp-electronic-stability-programme/`", Accessed: 2019-05-19.

[49] V. Schereiber, V. Ivanov, K. Augsburg, M. Noack, B. Shyrokau, C. Sandu, P. Schalk Els, "Shared and Distributed X-in-the-Loop Tests for Automotive Systems: Feasibility Study", *IEEE Access*, vol. 6, pp. 4017-4026, January 2018.

[50] M. Schlager, M. Conrad, "Hardware-in-the-Loop Simulation', *VDM Verlag*, Saarbrücken, Germany, 2008.

[51] O. Schneider, T. Mindel, J. Liebmann, R.G. Ramos, "Model based software development – solutions for series software", *15th Interationales Stuttgarter Symposium*, pp. 1551-1560, Stuttgart, Germany, May 2015.

[52] S. K. Shukla, "Model-Driven Engineering and Safety-Critical Embedded SW", *Embedded Computing published by the IEEE Computer Society*, September 2009.

[53] J. Sobota, M. Goubej, J. Königsmarková, M. Čech, "Raspberry Pi-based HIL simulators for control education", *Pre-print accepted for IFAC ACE 2019*, Philadelphia, United States, July, 2019. J. Sobota, M. Goubej, J. Konigsmarkova, M. Cech Raspberry Pi-based HIL simulators for control education Pre-print accepted for IFAC ACE 2019, July 2019, Philadelphia, USA

[54] R. Siegel, History of obsolete car audio, part 1: Early radio, "`https://www.hagerty.com/articles-videos/articles/2017/12/11/history-of-early-radio`", December 2017, Accessed: 2019-05-19.

[55] A. Spillner, "The W-MODEL – Strengthening the Bond Between Development and Test", January 2000.

[56] N. Srinivas, N. Panditi S. Schmidt, R. Garrelfs, "MIL/SIL/PIL Approach A new paradigm in Model Based Development", *The Journal of Systems and Softwares*, Continental AG, July 2014.

[57] D. Ståhl and J. Bosch, "Modeling continuous integration practice differences in industrySW development", *The Journal of Systems and Softwares*, vol. 87, pp. 48-59, 2014.

[58] E. Şaykol "An Economic Analysis of Software Development Process based on Cost Models", *International Conference on Eurasian Economies*, At Almaty, Kazachstan, October 2012.

[59] Tesla Inc., "`https://www.tesla.com/`", Accessed: 2019-05-19.

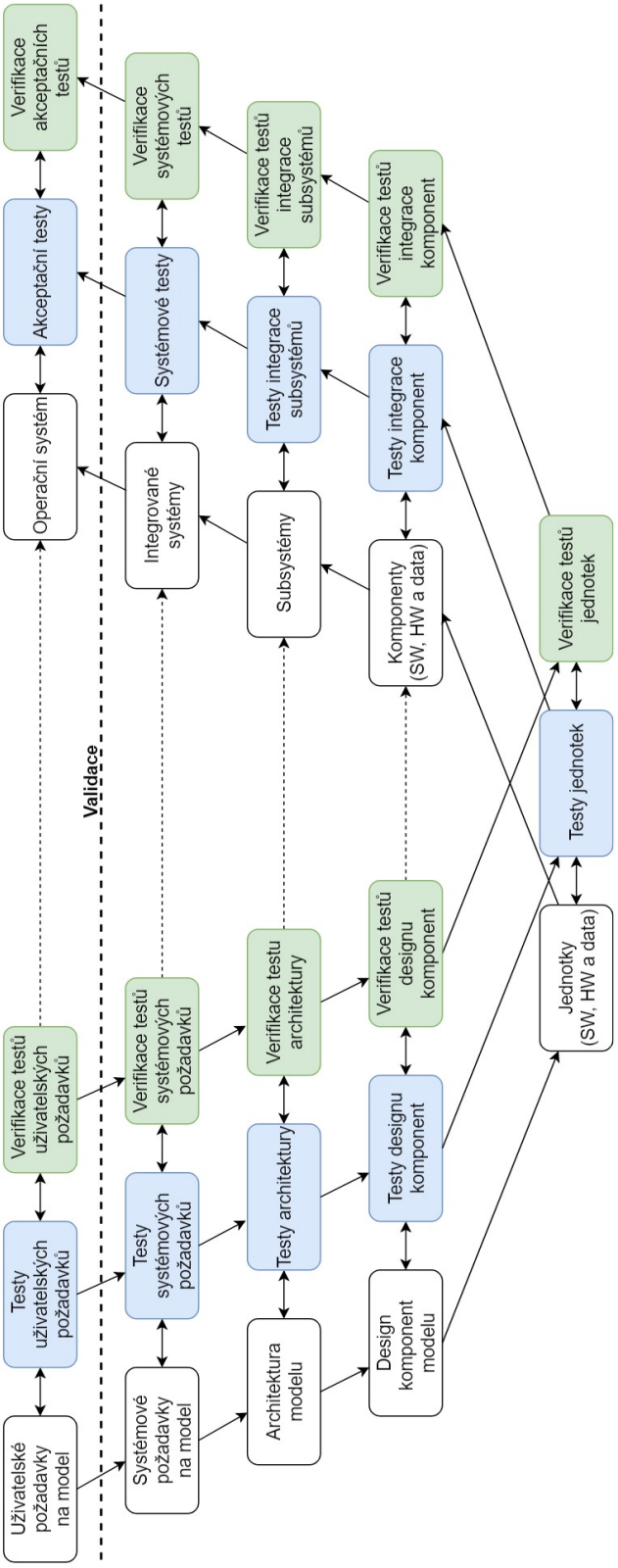
[60] G. Tibba, Ch. Malz, Ch. Stoermer, N. Nagarajan, L. Zhang, S. Chakraborty, "Testing Automotive Embedded Systems under X-in-the-Loop Setups", *35th International Conference on Computer-Aided Design*, Austin, Texas, November 2016.

[61] S. Tsugawa, "TRENDS AND ISSUES IN SAFE DRIVER ASSISTANCE SYSTEMS– Driver Acceptance and Assistance for Elderly Drivers", *IATSS Research Volume 30, Issue 2* , pages 6-18, 2006.

[62] Vector Informatik GmbH, "User manual ASAP2 Tool-Set", version 13.0, 2018.

[63] TURBOSQUID, shop3ds, "`https://www.turbosquid.com/3d-models/3d-model-benz-patent-motor-car/911673`", Accessed: 2019-05-19.

7 Přílohy



Obrázek 55: Trojité V - model pro vývoj SW

Listing 1: Vygenerovaný kód z příkladu uvedeného v kapitole 4.4.1

```
1  \*-----------------------------------------------------------------------*/
2  #include <math.h>
3  #include "Prj_Common.h"
4  /*************************************************************************\
5     SLGlobal: Default storage class for global variables | Width: 32
6  \*************************************************************************/
7  fl32 Sa1_Out1;
8  fl32 Sa1_X;
9  fl32 Sa1_Y;
10 fl32 Sa1_Z;
11
12 void Sa1_Prj_Common(void)
13 {
14    /* SLLocal: Default storage class for local variables | Width: 32 */
15    fl32 Sa1_Add;
16    fl32 Sa1_Divide;
17    fl32 Sa1_Divide1;
18    fl32 Sa1_Divide2;
19    fl32 Sa1_Gain;
20    fl32 Sa1_Math_Function;
21    fl32 Sa1_Product;
22    fl32 Sa1_Sqrt;
23
24    /* Math: Prj_Common/Math Function */
25    Sa1_Math_Function = expf(Sa1_X);
26    /* Product: Prj_Common/Divide2 */
27    if (Sa1_Z != 0.F) {
28       Sa1_Divide2 = Sa1_Math_Function / Sa1_Z;
29    }
30    else {
31       /* Prj_Common/Divide2: Numerator always greater than or
32       equal to zero. */
33       Sa1_Divide2 = 3.402823466e+38F;
34    }
35    /* Product: Prj_Common/Product */
36    Sa1_Product = Sa1_X * Sa1_Y;
37    /* Sum: Prj_Common/Add
38       # combined # Trig: Prj_Common/TrigoFcn */
39    Sa1_Add = sinf(Sa1_Divide2) + Sa1_Product;
40    /* Sqrt: Prj_Common/Sqrt */
41    if (Sa1_Add <= 0.F) {
42       Sa1_Sqrt = 0.F;
43    }
44    else {
45       if (Sa1_Add >= 0.F) {
```

```
46            Sa1_Sqrt = sqrtf(Sa1_Add);
47         }
48         else {
49            Sa1_Sqrt = -sqrtf(-Sa1_Add);
50         }
51      }
52      /* Gain: Prj_Common/Gain */
53      Sa1_Gain = Sa1_Y * 5.F;
54      /* Product: Prj_Common/Divide */
55      if (Sa1_Gain != 0.F) {
56         Sa1_Divide = Sa1_Sqrt / Sa1_Gain;
57      }
58      else {
59         if (Sa1_Sqrt < 0.F) {
60            Sa1_Divide = -3.402823466e+38F;
61         }
62         else {
63            Sa1_Divide = 3.402823466e+38F;
64         }
65      }
66      /* Product: Prj_Common/Divide1 */
67      if (Sa1_Z != 0.F) {
68         Sa1_Divide1 = Sa1_Y / Sa1_Z;
69      }
70      else {
71         if (Sa1_Y < 0.F) {
72            Sa1_Divide1 = -3.402823466e+38F;
73         }
74         else {
75            Sa1_Divide1 = 3.402823466e+38F;
76         }
77      }
78      /* TargetLink outport: Prj_Common/Out1
79         # combined # Sum: Prj_Common/Add2
80         # combined # Product: Prj_Common/Product1
81         # combined # Gain: Prj_Common/Gain1
82         # combined # Sum: Prj_Common/Add1 */
83      Sa1_Out1 = ((Sa1_Math_Function * 10.F) + Sa1_Divide) -
84      ((Sa1_Product + Sa1_Divide1) * 0.001F);
85   }
```