ZÁPADOČESKÁ UNIVERZITA V PLZNI

FAKULTA ELEKTROTECHNICKÁ

KATEDRA ELEKTRONIKY A INFORMAČNÍCH TECHNOLOGIÍ

DIPLOMOVÁ PRÁCE

Simulace kanálů pro řízení ventilů automatické převodovky

Alona Hepenko **2021**

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

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

Jméno a příjmení:	**Bc. Alona HEPENKO**
Osobní číslo:	**E19N0058P**
Studijní program:	**N2612 Elektrotechnika a informatika**
Studijní obor:	**Elektronika a aplikovaná informatika**
Téma práce:	**Simulace kanálů pro řízení ventilů automatické převodovky**
Zadávající katedra:	**Katedra elektroniky a informačních technologií**

Zásady pro vypracování

Navrhněte simulátor HW pro řízení ventilů automatické převodovky

1. Systém navrhněte parametrizovatelný a modulární
2. Navrhněte interface, který by umožnil integrací do celého systému
3. Simulace musí být prováděna cyklicky podle periody hlavní smyčky systému s nastavitelnou periodou
4. Umožněte simulace základních chyb – zkrat na zem, zkrat na baterii, zkrat mezi kanály, rozpojený obvod
5. Otestujte systém porovnáním s chováním reálné řídící jednotky

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

Seznam doporučené literatury:

1. PINKER, Jiří a KOUCKÝ, Václav. Analogové elektronické systémy. 4. vyd. Plzeň: Západočeská univerzita, 2010. 2 sv. ISBN 978-80-7043-917-3.
2. Barone L.M., Marinari E., Organtini G., Tersenghi F.R. Scientific Programming: C-Language, Algorithms and Models in Science. World Scientific Publishing Co. Pte. Ltd, 2013. ISBN 978-981451340.

Vedoucí diplomové práce: **Ing. Petr Weissar, Ph.D.**
Katedra elektroniky a informačních technologií

Datum zadání diplomové práce: **9. října 2020**
Termín odevzdání diplomové práce: **27. května 2021**

L.S.

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

Doc. Ing. Jiří Hammerbauer, Ph.D.
vedoucí katedry

V Plzni dne 9. října 2020

Abstrakt

Diplomová práce se zabývá problematikou softwarové simulace řízení ventilů automatické převodovky. První část je věnována automobilové elektronice, principu funkčnosti řídicí jednotky a rozdělení elektroniky na funkční domény. Druhou částí je popis mechanické funkčnosti automatické převodovky. Třetí část je zaměřená na vysvětlení procesu vývoje softwaru v automobilovém průmyslu podle V-modelu. Čtvrtá část je věnována vysvětlení zapojení schématu pro řízení ventilu a výpočtům výstupních hodnot pro stav bez poruchy a pro poruchové stavy. V páté části je popsán vývoj simulace řízení ventilů, který je realizován podle V-modelu. Zahrnuje následující fáze: analýza požadavků, vytvoření designu, implementace a testování.

Klíčová slova

Automatická převodovka, automotive, elektronika, SIL, simulace, software, TCU, ventil, V-model.

Abstract

The diploma thesis deals with the issue of software simulation of automatic transmission valve control. The first part is devoted to automotive electronics, the principle of control unit functionality and the division of electronics into functional domains. The second part is a description of the mechanical functionality of the automatic transmission. The third part is focused on explaining the process of software development in the automotive industry according to the V-model. The fourth part is devoted to the explanation of the circuit diagram for the valve control and the calculations of the output values for the no error state and for the error states. The fifth part describes the development process of valve control simulation, which is implemented according to the V-model, therefore it contains the following phases: requirements analysis, design, implementation and testing.

Key words

Automatic transmission, Automotive, Electronics, SIL, Simulation, Software, TCU, Valve, V-model.

Prohlášení

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

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

v.r. Hepenko Alona

podpis

V Plzni dne 25.5.2021 Alona Hepenko

Poděkování

Tímto bych ráda poděkovala vedoucímu diplomové práce, Ing. Petru Weissarovi, Ph.D., za cenné profesionální rady, připomínky a metodické vedení práce. Mé poděkování patří též Ing. Václavu Mulačovi za cenné rady a odborný dohled.

Obsah

Úvod

Předkládaná diplomová práce je vypracována v rámci zaměstnání ve společnosti ZF Engineering Plzeň s.r.o. Tato práce je zaměřená na vytvoření softwarové simulace části řídicí jednotky převodovky, která je odpovědná za řízení ventilů. Tyto ventily jsou součástí automatické převodovky a míra jejich otevření zajišťuje proces automatického řazení. Proces řazení je klíčovým z hlediska správné funkčnosti převodovky a bezpečnosti řízení vozidla. Z tohoto důvodu součástí softwaru, který cyklicky běží v řídicí jednotce, je diagnostická komponenta, jež kontroluje správnou funkčnost ventilů, a v případě poruchy ji nahlásí dalším komponentám. Software pro automobilový průmysl se pořád vyvíjí, vznikají nové požadavky, které musí být implementované. Po každé změně v kódu musí být software otestován. V automobilovém průmyslu existuje několik prostředí pro provedení testů, například SIL a HIL. Aby bylo možné otestovat některé komponenty v prostředí SIL, musí být vytvořena softwarová simulace hardwaru řídicí jednotky. Diagnostická komponenta vyžaduje simulaci řízení ventilů pro provedení testů v prostředí SIL. Cílem diplomové práce je vytvořit požadovanou simulaci.

Navržená simulace musí být parametrizovatelná. To znamená, že musí existovat soubor vstupních dat obsahující parametry, které je možné měnit bez vlivu na správnost funkčnosti systému, například hodnoty součástek. Simulace musí být modulární, protože existuje několik druhů ventilů, přičemž modularita umožňuje jednoduchý a jednotný způsob konfigurace pro různé druhy ventilů. Musí být provedena integrace do již existujícího prostředí pro simulaci, to znamená, že simulace řízení ventilu musí poskytovat interface, který tuto integraci umožní. Jelikož simulace musí být zaintegrována do systému, musí správně fungovat pro zvolenou periodu hlavní smyčky systému. Pro umožnění testování diagnostiky chyb musí být implementována simulace základních chyb: zkrat na zem, zkrat na baterii, zkrat mezi kanály a rozpojený obvod. Ověření správné funkčnosti softwarové simulace musí být provedeno porovnáním hodnot ze simulace a reálných hodnot naměřených v prostředí HIL.

Seznam symbolů a zkratek

Atd. ..... a tak dále

ECU ..... Engine Control Unit

HIL ..... Hardware In the Loop

HMI ..... Human-Machine Interface

MISRA ..... Motor Industry Software Reliability Association

OCG..... Output Current Ground

ODG ..... Output Digital Ground

PWM..... Pulse Width Modulation

SIL ..... Software In the Loop

TCU ..... Transmission Control Unit

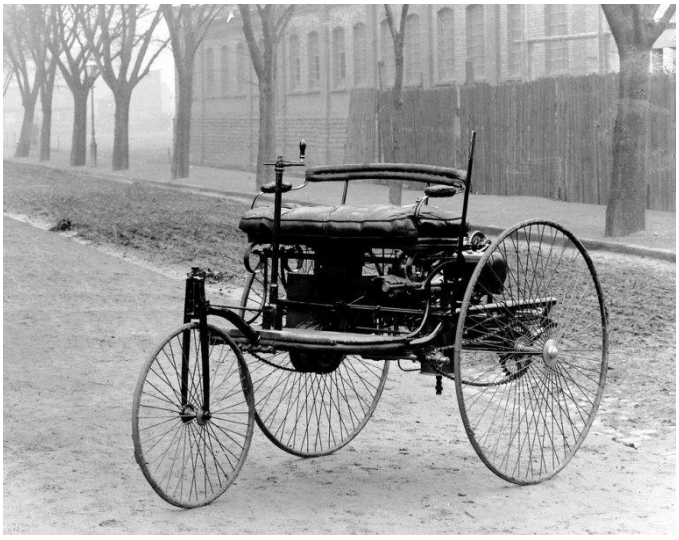
Tzv. ..... takzvaný

UML ..... Unified Modeling Language

VPS..... Voltage Power Supply

1 Elektronika v automobilu

První automobil byl patentován v roce 1886 Karlem Benzem (Obr. 1). Jako každý nový vynález i automobil ještě dlouhou dobu nebyl dostupný pro širokou veřejnost. Byl to však první krok ke zvládnutí větších vzdáleností, dosažení větších rychlostí, přepravě větších nákladů a zvýšení bezpečnosti. Od té doby byl tento stroj významně zmodifikován a proces vývoje pokračuje i nadále [35].



Obr. 1 První automobil

Významným krokem pro rozvoj automobilu bylo zavedení elektroniky. První elektronikou, která se objevila v autě, bylo elektronkové autorádio v roce 1930. Nemělo vliv na proces řízení, ale zpříjemňovalo cestování. Vývoj polovodičů po druhé světové válce značně rozšířil použití elektroniky v automobilech. Dalším významným krokem bylo zavedení tranzistorového zapalovacího systému v roce 1955. V následujících letech se do auta zavedla spousta dalších elektronických systémů a tento proces pokračuje i nadále [35].

V současné době se v autě může nacházet kolem 70 řídicích jednotek, které jsou zodpovědné za řízení motoru a převodovky, aktivní a pasivní bezpečnost, komfort ve vnitřním prostoru, zábavu. Pomocí elektroniky se dá zlepšit hodně parametrů, například plynulost a jednoduchost řízení, spotřeba paliva, bezpečnost, pohodlnost vnitřního prostoru [35].

1.1 Základní princip funkčnosti řídicí jednotky

Základní princip řízení pomocí elektroniky spočívá v tom, že řídicí jednotka dostává signály ze senzorů a od dalších řídicích jednotek, zpracovává je podle vnitřní logiky a posílá signály do akčních členů a do dalších řídicích jednotek. Když se podíváme na vnitřní logiku řídicí jednotky, základní princip spočívá v definici optimálních parametrů a jejich porovnání se skutečnými parametry. Po spuštění motoru jsou signály z mnoha senzorů přenášeny do řídicí jednotky, poté jednotka porovnává data s předem zapsanými parametry v paměti. Pokud je detekována odchylka od normy, jednotka generuje řídicí signály pro korekci, které jsou přenášeny na akční členy. Pokud není možné opravit činnost konkrétního systému, tj. data ze snímače stále neodpovídají přípustným normám zapsaným v paměti jednotky, potom řídicí jednotka detekuje chybu.

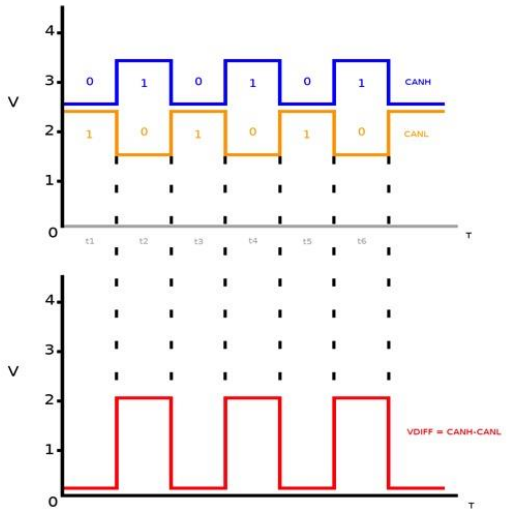
1.2 Sběrnice

Důležitou podmínkou pro správnou funkčnost systému je použití robustních sběrnic, které propojují senzory a řídicí jednotky, řídicí jednotky mezi sebou a řídicí jednotky s akčními členy. Na sběrnice, které mohou být použité v automotive, jsou kladené velké nároky: odolnost vůči rušení, teplotě, vlhkosti, vibracím, mechanická pevnost, rychlý přenos signálu, možnost přenosu velkého počtu signálů.

Všechny řídicí jednotky jsou zařazené do určitých sítí podle funkční domény. Kdyby všechny jednotky byly připojené do jedné sítě, počet požadavků o komunikaci v jeden okamžik by byl hodně velký, což by zvýšilo latenci zpracování jednotlivých požadavků. Pro řídicí jednotky motoru, převodovky a bezpečnostních systému taková situace nesmí nastat, mohlo by to vést k nehodě. Proto se ve vozidle používá několik druhů sběrnic, které tvoří nezávislé sítě. Nejpoužívanějšími druhy sběrnic jsou CAN, LIN, MOST [1].

Nejpoužívanější sběrnicí je CAN. Jako přenosové médium je použita kroucená dvoulinka. Vedení musí být schopno přenášet dva logické stavy: dominantní – logická „0“ a recesivní – logická „1“. Jeden vodič přenáší signál ve stejné podobě, v jaké byl odeslán, druhý vodič přenáší signál komplementární k původnímu. Potom se tyto dva signály zpracovávají. Díky tomu, že vodiče jsou zkroucené, se indukované rušení navzájem odečte, což dělá sběrnici CAN velmi odolnou(Obr. 2). Maximální délka sběrnice může být 10 km, a to s přenosovou rychlostí 5 kbit/s, což je pro použití v automotive málo.

Nejčastěji jsou použity rychlosti 1 Mbit/s (maximální délka 40 m) nebo 500 kbit/s (maximální délka 130 m) [1].



Obr. 2 CAN signály

Další sběrnicí je LIN – Local Interconnect Network. Tato sběrnice se používá pro propojení jednotek řízení klimatizace, oken, zámků, dveří. Na řízení těchto systémů nejsou kladeny tak vysoké požadavky ohledně bezpečnosti a rychlosti přenosu, tato sběrnice je levnou alternativou CAN. Maximální rychlost přenosu je 20 kbit/s [1].

V automobilech se běžně používá také sběrnice MOST – Media Oriented System Transport. Je určená k přenosu dat mezi řídící jednotky audio, video a telematiky. Přenosovým médiem je optické vlákno, což řeší dva problémy: odolnost vůči rušení (optický signál není ovlivněn šumem) a rychlost (multimediální přenos musí mít rychlost kolem 25 Mbit/s) [1].

1.3 Funkční domény

Podle ITEA EAST-EET projektu lze elektroniku ve vozidle rozdělit na funkční domény. Existuje pět domén: pohonné jednotky, podvozek, karoserie, HMI a telematika [25].

Doména pohonných jednotek zahrnuje systémy, které jsou odpovědné za pohon vozidla: řídicí jednotky motoru, převodovky [25]. Řídicí jednotka motoru (ECU) je klíčovým systémem ve vozidle. Provádí neustálou výměnu dat s řídicími moduly jiných systémů a s velkým množstvím různých senzorů. ECU umožňuje dosáhnout významné optimalizace mnoha procesů: nejlepší míra spotřeby paliva, parametry vstřikování paliva

a přívodu vzduchu na vstupy jsou dynamicky upravovány, zvýšení výkonu motoru, úplné spalování směsi paliva a vzduchu ve válcích, snížená toxicita výfukových plynů atd. Mezi základní funkce řídicí jednotky motoru patří: ovládání zapalování, analýza polohy škrticí klapky, kontrola a řízení procesů vstřikování paliva, ovládání systému variabilního časování ventilů, řízení teploty motoru a chladicího systému motoru, řízení systému recirkulace výfukových plynů [8] [11].

Řídicí jednotka převodovky řídí převody na základě vstupů z různých senzorů a také podle údajů poskytovaných řídicí jednotkou motoru. Zpracovává tyto vstupy, rozhoduje jak a kdy řadit rychlostní stupně v převodovce. Dále generuje signály, které pohánějí akční členy, aby dosáhly řazení. Software v TCU je navržen tak, aby optimalizoval výkon vozidla, kvalitu řazení a účinnost paliva. Elektronické senzory sledují polohy voliče, rychlost vozidla, polohu škrticí klapky a řadu dalších parametrů. Na základě těchto informací řídicí modul upravuje proud dodávaný solenoidům v převodovce, které řídí polohu různých ventilů a převodů. Tato diplomová práce se zabývá softwarovou simulací řízení těchto ventilů [12].

Obr. 3 Řídicí jednotka převodovky [37]

Doména podvozku ovládá kola, jejich polohu a pohyb [25]. Jsou to systémy řízení a brzdění, jako například ABS (anti-lock braking system). ABS patří k prostředkům aktivní bezpečnosti, to znamená, že zajišťuje, aby nedošlo k nehodě. Tento systém zabraňuje zablokování kol, a to i při panickém brzdění. U vozidel bez ABS může panické brzdění vést k zablokování kol, dojde-li tak ke smyku, řidič nebude schopen auto ovládat. Senzory rychlosti kol detekují, zda má kolo tendenci k zablokování. V případě, že taková tendence

je, řídicí jednotka snižuje brzdný tlak na jednotlivá kola, totiž uvolní brzdný tlak těsně před blokováním kol [2].

Podle definice EAST-EEA zahrnuje doména karoserie subjekty, které nepatří k dynamice vozidla, tedy takové, které podporují uživatele automobilu, například: airbag, ovládání stěračů, osvětlení, oken, dveří, klimatizace, vybavení sedadel atd. [25]. Airbag je další důležitou součástí moderního vozidla. Je to prostředek pasivní bezpečnosti, to znamená, že pokud dojde k nehodě, musí snížit škodu této nehody. Cílem airbagu je ztlumit sílu nárazu člověka ve vozidle při nehodě a zabránit nárazu do předmětu v autě. Princip funkčnosti airbagu spočívá v tom, že řídicí jednotka airbagu zpracovává data ze senzorů zrychlení a některých dalších senzorů, a když je překročena určitá prahová hodnota zrychlení, tedy došlo k nárazu, spustí zapalování plynového generátoru a rychle nafoukne látkový vak. Když se člověk ve vozidle srazí s vakem, zmáčkne jej a plyn začne unikat malými odvzdušňovacími otvory [2].

Doména HMI zahrnuje zařízení umožňující výměnu informací mezi elektronickými systémy a řidiči: displeje a tlačítka. Tyto systémy jsou odpovědné za příjem požadavků na multimediální zařízení a zobrazení výsledků těchto požadavků. Tyto systémy také slouží k zobrazení informací o stavu vozidla: rychlost, otáčky, stav oleje, stav dveří, světel atd. [25].

Systémy, které patří telematické doméně, zajišťují výměnu informací mezi vozidlem a vnějším světem: rádio, navigační systém, přístup na internet, platby [25].

2 Mechatronika automatické převodovky

Pod pojmem automatická převodovka se nejčastěji rozumí převodovka s hydrodynamickým měničem. Výhoda použití automatické převodovky je ve větším pohodlí řízení auta, řidič nemusí používat řadicí páku a nemusí mačkat spojku, aby přepnul rychlostní stupeň. Změna rychlostního stupně se uskutečňuje automaticky na základě polohy voliče, zatížení motoru, otáček hřídele, polohy pedálu plynu a dat z různých senzorů. Ve výsledku se přepínání rychlostního stupně uskutečňuje plynule, prvky převodovky a podvozku jsou chráněny před velkým zatížením. Existuje druh převodovky, který má možnost automatického i manuálního řazení [9].

Automatická převodovka má i určité nevýhody. Z mechanického hlediska je to složitá a drahá konstrukce v porovnání s manuální převodovkou. Údržba a oprava jsou dražší a složitější. Vozidlo s tímto typem převodovky navíc spotřebovává více paliva, automatická převodovka předává menší točivý moment na kola, účinnost je tedy poněkud snížena. Nové modely však tento problém už téměř vyřešily. Další nevýhodou je, že řízení vozidla s tímto typem převodovky má určitá omezení. Automatická převodovka např. musí být před jízdou zahřátá, je vhodné se vyhnout prudkým rozjezdům a brzdění [20].

Automatická převodovka se ovládá pomocí voliče, který může být zařazen do následujících režimů [9]:

- **Režim P:** parkovací poloha, zařazení parkovací západky.
- **Režim R:** jízda vzad.
- **Režim N**: neutrál.
- **Režim D:** jízda vpřed s automatickým řazením.

Mohou existovat i jiné režimy. Například režim L2 znamená, že při jízdě vpřed může být zařazen pouze první a druhý rychlostní stupeň. Režim L1 znamená, že může být zařazen pouze první rychlostní stupeň. Režim S je sportovní. Existují však i různé zimní režimy atd. [9].

Automatická převodovka se skládá z následujících základních prvků:

- **Hydrodynamický měnič:** zabezpečuje změnu a přenos točivého momentu z motoru na převodovku.
- **Planetové soukolí:** zabezpečuje změnu převodového poměru.
- **Brzdové pásy a spojky:** zabezpečují blokování různých soukolí, umožňují plynulé a včasné řazení.
- **Hydraulický rozvaděč:** ovládací část převodovky, určuje, který převodový poměr musí být zařazen v daný okamžik. Tato jednotka obsahuje olejovou vanu, zubové čerpadlo a skříň s ventily. Simulací řízení těchto ventilů se zabývá daná diplomová práce [9].

2.1 Konstrukce a princip činnosti hydraulického měniče točivého momentu

Hydrodynamický měnič je určen ke změně a přenosu točivého momentu z motoru na převodovku. Díky činnosti měniče jsou potlačeny vibrace. Je součástí automatické převodovky, ale má samostatné pouzdro vyrobené z odolného materiálu naplněného pracovní kapalinou [21].

Měnič točivého momentu se skládá z následujících prvků – viz Obr. 4 [21]:

- **čerpadlové kolo:** vstupní prvek poháněný přímo od motoru;
- **turbínové kolo:** výstupní prvek hydraulicky poháněný čerpadlovým kolem;
- **reaktorové kolo:** reakční prvek, násobič momentu;
- **blokovací spojka:** mechanicky spojuje čerpadlo s turbínou.

Obr. 4 Hydrodynamický měnič [14]

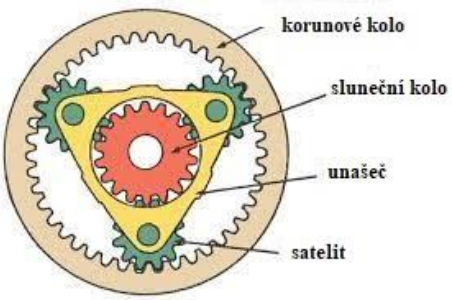
Čerpadlové kolo je připojeno ke klikové hřídeli motoru. Turbínové kolo je připojeno bezprostředně k převodovce. Mezi turbínou a čerpadlovým kolem je reaktorové kolo. Každé z kol má lopatky, které se liší tvarem. Mezi lopatkami jsou kanály, kterými prochází převodová kapalina [13] [21].

Turbínové kolo uvádí do pohybu kapalinu uvnitř měniče. Pokud jsou otáčky hřídele motoru nízké, kapalina zasahuje reaktorové kolo a uvádí ho do pohybu. Lopatky reaktorového kola přesměrují kapalinu zpět na turbínové kolo, díky čemuž se zvyšuje účinnost mechanizmu. Část kapaliny, která se nedostala na turbínové kolo, se přes lopatky reaktorového kola vrací na čerpadlové kolo, díky tomu se zvyšuje točivý moment. Z toho plyne, že maximální točivý moment je dosažen při nejnižších otáčkách hřídele motoru [13] [21].

2.2 Konstrukce a princip činnosti planetového soukolí

Planetové soukolí se skládá z následujících prvků – viz Obr. 5 [30]:

- sluneční kolo;
- satelity;
- korunové kolo;
- unašeč.

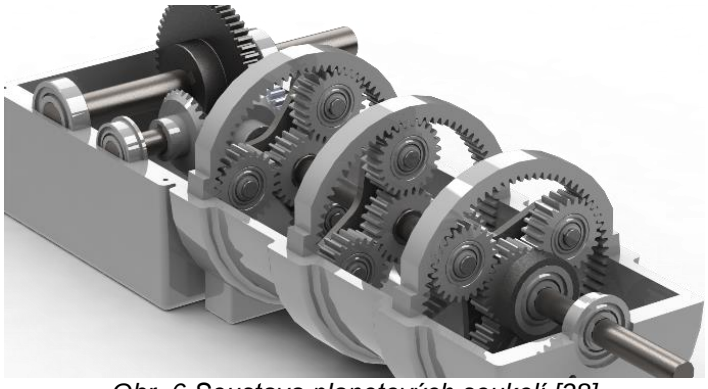


Obr. 5 Planetové soukolí [30]

Změna točivého momentu se dosáhne pomocí blokace jednoho nebo dvou prvků planetového soukolí: centrálního kola, korunového kola nebo unašeče. Satelity nemohou být zablokované. Pokud je zablokováno korunové kolo, dojde ke zvýšení převodového poměru. Pokud je zablokováno centrální kolo, převodový poměr se sníží. Zablokovaný unašeč znamená, že došlo ke změně směru otáčení [30].

Blokování prvků soukolí se uskutečňuje pomocí brzdových pásů a spojek. Spojky spojují mezi sebou jednotlivé části soukolí, brzdový pás blokuje prvky pomocí jejich připevnění k tělu převodovky. Blokování a odblokování těchto prvků se řídí prostřednictvím hydraulických válců. Tyto hydraulické válce jsou ovládány elektronicky ze speciálního modulu – hydraulického rozvaděče [30].

Pro dosažení většího počtu převodových stupňů se v převodovce spojuje několik planetových soukolí (Obr. 6). Korunové kolo je připojené k centrálnímu kolu následujícího soukolí. Jednoduché automatické převodovky mají čtyři stupně, zatímco moderní řešení mohou mít šest, sedm, osm, nebo dokonce devět stupňů [28].



Obr. 6 Soustava planetových soukolí [28]

2.3 Hydraulický rozvaděč

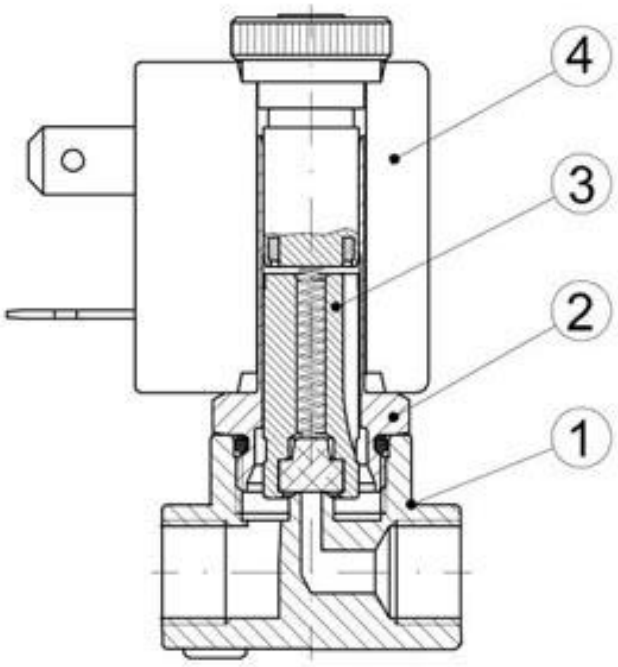
Změna převodových stupňů se uskutečňuje pomocí zapnutí a vypnutí spojek a brzdových pásů. Řízení těchto prvků je realizováno prostřednictvím softwaru, který běží v řídicí jednotce převodovky – TCU. Snímače připojené k TCU sledují polohu pedálu plynu, polohu voliče, teplotu převodové kapaliny atd. TCU dostává data i od jiných řídicích jednotek, například od motoru dostává informaci o jeho zatížení a další data. TCU zpracovává přijaté signály a na základě těchto dat odesílá příkazy akčním členům v hydraulickém rozvaděči (Obr. 7) [12].

Obr. 7 Hydraulický rozvaděč [15]

Princip činnosti hydraulického rozvaděče je následující: na základě elektrických signálů z TCU přes určité kanály probíhá rozdělení tlaku převodové kapaliny, která teče z čerpadla do určitého spojkového bubnu. Podle toho, do kterého bubnu se kapalina dostala a s jakým tlakem, se určí stav spojek a brzdových pasů, čímž se zařadí určitý převodový stupeň [12].

Regulaci tlaku zajišťují ventily. Ventily otevírají a uzavírají kanály v hydraulickém bloku, kterými teče převodová kapalina směrem k mechanické části převodovky. Činnost ventilů je řízena řídicí jednotkou převodovky. Řídicí jednotka určuje proud, který teče do solenoidu, čímž otevírá/zavírá ventil [27].

Existuje hodně druhů ventilů, které se liší konstrukcí a určením. Na Obr. 8 je znázorněna obecná konstrukce, která vysvětluje základní princip činnosti [27].



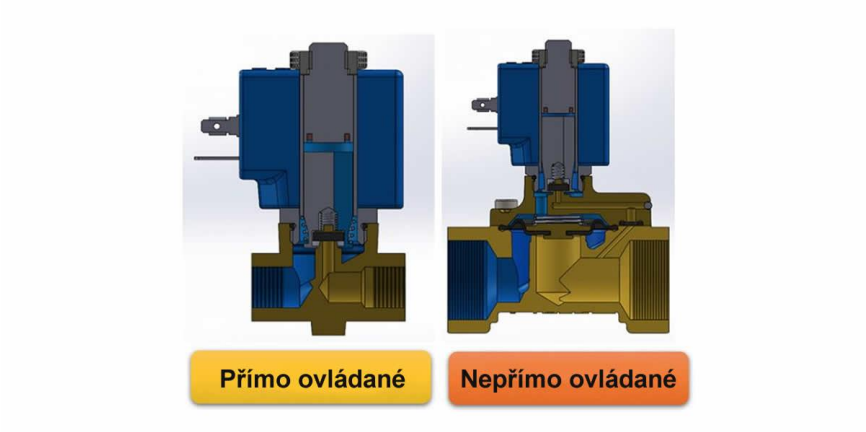
Obr. 8 Konstrukce ventilu [27]

1. Tělo ventilu, které se skládá ze vstupu, výstupu a sedla.
2. Trubka s jádrem, na kterém je namontována cívka.
3. Píst, který se zasouvá do trubky, v některých případech slouží jako těsnění.
4. Elektromagnetická cívka, která vytváří magnetické pole potřebné k pohybu pístu.

Ventil je vybaven solenoidem, což je cívka s pohyblivým feromagnetickým jádrem uprostřed. Toto jádro se nazývá píst. V klidové poloze píst uzavírá otvor. Elektrický proud procházející cívkou vytváří magnetické pole. Magnetické pole vyvíjí sílu na píst, což způsobuje, že se píst táhne směrem ke středu cívky, takže se otvor otevírá. Takovým způsobem funguje ventil, kterému se říká normálně zavřený, to znamená, že když na něj nepůsobí žádné magnetické pole, je zavřený, ale pod vlivem magnetického pole je otevřený. Také existují ventily normálně otevřené, totiž bez vlivu magnetického pole jsou otevřené, ale jakmile na ně začne magnetické pole působit, uzavírají se [27].

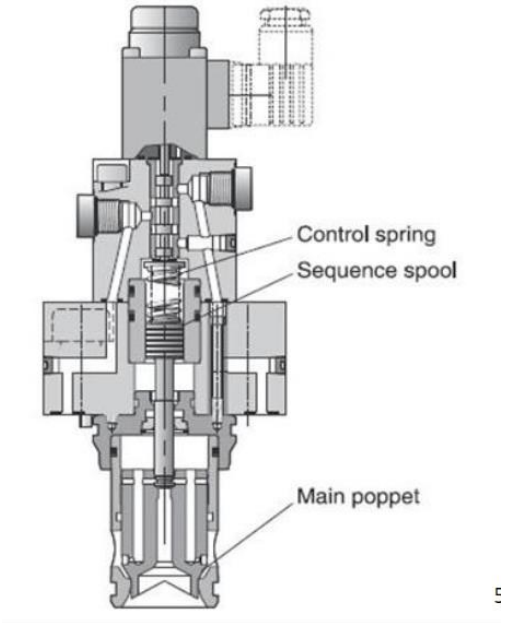
Ventily lze rozdělit na přímo ovládané a nepřímo ovládané (Obr. 9). Princip, který se používá u ventilu přímo ovládaného, je vhodný pro otvory přibližně do 9 mm, pro větší otvory tento princip nemůže zajistit dostatečné těsnění. Proto existují nepřímo ovládané ventily. Průměr otvoru těchto ventilů je od 12 mm do 80 mm. Dobré těsnění je zajištěno

použitím membrány. Nepřímé ovládání znamená, že tlak na membráně, který zajišťuje těsnění ventilu, vytváří diferenční tlak, proto se membrána posune nahoru. Tím se ventil otevře [3].



Obr. 9 Druhy ventilů [3]

ON/OFF ventily, jak plyne z názvu, mohou mít jenom dva stavy: otevřeno a zavřeno. Konstrukce tohoto druhu ventilu je jednoduchá, ale jejich funkcionalita není dostatečná pro složitější aplikace. Existují proporcionální ventily, míra jejich otevření se řídí PWM signálem. Změnou elektrického proudu na vstupu proporcionálního ventilu je průtok tekutiny ventilem plynule regulován v rozsahu od 0 % do 100 % maximálního jmenovitého průtoku. V proporcionálních ventilech je magnetické pole přímo úměrné proudu protékajícímu cívkou. Hlavní odlišností proporcionálního ventilu je jiný tvar elektromagnetu – viz Obr. 10 [26].



Obr. 10 Konstrukce proporcionálního ventilu [26]

3 Vývoj softwaru v automobilovém průmyslu. V-model

Vývoj softwaru v automobilovém průmyslu je komplikovaným procesem. Systém je rozsáhlý a složitý, proto je náročné splnit požadavky na kvalitu, splnit očekávání zákazníků a dodržet časový úsek určený pro vykonání projektu.

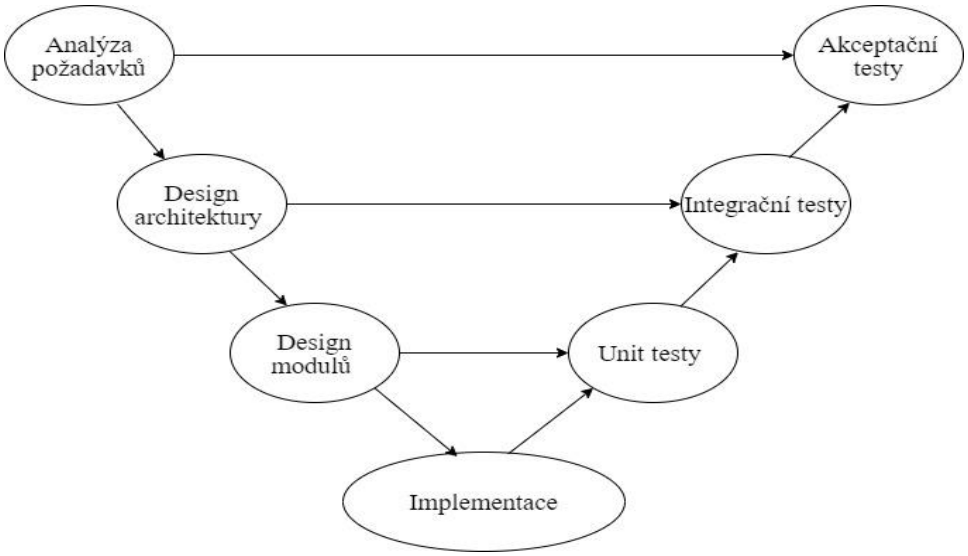
Proces používaný softwarovým průmyslem k navrhování, vývoji a testování vysoce kvalitního softwaru se nazývá životní cyklus vývoje softwaru. Skládá se z podrobného plánu popisujícího, jak vyvíjet, udržovat a vylepšovat konkrétní software. Životní cyklus definuje metodiku zlepšování kvality softwaru a celkového procesu vývoje [34].

Existují různé modely životního cyklu vývoje softwaru. Každý tento model obsahuje řadu kroků specifických pro daný model. Nejpoužívanějšími modely jsou [34]:

- Model vodopádu;
- Iterativní model;
- Spirálový model;
- V-model;
- Model velkého třesku.

V dané diplomové práci je podrobně popsán V-model, protože právě tento model byl použit při vývoji simulace řízení kanálů ventilů.

Jako V-model se nazývá proto, že proces vývoje se dá znázornit ve tvaru písmena „V“ (Obr. 11). Také se tomuto modelu říká „model verifikace a validace“. Z levé strany písmena „V“ jsou zobrazené fáze verifikace, z druhé pak fáze validace. Tyto dvě větve jsou spojené fází kódování. Hlavní vlastností V-modelu je to, že během vývojové fáze se paralelně vypracuje plán pro příslušnou testovací fázi [34].



Obr. 11 V-model

3.1 Analýza požadavků

Toto je první fáze vývojového cyklu, kdy zákazník musí určit požadavky na produkt. Tato fáze zahrnuje komunikaci se zákazníkem, musí se docílit přesného pochopení toho, co zákazník od výsledného produktu očekává. Jedná se o velmi důležitou činnost a je třeba ji dobře řídit, protože většina zákazníků si není jistá, co přesně potřebují. Všechny požadavky musí být napsané srozumitelně a jednoznačně, musí být ověřeno, zda je možné tyto požadavky implementovat. V této fázi se provádí plánování návrhu akceptačního testu [33].

Požadavky lze rozdělit do dvou kategorií: funkční a nefunkční. Funkční požadavky popisují, jak se systém musí chovat. Tyto požadavky obvykle popisují nějakou akci: „Když uživatel udělá x, systém udělá y.“ Většina produktů a aplikací určených k provádění užitečné práce obsahuje mnoho funkčních požadavků na software. Při definování funkčních požadavků by se měl najít zlatý střed mezi příliš podrobným popisem požadavku a obecným a nejednoznačným popisem. Například není zpravidla potřeba mít obecný funkční požadavek, jako je tento: „Když stisknete toto tlačítko, systém se zapne a funguje.“ Oproti tomu požadavek, který se skládá z několika stránek textu, se jeví jako příliš konkrétní. Zkušenosti ukazují, že definice požadavku, který sestává z jedné nebo dvou vět, je nejlepší, jestliže potřebujeme sladit potřebu uživatele s přijatelnou úrovní specifikace pro práci vývojáře [7].

Funkční požadavky popisují, jak by se měl systém chovat, když dostává určité vstupy. To však nestačí k úplnému popisu systémových požadavků. Je třeba vzít v úvahu také následující charakteristiky, které Grady (1992) nazval jako „nefunkční požadavky“: použitelnost, spolehlivost, výkon, podpora. Díky jejich srozumitelné klasifikaci se dá dozvědět více o systému, který je třeba vytvořit [7].

Požadavky musí být rozdělené podle důležitosti funkcionality, o kterou se jedná. Ve větě, která popisuje požadavek, musí být použita jedna z následujících konstrukcí [33]:

- **Must have**: tento druh požadavku musí být implementován v softwaru, jinak zákazník tento produkt nepřijme.
- **Should have**: vylepšení funkčnosti softwaru.
- **Could have:** bez implementace těchto požadavků software může stále správně fungovat, jejich implementace může být posunuta do dalšího vydání projektu.
- **Wish list**: tyto požadavky nejsou nutné ke správné funkčnosti softwaru, ale zákazník by chtěl, aby tato funkcionalita byla implementována.

Připravené požadavky musí být uložené a dostupné pro každého, kdo pracuje na daném projektu. Vždy musí být možnost se vrátit k dřívější verzi požadavku, proto se pro jejich ukládání musí používat verzovací systém. Jedním z programů, který k tomu slouží, je program DOORS. Tento program umožňuje verzování, strukturaci požadavku a traceabilitu mezi požadavky a designem.

3.2 Design

Pro posuzování požadavků zákazníků je vytvořen dokument, který se nazývá „Specifikace softwarových požadavků“, zatímco pro kódování a implementaci je potřeba specifičtějších a podrobnějších popisů z hlediska softwaru. Výstup tohoto procesu lze použít přímo při implementaci v programovacích jazycích [6].

Design lze rozdělit do dvou etap:

- design architektury;
- design modulů.

3.2.1 Design architektury

Design architektury se označuje jako „High Level Design“. Tento design popisuje strukturované a hierarchické uspořádání komponent včetně jejich interakcí a rozhraní. V této fázi je definován přenos dat mezi interními moduly a komunikace s jinými systémy. Moduly jsou považovány za „Black-Box“. S touto informací lze v této fázi navrhnout integrační testy [10].

3.2.2 Design modulů

Ve fázi designu modulů je specifikován podrobný design pro všechny systémové moduly, tento druh designu se označuje jako „Low Level Design“. Design modulů se zabývá implementační částí toho, co je v předchozím designu vnímáno jako systém, a jeho podsystémy. Poskytuje podrobný popis každého modulu, což znamená, že obsahuje skutečnou logiku pro každý modul systému a jeho rozhraní pro komunikaci s ostatními moduly. Je důležité, aby design modulu byl kompatibilní s ostatními moduly v architektuře systému a dalšími externími systémy. Unit testy lze navrhnout v této fázi na základě designu modulů. Unit testy jsou nezbytnou součástí každého procesu vývoje a pomáhají eliminovat velký počet chyb ve velmi rané fázi [10].

3.2.3 UML

Design je grafickým zobrazením projektu, pro vytvoření designu se používá UML (Unified Modeling Language). UML je standardizovaný modelovací jazyk skládající se ze sady diagramů, určený ke specifikaci, vizualizaci, konstrukci a dokumentaci designu softwarových systémů. UML definuje spoustu různých druhů diagramů. Důvodem je to, že na design se musí dívat lidé různého zaměření: programátoři, testeři, kvalitáři, zákazníci, proto je nutné zobrazit systém z různých pohledů. Všichni tito účastníci procesu se zajímají o různé aspekty systému a každý z nich vyžaduje jinou úroveň podrobností [36].

Diagramy můžeme rozdělit do dvou skupin: strukturní a diagramy chování [36].

Strukturní diagramy zobrazují statickou strukturu systému a jeho částí na různých úrovních abstrakce. Prvky ve strukturním diagramu představují smysluplné koncepty systému a mohou zahrnovat abstraktní a reálný svět a koncepty implementace. Existuje sedm typů strukturního diagramu [36]:

- diagram tříd;
- diagram komponent;
- diagram nasazení;
- objektový diagram;
- diagram balíčku;
- složený strukturní diagram;
- diagram profilu.

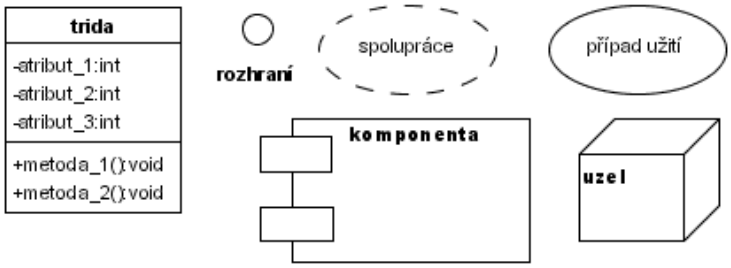
Diagramy chování ukazují dynamické chování objektů v systému, které lze popsat jako sérii změn v systému v průběhu času, existuje sedm typů diagramů chování [36]:

- diagram použití;
- diagram aktivit;
- stavový diagram;
- sekvenční diagram;
- diagram spolupráce;
- diagram interakcí;
- diagram časování.

Každý z těchto diagramů se dá sestavit pomocí určitých, přesně definovaných grafických jednotek: předmětů a relací [36].

Předměty se dělí do několika skupin:

- **předměty struktury**: definují statickou strukturu systému:



Obr. 12 Předměty struktury

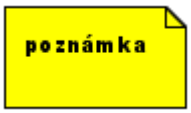
- **předměty chování:** definují dynamické chování objektů v systému:

Obr. 13 Předměty chování

- **předměty seskupení**: seskupují prvky modelu:

Obr. 14 Předměty seskupení

- poznámky:



Obr. 15 Poznámka

Relace ukazují, jak jsou předměty navzájem spojeny:

Obr. 16 Relace

- **závislost**: znázorňuje vztah mezi dvěma předměty, ve kterém změna v jednom prvku ovlivňuje i druhý;
- **asociace**: sada propojení, která spojuje prvky modelu UML;
- **zobecnění**: popisuje vztah dědičnosti ve světě objektů;
- **realizace**: jeden prvek popisuje nějakou funkčnost, druhý ji implementuje, tento druh relace se používá pro spojení rozhraní.

3.3 Implementace

V této fázi probíhá kódování na základě vytvořených požadavků a designu. O nejvhodnějším programovacím jazyce se rozhoduje na základě požadavků. Kódování se provádí na základě pokynů a standardů pro kódování. Kód prochází četnými kontrolami a je optimalizován pro nejlepší výkon [32].

Během psaní kódu je důležité dodržovat určitá pravidla, která se tykají pojmenování souborů, funkcí, proměnných, ukazatelů atd. Jedním z nejdůležitějších pravidel je to, že všechny názvy a komentáře musí být napsány v angličtině. Zdrojové soubory musí být komentovány dostatečně a srozumitelně. To se provádí pomocí blokového komentáře k modulům, třídám a funkcím nebo metodám a také komentářem přímo v kódu. Dalším důležitým pravidlem je použití prefixů, například pro lokální proměnnou se použije prefix loc, pro argument – arg, pro ukazatel – ptr. Toto jednoduché pravidlo dělá kód přehlednějším a pomáhá programátorovi lépe se v něm orientovat. Pro pojmenování funkcí se obvykle používá tak zvaný CamelCaps, to znamená, že několik slov je napsáno bez mezer a první písmeno každého slova je velké. Existují i další pravidla, které dělají kód

přehledný a čitelný. To je zvlášť důležité v automobilovém průmyslu, kde jednotlivé komponenty mohou být rozsáhlé, přičemž jejich špatná čitelnost zkomplikuje následující vývoj. To znamená, že nad provedením určité změny ve špatně čitelné komponentě programátor stráví více času, než kdyby tato komponenta byla napsána přehledně. Z toho vyplývají hodiny práce navíc, které firma musí zaplatit. Ve výsledku kvalita napsaného kódu výrazně ovlivňuje cenu vývoje [32].

Zdrojové soubory jsou výsledkem fáze kódování. Obvykle se zdrojové soubory dělí do několika skupin:

- **Zdrojový kód**: soubor ve formátu .c nebo .cpp v případě použití jazyku C nebo C++. Obsahuje definici parametrů, funkcí nebo metod.
- **Hlavičkový soubor**: soubor ve formátu .h nebo .hpp v případě použití jazyku C nebo C++. Obsahuje deklaraci parametrů, funkcí nebo metod.
- **Definující soubor**: soubor obsahující definice různých parametrů. Díky takovým definicím se zlepšuje čitelnost kódu. V kódu se nesmí objevit žádné číslo, nejlepším řešením je vytvořit makro pro každé číslo. Pokud toto nebylo provedeno, musí se alespoň okomentovat, co dané číslo znamená.
- **Inicializační soubor**: v tomto souboru lze definovat parametry na kterých závisí konfigurace komponenty. Tento soubor se může měnit ve vyšší vrstvě kódu a díky tomu se specifikuje chování komponenty.

Jelikož je automobilový průmysl oblastí s velice vysokými nároky na bezpečnost, musí se provést určitá analýza kódu, která potvrdí, že kód je napsán kvalitně a bezpečně.

Nejdůležitější analýzou je kontrola dodržování MISRA pravidel. MISRA znamená Motor Industry Software Reliability Association, je to spolupráce mezi výrobci vozidel, jejímž cílem je propagovat osvědčené postupy při vývoji elektronických systémů souvisejících s bezpečností a kvalitou softwarově náročných aplikací. Za tímto účelem MISRA vydává dokumenty, které poskytují informace pro inženýry a management, a pořádá akce umožňující výměnu zkušeností mezi odborníky. MISRA poskytuje standardy kódování pro vývoj systémů s vysokými nároky na bezpečnost [24].

Všechna tato pravidla lze seskupit do následujících kategorií:

- **Povinná**: například MISRA C 2012 pravidlo 13.6 říká, že argument funkce „size of“ nesmí obsahovat žádný výraz, který má potenciální vedlejší účinky [18].
- **Vyžadovaná**: odchylky od pravidel jsou povoleny, ale je doporučeno je zdokumentovat. Podle MISRA-C pravidla 15.3 musí konstrukce switch-case vždy mít možnost „default“, podle MISRA C 2012 pravidla 15.7 musí být podmínka „if“ vždy ukončena příkazem „else“. Uvedená pravidla zvyšují spolehlivost kódu, protože přinutí programátora přesně definovat, co se musí stát, pokud není splněna žádná z předchozích podmínek [17] [19].
- **Doporučená**: nejsou povinná. Například MISRA C 2012 pravidlo 15.1 říká, že příkaz „go to“ by se nesměl používat. Je to z toho důvodu, že tato instrukce dělá tok programu nepřehledným, takže program je těžko srozumitelný a těžko upravitelný [16].

Další analýza souvisí se složitostí kódu. Složitost kódu koreluje s mírou vad a robustností aplikace. Složitý kód je obtížné otestovat, ve složitém kódu je s největší pravděpodobností více chyb než v méně složitém kódu. Složitý kód je obtížné udržovat, to zvyšuje náklady. Měření složitosti kódu je nejlepším nástrojem k předpovědi pravděpodobnosti vady. Složitost kódu značně ovlivňuje efektivitu softwarového projektu. Je snadné, aby se kód stal komplikovaným. Někdy se stává tak komplikovaným, že již nemůže podporovat žádné významné vylepšení. V takových situacích je jediným způsobem dalšího vylepšení kódu nejprve snížení složitosti [5].

Zde jsou uvedené některé metriky používané k měření složitosti kódu:

- **Zdrojové řádky kódu**: spočítá počet řádků ve zdrojovém kódu. Jedná se o nejpřímější metriku používanou k měření velikosti programu. Nicméně se nedá určit, jestli je možné implementovat stejnou funkcionalitu ve výrazně menším kódu [5].
- **Cyklomatická složitost**: měří se počet toků v programu. Programy s podmíněnější logikou jsou zřejmě složitější. Určení tohoto parametru se provádí přímým měřením počtu existujících cest v kódu [5].

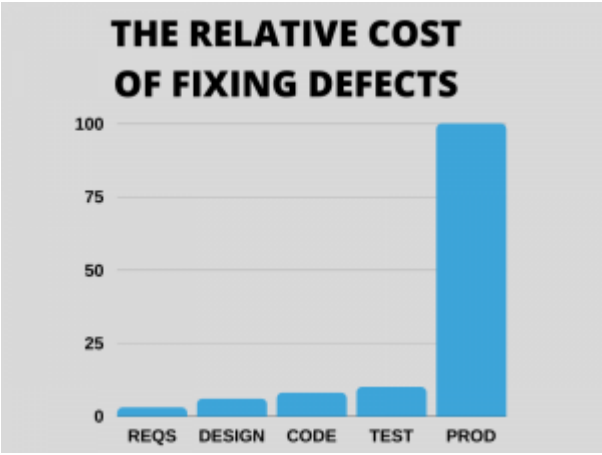
- **Halsteadův objem**: tato metrika říká, kolik informací je ve zdrojovém kódu. Vyhodnotí se počet proměnných, které se používají, a jak často se tyto proměnné v programu používají [5].
- **Index udržovatelnosti**: celkový index udržovatelnosti programu. Na rozdíl od cyklomatické složitosti a Halsteadova objemu je index udržovatelnosti spíše empirickým měřítkem. Je to poměr cyklomatické složitosti a Halsteadova objemu k počtu řádků kódu. Tato analýza poskytuje celkový obraz složitosti softwaru [5].

3.4 Validační fáze: testování

Jakmile je kód napsán a jsou provedeny nutné analýzy, musí se software otestovat. Kvalitní provedení testu je důležitou fázi vývoje, protože je to poslední fáze, kdy je oprava chyby relativně levná (Obr. 17). Když se chyba odhalí v průběhu výrobní fáze, náklady na její opravu jsou podstatně vyšší. Navíc, dodání chybného softwaru zákazníkovi, může způsobit jeho odchod ke konkurenční firmě a špatnou reputaci firmy. V důsledku toho firma přijde i o další zákazníky [5].

Softwarové testy jsou rozděleny do tří kroků:

- **Unit testy**: testování designu modulů a implementace.
- **Integrační testy**: testování rozhraní a funkční interakce mezi moduly, testování designu architektury.
- **Akceptační testy**: testování úplné a správné implementace požadavků.



Obr. 17 Relativní náklady na opravu chyb [5]

3.4.1 Unit testy

Unit testy jsou navržené ve fázi designu modulu, ale provádí se ve fázi validace. Unit testy pomáhají eliminovat určité chyby v rané fázi. Všechny druhy chyb pomocí unit testu nelze odhalit, zvlášť pokud se jedná o programování mikrokontroleru, kde lze chyby v kódu odhalit jenom pomocí testování na hardwaru.

Unit testy píše programátor pro svůj kód. Unit testy musí být rozdělené do bloků (tzv. test casů). Tyto bloky musí pokrýt celý kód. Existuje určitá analýza, která zkontroluje pokrytí kódu, výsledek této analýzy vždy musí být 100 %. Základní instrukcí v unit testech je instrukce „equal“, která porovnává hodnoty prvního a druhého argumentu a v případě, že se rovnají, vrátí stav PASSED. Například unit test na Obr. 18 zkontroluje, zda inicializace proběhla úspěšně.

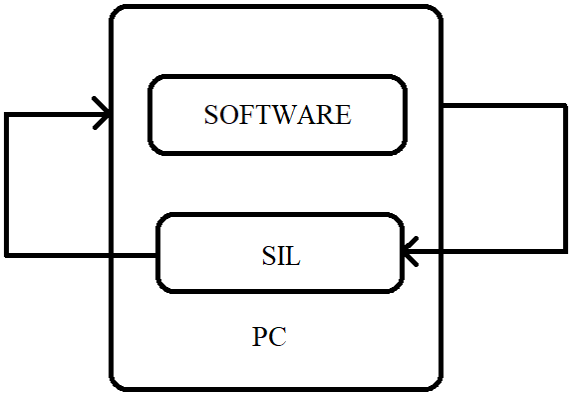
```
1  UNITTEST_BLOCK_BEGIN("Test Init")
2  equal(Init(), STATUS_OK)
3  UNITTEST_BLOCK_END
4
```

Obr. 18 *Příklad unit testu*

3.4.2 Integrační testy

Integrační testy jsou spojené s fází architektonického designu. Cílem integračních testů je ověřit správnost integrace modulů do celkového systému v souladu s designem architektury pomocí testování interakce těchto modulů mezi sebou. To znamená, že se testuje funkční interakce a rozhraní komponent.

Na rozdíl od unit testu musí být integrační a následující testy provedeny testerem, nikoliv programátorem, který tento kód psal. Programátor není schopen objektivně otestovat vlastní kód. Testování se provádí podle principu „Black Box“, to znamená, že tester neví, jak je kód napsán, má jenom informaci o rozhraních a dokumentaci o tom, jak se systém musí chovat. Tento druh testu se primárně provádí v prostředí SIL – software in the loop. SIL zahrnuje matematickou simulaci hardwaru, pomocí SIL můžeme simulovat systém v reálném čase bez fyzického připojení hardwaru. Z Obr. 19 je vidět, že softwarový algoritmus i simulovaný model běží na stejném PC. V tomto testovacím prostředí se ověřuje správnost algoritmů [23].



Obr. 19 *Blokové schéma SIL*

3.4.3 Akceptační testy

Akceptační testy jsou spojené s fází analýzy požadavků a zahrnují testování produktu v uživatelském prostředí. Účelem akceptačních testů je potvrdit, že integrovaný software splňuje stanovené požadavky. V této fázi je již možné ověřit nefunkční požadavky (například nevyhovující zatížení) a výkon ve skutečném uživatelském prostředí. Tento druh testu se provádí v prostředí HIL – hardware in the loop (Obr. 20).

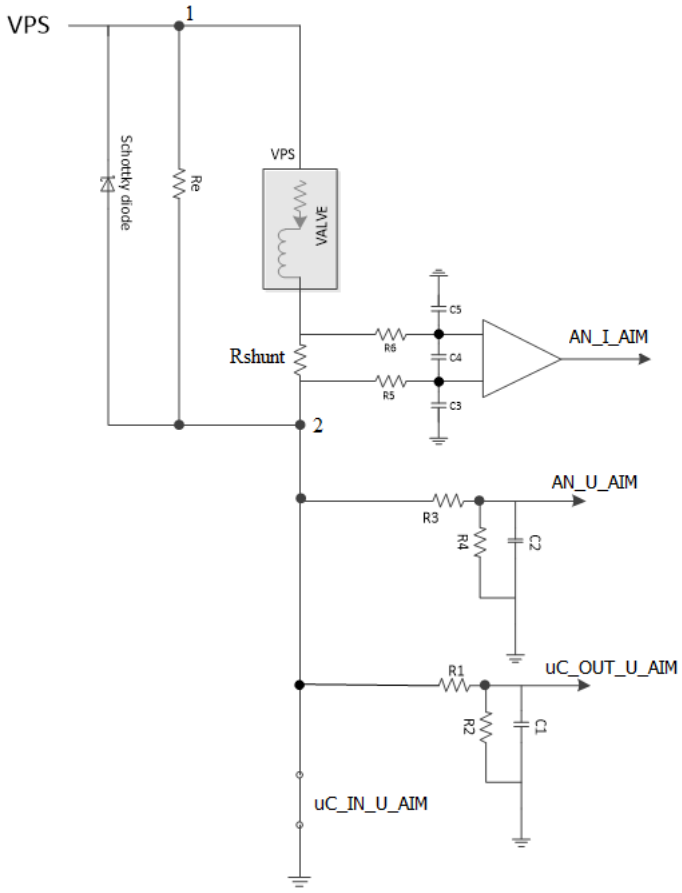
HIL je testovací systém, který obsahuje stejný hardware jako reálné auto. Testovaný software se nahrává do řídicí jednotky. V tomto testovacím systému vstupní signály jsou simulovány, což umožňuje testování softwaru při různých vstupních datech a simulaci elektrických chyb [22].

Obr. 20 HIL [22]

4 Analýza zapojení řízení ventilů

4.1 Popis schématu řízení ventilů

Na Obr. 21 je zobrazeno obecné schéma řízení ventilu, které je převzato z interní dokumentace. Na vstup spínače je zaveden signál uC_IN_U_AIM, je to PWM signál, který určuje, kdy spínač musí být sepnut a kdy rozepnut. Perioda a střída PWM signálu se vypočítávají softwarově na základě požadovaného proudu ventilem. Tento proud určuje míru otevření ventilu. Zapojení je napájeno z baterií VPS.



Obr. 21 Obecné schéma řízení ventilu

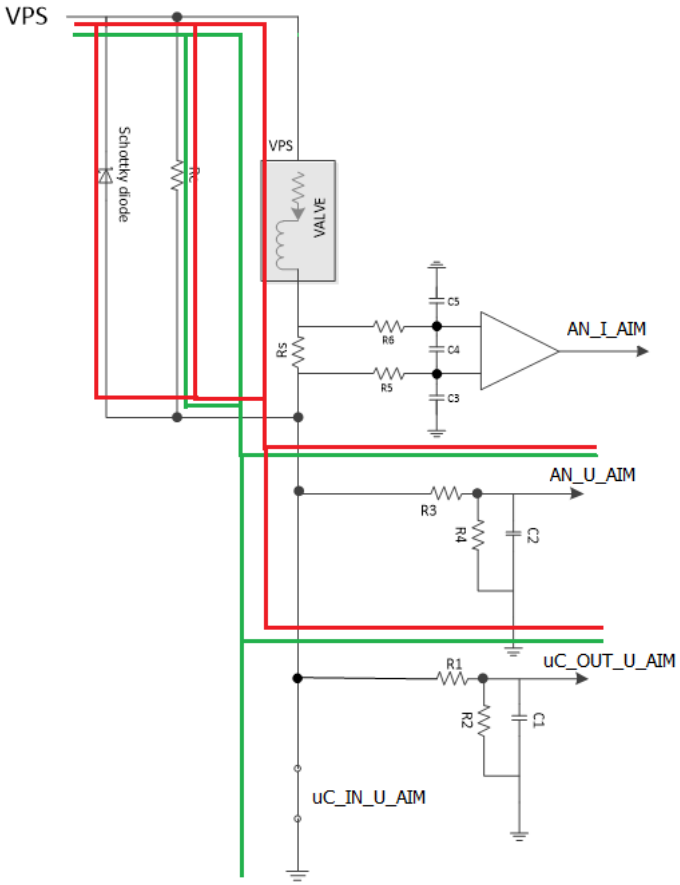
Na schématu jsou vidět tři výstupy, které se používají k diagnostice řízení ventilu:

- **AN_I_AIM**: proud ventilem. Tento proud se měří na rezistoru R_{shunt}. Jelikož proud protékající tímto rezistorem může být hodně malý, jeho hodnota je zesílená.
- **AN_U_AIM**: napětí mezi spínačem a ventilem. Aby diagnostika napětí neovlivnila proud protékající ventilem, a tím pádem i přesnost jeho řízení, napěťový výstup je vyveden, jak je ukázáno na Obr. 21.

- **uC_OUT_U_AIM**: PWM signál, který je kontrolován diagnostikou u proporcionálních ventilů. Tento výstup slouží k odhalení chyb „zkrat na baterii“ a „zkrat mezi ventily“. V bezporuchovém stavu se signál na tomto výstupu musí rovnat vstupnímu signálu uC_IN_U_AIM.

Všechny výstupy jsou opatřeny filtry, které vyhlazují signál.

Na Obr. 22 je zelenou barvou označena cesta protékání proudu se sepnutým spínačem, červenou barvou je označena cesta protékání proudu s rozepnutým spínačem.



Obr. 22 Schéma s označenými cestami protékání proudu

Celkový proud dodávaný do zapojení se rovná poměru VPS k celkovému odporu.

$$I_{source} = \frac{VPS}{R} \tag{1}$$

Celkový odpor se sepnutým spínačem:

$$R_{switchON} = (R_{filterU} \parallel R_{filterPWM} \parallel R_{switch}) + (R_e \parallel (R_{valve} + R_{shunt})), \quad (2)$$

kde R_{switch} = 0,34Ω.

Celkový odpor s rozepnutým spínačem:

$$R_{switchOFF} = (R_{filterU} \parallel R_{filterPWM}) + (R_e \parallel (R_{valve} + R_{shunt})) \quad (3)$$

Ve stavu, když je spínač rozepnut, je celkový odpor mnohem větší, proto do ventilu v tomto případě teče malý proud, který se může považovat za zanedbatelný. Když je spínač sepnut, do ventilu se dodává maximální možný proud, který může protékat ventilem při zapojení s danými parametry součástek. Perioda a střída PWM signálu určují střední hodnotu proudu, který protéká ventilem. Tento proud vytváří elektromagnetické pole, které nastaví feromagnetické jádro ventilu do požadované polohy. Jelikož je to solenoidový ventil, v něm se akumuluje energie, která se vybíjí přes Schottkyho diodu.

4.2 Výpočty výstupních parametrů obvodu

4.2.1 Stav bez chyby

Nejdříve vypočteme celkový odpor větví z uzlu 2 k zemi, pojmenujeme tento odpor $R_{branch\_switchON}$ pro stav se sepnutým spínačem a $R_{branch\_switchOFF}$ pro stav s rozepnutým spínačem. Tyto hodnoty budeme potřebovat i pro výpočty poruchových stavů.

$$R_{branch_{switchON}} = R_{filterU} \parallel R_{filterPWM} \parallel R_{switch}$$
$$R_{branch_{switchOFF}} = R_{filterU} \parallel R_{filterPWM} \quad (4)$$

Dále vypočteme celkový odpor zapojení pro stav se sepnutým a rozepnutým spínačem:

$$R_{switchON} = R_{branch_{switchON}} + (R_e \parallel (R_{valve} + R_{shunt}))$$
$$R_{switchOFF} = R_{branch_{switchOFF}} + (R_e \parallel (R_{valve} + R_{shunt})) \quad (5)$$

Dále vypočteme celkový proud dodaný z baterií:

$$I_{source_{switchON}} = \frac{VPS}{R_{switchON}} \qquad (6)$$
$$I_{source_{switchOFF}} = \frac{VPS}{R_{switchOFF}}$$

Vypočteme napětí mezi uzly 1 a 2:

$$U_{12_{switchON}} = I_{source_{switchON}} \cdot (R \parallel (R_{valve} + R_{shunt})) \qquad (7)$$
$$U_{12_{switchOFF}} = I_{source_{switchOFF}} \cdot (R \parallel (R_{valve} + R_{shunt}))$$

Z této hodnoty zjistíme proud protékající ventilem AN_I_AIM:

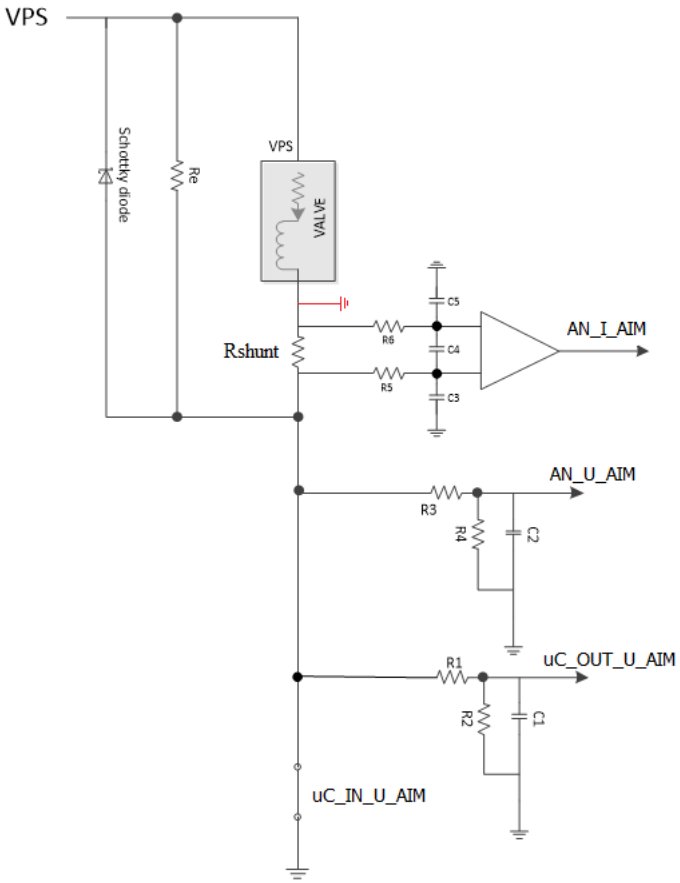
$$I_{valve_{switchON}} = \frac{U_{12_{switchON}}}{R_{valve} + R_{shunt}} \qquad (8)$$
$$I_{valve_{switchOFF}} = \frac{U_{12_{switchOFF}}}{R_{valve} + R_{shunt}}$$

Také z hodnoty U_{12} zjistíme napětí na výstupu AN_U_AIM:

$$U_{valve_{switchON}} = VPS - U_{12_{switchON}} \qquad (9)$$
$$U_{valve_{switchOFF}} = VPS - U_{12_{switchOFF}}$$

4.2.2 Zkrat na zem

Na Obr. 23 je zobrazeno schéma zapojení pro poruchový stav „zkrat na zem“.



Obr. 23 Schéma řízení ventilu pro poruchový stav „zkrat na zem“

Nejdříve vypočteme celkový odpor rezistorů R_{shunt}, R_e a R_{branch_switch} pro stav se sepnutým a rozepnutým spínačem:

$$R_{switchON} = (R_{branch_{switchON}} \parallel R_{shunt}) + R_e$$
$$R_{switchOFF} = (R_{branch_{switchOFF}} \parallel R_{shunt}) + R_e \quad (10)$$

Vypočteme proud protékající těmito rezistory:

$$I_{switchON} = \frac{VPS}{R_{switchON}}$$
$$I_{switchOFF} = \frac{VPS}{R_{switchOFF}} \quad (11)$$

Napětí AN_U_AIM:

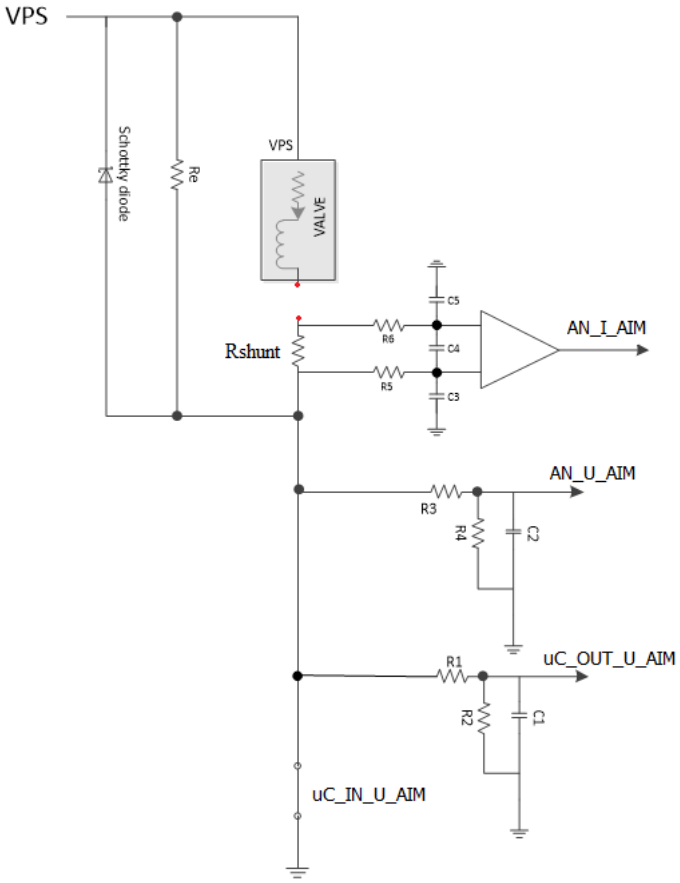
$$U_{valve_{switchON}} = I_{switchON} \cdot (R_{switchON} \parallel R_{shunt})$$
$$U_{valve_{switchOFF}} = I_{switchOFF} \cdot (R_{switchOFF} \parallel R_{shunt}) \quad (12)$$

Jelikož je na rezistoru R_{shunt} stejné napětí jako na filtru signálu AN_U_AIM, proud AN_I_AIM se vypočte následujícím způsobem:

$$I_{valve_{switchON}} = \frac{U_{valve_{switchON}}}{R_{shunt}}$$
$$I_{valve_{switchOFF}} = \frac{U_{valve_{switchOFF}}}{R_{shunt}} \quad (13)$$

4.2.3 Rozpojený obvod

Na Obr. 24 je zobrazeno schéma zapojení pro poruchový stav „rozpojený obvod“.



Obr. 24 Schéma řízení ventilu pro poruchový stav „rozpojený obvod“

Vypočteme celkový odpor zapojení pro stav se sepnutým a rozepnutým spínačem:

$$R_{switchON} = R_{branch_{switchON}} + R_e$$
$$R_{switchOFF} = R_{branch_{switchOFF}} + R_e \quad (14)$$

Dále vypočteme celkový proud dodaný z baterií:

$$I_{source_{switchON}} = \frac{VPS}{R_{switchON}}$$
$$I_{source_{switchOFF}} = \frac{VPS}{R_{switchOFF}} \quad (15)$$

Napětí AN_U_AIM:

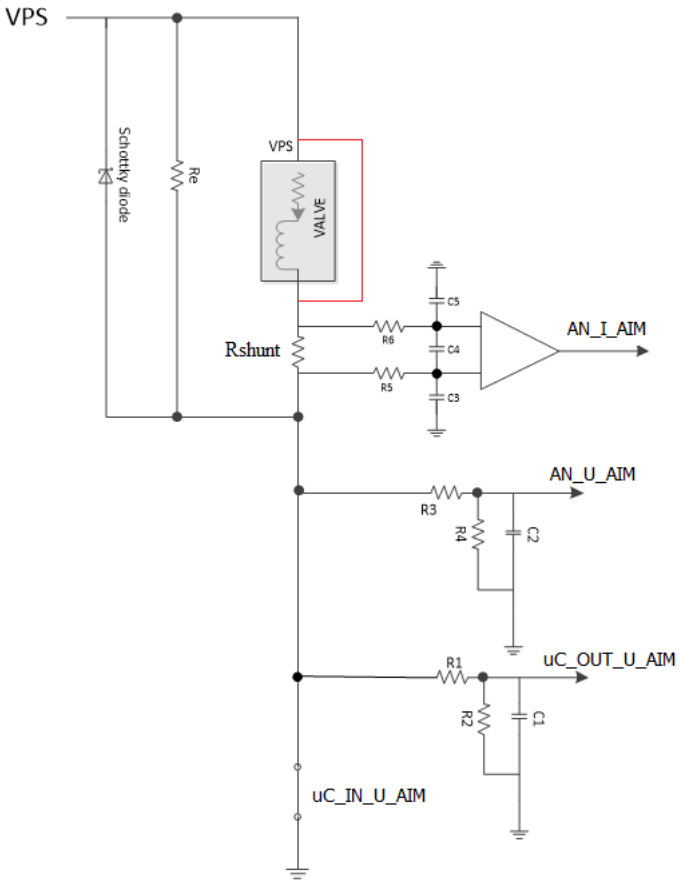
$$U_{valve_{switchON}} = I_{source_{switchON}} \cdot R_{branch_{switchON}}$$
$$U_{valve_{switchOFF}} = I_{source_{switchOFF}} \cdot R_{branch_{switchOFF}} \quad (16)$$

Proud AN_I_AIM se měří na rezistoru R_{shunt}, kterým žádný proud neprotéká:

$$I_{valve_{switchON}} = 0$$
$$I_{valve_{switchOFF}} = 0 \quad (17)$$

4.2.4 Zkrat na baterii

Na Obr. 25 je zobrazeno schéma zapojení pro poruchový stav „zkrat na baterii“.



Obr. 25 Schéma řízení ventilu pro poruchový stav „zkrat na baterii“

Vypočteme celkový odpor zapojení pro stav se sepnutým a rozepnutým spínačem:

$$R_{switchON} = R_{branch_{switchON}} + (R_e \parallel R_{shunt})$$
$$R_{switchOFF} = R_{branch_{switchOFF}} + (R_e \parallel R_{shunt}) \quad (18)$$

Dále vypočteme celkový proud dodaný z baterií:

$$I_{source_{switchON}} = \frac{VPS}{R_{switchON}}$$
$$I_{source_{switchOFF}} = \frac{VPS}{R_{switchOFF}} \quad (19)$$

Napětí AN_U_AIM:

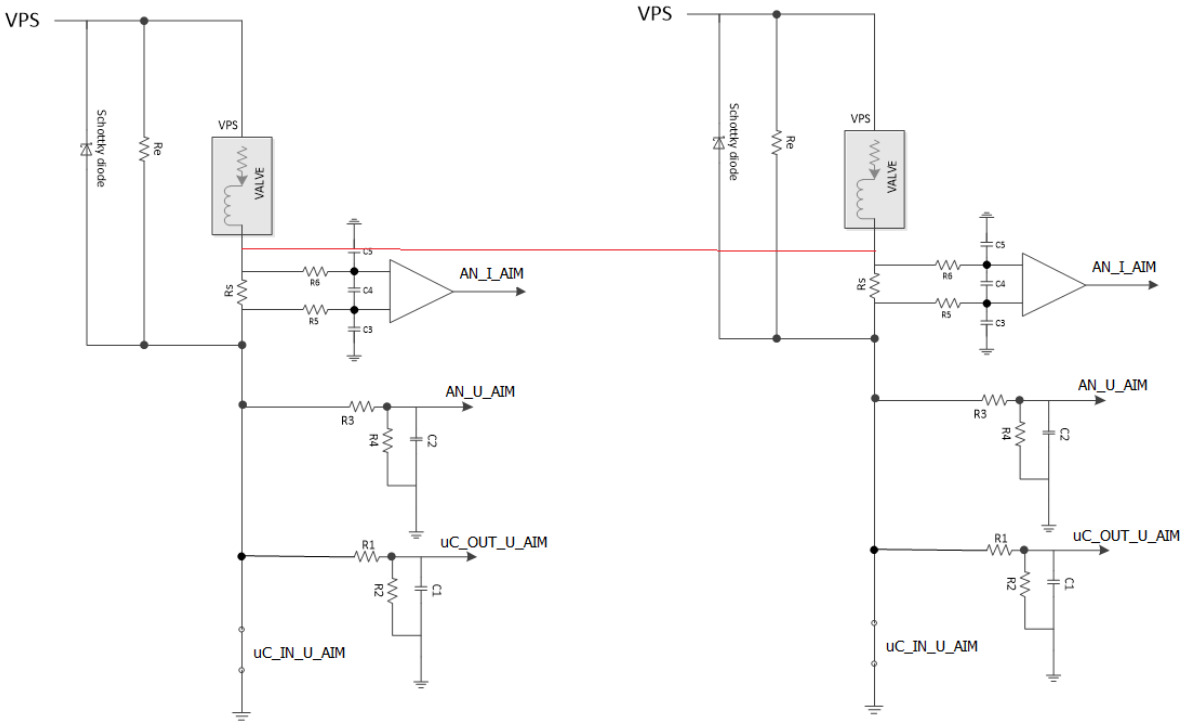
$$U_{valve_{switchON}} = I_{source_{switchON}} \cdot R_{branch_{switchON}}$$
$$U_{valve_{switchOFF}} = I_{source_{switchOFF}} \cdot R_{branch_{switchOFF}} \quad (20)$$

Proud AN_I_AIM:

$$I_{valve_{switchON}} = \frac{VPS - U_{valve_{switchON}}}{R_{shunt}}$$
$$I_{valve_{switchOFF}} = \frac{VPS - U_{valve_{switchOFF}}}{R_{shunt}} \quad (21)$$

4.2.5 Zkrat mezi ventily

Na Obr. 26 je vyobrazeno schéma zapojení pro poruchový stav „zkrat mezi ventily“.



Obr. 26 Schéma řízení ventilu pro poruchový stav „zkrat mezi ventily“

Výpočty pro tento stav jsou obtížné, může nastat zkrat i mezi několika ventily, hodnoty proudu a napětí se budou měnit v závislosti na kombinaci stavů všech spínačů. Pro odhalení této chyby diagnostika nekontroluje hodnoty výstupního proudu a napětí. Detekce této chyby se provádí kontrolou výstupního PWM signálu uC_OUT_U_AIM, proto jsou pro simulaci použity hodnoty proudu a napětí pro stav bez chyby, ale signál uC_OUT_U_AIM je změněn tak, aby se tato chyba odhalila pomocí diagnostiky.

5 Simulace řízení ventilů

V této části je popsána realizace simulace řízení ventilů na základě výpočtu podle V-modelu z kapitoly 4.

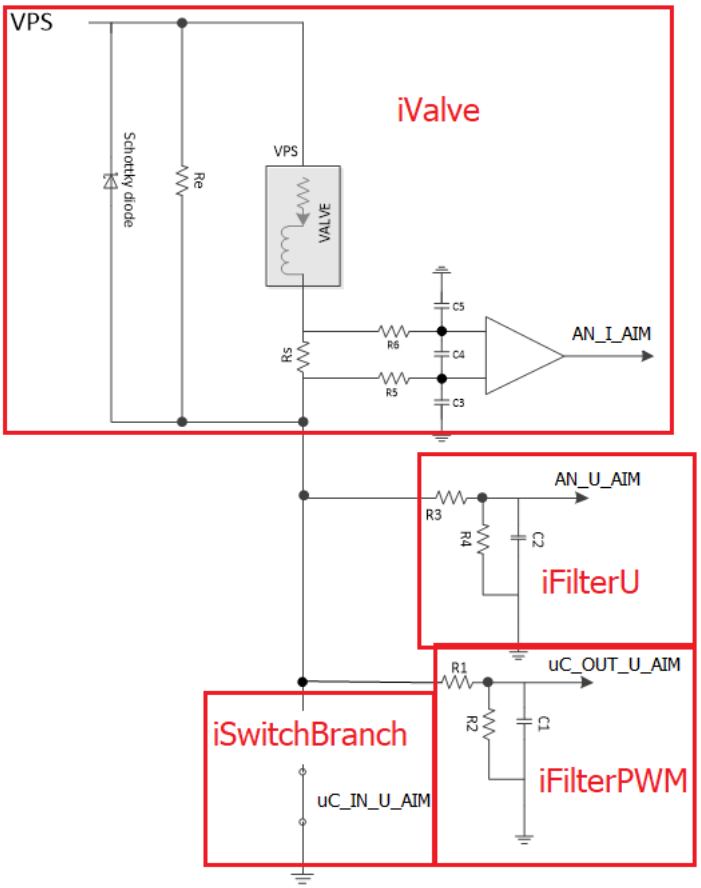
5.1 Požadavky

V daném případě požadavkům odpovídá zadání diplomové práce.

- Systém musí být parametrizovatelný a modulární.
- Simulovaný systém musí mít interface, který by umožnil integraci do celého systému.
- Musí být udělaná simulace jednoho kanálu a pomocí instancování by se měl nasimulovat systém s libovolným počtem kanálů.
- Simulace musí být prováděna cyklicky podle periody hlavní smyčky systému. Tato perioda musí být nastavitelná.
- Musí být umožněna simulace základních chyb: zkrat na zem, zkrat na baterii, zkrat mezi kanály, rozpojený obvod.
- Systém musí být otestován porovnáním s chováním reálné řídicí jednotky.

5.2 Design architektury

Navrhovaný systém musí být parametrizovatelný a modulární. Pro zajištění modularity je nejvhodnější použit objektově orientovaný programovací jazyk. Jelikož je systém, do kterého se bude integrovat simulace, napsán v jazycích C nebo C++, nejvhodnější volbou je jazyk C++. Tento jazyk využívá objektové paradigma, což znamená, že základním celkem je objekt [29].



Obr. 27 Rozdělení schématu na moduly

Na Obr. 27 je vidět rozdělení schématu na moduly. Každý z těchto modulů je realizován jako třída. Třída cValve popisuje tu část zapojení, která obsahuje ventil. Atributy této třídy jsou proměnné obsahující hodnoty součástek, hodnoty proudu a napětí na ventilu pro různé stavy se sepnutým a rozepnutým spínačem a další. Tato třída obsahuje metody pro generování časových značek proudu a napětí. Třída cFilter slouží k výpočtu celkového odporu filtru. Pro každý ventil jsou vytvořené dvě instance: iFilterPWM (pro filtraci PWM signálu) a iFilterU (pro filtraci signálu výstupního napětí). Hodnota celkového odporu filtru je předána instanci třídy cSwitchBranch pro následující výpočty. Třída cSwitchBranch obsahuje metody pro generování časových značek PWM signálu na základě střídy, periody PWM, periody hlavní smyčky, posunu fáze PWM signálu, počtu požadovaných časových značek.

Simulace musí být parametrizovatelná, proto je vytvořen soubor init.hpp, ve kterém jsou definovány parametry, které může měnit uživatel. Těmito parametry jsou střída, perioda hlavní smyčky, perioda PWM, hodnoty součástek zapojení, druh ventilu, počet ventilů.

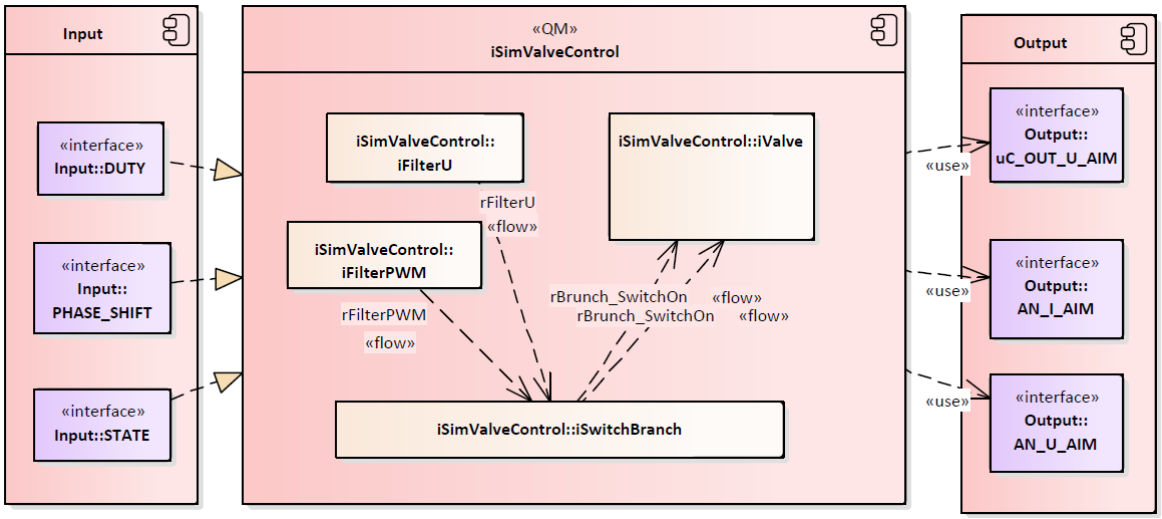
Třída cSimValveControl obsahuje public metody, které tvoří interface simulace. Tato třída obsahuje metodu Init() a CyclicUpdate(). Metoda Init() slouží k inicializaci hodnot a vytvoření instanci tříd, které již byly zmíněné. Metoda CyclicUpdate() musí být volána cyklicky, v ní probíhá aktualizace dat na základě vstupních parametrů. Design je vytvořen v programu Enterprise Architect (Obr. 28).

Vstupy komponenty:

- **DUTY** – střída PWM signálu.
- **PHASE_SHIFT** – posun fáze PWM signálu.
- **STATE** – stav ventilu: bez chyby nebo simulace některé z chyb.

Výstupy komponenty:

- **uC_OUT_U_AIM** – pole časových značek PWM.
- **AN_I_AIM** – výstupní proud.
- **AN_U_AIM** – výstupní napětí.

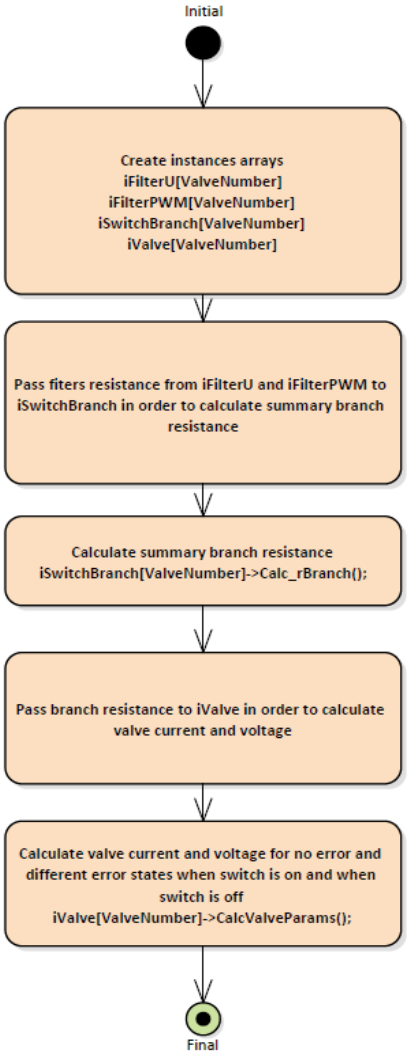


Obr. 28 Design architektury vytvořený v programu Enterprise Architect

5.3 Design modulů

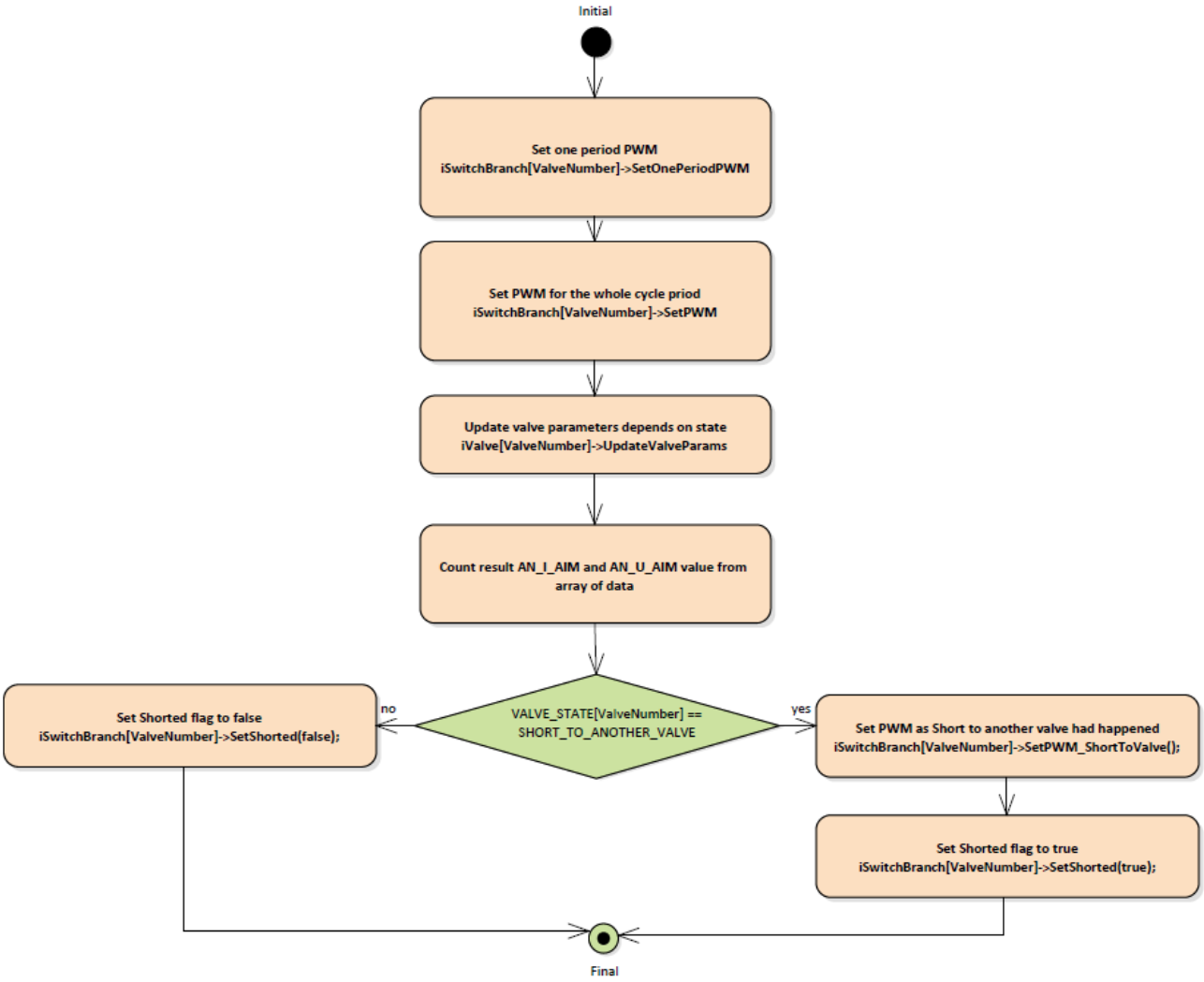
5.3.1 cSimValveControl

Init() metoda je metodou rozhraní a slouží k inicializaci simulace (Obr. 29). Nejdříve se vytvoří pole instancí podle počtu ventilů a v konstruktoru každé instance se provede nastavení hodnot podle druhu ventilu. Dále hodnoty odporů, které se vypočetly v konstruktorech instancí iFilterU a iFilterPWM se předají instanci iSwitchBranch pro následující výpočty. Pro ON/OFF ventil je odpor iFilterPWM nulový, protože tento druh ventilu neobsahuje kontrolu PWM signálu. Potom se pomocí metody iSwitchBranch::Calc_rBranch vypočte celkový odpor filtrů a spínače. Tato hodnota se předá instanci iValve. Potom se pomocí metody iValve::CalcValveParams() provede výpočet hodnot proudu a napětí ve stavu bez chyby a v různých chybových stavech se sepnutým a rozepnutým spínačem.



Obr. 29 Vývojový diagram pro metodu Init()

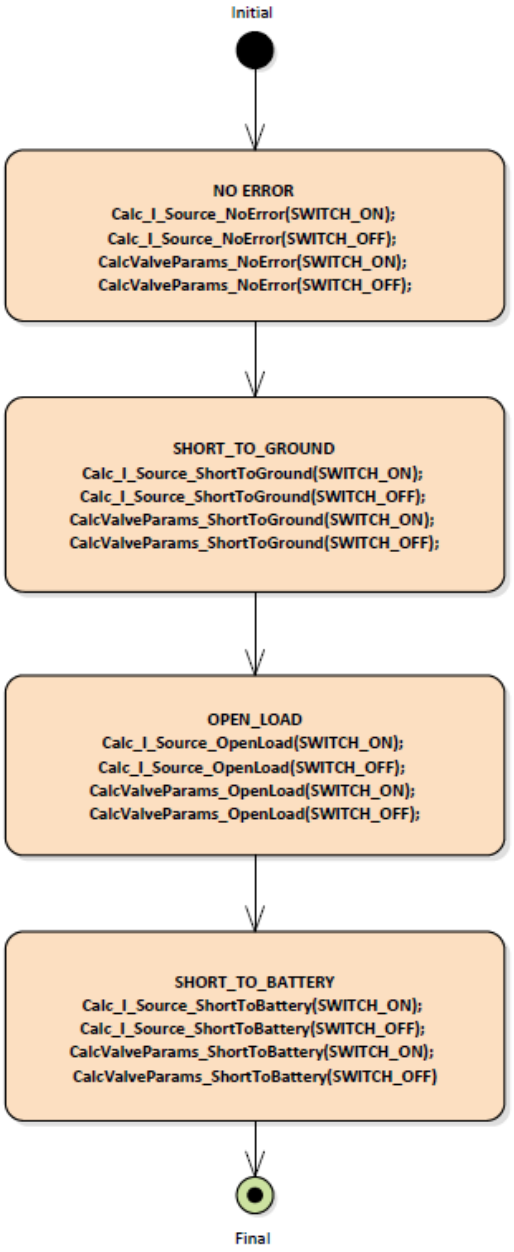
Metoda CyclicUpdate() musí být volána cyklicky (Obr. 30). Nejdříve se nastaví pole PWM hodnot pro jednu periodu PWM, potom se nastaví pole obsahující časové značky PWM hodnot pro celou periodu hlavní smyčky. Dále se přepočtou hodnoty proudu a napětí podle PWM a stavu. Jakmile je nastaven stav „zkrat mezi ventily“, zavolá se metoda iSwitchBranch::SetPWM_ShortToValve a pro daný ventil se nastaví příznak, který říká, že nastala tato chyba.



Obr. 30 *Vývojový diagram pro metodu CyclicUpdate()*

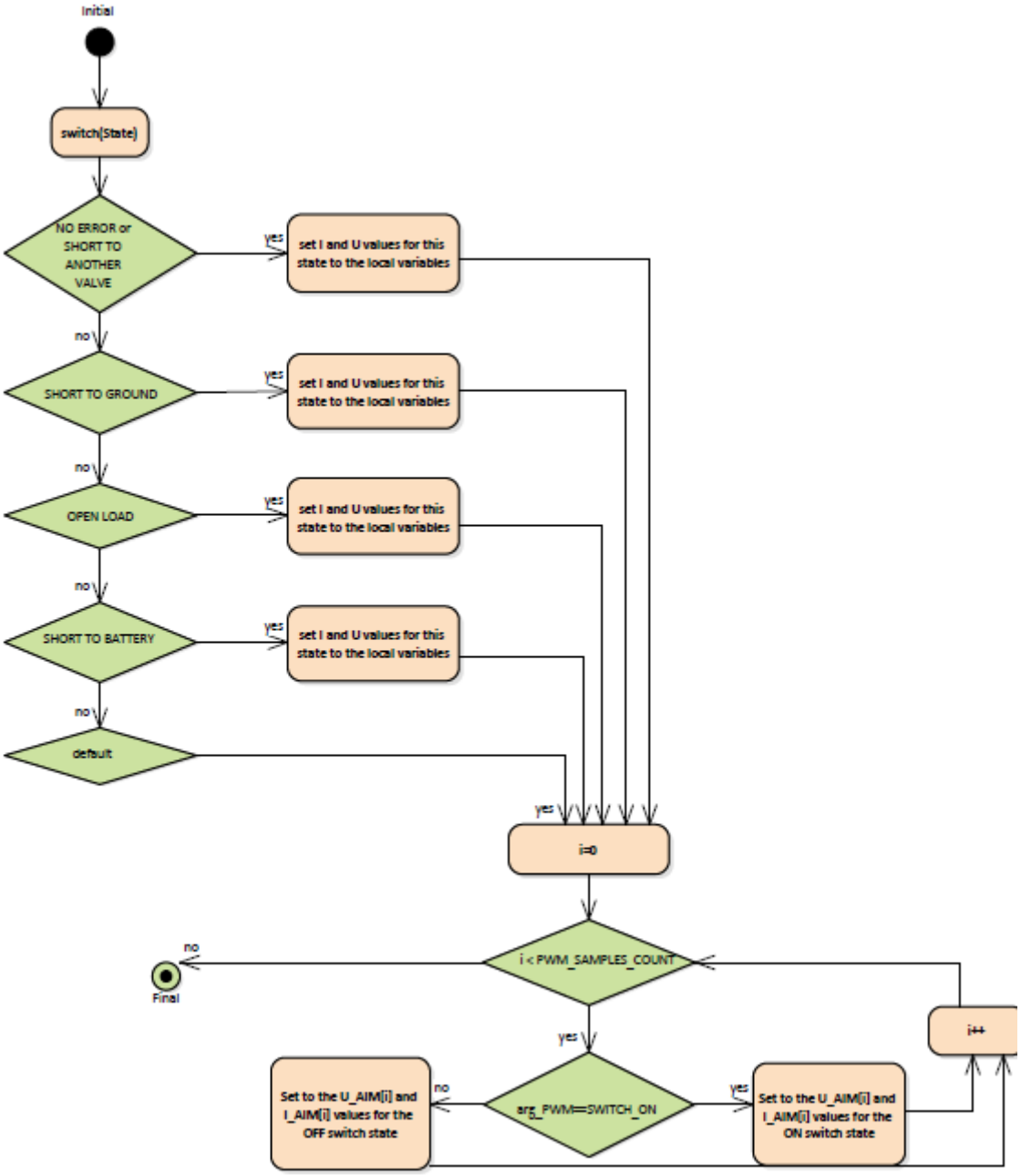
5.3.2 cValve

Metoda CalcValveParams() slouží k výpočtu hodnot proudu a napětí ve stavu bez chyby a v různých chybových stavech se sepnutým a rozepnutým spínačem (Obr. 31). Tato metoda musí být volána v inicializační fázi, jelikož musí připravit hodnoty pro cyklický přepočet parametrů.



Obr. 31 Vývojový diagram pro metodu CalcValveParams()

Metoda UpdateValveParams() slouží k naplnění polí časovými značkami proudu a napětí (Obr. 32). Nejdříve se podle stavu ventilu určí, které hodnoty proudu a napětí se použijí k naplnění polí, a potom se podle časových značek PWM zaplní pole I_AIM a U_AIM. Tato metoda musí být volána cyklicky.



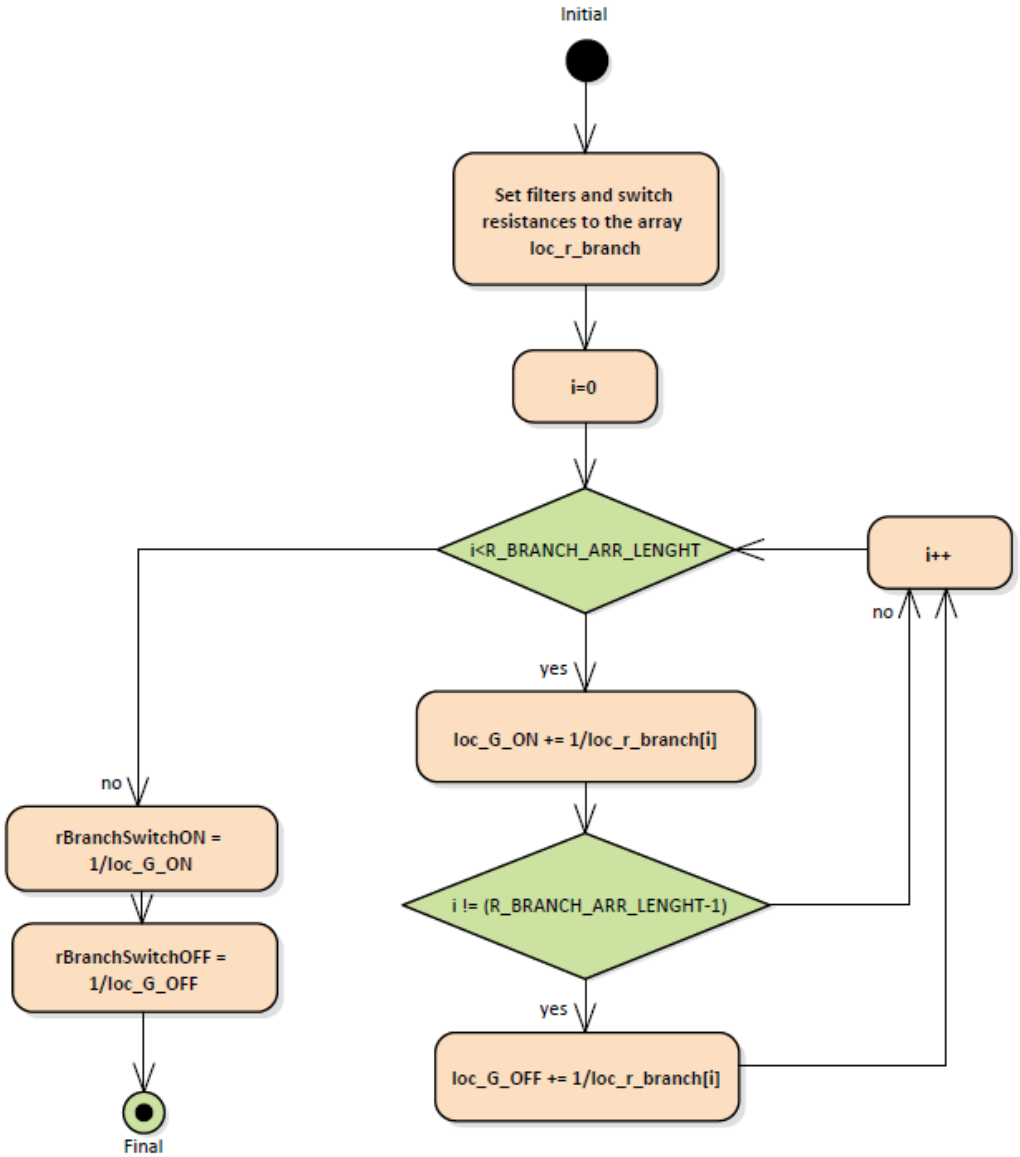
Obr. 32 Vývojový diagram pro metodu UpdateValveParams()

5.3.3 cFilter

Tato třída pouze vypočítává celkový odpor filtru, to se provede v konstruktoru. Proto grafický design pro tuto třídu není zapotřebí.

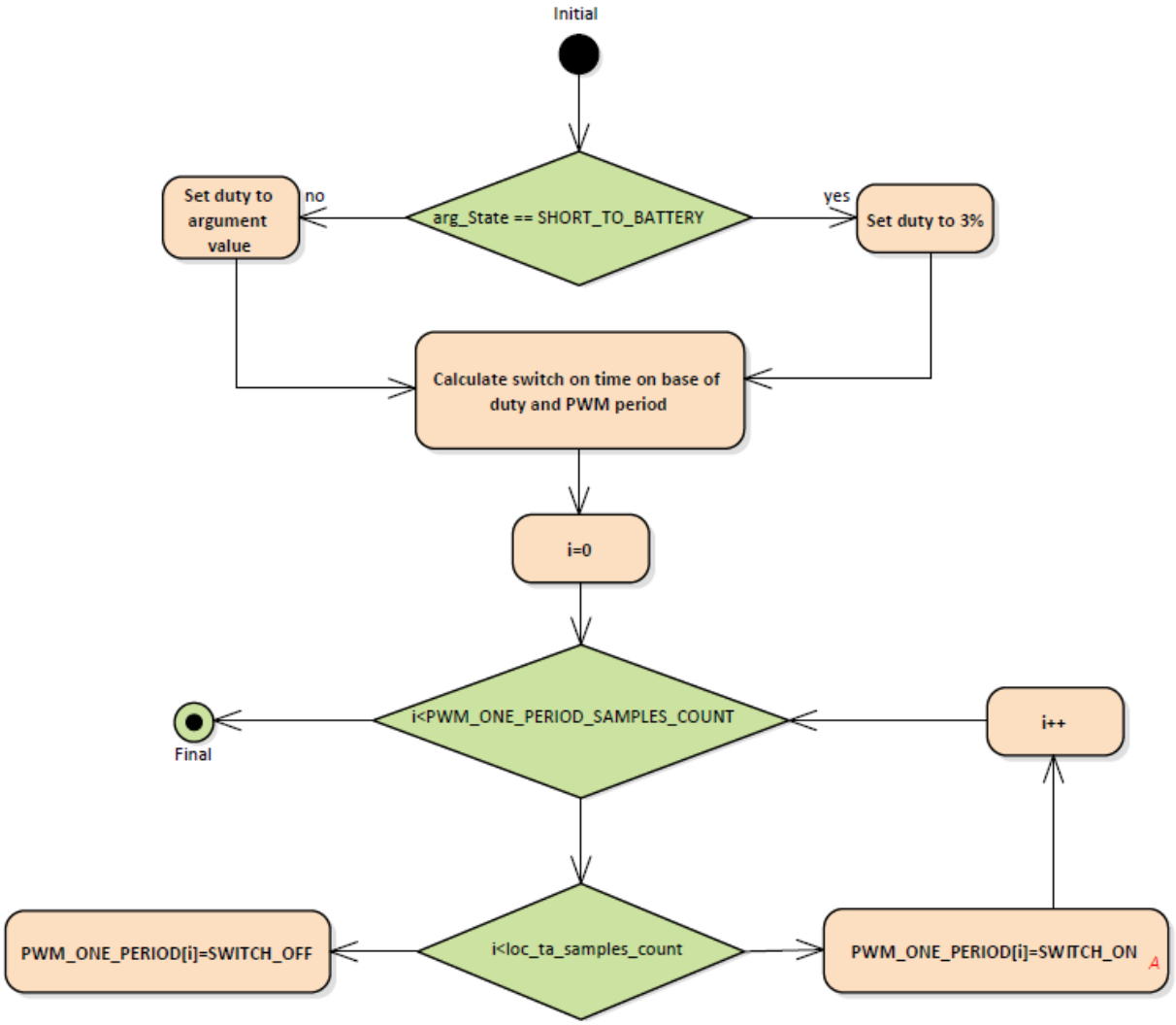
5.3.4 cSwitchBranch

Metoda Calc_rSwitchBranch() vypočítává celkový odpor spínače a filtrů pro stav se sepnutým a rozepnutým spínačem (Obr. 33).



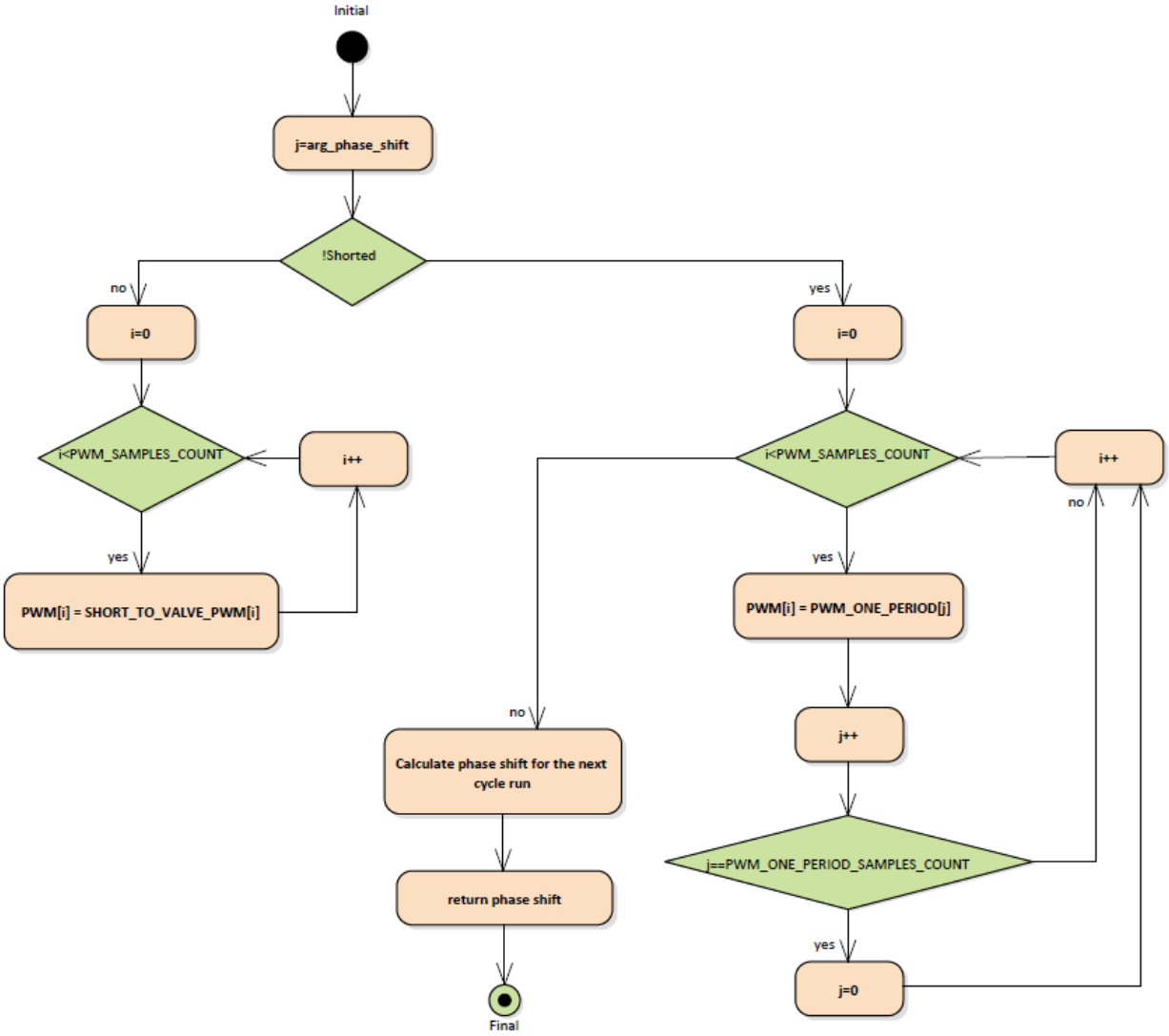
Obr. 33 Vývojový diagram pro metodu Calc_rSwitchBranch()

Metoda SetOnePeriodPWM() vyplňuje pole PWM_ONE_PERIOD hodnotou 0 pro stav s rozepnutým spínačem a hodnotou 1 pro stav s rozepnutým spínačem na základě střídy a periody PWM (Obr. 34). Pro stav „zkrat na baterii“, střída je nastavená na 3 %. V tomto chybovém stavu do spínače teče velký proud, kvůli tomu se spínač přehřívá a pomocí vnitřní ochrany se rozepíná, proto se střída mění na hodnotu blízkou 3 %.



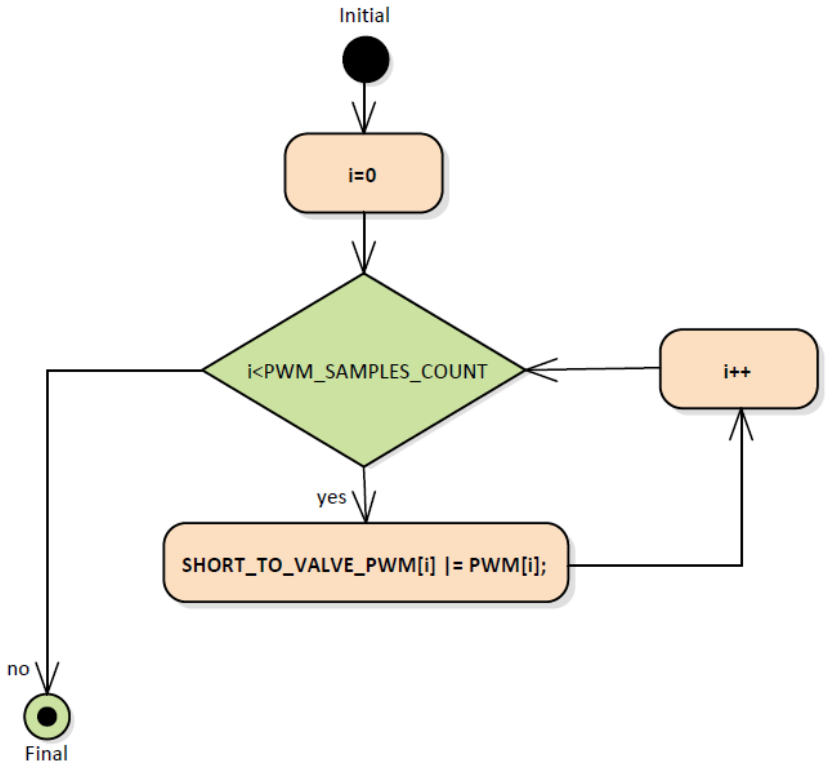
Obr. 34 Vývojový diagram pro metodu SetOnePeriodPWM()

Metoda SetPWM() naplňuje pole PWM takovým počtem polí PWM_ONE_PERIOD, které se vejdou do periody hlavní smyčky programu (Obr. 35). Díky takovému způsobu nastavení PWM simulace pracuje správně pro různé periody hlavní smyčky a různé periody PWM. Další problém, který řeší tento způsob, je výpočet fázového posunu PWM signálu pro jednotlivé ventily. Tento posun se může měnit v závislosti na různých kombinacích periody hlavní smyčky a periody PWM. V případě, že nastala chyba „zkrat mezi ventily“, pole PWM hodnot se naplní hodnotami z pole SHORT_TO_VALVE_PWM.



Obr. 35 Vývojový diagram pro metodu SetPWM()

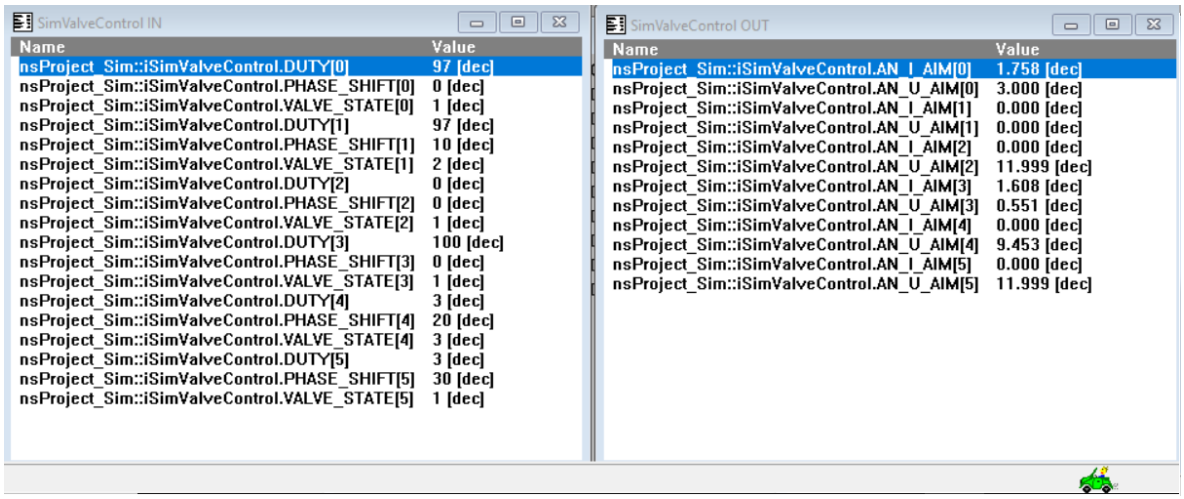
Metoda SetPWMShortToValve() vyplní pole hodnot SHORT_TO_VALVE_PWM (Obr. 36). Při poruchovém stavu „zkrat mezi ventily“ se PWM signály zkratovaných ventilů logicky sečtou.



Obr. 36 Vývojový diagram pro metodu SetPWMShortToValve()

5.4 Implementace

Implementace je provedená podle designu architektury a modulů. Simulace je zaintegrována do prostředí SIL (Obr. 37). Je vytvořeno pole s 6 instancemi, každá z nich znamená jednotlivý ventil. Ventily 0, 1, 4 a 5 jsou nastavené jako proporcionální, ventily 2 a 3 jsou typu ON/OFF. Program běží cyklicky, v každém cyklu probíhá přepočet výstupních hodnot na základě hodnot vstupních.



Obr. 37 Výstup z prostředí SIL

Pojmenování tříd, metod, proměnných, souboru je provedeno podle interních předpisů. Je provedena analýza složitosti kódu.

Struktura projektu:

source:

- Valve.cpp/Valve.hpp – zdrojový a hlavičkový soubor třídy Valve.
- Filter.cpp/Filter.hpp – zdrojový a hlavičkový soubor třídy Filter.
- SwitchBranch.cpp/SwitchBranch.hpp – zdrojový a hlavičkový soubor třídy SwitchBranch.
- SimValveControl.cpp/SimValveControl.hpp – zdrojový a hlavičkový soubor třídy SimValveControl.
- defs_sim_valve_control.hpp – nadefinované hodnoty.

default_cfg:

- init_sim_valve_control.cpp /init_sim_valve_control.hpp – inicializační hodnoty.

Analýza složitosti kódu je provedena pomocí nástroje CMT. Tento nástroj kontroluje následující metriky [31]:

- **v(G)** – cyklomatická složitost, kontroluje složitost toku kódu;
- **LOCphy** – počet fyzických řádků kódu;
- **LOCpro** – počet programových řádků kódu;

- **c%** - značka „-“ znamená, že procento komentářů je menší, než je doporučeno;
- **V** – Halsteadův objem, informační obsah kódu;
- **B** – Halsteadův odhadovaný počet chyb, kolik chyb je pravděpodobně v kódu, tato metrika je založená na vypočítané složitosti;
- **MI** – index udržovatelnosti.

V Tab. 1 jsou uvedeny limitní hodnoty těchto metrik pro celý soubor a pro jednotlivé funkce.

Tab. 1 Limity pro jednotlivé metriky (podle interní dokumentace)

	soubor	funkce
V(G)	1–1000	1–10
LOCpro	4–4000	4–40
c%	30–75	30–75
V	100–8000	20–1000
B	0–2	nedefinováno
MI	65 – nedefinováno	65 – nedefinováno

Pro metodu UpdateValveParams() není splněná limita programových řádků kódu. Limitní hodnota je překročena o 10 řádků. A díky tomu je překročená limita pro Halsteadův objem. Jelikož stejnou funkcionalitu nelze implementovat v 40 řádcích a rozdělení této metody na dvě části nemá logický význam, lze výsledek analýzy zanedbat. Výsledky analýzy pro ostatní soubory a metody simulace jsou v doporučeném rozmezí a nepřekračují limity. Příloha A obsahuje soubor s výsledky analýzy složitosti kódu provedené pomocí nástroje CMT.

5.5 Testy: porovnání s chováním reálného systému

Testování simulace je provedené porovnáním výsledků simulace v prostředí SIL s hodnotami naměřenými na HILu pomocí programu CANAPE. Pro proporcionální druh

ventilu jsou měření a simulace provedené pro různé hodnoty střídy PWM signálu: 3 %, 25 %, 50 %, 75 %, 97 %. Pro ON/OFF ventil jsou měření a simulace provedené pro stavy ON a OFF. Výstupní signály simulace jsou napojené na vstupy diagnostické komponenty. Popis parametrů, které kontroluje diagnostická komponenta pro určení aktuálního stavu, jsou převzaté z interní dokumentace.

5.5.1 Stav bez chyby

Pro stav bez chyby jsou hodnoty vypočtené v simulaci blízké skutečným hodnotám. Nepřesnosti ve výpočtech jsou způsobeny vlivem indukčnosti ventilu a vlivem kondenzátorů ve filtrech. Důvodem také je nelineární voltampérová charakteristika spínače. Ve výpočtech nejsou tyto vlivy zohledněné, proto se tyto nepřesnosti projeví i ve výpočtech pro chybové stavy.

Tab. 2 Stav bez chyby, proporcionální ventil

NO ERROR	Střída, %	3	25	50	75	97
měření	AN_I_AIM, mA	19	413	885	1348	1732
	AN_U_AIM, mV	12055	9583	6513	3451	755
simulace	AN_I_AIM, mA	66	549	1099	1648	2131
	AN_U_AIM, mV	11622	9187	6375	3562	1087

Tab. 3 Stav bez chyby, ON/OFF ventil

NO ERROR		ON	OFF
měření	AN_I_AIM, mA	1254	16
	AN_U_AIM, mV	276	11 886
simulace	AN_I_AIM, mA	1608	0
	AN_U_AIM, mV	551	11 999

5.5.2 Zkrat na zem

Ve stavu „zkrat na zem“ by odporem R_{shunt} neměl téct žádný proud. Rozdíl v hodnotách naměřených a vypočtených je způsoben tím, že ve skutečnosti tento zkrat nevede přímo na zem, ale přes obvody HILu, které vytváří odpor, jehož hodnotu není možné zjistit.

Tab. 4 Stav „zkrat na zem“, proporcionální ventil

SHORT TO GND	Střída, %	3	25	50	75	97
měření	AN_I_AIM, mA	62	60	62	63	62
	AN_U_AIM, mV	262	231	212	194	187
simulace	AN_I_AIM, mA	1	1	0	0	0
	AN_U_AIM, mV	0	0	0	0	0

Tab. 5 Stav „zkrat na zem“, ON/OFF ventil

SHORT TO GND		ON	OFF
měření	AN_I_AIM, mA	150	49
	AN_U_AIM, mV	114	131
simulace	AN_I_AIM, mA	0	1
	AN_U_AIM, mV	0	0

Diagnostika této chyby se provádí ve stavu, když je spínač rozepnut. Ve stavu se sepnutým spínačem nelze odlišit tuto chybu od chyby „rozpojený obvod“.

Následující podmínky musí být splněné, aby diagnostika detekovala tuto chybu:

- AN_I_AIM < prahová hodnota proudu.
- AN_U_AIM < prahová hodnota napětí.

Výsledky simulace ukazují, že hodnota proudu a napětí je výrazně menší než ve stavu bez chyby. Je ověřeno, že diagnostická komponenta rozpozná tuto chybu na základě simulovaných hodnot.

5.5.3 Rozpojený obvod

Naměřený proud se liší od vypočteného kvůli vlivu zesilovače. Při této chybě by do rezistoru R_{shunt} neměl téct žádný proud, ale kvůli tomu, že zesilovač v tomto případě funguje mimo pracovní oblast, naměří se malý proud. Rozdíl v naměřených a vypočtených hodnotách napětí je způsoben vlivem kondenzátorů.

Tab. 6 Stav „rozpojený obvod“, proporcionální ventil

OPEN LOAD	Střída, %	3	25	50	75	97
měření	AN_I_AIM, mA	17	20	22	24	25
	AN_U_AIM, mV	6800	6338	4700	3800	3500
simulace	AN_I_AIM, mA	0	0	0	0	0
	AN_U_AIM, mV	9170	7090	4727	2363	284

Tab. 7 Stav „rozpojený obvod“, ON/OFF ventil

OPEN LOAD		ON	OFF
měření	AN_I_AIM, mA	16	16
	AN_U_AIM, mV	4500	8217
simulace	AN_I_AIM, mA	0	0
	AN_U_AIM, mV	0	10350

Následující podmínky musí být splněné, aby diagnostika detekovala tuto chybu:

- AN_I_AIM < prahová hodnota proudu.

Velký rozdíl mezi skutečnou hodnotou odporu ventilu a vypočtenou:

$$R_{ValveCalc} = \frac{(VPS - AN\_U\_AIM)}{AN\_I\_AIM} \tag{22}$$

Výsledky simulace ukazují, že hodnota proudu je výrazně menší než ve stavu bez chyby. Je ověřeno, že diagnostická komponenta rozpozná tuto chybu na základě simulovaných hodnot.

5.5.4 Zkrat na baterii

Když nastává tato chyba, přes spínač teče hodně velký proud. Tento proud zahřívá spínač a ten se díky teplotní ochraně rozepíná. Až se jeho teplota opět vrátí do normální hodnoty, tak sepne. Většinu času ale je rozepnut, proto střída se mění na hodnotu kolem 3%. Vypočtená hodnota proudu je větší. Je to způsobeno tím, že v simulaci je napevno použita střída 3%, ale ve skutečnosti střída může kolísat od 0% do 3% nebo i více.

Tab. 8 Stav „zkrat na baterii“, proporcionální ventil

SHORT TO BAT	Střída, %	3	25	50	75	97
měření	AN_I_AIM, mA	332	455	455	455	455
	AN_U_AIM, mV	11946	11799	11704	11798	11804
simulace	AN_I_AIM, mA	783	783	783	783	783
	AN_U_AIM, mV	11906	11906	11906	11906	11906

Tab. 9 Stav „zkrat na baterii“, ON/OFF ventil

SHORT TO BAT		ON	OFF
měření	AN_I_AIM, mA	17	16
	AN_U_AIM, mV	12089	12082
simulace	AN_I_AIM, mA	0	0
	AN_U_AIM, mV	12000	12000

Následující podmínky musí být splněné, aby diagnostika detekovala tuto chybu:

- AN_U_AIM = VPS.
- uC_IN_U_AIM != uC_OUT_U_AIM – vstupní PWM signál se nerovná výstupnímu.

Je ověřeno, že diagnostická komponenta rozpozná tuto chybu na základě simulovaných hodnot.

5.5.5 Zkrat mezi ventily

V Tab. 10 jsou uvedeny naměřené hodnoty pro stav „zkrat mezi dvěma proporcionálními ventily“. Střída obou ventilu je stejná. V Tab. 11 jsou uvedeny naměřené hodnoty pro stav „zkrat mezi dvěma ventily“, jeden proporcionální, druhý ON/OFF ve stavu OFF. V Tab. 12 jsou evidovány naměřené hodnoty pro stav „zkrat mezi dvěma ventily“, jeden proporcionální, druhý ON/OFF ve stavu ON.

Tab. 10 Stav „zkrat mezi dvěma proporcionálními ventily“

SHORT TO VALVE	Střída, %	3	25	50	75	97
	AN_I_AIM 1, mA	832	491	1198	2036	2063
	AN_U_AIM 1, mV	12	9522	6605	3621	90
	AN_I_AIM 2, mA	24	295	405	319	1029
měření	AN_U_AIM 2, mV	12	9534	6048	4188	1970

Tab. 11 Stav „zkrat mezi dvěma ventily“, jeden proporcionální, druhý ON/OFF ve stavu OFF

SHORT TO VALVE	Střída, %	3	25	50	75	97
	AN_I_AIM 1, mA	20	440	1089	1908	2065
	AN_U_AIM 1, mV	12 063	9595	6600	3551	970
	AN_I_AIM 3, mA	22	255	345	265	53
měření	AN_U_AIM 3, mV	12 063	9649	6825	4160	1876

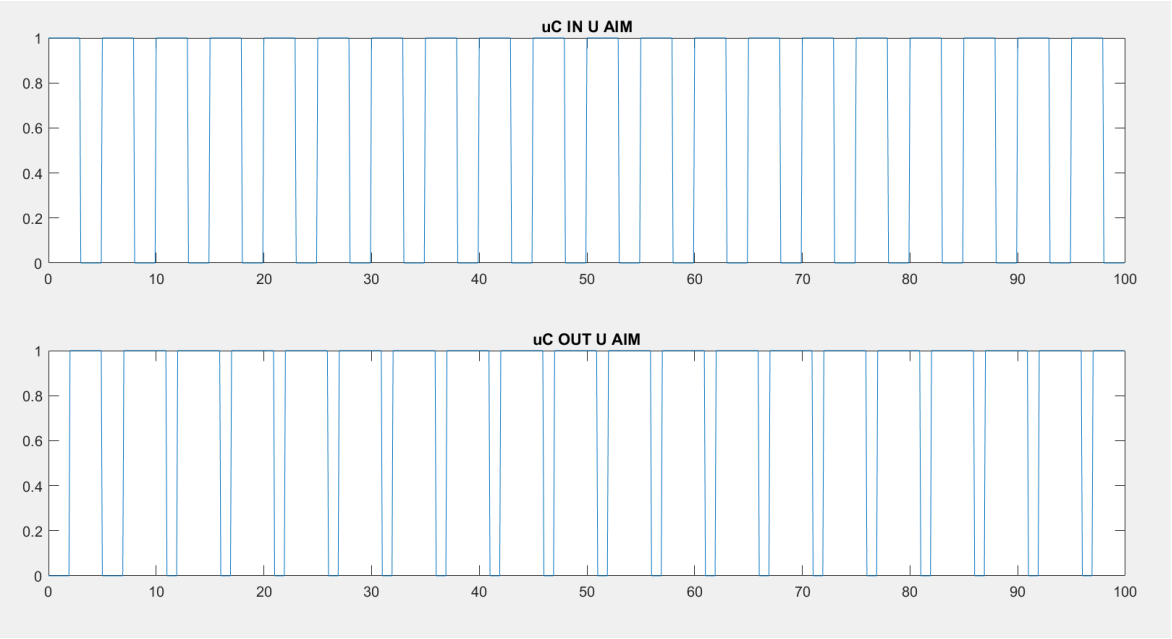
Tab. 12 Stav „zkrat mezi dvěma ventily“, jeden proporcionální, druhý ON/OFF ve stavu ON

SHORT TO VALVE	Střída, %	3	25	50	75	97
	AN_I_AIM 1, mA	18	357	713	1075	1392
	AN_U_AIM 1, mV	1551	1443	1200	1160	1020
	AN_I_AIM 4, mA	2066	2066	2066	2067	2066
měření	AN_U_AIM 4, mV	582	559	535	498	451

Z naměřených hodnot je vidět, že tyto hodnoty závisí na střídě obou ventilů, na fázovém posunu PWM signálu. Výpočty pro tento stav jsou obtížné a v podstatě nemají význam, protože diagnostika tohoto druhu chyby probíhá pomocí porovnání časových značek vstupního a výstupního PWM signálu:

- uC_IN_U_AIM != uC_OUT_U_AIM – vstupní PWM signál se nerovná výstupnímu.

V simulaci se tyto dva signály nerovnají (Obr. 38), proto je tato chyba detekována diagnostickou komponentou.



Obr. 38 Vstupní a výstupní PWM signál

Závěr

Diplomová práce se zabývá vytvořením softwarové simulace proudového řízení ventilů automatické převodovky. V první části práce bylo popsáno, jak funguje elektronika ve vozidle, jaké řídicí jednotky mohou být součástí vozidla a jakou funkci tyto jednotky plní. V druhé části byla popsána mechatronika automatické převodovky, princip automatického řazení pomocí ventilů, druhy ventilů a princip jejich funkčnosti. Ve třetí části byl nastíněn proces vývoje softwaru podle V-modelu. Ve čtvrté části bylo popsáno obecné schéma proudového řízení ventilů pomocí PWM signálu a byly provedeny výpočty pro stav bez chyby a pro chybové stavy. V poslední části je představen proces vývoje softwaru simulace. Nejdřív byly určeny požadavky, v daném případě požadavky odpovídají zadání diplomové práce.

Na základě požadavků byl vytvořen design. V designu architektury byl definován interface, vstupní a výstupní signály a parametry, čímž se zajistila parametrizovatelnost simulace. V této fázi bylo provedeno rozdělení simulace na moduly. Další fází bylo vytvoření designu modulů. Byl vytvořen podrobný popis metod, které každý z definovaných modulů obsahuje. Základní princip simulace spočívá v tom, že v každém cyklu se vypočítává určitý počet časových značek výstupního proudu, napětí a PWM signálu. Tento přístup umožňuje nastavení hodnot periody hlavní smyčky a periody PWM. Design architektury a modulů byl graficky znázorněn v programu Enterprise Architect. Následující fází bylo přenést design do podoby kódu. Pro implementaci byl použit jazyk C++, protože systém, do kterého byla simulace integrována je napsán v tomto jazyce. Tento jazyk perfektně vyhovuje požadavku modularity systému, která byla realizována pomoci tříd. Byla provedena analýza složitosti kódu pomocí nástroje CMT.

Po provedení implementace a integrace kódu, byl udělán test správné funkčnosti simulovaného systému porovnáním s reálním systémem. V prostředí HIL byly naměřeny reálné fyzické hodnoty výstupního proudu a napětí pro proporcionální ventil (střída: 3 %, 25 %, 50 %, 75 %, 97 %) a pro ventil ON/OFF (stav: ON, OFF). Tato měření byla provedena pro stav bez chyby, zkrat na baterii, zkrat na zem, rozpojený obvod a zkrat mezi ventily. Odpovídající parametry byly vypočteny v simulaci. Celkově se dá říct, že výsledky simulace a měření na Hilu jsou si podobné. Případné neshody jsou popsány v příslušné kapitole. Nejdůležitější ale je, že s takovými výsledky simulace diagnostická komponenta

detekuje stav bez chyby a chybové stavy. Výstupy ze simulace jsou napojeny na diagnostickou komponentu, čímž se ověřilo, že simulace řízení ventilů generuje správné výstupy pro odhalení uvedených chyb. Díky provedené simulaci je možné testovat diagnostickou komponentu v prostředí SIL, což usnadní a urychlí budoucí vývoj této komponenty.

Seznam literatury a informačních zdrojů

[1] SCHMID, M. Automotive Bus Systems. In: *Umd.edu* [online]. © 2011 [cit. 2021-05-04]. Dostupné z: https://user.eng.umd.edu/~austin/enes489p/project-resources/SchmidAutoBusSystems.pdf

[2] SHARMA, B. Role of Embedded Systems in Automobile Technology. *International Journal for Scientific Research & Development*. 2014, Vol. 1, Iss. 12, pp. 2759–2761. ISSN 2321-0613.

[3] MCMICHAL. Jak elektromagnetický ventil funguje? In: *Smstork.cz* [online]. 24. 4. 2015 [cit. 2021-05-04]. Dostupné z: https://www.smstork.cz/jak-elektromagnetick y-ventil-funguje/

[4] BŘOUŠEK, J. *Model planetového soukolí.* Liberec, 2011. Bakalářská práce. Technická univerzita v Liberci, Fakulta strojní. Vedoucí práce Josef Břoušek.

[5] CODEGRIP. A Simple Understanding of Code Complexity. In: *Codegrip.tech* [online]. 9. 6. 2020 [cit. 2021-05-04]. Dostupné z: https://www.codegrip.tech/productivity/a-simple-understanding-of-code-complexity/

[6] CODERLESSONS. Osnovy projektirovanija programnogo obespechenija. *Coderlessons.com* [online]. © 2020 [citováno 2021-05-04]. Dostupné z: https://coderlessons.com/tutorials/akademicheskii/programmnaia-inzheneriia/osnovy-proektirovaniia-programmnogo-obespecheniia

[7] LEFFINGWELL, D. and D. WIDRIG. *Managing software requirements.* New York: Addison-Wesley, 2001. ISBN 0-2016-1593-2.

[8] ECU TESTING LTD. Ecu explained. *Ecutesting.com* [online]. © 2021 [cit. 2021-05-04]. Dostupné z: https://www.ecutesting.com/categories/ecu-explained/

[9] ELUC. Automatické převodovky. *Eluc.kr-olomoucky.cz* [online]. © 2021 [cit. 2021-05-04]. Dostupné z: https://eluc.kr-olomoucky.cz/verejne/lekce/1477

[10] GEEKSFORGEEKS. Difference between High Level Design and Low Level Design. In: *Geeksforgeeks.org* [online]. 13. 10. 2020 [cit. 2021-05-04]. Dostupné z: https://www.geeksforgeeks.org/difference-between-high-level-design-and-low-level-design/

[11] RANGAM, K. What Is An ECU? Electronic Control Unit (ECU) Explained. In: *Gomechanic.in* [online]. 4. 10. 2020 [cit. 2021-05-04]. Dostupné z: https://gomechanic.in/blog/ecu-electronic-control-unit-explained/

[12] RANGAM, K. What Is a TCU? Transmission Control Unit (TCU). In: *Gomechanic.in* [online]. 5. 10. 2020 [cit. 2021-05-04]. Dostupné z: https://gomechanic.in/blog/tcu-transmission-control-unit-explained/

[13] SUVIK. Hydrodynamický měnič točivého momentu. *Suvik.cz* [online]. © 2021 [cit. 2021-05-04]. Dostupné z: https://suvik.cz/clanky/hydromenic.html

[14] KAPS AUTOMATIC. Hydroměniče. *Kaps.cz* [online]. © 2021 [cit. 2021-05-04]. Dostupné z: https://www.kaps.cz/sluzby/hydromenice

[15] KAPS AUTOMATIC. Revize hydraulických rozvaděčů. *Kaps.cz* [online]. © 2021 [cit. 2021-05-04]. Dostupné z: https://www.kaps.cz/revize-hydraulickych-rozvadecu-1226

[16] KLOCWORK. MISRA.GOTO. *Bullwhip.physio-control.com* [online]. © 2021 [cit. 2021-05-04]. Dostupné z: https://bullwhip.physio-control.com/documentation/help/reference/misra.goto.htm

[17] KLOCWORK. MISRA.IF.NO_ELSE. *Bullwhip.physio-control.com* [online]. © 2021 [cit. 2021-05-04]. Dostupné z: https://bullwhip.physio-control.com/documentation/help/reference/misra.if.no_else.htm?highlight=if+else%C2%A8

[18] KLOCWORK. MISRA.SIZEOF.SIDE_EFFECT. *Bullwhip.physio-control.com* [online]. © 2021 [cit. 2021-05-04]. Dostupné z: https://bullwhip.physio-control.com/documentation/help/reference/misra.sizeof.side_effect.htm?highlight=mandatory

[19] KLOCWORK. MISRA.SWITCH.NODEFAULT. *Bullwhip.physio-control.com* [online]. © 2021 [cit. 2021-05-04]. Dostupné z: https://bullwhip.physio-control.com/documentation/help/reference/misra.switch.nodefault.htm?highlight=switch

[20] KRUTIMOTOR. EBU AKPP: Ustrojstvo i princip raboty. *Kurimotor.ru* [cit. 2021-05-04]. Dostupné z: http://krutimotor.ru/ebu-akpp-ustrojstvo-neispravnosti-remont/

[21] KRUTIMOTOR. Gidrotransformator. *Krutimotor.ru* [online]. © 2021 [cit. 2021-05-04]. Dostupné z: http://krutimotor.ru/gidrotransformator-akpp-ustrojstvo-printsip-raboty/

[22] THE MATHWORKS. Hardware-in-the-Loop (HIL) Simulation. *It.mathworks.com* [online]. © 2021 [cit. 2021-05-04]. Dostupné z: https://it.mathworks.com/discovery/hardware-in-the-loop-hil.html

[23] THE MATHWORKS. Software-in-the-Loop Simulation. *Mathworks.com* [online]. © 2021 [cit. 2021-05-04]. Dostupné z: https://www.mathworks.com/help/ecoder/software-in-the-loop-sil-simulation.html

[24] *MISRA* [online]. © 2021 [cit. 2021-05-04]. Dostupné z: https://www.misra.org.uk/

[25] NAVET, N. and F. SIMONOT-LION. *Automotive Embedded Systems Handbook.* Boca Raton: Taylor & Francis Inc, 2009. ISBN 9780849380266.

[26] OOO "APLI-SENSOR". Proporcionalnyj klapan. *Aplisensor.com.ua* [online]. © 2021 [cit. 2021-05-04]. Dostupné z: http://www.aplisensor.com.ua/solenoid-valves/proporcionalnyj-klapan.html

[27] OOO "ITALGAZ". Solenoidnyj klapan. *Italgaz.com.ua* [online]. © 2021 [cit. 2021-05-04]. Dostupné z: https://www.italgaz.com.ua/wiki/solenoid-valve.html

[28] DAVIES, O. Planetary Gearbox. In: *Trapeziumengineering.com* [online]. 23. 10. 2014 [cit. 2021-05-04]. Dostupné z: http://www.trapeziumengineering.com/2014/10/23/planetary-gear-box/

[29] PRATA, S. Mistrovství v C++. 2. aktualiz. vyd. Brno: Computer Press, 2004. ISBN 978-80-251-0098-1.

[30] ŠIMČÁK, S. Zjednodušený 3D model samočinné převodovky ZF 4HP20. Praha, 2020. Bakalářská práce. ČVUT, Fakulta strojiní. Vedoucí práce Jaroslav Kaněra.

[31] TESTWELL CMT++. Complexity Measurement Tool for C/C++/C#. *Testwell.fi* [online]. © 2021 [cit. 2021-05-04]. Dostupné z: https://www.testwell.fi/cmtdesc.html

[32] TUTORIALS POINT. Software Implementation. *Tutorialspoint.com* [online]. © 2021 [cit. 2021-05-04]. Dostupné z: https://www.tutorialspoint.com/software_engineering/software_implementation.htm

[33] TUTORIALS POINT. Software Requirements. *Tutorialspoint.com* [online]. © 2021 [cit. 2021-05-04]. Dostupné z: https://www.tutorialspoint.com/software_engineering/software_requirements.htm

[34] TUTORIALS POINT. SDLC - V-Model. *Tutorialspoint.com* [online]. © 2021 [cit. 2021-05-04]. Dostupné z: https://www.tutorialspoint.com/sdlc/sdlc_v_model.htm

[35] CHEN, A. Automotive Systems. In: *Bristol.ac.uk* [online]. 22. 10. 2018 [cit. 2021-05-04]. Dostupné z: https://blect.blogs.bristol.ac.uk/2018/10/22/automotive-systems/

[36] VISUAL PARADIGM. What is Unified Modeling Language (UML)? *Visual-paradigm.com* [online]. © 2021 [cit. 2021-05-04]. Dostupné z: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/#:~:text=UML%2C%20short%20for%20Unified%20Modeling,business%20modeling%20and%20other%20non%2D

[37] ZF FRIEDRICHSHAFEN AG. EST 54. *Zf.com* [online]. © 2021 [cit. 2021-05-04]. Dostupné z: https://www.zf.com/products/en/trucks/products_50267.html

Seznam obrázků

Seznam tabulek

Přílohy

Příloha A – Výsledek analýzy kódu pomocí nástroje CMT

File: \frdcc_esgi_basic\example\source\sim\project_sim.cpp

Line	Measured object	v(G)	LOCphy	LOCpro	c%	V	B	MI
128	cMainSim::							
128	SoftcarReference()	1	7	4		40	0.01	163
136	SoftcarInit()	1	10	5		56	0.01	157
147	SoftcarCyclic()	1	10	5		94	0.02	154
158	SoftcarTerminate()	1	6	4		20	0.00	171
165	project_sim.cpp	4	165	79		3648	0.93	121

File: \frdcc_esgi_basic\example\source\sim\project_sim.hpp

Line	Measured object	v(G)	LOCphy	LOCpro	c%	V	B	MI
71	project_sim.hpp	7	71	27		402	0.09	114

File: \frdcc_esgi_basic\example\source\sim\sim_valve_control\SimValveControl.cpp

Line	Measured object	v(G)	LOCphy	LOCpro	c%	V	B	MI
15	cSimValveControl::							
15	cSimValveControl()	1	10	10		117	0.02	109
32	InitSimValveControl()	4	37	26		994	0.22	103
70	CyclicUpdateSimValveControl()	3	76	22		723	0.19	112
147	InitSoftcarVariables()	1	11	4		20	0.00	164
159	CountResultParams()	2	12	12		300	0.08	136
175	SimValveControl.cpp	10	175	89		3006	0.84	119

File: \frdcc_esgi_basic\example\source\sim\sim_valve_control\init.cpp

Line	Measured object	v(G)	LOCphy	LOCpro	c%	V	B	MI
63	init.cpp	2	63	45		581	0.09	116

File: \frdcc_esgi_basic\example\source\sim\sim_valve_control\Filter.cpp

Line	Measured object	v(G)	LOCphy	LOCpro	c%	V	B	MI
5	cFilter::							
5	cFilter()	1	8	5		104	0.03	148
14	Get_rFilter()	1	4	4		33	0.01	130
18	Filter.cpp	1	18	10		172	0.04	142

File: \frdcc_esgi_basic\example\source\sim\sim_valve_control\SwitchBranch.cpp

Line	Measured object	v(G)	LOCphy	LOCpro	c%	V	B	MI
6	cSwitchBranch::							
6	cSwitchBranch()	1	13	10		164	0.03	131
20	Set_rFilterU()	1	4	4		42	0.01	129
25	Set_rFilterPWM()	1	4	4		42	0.01	129
30	Calc_rBranch()	4	19	19		425	0.13	120
50	Get_rBranchSwitchON()	1	4	4		33	0.01	130
55	Get_rBranchSwitchOFF()	1	4	4		33	0.01	130
60	SetOnePeriodPWM()	4	27	25		412	0.12	106
89	SetPWM()	6	33	30		616	0.23	80
123	GetPWM()	1	4	4		38	0.01	129
128	SetPWM_ShortToValve()	2	7	7		119	0.03	114
136	SetShorted()	1	4	4		42	0.01	129
141	GetShorted()	1	4	4		33	0.01	130
143	SwitchBranch.cpp	13	143	121		2768	0.94	120

File: \frdcc_esgi_basic\example\source\sim\sim_valve_control\Valve.cpp

Line	Measured object	v(G)	LOCphy	LOCpro	c%	V	B	MI
4	cValve::							
4	cValve()	1	21	18		347	0.05	128
26	Set_rBranchSwitchON()	1	4	4		42	0.01	129
31	Set_rBranchSwitchOFF()	1	4	4		42	0.01	129
36	Get_I_AIM()	1	4	4		38	0.01	129
40	Get_U_AIM()	1	4	4		38	0.01	129
45	Calc_I_Source_NoError()	3	11	11		204	0.04	104
57	Calc_I_Source_ShortToGround()	3	17	14		345	0.07	124
75	Calc_I_Source_OpenLoad()	3	11	11		183	0.04	104
87	Calc_I_Source_ShortToBattery()	3	13	12		260	0.05	121
101	CalcValveParams_NoError()	3	15	14		469	0.11	114
117	CalcValveParams_ShortToGround()	3	18	16		431	0.10	121
136	CalcValveParams_OpenLoad()	3	13	13		235	0.04	100
150	CalcValveParams_ShortToBattery()	3	13	13		311	0.07	127
164	CalcValveParams()	1	26	19		290	0.05	117
191	UpdateValveParams()	7	53	50-		1117-	0.34	83
245	Valve.cpp	23	245	208		5760	2	117

File: \frdcc_esgi_basic\example\source\sim\sim_valve_control\SimValveControl.hpp

Line	Measured object	v(G)	LOCphy	LOCpro	c%	V	B	MI
47	SimValveControl.hpp	7	47	36		635	0.13	89

File: \frdcc_esgi_basic\example\source\sim\sim_valve_control\defs.hpp

Line	Measured object	v(G)	LOCphy	LOCpro	c%	V	B	MI
75	defs.hpp	3	75	43		791	0.11	96

File: \frdcc_esgi_basic\example\source\sim\sim_valve_control\init.hpp

Line	Measured object	v(G)	LOCphy	LOCpro	c%	V	B	MI
26	init.hpp	3	26	18		325	0.04	108

File: \frdcc_esgi_basic\example\source\sim\sim_valve_control\Filter.hpp

Line	Measured object	v(G)	LOCphy	LOCpro	c%	V	B	MI
27	Filter.hpp	4	27	19		241	0.05	103

File: \frdcc_esgi_basic\example\source\sim\sim_valve_control\SwitchBranch.hpp

Line	Measured object	v(G)	LOCphy	LOCpro	c%	V	B	MI
45	SwitchBranch.hpp	4	45	35		741	0.14	85

File: \frdcc_esgi_basic\example\source\sim\sim_valve_control\Valve.hpp

Line	Measured object	v(G)	LOCphy	LOCpro	c%	V	B	MI
58	Valve.hpp	4	58	47		1133	0.21	78