ZÁPADOČESKÁ UNIVERZITA V PLZNI
FAKULTA ELEKTROTECHNICKÁ

Katedra aplikované elektroniky a telekomunikací

DIPLOMOVÁ PRÁCE

Naprogramování a zprovoznění řídicí jednotky se zaměřením pro automobilový průmysl

vedoucí práce: **Ing. Michal Kubík**
autor: **Luděk Elis**

2012

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta elektrotechnická
Akademický rok: **2011/2012**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Luděk ELIS**
Osobní číslo:	**E10N0156P**
Studijní program:	**N2612 Elektrotechnika a informatika**
Studijní obor:	**Dopravní elektroinženýrství a autoelektronika**
Název tématu:	**Naprogramování a zprovoznění řídicí jednotky se zaměřením pro automobilový průmysl**
Zadávající katedra:	**Katedra aplikované elektroniky a telekomunikací**

Zásady pro vypracování:

1. Seznamte se s modelovacím prostředím Matlab-Simulink včetně prostupu generování kódu z modelu a prostudujte dokumentaci emulátoru řídicí jednotky.
2. V prostředí Matlab-Simulink vytvořte kód pro naprogramování emulátoru řídicí jednotky. Vytvořený kód by měl využívat dostupná rozhraní emulátoru.
3. Proveďte oživení naprogramovaného emulátoru řídicí jednotky ve spojení s HiL simulátorem.
4. Vytvořte sadu testů pro otestování oživené řídicí jednotky.

Rozsah grafických prací: **podle doporučení vedoucího**

Rozsah pracovní zprávy: **30 - 40 stran**

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

Student si vhodnou literaturu vyhledá v dostupných pramenech podle doporučení vedoucího práce.

Vedoucí diplomové práce: **Ing. Michal Kubík**
Katedra aplikované elektroniky a telekomunikací

Konzultant diplomové práce: **Dr. Radek Maňásek**
MBtech Bohemia s.r.o Plzeň

Datum zadání diplomové práce: **17. října 2011**

Termín odevzdání diplomové práce: **11. května 2012**

Doc. Ing. Jiří Hammerbauer, Ph.D.
děkan

L.S.

Doc. Dr. Ing. Vjačeslav Georgiev
vedoucí katedry

V Plzni dne 17. října 2011

Anotace

Předkládaná diplomová práce se zabývá naprogramováním emulátoru řídicí jednotky automobilu pomocí modelovacího prostředí Matlab-Simulink. Úvodní část se zabývá použitím universálních testovacích emulátorů a možnostmi jejich programování. Následuje popis konkrétně použitého emulátoru včetně všech jeho rozhraní a možného využití. Dále jsou uvedeny vlastnosti Simulinku a jeho nastavení pro základní používání knihovny Embedded Coder. Pátá kapitola se zabývá konfigurací programu PROVEtech:TA. Další dvě části jsou věnovány popisu vytvořených modelů. Model řídící jednotky nahraný v emulátoru a model prostředí nahraný v μHiLu. V poslední kapitole je popis výsledků této soustavy.

Klíčová slova

Emulátor, ECU, HiL, Testování, Simulink, Embedded Coder, Model, PROVEtech, Rychlý vývoj, Automobilová technika

Abstract

Presented master thesis deals with programming of an emulator for a car controlling unit using the Matlab-Simulink software. The first part deals with the usage of universal testing emulators and their programming options. In the following part is description of used emulator with all of its interfaces, and potential uses. Then there are the properties of Simulink and setting for a basic library Embedded Coder. The fifth part is focused on configuration of PROVEtech:TA software. The next two sections describes the developed models. Control unit model recorded in the emulator and environment model recorded in μHiL. In the last chapter is a description of the results of this system.

Key words

Emulator, ECU, HiL, testing, Simulink, Embedded Coder, model, PROVEtech, Rapid prototyping, Automotive

Prohlášení

Předkládám tímto k posouzení a obhajobě diplomovou práci, zpracovanou na závěr studia na Fakultě elektrotechnické Západočeské univerzity v Plzni. Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně, s použitím odborné literatury a pramenů uvedených v seznamu, který je součástí této diplomové práce. Dále prohlašuji, že veškerý software, použitý při řešení této diplomové práce, je legální.

V Plzni dne 9.5.2012 Luděk Elis

.....

Poděkování

Na tomto místě bych rád poděkoval svým rodičům a blízké rodině, za podporu a zázemí během mých studií. Dále bych rád poděkoval vedoucímu diplomové práce Ing. Michalu Kubíkovi za metodické vedení práce. Nemalý dík patří rovněž Ing. Jakubu Tallovi, za cenné rady a připomínky k práci.

Obsah

1 Úvod

Tato diplomová práce navazuje na diplomovou práci Jana Hynka, v níž byla vyvinuta HW koncepce řídící jednotky na bázi obecně přístupného mikroprocesoru, jenž je možné programovat pomoci SW Matlab-Simulink. Vzniknul tak universální emulátor řídící jednotky. Ten je možné použít při vývoji a návrhu nových elektronických řídících jednotek (ECU – Electronic Control Unit).

V průběhu tvorby modelu určeného k řízení systému je možné software otestovat, právě na těchto universálních zkušebních deskách, které byly pro tyto účely vytvořeny. Jedná se o multifunkční ECU poskytující většinu možných rozhranní, kterými by mohla skutečná ECU disponovat. Pod universální testovací deskou si lze představit například Ing. Hynkem vytvořený emulátor. Každá firma zabývající se návrhem a vývojem softwaru i hardwaru vlastní takovéto universální zkušební vývojové prostředky používané pro interní potřeby firmy.

Za tímto účelem byly vytvořeny universální vývojové desky, které jsou osazeny stejnými procesory, jako budou v budoucnu řídit reálný systém. Požadavek na shodnost procesoru v universální testovací desce a finální ECU je důležitý. Testování programu na vývojových prostředcích osazených stejnými procesory jako jsou použity na hotových ECU je označováno jako PiL (Procesor in the Loop). Je použit konkrétní procesor, který je testován, ale nejedná se o finální ECU.

Další inovací vedle univerzálních emulátorů, kde lze ušetřit čas a tím zrychlit fázi vývoje je použití lepších programů pro návrh řídícího softwaru. Klasické programování v C, C++, nebo dalších programovacích jazycích nemusí být tak přehledné jako nové způsoby, které lze použít k vytvoření kódu. Programátor, který píše program pro konkrétní procesor je nucen důkladně tento kód popisovat komentáři, aby zajistil lepší orientaci čtenáře s konkrétními kroky, které se v programu provádějí. Přesto je tato struktura tvorby kódu oproti novým vývojovým prostředkům méně přehledná a i samotný řešitel kódu se při veliké složitosti programu lehce ztratí. Navíc při změně procesoru nemusí být úprava kódu jednoduchá. Většinou je problematické původní program implementovat do jiného procesoru bez velkých změn. Oproti tomu již zmíněný Matlab-Simulink nebo LabVIEW umožňuje grafické programování. Jednotlivé příkazy psané do řádků, jsou nahrazeny schematickými bloky, které lze myší skládat do bloků a vzájemně je propojovat. Kód je tak velice přehledný a uživatel má možnost vidět celou strukturu programu v jednom schématu – modelu.

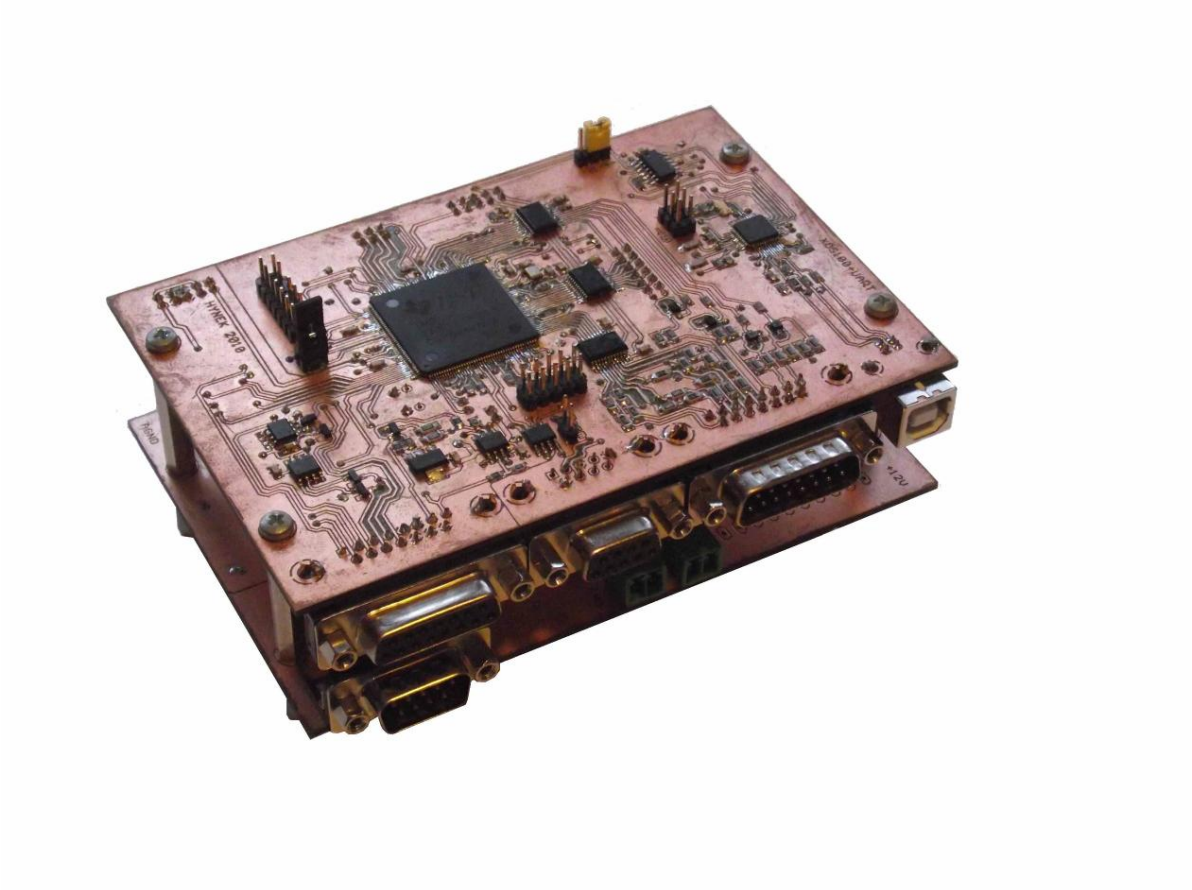
Tvorba kódu formou modelů a schémat je dnes upřednostňována většinu firem, jež potřebuje vytvořit řídící algoritmus. Převážná část aplikací, pro které je vhodné tento software použít k programování, je elektronika řízená jednočipovými mikroprocesory. A právě do této skupiny spadají i řídící jednotky automobilu, které mají na starosti řízení motoru, klimatizace a dalších podsystémů celého auta.

V této práci bude použito programu Matlab-Simulink k vytvoření modelu řídící jednotky dveří, což je i hlavním cílem této práce. Model by měl popisovat funkcionalitu zvolené řídící jednotky a měl by být syntetizovatelný pro vygenerování kódu s ohledem na použitý procesor v emulátoru. V tomto případě je emulátor osazen procesorem TMS320F2812. Matlab-Simulink umožňuje vytvořené modely převést do přeložitelného kódu, který je možné nahrát do jednočipového procesoru. Hierarchická struktura modelů umožňuje koncipovat i velmi složité systémy do přehledné soustavy subsystémů prakticky bez omezení počtu bloků. Návrh řídícího algoritmu pomocí tohoto softwaru je nejen rychlejší, ale i více univerzálnější pro širší použití cílových skupin.

V Simulinku nemusí být vytvářeny jen *modely řídicích systémů*, ale je zde možné navrhovat i *modely řízených systému*. Tyto modely je opět možné převést do přeložitelného kódu, určeného pro specializované testovací systémy založené převážně na technologii FPGA. PROVEtech:µHiL vytvořený společností MBtech je jedním z takových systémů, umožňujících řešení pro funkční testování elektronických řídicích jednotek (ECU), které lze použít ve všech fázích vývoje. Stačí nahrát správný model a propojit s testovanou ECU.

2 Emulátor řídicí jednotky

Ing. Hynkem byl vytvořen emulátor řídicí jednotky, kde celá jeho struktura a popis jednotlivých částí je uvedena v [1]. Na následujícím obrázku je vidět, jak může testovací vývojový prostředek vypadat. Jelikož nejsou tyto universální zkušební prostředky určené do provozního prostředí vozidla, není nutné, aby jejich provedení odolávalo nepříznivým vlivům tohoto prostředí. Použití emulátorů obecně je určené do laboratorních podmínek a zde nehrozí vibrace, vlhkost ani jiné vlivy, které by emulátor měly poškodit, nebo jiným způsobem jej činit nepoužitelným.



Obr. 2.1: Emulátor řídicí jednotky. [1]

Jak je z obrázku patrné, jedná se o dvě desky plošných spojů, které jsou mezi sebou propojeny. Z fotografie je vidět, že dominantou celého emulátoru je právě procesor, který je srdcem celého zařízení. Důvodem proč nejsou tyto vývojové prostředky navrženy jako masivní a odolná zařízení je především cena a pak také jednoduchost návrhu. Při vývoji a následném testování hraje velikou roli čas. Proto je potřeba, aby bylo universální zařízení

určené pro testování softwaru vytvořeno v co možná nejkratším čase.

Ve 4. kapitole p. Hynek popisuje, jakým způsobem byla řešena jednotlivá rozhraní, kterými emulátor disponuje. Mezi důležitá rozhranní, se kterými souvisí použití emulátoru jako universální řídicí jednotky, jsou považovány:

- analogové vstupy a výstupy
- digitální vstupy a výstupy
- měření odporu
- výkonové 12V výstupy
- PWM
- CAN

2.1 Ověření funkčnosti emulátoru

Pokud by měl být emulátor používán jako universální řídicí jednotka, měl by být v bezvadném stavu, tj. veškerá jeho rozhranní nesmí projevovat sebemenší známky poruch. Toto otestování je velice důležité, protože pokud bude nějaký vstup, respektive výstup, nefunkční nebo dokonce jen částečně funkční, budou se při testování vytvořeného modelu tyto poruchy projevovat. Není tak zcela jasné, zda je chyba v modelu, nebo hardware není funkční. Programátor netušící o hardwarovém poškození testovacího emulátoru, bude špatné chování stále přisuzovat programu, který vytvořil. Je tak jasné, že ani nemá možnost tento problém vyřešit a jen ztrácí čas, který by mohl věnovat zdokonalení řídícího algoritmu. V případě že se jen budeme spoléhat na bezvadnou funkci hardwaru, mnohdy se k funkčnímu modelu nemusíme ani dostat.

Hlavní části emulátoru byly p. Hynkem testovány zkušebními modely, kterými ověřoval funkčnost jednotlivých bloků. Ve většině případů byly pro testování použity již vytvořené modely z Matlab-Simulink demonstrující příkladové funkce. Nicméně bylo nutné provést revizi, k ověření správnosti zapojení celého zařízení a otestovat správnou funkci veškerých použitých rozhranní emulátoru. Bylo odhaleno několik neshod s přiloženou dokumentací.

Jednou z hlavních nedostatků, byly nepopsané veškeré vstupní a výstupní rozhranní emulátoru. Celá koncepce emulátoru je navržena jako sandwitchová struktura, která je složena ze dvou desek plošných spojů. Horní deska, na které je umístěn DSP procesor a ostatní bloky pracující s 5 V a druhá spodní deska, kde jsou napájecí zdroj, výkonové spínací prvky a bloky pracující s 12 V. Na těchto deskách jsou rozmístěny konektory se vstupy a výstupy.

2.1.1 Deska s procesorem

Na této desce jsou umístěny celkem čtyři konektory:

- USB sloužící pro připojení k PC
- CAN
- analogové vstupy a výstupy
- digitální vstupy a výstupy

Všechny výše uvedená rozhranní byla otestována a jejich zapojení ověřeno s přiloženou dokumentací. Neshody nebyly nalezeny.

2.1.2 Deska s výkonovými obvody

Deska je osazena rozhranními pracující s 12 V a svorkou pro napájení.

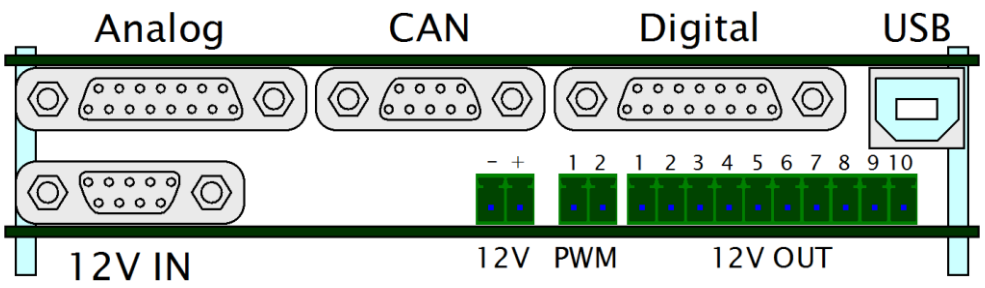
- 12V vstupy
- napájecí
- PWM
- výkonové výstupy

Tato deska již není tak podrobně v dokumentaci popsána a výše zmíněné konektory nejsou popsány vůbec. Bylo tedy nutné z doložené dokumentace tyto konektory správně popsat. Tento proces byl velice obtížný a byly odhaleny další nepřesnosti ve schématech. Veliký nedostatek je v různorodém značení pinů sloužících k propojení mezi oběma deskami. Číslování těchto pinů bylo rozdílné, což velice komplikovalo identifikaci. Od odvozování konkrétních pinů konektoru vstupů a výstupů emulátoru ze schématu bylo upuštěno a bylo nutné cesty zdokumentovat přímo z Layoutu.

Druhý vážný nedostatek byl nalezen u výstupu PWM. Byl zde zvolen shodný budič, který je použit u digitálních výkonových výstupů. Pro buzení digitálních výstupů je tento obvod dostatečný ovšem v případě výkonového výstupu PWM již tento obvod nedostačuje. Výrobcem udávaná doba vzestupné hrany je v rozmezí 100 až 400 µs. Obvod byl změřen a tento čas byl více než 400 µs. I přesto pokud by doba byla menší než 200 µs, nebylo by možné obvod použít s nosnou frekvencí převyšující 1 kHz. Použít takto nízkou frekvenci není možné, protože spadá do oblasti slyšitelného pásma, kde je velice výrazná.

2.2 Rozmístění konektorů Emulátoru

Samotný emulátor není dokonale označen a práce s ním je tak velice obtížná. Jednotlivé konektory umístěné na emulátoru nemají žádné popisy, podle kterých by bylo možné konkrétní konektor jednoznačně a určit. Toto označení není uvedeno ani na plošných spojích. Pro zjednodušení práce s Emulátorem a hrubou představu, kde jsou umístěny jednotlivé konektory a jaké jsou jejich funkce, je na následujícím obrázku naznačeno jejich umístění.



Obr. 2.2: Rozmístění konektorů na emulátoru.

2.3 Zapojení jednotlivých konektorů

V této kapitole jsou uvedeny tabulky popisující zapojení jednotlivých konektorů a jejich význam. Vždy je v prvním sloupci uvedeno číslo pinu konektoru, ve druhém název pinu procesoru, ve třetím o jaký typ rozhraní emulátoru se jedná a v posledním sloupci je uvedeno použití tohoto signálu – jakou bude mít funkci.

2.3.1 Konektor - 12 V

Svorka označená *12 V* slouží pro připojení napájení. Levý pin konektoru je určen k připojení země a pravý pin k připojení kladného potenciálu baterie. Jelikož je emulátor určen pro automobilový průmysl, velikost napájecího napětí je určena palubní sítí, tedy jmenovitých 12 V.

Tab. 2.1: Zapojení konektoru 12 V.

Pin konektoru	Pin DSP	Funkce vstupu	Funkce Emulátoru
1	-	+ 12V napájení	-
2	-	GND	-

2.3.2 Konektor - PWM

Jak již samotný název napovídá, jedná se o dvoukanálový PWM výstup. První kanál je vlevo a druhý vpravo. Pro naši funkci emulátoru nebyly tyto výstupy použity.

Tab. 2.2: Zapojení konektoru pro výstup PWM.

Pin konektoru	Pin DSP	Funkce vstupu	Funkce Emulátoru
PWM1	GPIOB14	Digitální 12V vstup	-
PWM2	GPIOB15	Digitální 12V vstup	-

2.3.3 Konektor - 12 V OUT

Následující konektor má deset pinů výkonových výstupů. Opět první kanál je vlevo a směrem doprava pokračují ostatní.

Vzhledem k použití emulátoru jako řídicí jednotky dveří spolujezdce je potřeba velkého množství právě těchto výstupů. Na dveřích se nacházejí nejrůznější motory, které musejí být připojeny právě k těmto výkonovým výstupům.

Tab. 2.3: Zapojení konektoru pro výkonové výstupy 12 V (12 V OUT).

Pin konektoru	Pin DSP	Funkce vstupu	Funkce Emulátoru
D12OUT1	GPIOB12	Digitální 12V výstup	motor - okno nahoru
D12OUT2	GPIOB15	Digitální 12V výstup	motor - okno dolu
D12OUT3	GPIOB13	Digitální 12V výstup	motor - zamknout
D12OUT4	GPIOB9	Digitální 12V výstup	motor - odemknout
D12OUT5	GPIOB7	Digitální 12V výstup	motor - zrcátko nahoru
D12OUT6	GPIOB6	Digitální 12V výstup	motor - zrcátko dolu
D12OUT7	GPIOB8	Digitální 12V výstup	motor - zrcátko do leva
D12OUT8	GPIOB10	Digitální 12V výstup	motor - zrcátko doprava
D12OUT9	GPIOB14	Digitální 12V výstup	blinkr
D12OUT10	GPIOB11	Digitální 12V výstup	vyhřívání zpětného zrcátka

Tyto tři popsané konektory se od ostatních liší použitým typem konektoru. Ve všech třech případech se jedná o svorky určené pro přenos s větším výkonem při napětí 12 V a používají shodný typ svorky. Může tak dojít k jejich neúmyslné záměně a tím i k možnému poškození.

2.3.4 Konektor - 12 V IN

Tento konektor se od předcházejících liší tím, že není výkonový a je konstrukčně použita jiná svorka *D-SUB vidlice*.

Tab. 2.4: Zapojení konektoru s digitálními vstupy.

Pin konektoru	Pin DSP	Funkce vstupu	Funkce Emulátoru
D12IN1	GPIOA15	Digitální 12V vstup	horní kontakt okna
D12IN2	GPIOA14	Digitální 12V vstup	dolní kontakt okna
D12IN3	GPIOA12	Digitální 12V vstup	dveřní kontakt
D12IN4	GPIOA11	Digitální 12V vstup	-
D12IN5	GPIOA10	Digitální 12V vstup	-
D12IN6	-	GND	
D12IN7	GPIOA13	Digitální 12V vstup	-
D12IN8	-	GND	
D12IN9	-	GND	

2.3.5 Konektor - Analog

Na vrchní desce plošných spojů je umístěn konektor s analogovými vstupy. Zde pro toto rozhraní je použito konektoru *D-SUB zásuvka*.

Tab. 2.5: Zapojení konektoru s analogovými vstupy.

Pin konektoru	Pin DSP	Funkce vstupu	Funkce Emulátoru
A1	ADCINA7	Anologový vstup	proud okna
A2	ADCINA4	Anologový vstup	-
A3	ADCINA1	Měření odporu	tlačítko okna
A4	ADCINA3	Měření odporu	-
A5	-	AGND	
A6	-	AGND	
A7	SPI + GPIOF13	Analogový výstup CH-A	
A8	SPI + GPIOF13	Analogový výstup CH-B	
A9	ADCINA6	Anologový vstup	ZV zrcátko vertikálně
A10	ADCINA5	Anologový vstup	ZV zrcátko horizontalně
A11	ADCINA2	Měření odporu	vložka zámku
A12	ADCINA0	Měření odporu	ZV zámek
A13	-	AGND	
A14	-	AGND	
A15	-	AGND	

2.3.6 Ostatní konektory (CAN, USB, Digital)

Zbývající konektory není nutné popisovat, protože je jejich zapojení a použití zcela jasné. Konektor CAN je zapojen podle příslušného standardu a konektor USB, určený pro připojení k PC a naprogramování procesoru, je zapojen také standardním způsobem.

Výjimku tvoří konektor ***Digial***, který obsahuje digitální vstupy i výstupy. Jejich detailní zapojení je uvedeno v [1]. Tyto digitální vstupy a výstupy nejsou zapojeny k žádným prvkům systému a emulátor jich tedy nevyužívá. Obecné použití rozhraní s napěťovou úrovní 5 V v automobilu je jen velmi zřídka a jejich využití je velice ojedinělé a to ve speciálních případech, kdy je nutné řídicí jednotku spojit digitálními vstupy a výstupy s jinou elektronickou jednotkou pracující se shodnou napěťovou úrovní.

2.4 Code Composer Studio V3.3

Software použitý v této práci pro přeložení kódu a jeho nahrání do paměti procesoru je Code Composer Studio™ v3.3 (CCStudio v3.3). Jedná se o komplexní vývojové prostředí (IDE) pro DSP procesory, mikrokontroléry a aplikační procesory od firmy Texas Instruments.

2.5 Programátor XDS100v1

Aby mohl být procesor naprogramován, je nutné k tomu použít například uvedený programátor XDS100. Je kompatibilní s Code Composer Studiem v3.3, využívající USB rozhraní a poskytující přístup k zařízení přes JTAG. Obvod od firmy Texas Instruments je postaven na integrovaném obvodu FT2232D. Je umístěn přímo na desce emulátoru. Počítač lze tedy připojit pomocí USB kabelu přímo k emulátoru a není potřeba jiného hardwaru.

2.6 Uvedení emulátoru do provozu

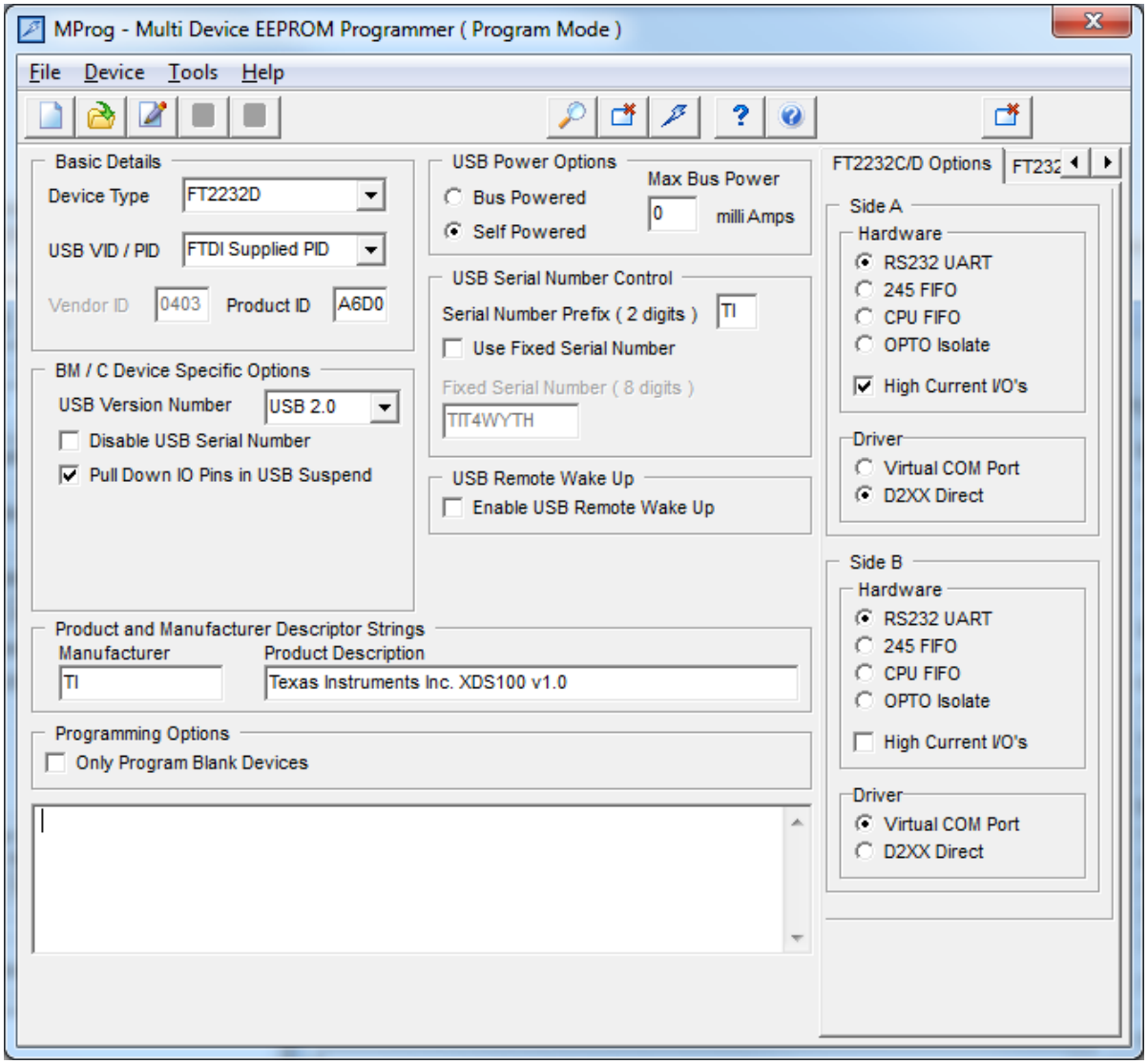
Abychom mohli emulátor použít, je potřeba nakonfigurovat počítač:

1. Nejprve je nutné nainstalovat CCStudio v3.3[1].
2. Nainstalujeme ovladače pro XDS100v1, které jsou dostupné ze stránek Texas Instruments Embedded Processors Wiki [12].
3. Dále je nutné nainstalovat ovladače k FT2232D, které jsou dostupné na stránkách

[1] Novější verze CCStudio není v Simulinku podporována.

výrobce [13].

4. Již je možné připojit emulátor k počítači pomocí USB a zapnout napájení.
5. Nahrajeme do paměti EEPROM obvodu FT2232D konfigurační data[2] pomocí programu MProg dostupného z [14]. Nastavení všech polí je vidět na *Obr. 2.3*.
6. Spustíme *Setup Code Composer Studio v3.3*, v záložce *Factory Boards* vyhledáme *F2812 XDS100 USB Emulátor*, přidáme jej do *Systém Configuration* a uložíme nastavení.
7. Spustíme *Code Composer Studio v3.3* a připojíme emulátor k procesoru *Debug → Connect.*



Obr. 2.3: Nastavení konfiguračních dat pro FT2232D.

[2] Konfigurační soubor *XDS100_HILsimulator.ept* je přiložen na CD

3 Řídící jednotky automobilu

Jedny z prvních automobilů měly jen velmi jednoduchou elektroniku. Možná by bylo vhodnější bavit se spíše o základní elektro-výzbroji sloužící především k chodu motoru a možná i k bezpečnosti, jako je například osvětlení vozidla, stěrače nebo klakson. Postupem času se auta vyvíjela, modernizovala a elektrických zařízení v autě přibývalo. Tak se do výbavy vozidla dostaly věci jako například autorádio, elektricky ovládaná okna a mnoho dalšího. Tento nárůst elektrických zařízení v autě není připisován pouze komfortním záležitostem, jak by se na první pohled mohlo zdát, ale i bezpečnostním systémům, nebo elektrické výzbroji kolem motoru.

Postupem času bylo těchto elektrických zařízení tolik, že množství kabelů v autě vybaveného nejnovější technikou, bylo neúnosné. S nástupem elektroniky a hlavně mikroprocesorů, dochází k velikému rozvoji a obrovským možnostem řízení všech elektrických i elektronických zařízení v autě. Již při použití jedné řídicí jednotky v autě starající se o běh motoru, přesněji pro zapalování a vstřikování paliva, skýtá veliké možnosti, jak tuto jednotku vhodně nastavit a naprogramovat. A právě zde, od místa vzniku řídicí jednotky motoru, začíná veliký rozvoj elektroniky vozidla.

3.1 Testování řídících jednotek

Při návrhu řídícího programu pro ECU je nutné průběžně tento kód testovat aby byla zajištěna přeložitelnost vytvořených modelů. Průběžným testováním také sledujeme, zda se řídicí jednotka chová správně a program koná správnou funkci a jeho chování se shoduje s provedenými simulacemi. Toto průběžné testování je důležité provádět s procesorem, který bude použitý v konkrétní řídicí jednotce. Každý výrobce má vlastní strukturu, které se mohou mezi sebou více či méně podobat, ale téměř nikdy nejsou zcela shodné. Odlišnost nemusí být dána pouze tím, že se jedná o jiného výrobce, ale i typové řady procesorů jednoho výrobce se mohou ve struktuře lišit. To může zapříčinit různé chování jednoho programu v různých typech procesorů. Je potom velmi těžké určit, zda tento rozdíl bude mít zásadní vliv na funkci řídicího algoritmu. Proto se průběžné testování řídícího programu testuje v definovaném typu procesoru. Ještě lepší by bylo testovat program již na finální ECU. Ve většině případů je ale ECU vyvíjena souběžně s řídícím programem a stejně jako program je ve fázi vývoje a pro testování pokroků v návrhu řídícího softwaru ji není možné použít.

Každou řídící jednotku je potřeba naprogramovat. Pokud ovšem programátor udělá chybu, není zcela jasné, jak chybu odhalit. Vložit nový neověřený program do nových vozidel určených k prodeji je nemožné. Nikdo není neomylný a každý dělá chyby. Nejjednodušší kontrolou je nechat si program zkontrolovat od více lidí. To ale neodstraňuje faktor neomylnosti člověka. Stále zde zůstává kontrola na lidech, o kterých jsme si řekli, že není možné, aby byly neomylní.

Jednou z možností je nechat systém, v našem případě řídicí jednotku, otestovat na všechny možné stavy. Pokud se jedná o primitivní řízení, je tato metoda velice účinná v nalezení případné chyby v programu. Ovšem pokud již chceme otestovat komplikovanější funkcionalitu jiné, složitější jednotky, tato metoda se stává neefektivní a v některých případech ani není možné tuto metodu pro testování použít. Problémem je veliká složitost a pak pokud netestujeme kombinační, ale sekvenční obvod, je testování všeobecně velice náročné. V případě kombinačních obvodů je velice jednoduché sestavit kombinaci vstupních proměnných a sledovat jak se řídicí jednotka zachová. Pokud se ovšem nejedná o kombinační, ale sekvenční obvod, což ve většině případů bývá, složitost testování enormě stoupá. Již není jisté, že při jedné kombinaci vstupů bude výstupní odezva pokaždé stejná. Závisí zde i na předchozích stavech, ve kterých se řídicí jednotka nacházela dříve.

3.2 Typy řídících jednotek

Vývoj v oblasti elektroniky zaznamenal v několika málo letech veliký pokrok a v automobilovém průmyslu nachází stále větší uplatnění. Dnešní automobily obsahují nejednu řídicí jednotku a jejich počet stále roste. Tento trend však není jen u vozidel z vyšší cenové kategorie, ale i u levných vozů. Většina automobilek se nezaměřuje pouze na jednu cenovou třídu, ale snaží se obsáhnout vetší skupinu zákazníků. Vývoj není levná záležitost, a pokud je navržena funkční jednotka, je nutné pro ni nalézt uplatnění, aby mohla být masově nasazena a tím došlo ke snížení výrobních nákladů. Proto je obvyklé, aby stejná řídicí jednotka byla použita jak ve velice drahém, tak i naopak velmi levném voze. Jde hlavně o co nejširší možné využití.

Každá jednotka v automobilu plní jinou funkci a je připojena k různým snímačům a akčním členům. Například řídicí jednotka motoru bude mít určitě jiný počet připojených snímačů, než řídicí jednotka zadních dveří. Pro tuto práci bylo nutné zvážit rozsah vstupních a výstupních bran universální jednotky a podle toho vybrat konkrétní řídicí jednotku, kterou by

bylo možné s vytvořeným hardwarem realizovat.

Jako příklad, na kterém by bylo možné demonstrovat použití emulátoru, byla zvolena jednotka řídící dveře spolujezdce. Splňuje zvolené požadavky a bude ji tedy na univerzálním emulátoru možné realizovat. Abychom pro tuto jednotku mohli vytvořit řídící model, musíme přesně definovat její chování.

3.3 Funkční chování řídicí jednotky dveří spolujezdce

- Dveře z vnitřní strany nesou ovládací tlačítka pro stahování okna, jež má pět poloh:
  - výchozí – základní poloha
  - zavřít automaticky
  - otevřít automaticky
  - zavřít manuálně
  - otevřít manuálně

- Z vnější strany auta je vložka zámku, do níž lze vložit klíč. Od středové základní je možné klíčem otočit na obě strany. Celkem tři polohy:
  - výchozí – základní poloha
  - zamknout
  - odemknout

- Zámek dveří, okno a zpětné zrcátko jsou ovládány pomocí servomechanizmů.

- Ve zpětném zrcátku je umístěn blinkr.

- Zpětné zrcátko je vyhřívané.

- Řídicí jednotka dveří spolujezdce je připojena na společnou sběrnici CAN. Přes tuto sběrnici komunikuje s ostatními řídícími jednotkami:
  - Centrální řídící jednotka
  - Řídící jednotka dveří řidiče
  - Centrální řídící jednotka komfortní elektroniky

4 Matlab-Simulink

Simulink je prostředí pro multioborové simulace a modelově založené návrhy pro dynamické a embedded systémy. Zatímco u Matlabu je stále nejdůležitější příkazový řádek, Simulink poskytuje interaktivní grafické prostředí a soubory blokových knihoven, které umožňují navrhovat, simulovat, implementovat a testovat různé časově závislé systémy, včetně kontroly nebo zpracování signálu.

Umožňuje také spouštět části simulačních schémat na základě výsledku logické podmínky. Tyto spouštěné a povolované subsystémy umožňují použití programu v náročných simulačních experimentech. Hierarchická struktura modelů umožňuje koncipovat i velmi složité systémy do přehledné soustavy subsystémů prakticky bez omezení počtu bloků. Simulink, stejně jako Matlab, dovoluje připojovat funkce napsané uživateli v jazyce C. Vynikající grafické možnosti Simulinku je možné přímo využít k tvorbě dokumentace. Mezi neocenitelné vlastnosti Simulinku patří nezávislost uživatelského rozhraní na počítačové platformě. Přenositelnost modelů a schémat mezi různými typy počítačů umožňuje vytvářet rozsáhlé modely, které vyžadují spolupráci většího kolektivu řešitelů na různých úrovních. [8]

4.1 Bloky a modely Simulinku

S tímto prostředím můžete rychle vytvářet modely a udržovat podrobné blokové schéma systému, pomocí komplexní sady předdefinovaných bloků. Simulink poskytuje nástroje pro hierarchické modelování, správu dat a přizpůsobení subsystému, takže je snadné vytvořit stručný a přesný popis systému, bez ohledu na jeho složitost.

Simulink obsahuje rozsáhlou knihovnu funkcí běžně používaných při modelování systémů. Mezi ně patří:

- spojité a diskrétní dynamické bloky
- bloky pro tvorbu algoritmů
- strukturální bloky

Otevřená architektura Simulinku vedla ke vzniku knihoven bloků, nazývaných blocksety, které rozšiřují základní knihovnu bloků Simulinku a umožňují použití programu v příslušných vědních a technických oborech, jako je například letectví, komunikace, a další aplikace. Knihovny je možné rozšiřovat i o vlastní bloky, vytvořené uživatelem.

4.2 Embedded Coder

Pro rychlé generování čitelného a kompaktního kódu C a C++ je možné použít rozšíření **Embedded Coder™**, součást Simulinku. Ve starších verzích Simulinku se tato část nazývala **Real-Time Workshop**. Lze jej použít na sériově vyráběné procesory a cíleně je programovat metodou Rapid Prototyping. Embedded Coder umožňuje konfigurace a pokročilou optimalizaci a kontrolu nad generovaným kódem funkcí, souborů a dat. Tyto konfigurace zefektivňují proces výroby a zásadním způsobem zrychlují návrh systému řízeného jednočipovým procesorem. Pomocí tohoto rozšíření lze integrovat kód pro **Code Composer Studio™** od Texas Instruments, nebo pro **VisualDSP++®** od Analog Devices™ a dalších vývojových prostředí různých výrobců.

Embedded Coder umožňuje automatické generování samostatného kódu v jazyce C a C++, který je určen pro nasazení na embedded procesory, prototypové vývojové desky cílových platforem a mikroprocesory používané v hromadné produkci. Generovaný zdrojový kód je přenositelný a snadno čitelný. Embedded Coder podporuje podmnožinu vlastností základního jazyka Matlab, včetně konstrukcí pro řízení chodu programu, výpočetních funkcí a maticových operací. Tyto optimalizace zlepšují efektivitu kódu a usnadňují integraci se stávajícím kódem, datovými typy a parametry nastavení používanými ve výrobě. Z generovaného zdrojového kódu je možné nechat vytvořit spustitelný soubor, statickou nebo dynamickou knihovnu, nebo jen začlenit vývojová prostředí cílových platforem sloužící k vytvoření spustitelného programu pro zvolený embedded procesor.

4.2.1 Nastavení Embedded Coder

Abychom mohli z vytvořeného modelu vygenerovat přeložitelný kód pro konkrétní procesor, je nutné správně nastavit pracovní prostředí modelu a parametry pro generování kódu. Než ale můžeme prostředí nastavit, musíme nejprve založit nový model a spustit Simulink. To je možné několika způsoby. Jednou z možností jak Simulink spustit je do příkazového okna Matlabu napsat `Simulink`. Další možností je kliknutím na ikonu:

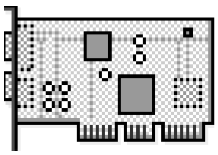
Obrázek 4.1: Ikona pro spuštění knihovny Simulinku

Posledním místem, kde můžeme Simulink spustit, je *Start → Simulink → Library Browser*. Takto jsme otevřeli knihovnu s bloky, které budeme používat.

Podobným způsobem otevřeme nový model. Opět existuje několik jednoduchých způsobů. Jednou možností je v Matlabu kliknout na *File → New → Model*. Pokud již máme otevřenou knihovnu Simulinku, lze nový model založit stejnou cestou v okně Simulinku.

Máme založený nový model, otevřenou knihovnu Simulinku a je potřeba jej správně nastavit. To lze provést příkazy v *Command Window* nebo spouštěcím skriptem. Uvedené metody, zvláště ta druhá, jsou vhodné, pokud bude toto nastavení sloužit pro několik modelů. Uživatel napíše skript, který je možné použít vždy při nastavování a konfiguraci nového modelu, určeného pro stejný hardware. Nevýhodou této metody je, že uživatel musí znát parametry, které má nastavit. Pokud by nebyly nastaveny všechny potřebné záležitosti podle konkrétního zadání, zůstávají tyto parametry v defaultní hodnotě, a pokaždé to nemusí být žádoucí. Tato metoda je určena spíše pokročilým uživatelům.

Další možnost, jak nastavit modelovací prostředí, je mnohem jednoduší a bezpečnější pro správné nastavení. V podstatě se jedná o automatické nastavení za pomoci průvodce, kde se nastaví základní parametry dle použitého procesoru. Provede se to tak, že do okna modelu se z knihovny Simulinku přetáhne blok *Target Preferences*, který nalezneme v *Embedded Coder → Embedded Targets*.



*Obr. 4.2: Blok **Target Preferences** pro automatické nastavení Modelu.*

Tento blok nastaví informace o cílovém procesoru a o propojení Simulinku s CCStudiem. Jeho nastavení je možné zkontrolovat, případně manuálně nastavit a provedeme jej v konfiguračním okně (*Tools → Code Generation → Options…*). V záložce Solver nastavujeme, jaký typ simulace má být prováděn.

Jelikož je model určen pro procesor, který pracuje s diskrétním časem, měly by být vložením bloku *Target Preferences* parametry nastaveny podle následujících bodů:

Solver

Solver options

- Type: Fixed-step
- Solver: discreate (no continuous states)
- Fixed-step size: auto

Simulation time

- Start time: 0.0
- Stop time: 10.0

Code Generation

Target selection

- System target file: idelink_ert.tlc[3]
- Language : C

Pro simulaci jsou důležité parametry nastavené v záložce *Solver*. Je možné změnit časový krok výpočtu *Fixed-step size* z automaticky nastavovaného (*auto*) podle bloků v modelu na námi zvolenou konkrétní hodnotu (v sekundách). Ta však musí být minimálně stejná, jako je nejmenší vzorkovací čas nastavený u jednotlivých bloků umístěných v modelu. Parametry *Start time* a *Stop time* určují časový rozsah řešení simulace. Pro kontinuální simulaci je možné zadat *inf* do *Stop time*. Simulaci je potom nutné manuálně zastavit.

Soubor *idelink_ert.tlc* udává informace o konkrétním procesoru a propojení Simulinku s dalším programem. Je tedy používán pro generování kódu.

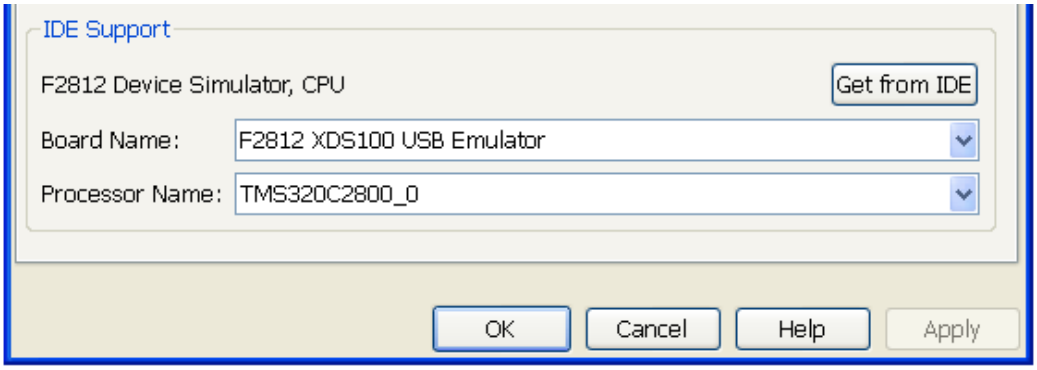
4.2.2 Nastavení bloku *Target Preferences*

Po vložení tohoto bloku do nového modelu se zobrazí dialogové okno. V tomto okně se nastavuje konkrétní procesor, který bude použit, a jeho programovací prostředí. Jelikož budeme používat procesor od Texas Instruments, vybereme *IDE/Tool Chain:* ***Texas Instruments Code Composer Studio***. Pro náš konkrétní procesor *TMS320F2812* můžeme

[3] Podle verze Simulinku je také možné použít *ccslink_ert.tlc*.

volit mezi dvěma tipy *Boardů*. Liší se pouze místem, kam se m program nahrát, zda do paměti RAM nebo do paměti Flash. Pokud jsme ve fázi vývoje a testujeme vytvořené programy, zvolíme možnost ***SD F2812 eZdsp***, která nahraje program pouze do paměti RAM. V této paměti je program, pouze pokud je procesor napájen. Druhá možnost je program uložit do paměti Flash (***SD F2812 eZdsp (boot from flash)***), ze které se program po spuštění nahraje do paměti RAM. Tuto možnost volíme až v případě, že je program finální a již nepředpokládáme jeho změnu v konkrétním procesoru. Paměť Flash je omezena pouze na 100 přepisů. Ve vývoji a testování, bychom počet zápisů do této paměti rychle vyčerpali.

Takto jsme nastavili základní parametry pro použití modelu ke konkrétnímu procesoru. Detailní nastavení hardwaru se provádí v dialogovém okně, které se zobrazí při poklepání na *Target Preferenses* vložený do modelu. Zde je možné nastavit frekvenci procesoru, mapu paměti a další parametry. V záložce *Board* musíme vybrat rozhraní, kterým bude nahráván kódu do procesoru. To provedeme podle následujícího obrázku.



Obr. 4.3: Nastavení rozhraní pro programování.

Nastavení paměťového prostoru a sekce ukládání jednotlivých částí programu ponecháme a v defaultním stavu. Poslední záložka *Peripherals* slouží pro nastavení periférií procesoru. Jejich detailním nastavením se budeme zabývat následovně.

ADC

Časování AD převodníku je odvozeno od systémových vysokorychlostních hodin HSPCLK. ADCLKPS udává 4-bitovou hodnotu předěličky, která dělí frekvenci hodin procesoru. Dále je možné tento signál dělit dvěma, pokud je CPS nastaven na 1. Poslední parametr ACQ_PS určuje šířku vzorkovacího okna S&H jednotky.

eCAN

Nastavení se týká bloků *C281x eCAN Receive* a *C281x eCAN Transmit*.

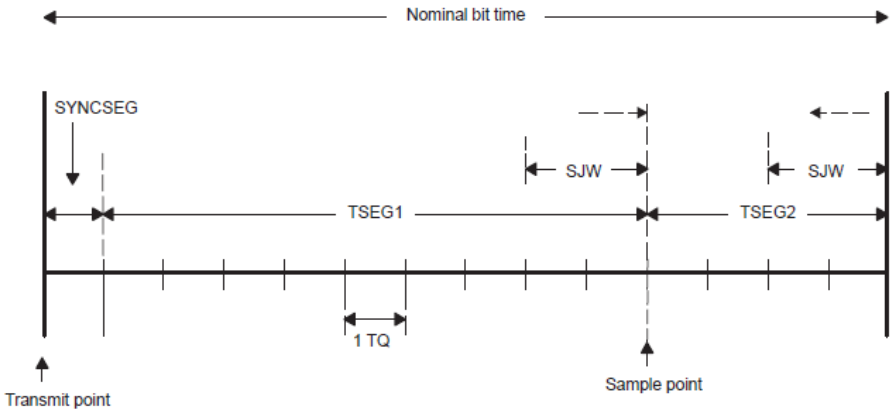
Přenosová rychlost *Bitrate* v bitech za sekundu se vypočítá dle vztahu *4.1*,

$$Bitrate = \frac{SYSCLKOUT}{BRP \cdot BitTime} \qquad 4.1$$

kde *BitTime* je počet časových kvant *TQ* na jeden bit a je definovaný dle vztahu *4.2*. SYSCLKOUT je frekvence CAN modulu, v našem případě 150 MHz. BRP je *Baud Rate Prescaler*.

$$BitTime = TSEG1 + TSEG2 + 1 \qquad 4.2$$

Každý bit je vzorkován a místo kde k tomu dochází je určeno právě TSEG1 a TSEG2.



Obr. 4.4: Časování vzorkování v modulu CAN. [10]

Pokud při vzorkování dojde k desynchronizaci, tak k opětovnému synchronizování lze použít SJW *Synchronization Jump Width*, určující o kolik *TQ* se může skočit. Posledním parametrem je SAM. Zde můžeme vybrat mezi jednoduchým a trojitým vzorkováním.

Následují dvě zaškrtávací okénka, z nichž prvním (*Enhanced CAN mode*) aktivujeme rozšiřující možnosti CAN řadiče. To umožňuje ke stávajícím 16 mailboxům přidat dalších 16. Zaškrtnutím druhého okénka (*Self test mode*) zajistíme samostatnou funkci procesoru při testování, kdy není potřeba připojovat další zařízení, které by odesílalo *acknowledge* – informaci o úspěšném přijetí zprávy jinou stanicí.

SPI

Vedle základního nastavení, zda se jedná o *Master* nebo *Slave*, je zde několik dalších parametrů komunikace SPI. Jedním z nich je přenosová rychlost *SPIBautrate*, která se vypočte:

$$SPIBitrate = \frac{LSPCLK}{SPIBRR+1} \qquad 4.3$$

kde LSPCLK jsou pomalejší systémové hodiny, jejichž kmitočet je HSPCLK/4 tedy 37,5 MHz. SPIBRR je *SPI Baud Rate Register*, jehož hodnota se nastavuje v rozsahu 3 až 127 hodnotou v poli *Baud rate factor*. Je také nutné nastavit délku vysílaných a přijímaných znaků parametrem *Data bits* od 1 do 16 bitů. Dále je možné nastavit vzorkování na vzestupnou nebo sestupnou hranu (*Clock polarity*) a fázi hodinového signálu vůči datovému (*Clock phase*) a to buď bez posunu, nebo se zpožděním půl cyklu. Poslední parametr *Suspension mode* určuje chování při ladění programu. Pokud program narazí na *breakpoint*, zvolený režim pozastavení určuje, jak se má v programu dále pokračovat. Dostupné volby jsou *Hard_abort*, *Soft_abort* a *Free_run*. *Hard_abort* zastaví program okamžitě. *Soft_abort* se zastaví, když je příjem nebo vysílání sekvence kompletní. *Free_run* pokračuje v běhu, bez ohledu na *breakpoint*.

Následují dvě zaškrtávací políčka. První zajistí interní připojení pinu Tx v procesoru na pin Rx a můžeme přenášet data z výstupního portu na vstupní pro kontrolu integrity přenosu. Druhé aktivuje vyvolání přerušení pro konkrétní úrovně zaplnění FIFO pamětí.

SCI

Použitý procesor disponuje dvěma sériovými rozhraními SCI a jejich nastavení je zcela shodné.

Enable loopback – Zaškrtnutím tohoto políčka se zapíná selftest pro diagnostické účely podobně jako u komunikace SPI.

Suspension mode – Umožňuje vybrat různé druhy zastavení při ladění programu, podobně jako u SPI.

Number of stop bits – Udává, kolik stopbitů bude mít přenášený znak.

Parity mode – Zde se volí lichá nebo sudá parita pro základní kontrolu.

Character lenit bits – Počet bitů přijímaného a vysílaného znaku.

Baud Rate – Přenosová rychlost vysílaných a přijímaných dat.

Communication mode – Na výběr je ze dvou položek: *Raw_data* a *Protocol*. *Raw_data* je neformátovaný režim a data jsou vysílána okamžitě, ať je přijímací zařízení připraveno nebo ne. V tomto režimu je možné, že přijímací strana může ”promeškat” data. Nejsou-li však kritická, je vhodné použít tento režim, protože nebudou kvůli přenosu dat blokované žádné procesy. V komunikačním módu *Protocol* dochází mezi procesorem a hostem k handshaku. Vysílací strana vyšle zprávu SND, když je připravena na vyslání dat. V případě, že je přijímací strana připravená přijímat, vysílá zprávu RDY . Po odeslání dat se ještě navíc posílá checksum.

Blocking mode – Když je toto políčko vybrané, systém čeká na příchozí data. Ve druhém případě systém periodicky kontroluje FIFO, zda-li nepřišla nová data. Když jsou nová data přítomna, systém je předá k dalšímu zpracování. Nejsou-li k dispozici, drží se na výstupu poslední hodnota.

Data byte order – Označuje pořadí bytů (*LittleEndian a BigEndian*)

Data swap width – Na výběr je z 8 bitového nebo 16 bitového odkládacího prostoru. V případě volby pořadí bytů *BigEndian* musí být nastaven na *8_bits*.

Watchdog

Tato funkce umožňuje zotavení procesoru v případě poruchy. Zaškrtnutím *Enable watchdog* povolíme watchdog timer modul.

Parametr *Counter clock* nastavuje periodu watchdog timeru vzhledem k OSCCLK/512. OSCCLK je hodinový signál odpovídající kmitočtu externího oscilátoru, čili 30MHz.

Timer period in seconds pouze zobrazuje periodu časovače v sekundách. Hodnota je automaticky přepočítávána podle parametru *Counter clock*.

Time out event určuje chování po načítání watchdog čítače.

- *Chip reset* – generuje signál WDRST, který resetuje procesor.
- *Raise WD Interrupt* – generuje watchdog interrupt sigál WDINT a neprovede se reset procesoru. Tento signál lze využít k přechodu procesoru z IDLE nebo STANDBY módu.

GPIO

Vybrané brány GPIO (A, B, D, E) jsou vybaveny tzv. kvalifikačním obvodem. Tento obvod funguje jako jednoduchý filtr nežádoucích zákmitů na vstupech. Aby došlo ke změně výstupu kvalifikačního obvodu, musí na jeho vstupu být signál o stejné hodnotě po dobu nejméně šesti taktů hodinového signálu SYSCLKOUT.

Flash_loader

Flash_loader můžeme použít pro:

- automatické nahrání kódu do paměti Flash po přeložení
- ruční mazání, programování nebo ověřování konkrétních sektorů paměti Flash

K používání této funkce je nutné stáhnout a nainstalovat plugin *TI Flash API*.

PLL

Zde se nastavuje hodnota *PLL Control Registeru*. Výchozí hodnota je nastavena, aby CPU hodiny (CLKIN) běželi na maximální frekvenci. Předpokládáme, že externí oscilátor na desce (OSCCLK) je podle doporučení dodavatele konkrétního procesoru.

Změnu nastavení PLL děláme jestliže:

- chceme změnit frekvenci procesoru
- externí oscilátor se liší od výrobcem doporučené hodnoty

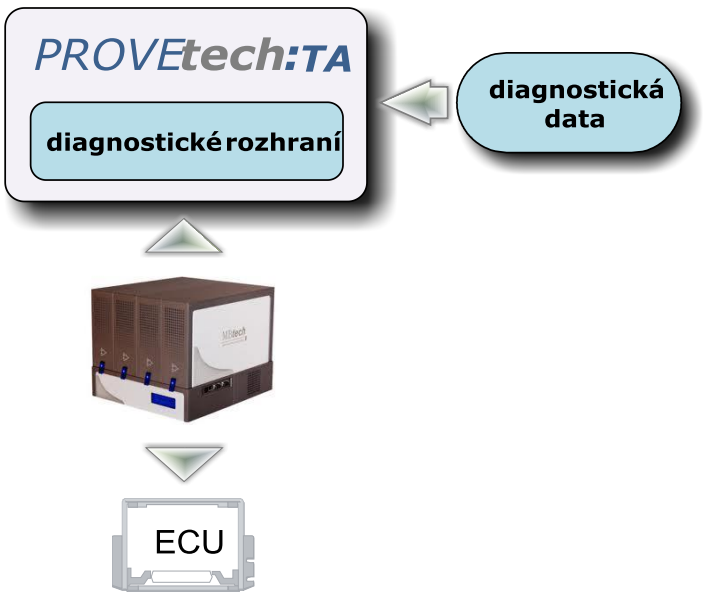
4.2.3 Používání knihovny Embedded Coder

Tato knihovna obsahuje bloky pro procesory, které je možné programovat pomocí Simulinku. Nalezneme zde několik typů procesorů různých výrobců a jejich rodiny. V každé knihovně konkrétního procesoru, nebo její řady, se nacházejí bloky starající se o hardwarové prvky uvnitř procesoru, jako jsou GPIO, ADC, CAN, PWM a další. Bloky používané v této práci s procesorem *TMS320F2812* nalezneme v *Embedded Coder → Embedded Targets → Processors → Texas Instruments C2000 → C281x*.

5 PROVEtech:TA

Tento software je používán mnoha výrobci automobilů a dodavateli svých testovacích systémů HiL[4]. Nabízí ideální základ pro spouštění automatizovaných testů v reálném čase. Struktura tohoto programu je hardwarově nezávislá a umožňuje přístup více uživatelů, což ho činí optimální pro použití v rámci pracovní skupiny. Integruje využívání databáze s cílem zajistit konzistentní zpracování dat a řízení.

PROVEtech:TA nabízí přátelské uživatelské rozhraní a především sjednocuje většinu modulů užívaných v testování a vývoji. Pod jedním grafickým rozhraním je tak možné spravovat testovací knihovny, tvořit série zkoušek a testovací sestavy, včetně správy výsledků z testů. Všechny tyto procesy spadající do Test Managementu mají v programu pro svou funkci odpovídající knihovny. Díky široké paletě ovládací prvků, které je možné umístit na pracovní plochu, usnadňuje přístup k signálům potřebných při testování a tvorbě testovacích scénářů. Uživatel si může nastavit, například pozici plynového pedálu a poté měřit výslednou rychlost vozidla. Mezi další moduly, které z tohoto nástroje činí kompletní software pro návrh, vykonávání a řízení testů, jsou moduly pro návrh diagnostiky a modul simulace poruch.



Obr. 5.1: Blokové uspořádání testovací sestavy řízené PROVEtech:TA

[4] HiL (Hardware in the Loop) – testovaní ECU ve smyčce

5.1 Workpage

Pracovní plocha (*Workpage*) je důležitá část programu *PROVEtech:TA* v níž je možné užívat širokou škálu zobrazovacích a ovládacích prvků pro stimulování a zobrazování signálů. Uživatel si může vybrat ze seznamu všech dostupných signálů a nastavit jim požadované hodnoty za pomoci různých řídicích bloků, například posuvnými prvky nebo tlačítky. Těchto pracovních ploch je možné vytvořit více a samostatně je pojmenovat. Přepínání mezi nimi je pomocí záložek v dolní části okna.

Mimo hlavní pracovní plochu je možné použít prostor nalevo od *Workpage*, tzv. palubní desku (*cocpit area*). Použití je naprosto stejné, ovšem oproti *Workpage* je vždy viditelná.

Rozložení ovládacích a zobrazovacích prvků na pracovní ploše tak i na palubní desce je možné uložit každé samostatně. Je tak možné vytvářet různé varianty pracovní plochy a vždy nahrávat tu, která bude uživateli nejvíce vyhovovat pro konkrétní projekt.

Do těchto oken jsou prvky odkazující na jednotlivé signály přidávány z okna *Signal Selection*, které je možné vyvolat *Extras* → *Signals*, nebo stiskem příslušné ikony v hlavní liště programu. V tomto okně jsou veškeré signály, se kterými je možné pracovat.

5.2 Nastavení Matlab-Simulink pro generování souborů do PROVEtech:RE

Aby bylo možné vložit modely do *PROVEtech:TA* je nutné provést nastavení Simulinku, který je součástí Matlabu. Do některých jeho složek se musí vložit specifické soubory vytvořené po instalaci programu *PROVEtech:RE*. Celkové nastavení a změny v Matlabu musí být provedeny podle následujícího postupu[5]:

1. Nainstalujeme *PROVEtech:RE*. Vytvoří se složka *RE* s potřebnými soubory.
2. Zkopírujeme soubory *grt_dll.def, grt_dll_main.c* a *grt_lcc_dll.tmf* [6] ze složky *RE\Goodies\Model\Matlab\Step 1* do složky *<MatlabRoot>\rtw\c\grt*.
3. Provedeme Patch Matlabu pomocí *MatlabPatch.exe*, který nalezneme ve složce *RE\Goodies\Model\Matlab\Step 2*. Tím budou nahrazeny tři *.tlc* soubory novými a původní budou přejmenovány s příponou *.bac*. Jedná se o soubory *commonhdr_blkiolib.tlc, commonprmlib.tlc* a *commonhdrlib.tlc* ve složce *<MatlabRoot>\rtw\c\tlc\mw*.

[5] Pro verze Matlabu R14 SP1+SP2+SP3, 2006a+b, 2007a+b, 2008a+b, 2009a+b, 2010a+b, 2011a
[6] Soubor *grt_lcc_dll.tmf* odpovídající nainstalované verzi Matlabu

4. V otevřeném modelu je nutné nastavit konfigurační parametry pro správné provedení simulací a generování kódu. To provedeme v konfiguračním okně (*Tools → Code Generation → Options…*).

Solver

- Stop time: inf
- Type: Fixed-step
- Solver: discreate (no continuous states)
- Fixed-step size: 0.01

Data Import/Export

- Load from workspace: vše vypnuto
- Save to workspace: vše vypnuto

Optimization

- Block reduction: vypnuto
- Signal storage reuse: vypnuto

Code Generation

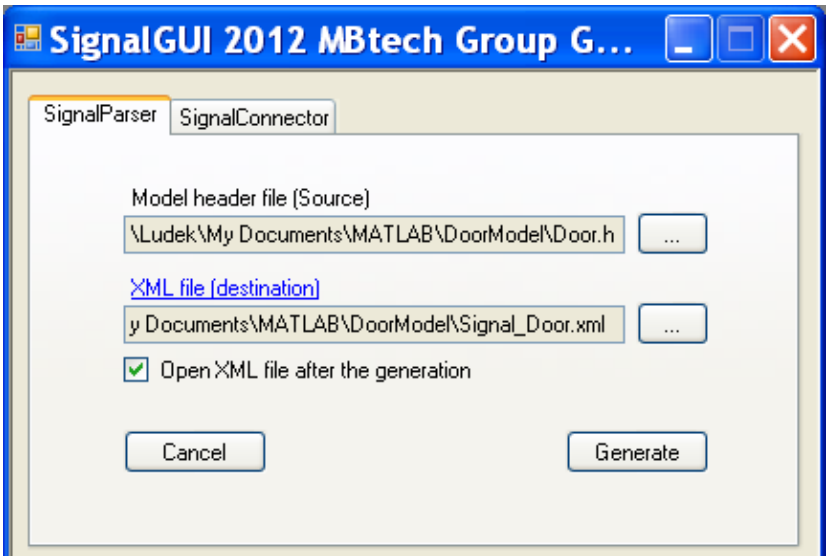
- System target file: grt.tlc (Generic Real-Time Target)
- Template makefile: grt_lcc_dll.tmf

5. Ostatní parametry nastavení modelu ponecháme v defaultním stavu.
6. Kombinací kláves Ctrl + B spustíme generování kódu a kompilaci dll souborů.

Ve stejné složce, ve které je uložený model, se vygenerovaly potřebné soubory. Dále je ovšem nutné vytvořit soubor *.xml*, potřebný pro *PROVEtech:TA*. Soubor vytvoříme pomocí programu *SignalGUI.exe*, který nalezneme v nainstalované složce *RE\Goodies\Model\SignalGUI*.

Prostředí programu je vidět na *Obr. 5.2*. Program má dvě záložky. Je nutné se přepnout do okna *SignalParser*. Zde musíme do kolonky *Model header file (Source)* zadat cestu k *.h* souboru, který byl vygenerován Matlabem. Tento vygenerovaný soubor je uložen ve stejné složce jako model a také má stejné jméno, ovšem s příponou *.h*.

V další kolonce je nutné zadat cestu, kam se má soubor *.xml* vytvořit. Je dobré zadat stejnou složku, ve které je uložený model.



Obr. 5.2: Program pro vygenerování .xml souboru.

Následně je již možné vygenerovat soubor stisknutím tlačítka *Generate*. V námi zvolené cílové složce se vytvořil *.xml* soubor s přednastaveným názvem *Signal_JmenoModelu.xml*.

Soubory vygenerované Matlabem potřebné pro *PROVEtech:TA* jsou:

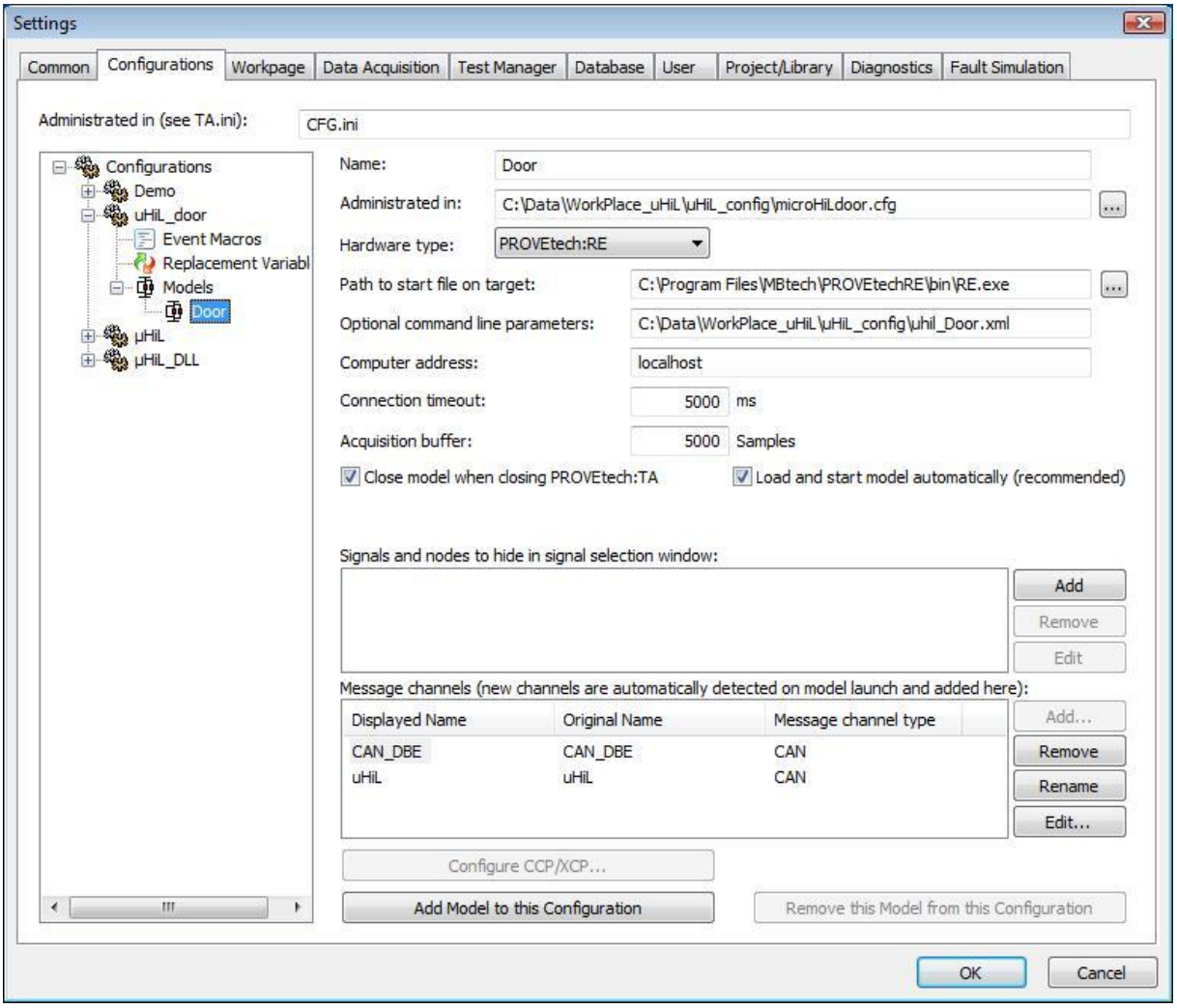
- *JmenoModelu.dll*
- *Signal_JmenoModelu.xml*

5.3 Konfigurace modelu (*PROVEtech:RE*) v *PROVEtech:TA*

Pro další práci je vhodné si vytvořit pracovní složku, ve které budeme mít uložené soubory, se kterými budeme pracovat. Do této složky zkopírujeme již vytvořené soubory *.dll* a *.xml* vygenerované z modelu v Simulinku a konfigurační soubor *microHil.xml*.

Po spuštění programu *PROVEtech:TA* musíme provést konfiguraci pro náš konkrétně vkládaný model. Toto nastavení je rozdílné pro jednotlivé typy platforem, na kterých bude simulační model běžet. *PROVEtech:TA* nabízí několik cílových Hardwarů se kterými je schopný pracovat. Podle konkrétně vybraného se liší i konfigurace a převážně soubory, potřebné ke spuštění modelu.

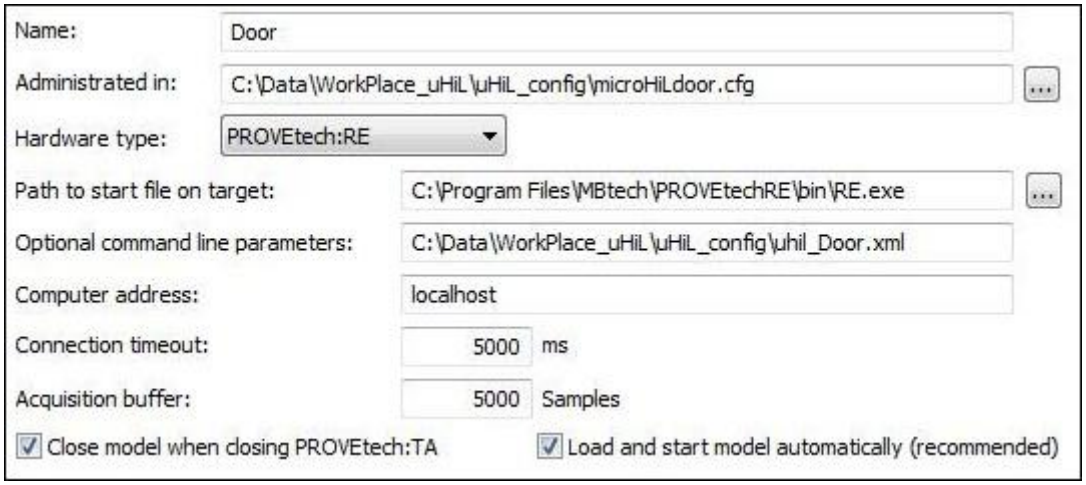
Aby bylo možné pracovat s *PROVEtech:µHiL* na platformě *PROVEtech:RE*, je důležité pracovní prostředí správně nastavit. Na *Obr. 2.1* je ukázka konfigurace. Toto okno vyvoláme *Extras → Configurations*. Důležitá záložka, která nás bude zajímat je *Configurations*, kam budeme vkládat vygenerované soubory. Konfiguraci je možné provést několika způsoby. Vytvořením nové, kde je nutné nastavit veškeré cesty k souborům a jednotlivé prvky, nebo duplikováním již některé vytvořené.



Obr. 5.3: Konfigurace modelu

Pokud je již nějaký model se stejnou konfigurací založený, postačí tuto konfiguraci duplikovat. Tím se vytvoří nová konfigurace, které musíme zadat nové jméno (*Name:*) aby nedošlo k přepsání původní konfigurace. Celé nastavení konfigurace je uloženo v souboru *.cfg*, k němuž je cesta zadaná v kolonce *Administrated in:*. Zadáme cestu do naší vytvořené pracovní složky a přidělíme konfiguračnímu souboru název.

Následující nastavení je určeno pro *PROVEtech:RE*. Pokud bychom chtěli konfiguraci na jinou platformu, provedeme nastavení podle manuálu, který nalezneme ve složce *MBtech\PROVEtechTA\doc* pod názvem *Manual.pdf* v kapitole *3.4.3.2.3 Models*.



Obr. 5.4: Konfigurační nastavení pro PROVEtech:RE

Do nové konfigurace, která byla vytvořena, je nutné zadat správné hodnoty a pracovní soubory. Konkrétní nastavení je vidět na *Obr. 5.4*.

Name:

Zde jsme již při vytvoření nové konfigurace nebo při duplikaci již existujícího zadali jméno naší konfigurace.

Administrated in:

Každá konfigurace je definována speciálním souborem *.cfg*, který je uložen ve složce uvedené v této kolonce.

Hardware type:

Pro základní testovací systém *PROVEtech:RE* je v této kolonce volba *PROVEtech:RE*.

Patch to start file on target:

Zde je uvedena cesta ke spustitelnému souboru *RE.exe*.

Optional command line parameters:

Zde je uvedená cesta ke konfiguračnímu souboru *.xml*. Jedná se o spustitelný soubor, v němž jsou všechny používané signály a také cesty k souborům *.xml* a *.dll*,vygenerované z modelu v Simulinku.

Pokud tento soubor není zadán, *PROVEtech:RE* využívá soubor *Konfig.xml*, uložený ve stejné složce, jako je *RE.exe*.

Jestliže cesta ke konfiguračnímu souboru obsahuje prázdné mezery, je nutné celý obsah této kolonky vložit do uvozovacích znaků (" ").

Computer address:

Název počítače je "localhost", nebo *127.0.0.1* v případě, že se *PROVEtech:RE* nachází na stejném počítači jako je spuštěný *PROVEtech:TA*.

Connection timeout:

Po spuštění se program pokouší připojit k modelu po dobu uvedenou v této kolonce (v milisekundách). V případě, že připojení trvá po startu velice dlouho, je nutné tento čas ještě zvýšit.

Acquisition buffer:

Do vyrovnávací paměti počítače se ukládají vzorky, dokud model běží. Malá hodnota může způsobit ztrátu dat přetečením vyrovnávací paměti. Naopak veliká hodnota zabírá velikou část operační paměti počítače.

6 Model ECU

Celý model ECU je rozdělen na tři části, které jsou samozřejmě součástí jednoho společného modelu. S tím souvisí i sekvence zpracovávání dat a celá logická funkce emulátoru. Pokud se podíváme na základní princip činnosti emulátoru, můžeme být sekvence zpracování následující:

1. **Čtení jednotlivých vstupů.**
    - analogové vstupy (ADC)
    - digitální vstupy
    - přijaté CAN zprávy
2. **Zpracování a vyhodnocení přijatých signálů.**
3. **Nastavení jednotlivých výstupů.**
    - analogové výstupy (DAC, PWM)
    - digitální výstupy
    - odeslané CAN zprávy

Na začátku většiny řízení je potřeba mít informace o řízeném systému. V našem případě se jedná o informace z analogových a digitálních vstupů a informace z přijaté ze sběrnice CAN, kde analogovými vstupy rozumíme přímé měření odporu k zemi a napětí do 3 V.

Je patrné, že pokud nebudeme mít informaci o chování systému a jeho aktuálním stavu, v našem případě okolí ECU, není možné tento systém jakýmkoli způsobem řídit. Proto je vždy důležité, aby přijatá data a snímané signály byly nejen správné, ale byly včas. Pokud máme data, která jsou podrobná a přesná, ale nejsou včas, není opět možné efektivně systém řídit.

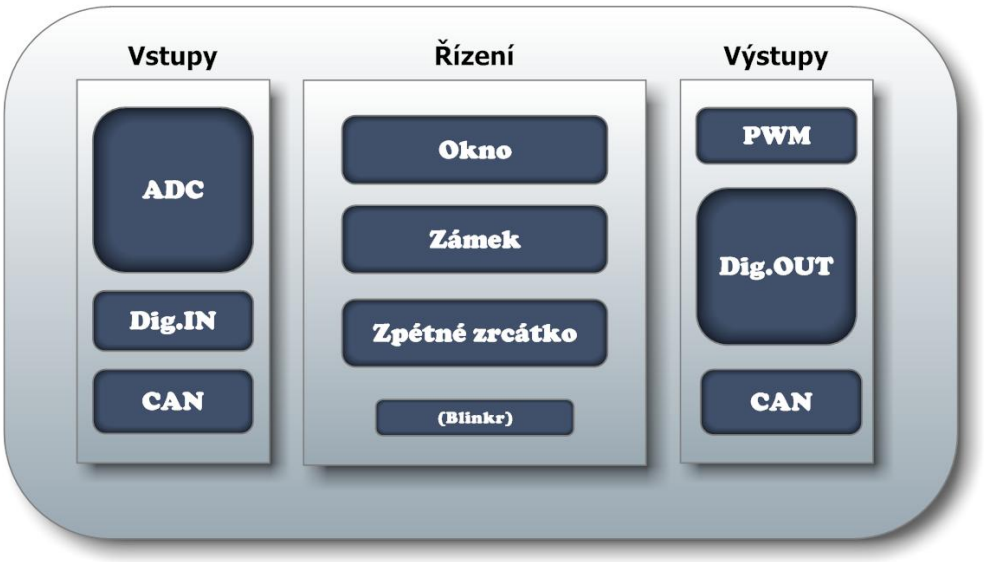
Pokud již máme změřené vstupní signály a přijatá data z CAN zprávy, je nutné signály vyhodnotit a vhodně na ně reagovat. O zpracování se stará logika řízení, která je jádrem celého řídícího modelu. Popis jakým způsobem se chová blok řízení na jednotlivé podněty, signály a stavy v CAN zprávě je popsáno v kapitole 6.2.

Nakonec jsou jednotlivé výstupy, kterými emulátor disponuje, nastaveny podle rozhodnutí řídící logiky. Nejčastěji používaný výstup, pro zásah do systému, je digitální a to výkonový nebo logický (CMOS). Vedle těchto dvou, emulátor disponuje výkonovým

výstupem PWM a výkonovým logickým výstupem. Tato jednoduchá struktura posloupnosti řídících kroků může být použita jako vhodné rozdělení řídícího modelu na tři základní bloky:

- vstupy do jednotky
- výstupy z jednotky
- hlavní řídící smyčka

Výše naznačené rozdělení je jen pro jednodušší vysvětlení jednotlivých částí modelu a umožňuje to lepší orientaci ve struktuře celého modelu. Grafické upořádání, tohoto strukturálního rozdělení je na následujícím obrázku.



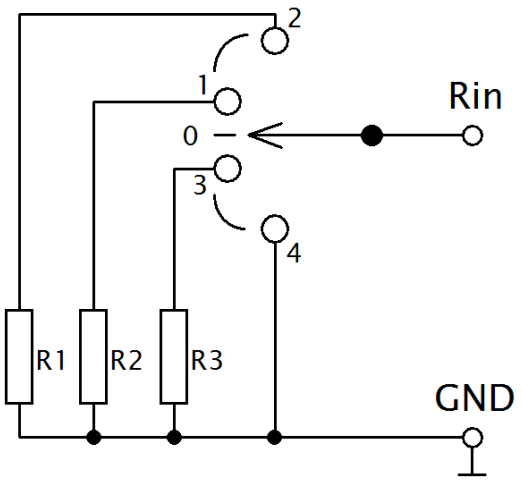
Obr. 6.1: Struktura modelu ECU

6.1 Vstupní bloky

Vstupní rozhranní, kterými emulátor disponuje, se nacházejí na levé straně obrázku. V této části modelu jsou veškeré bloky starající se o sledování okolí ECU. Je tím myšleno snímání nejrůznějších tlačítek, čidel a senzorů, jako i příjem zpráv po sběrnici CAN. U každého vstupu je možné nastavit periodu čtení, nebo-li jak často má být vstupní brána aktualizována. U každého bloku je možné tuto periodu nastavit rozdílně, ovšem pro snazší zpracování je vhodné u všech vstupních bloků periodu nastavit shodně. Není to nutné, ale doporučení zajistí nejen shodnou aktuálnost dat, ale i jednoduší práci s těmito signály.

6.1.1 Analogové vstupy

Největší zastoupení ve vstupním bloku má ADC vstup procesoru. Jelikož je snaha výrobců automobilu výrobní náklady co nejvíce minimalizovat, snaží se používat levné senzory nebo dokonce fyzikální veličiny měřit bezsenzoricky. Úsporám se nevyhnuly ani propojovací vodiče mezi snímači a řídicí jednotkou. Cílem je přivést data od několika snímačů stejné funkce do řídicí jednotky přes co nejmenší počet vodičů, v ideálním případě přes jeden. Pro snímání poloh ovládacích tlačítek je dnes téměř bez výhrady použito tzv. odporové kódování. Příklad jeho vnitřního zapojení zobrazuje *Obr. 6.2.*



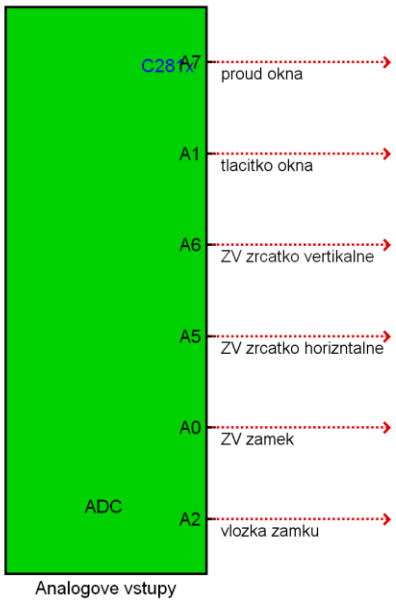
Obr. 6.2: Zapojeni ovládacího tlačítka pomocí odporového kódování

Jedná se o konkrétní zapojení tlačítka pro elektrické stahování okna v autě. Toto tlačítko má celkem 5 poloh, z nichž střední poloha je výchozí (aktuálně zobrazený stav – poloha 0). Jak je z obrázku patrné, postačí pro jedno tlačítko jediný vodič který je připojen k analogovému vstupu řídicí jednotky. K detekci konkrétní polohy tlačítka, snímá aktuální odpor mezi svorkami *Rin* a *GND*. Podle naměřeného odporu vyhodnotí pozici. Ve výchozí poloze je odpor k zemi nekonečný a naopak v poloze 4 je nulový. Ostatní polohy jsou určeny podle známého odporu, který je naměřen. V poloze 1 je připojen pouze rezistor *R2* a v poloze 3 pouze rezistor *R3*. To jsou první polohy při lehkém stisku, ale pokud je toto ovládací tlačítko stisknuto až do krajní polohy, dojde k připnutí dalšího rezistoru k již stávajícímu. V krajní poloze 2 je tak odpor daný paralelním spojením rezistorů *R1* a *R2*. Pro rozlišení jednotlivých poloh jsou odpory jednotlivých rezistorů odlišné. Jejich hodnoty musí být zvoleny tak, aby nemohlo dojít k záměně poloh a řídicí jednotka tak mohla jednoznačně určit

konkrétní polohu. Hodnoty rezistorů *R2* a *R3* nesmějí být shodné a taktéž v poloze 2 nesmí být výsledná hodnota odporu, paralelní kombinace rezistorů *R1* a *R2*, stejná jako hodnota odporu *R3*. Konkrétní hodnoty jsou u různých výrobců různé a není tak striktně určeno jakou mají mít přesnou velikost.

Vedle odporového kódování tlačítek je vstup, schopný měřit odpor, využit také ke zjištění polohy například zpětného zrcátka. Natočení je určeno polohou potenciometru, který je k zrcátku mechanicky připojen.

V modelu jsou tyto vstupy měřeny pomocí bloku *C281x ADC*, kde je využito celkem 6 analogových vstupů.

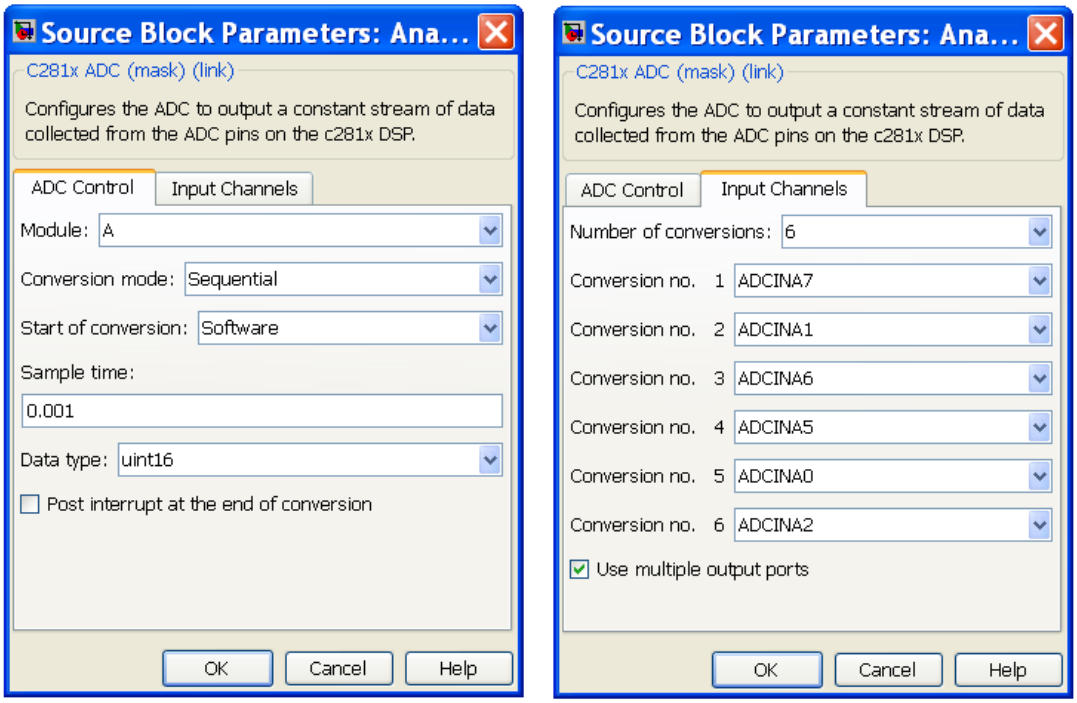


*Obr. 6.3: Blok **C281x ADC** – analogové vstupy*

Procesor použitý na emulátoru je vybaven dvěma kanály AD převodníku A a B, z nichž každý je schopen sekvenčně měřit 8 vstupů. Hardwarově je však zapojen pouze kanál A a jsme tak omezeni na 8 vstupů, které je možné měřit pouze sekvenčně. Simultánní měření, jenž je schopné měřit ve dvou kanálech ve stejný okamžik nelze v tomto zapojení využít, jelikož druhý kanál B, který je pro tento mód nutný, není zapojen. Emulátor využívá zapojených 8 vstupů kanálu A k měření analogových veličin, konkrétně k měření analogového napětí v rozsahu 0 – 5 V a k přímému měření odporu k zemi do velikosti 10 kΩ. Jakým způsobem jsou jednotlivé vstupy zapojeny, uvádí *Tab. 2.*5. Je v ní také uvedeno, k jakému účelu jsou konkrétní vstupy použity a jaký mají význam v Modelu ECU. Celkem je využito 6 vstupů,

kterými je měřena poloha potenciometrů pro snímání polohy nebo stav ovládacích tlačítek.

Nastavení bloku *C281x ADC* je patrné dle následujícího obrázku.



*Obr. 6.4 : Nastavení bloku **C281x ADC** pro měření analogových veličin.*

Jak bylo řečeno výše, je použit pouze kanál A v sekvenčním režimu, tzn. 6 využitých vstupů emulátoru je měřeno postupně v pořadí, které naznačuje *Obr. 6.4* vpravo. Zvolená perioda čtení (*Sample time*) nastavená na *0,001 s* (*1 kHz*) je zcela postačující, neboť jsou snímány pouze časově nekritické signály.

6.1.2 Digitální vstupy

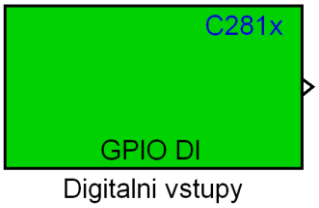
Vedle analogových vstupů je emulátor schopen snímat signály logické a to ve dvou napěťových úrovních 12 V a 5 V. V obou případech se jedná o klasické GPIO vstupy procesoru, pouze jsou na brány procesoru připojeny rozdílné budiče a převodníky. Je tak zajištěno snímání logických signálů s úrovní 5 V a s napěťovou úrovní 12 V, která je pro použití v automobilové logice přednostněji použita u nejrůznějších snímačů, kontaktů a čidel.

Jak již bylo zmíněno v předchozí kapitole, dnešním trendem je velkému počtu vodičů se vyhnout. Pokud bychom například chtěli připojit řídící jednotku k ovládacímu tlačítku okna, dostaneme několikabitovou informaci o stavu, kterou je nutné přenést. První variantou je použití vícežilového vodiče. Pro tlačítko se 4 polohami a jednou výchozí, je poté potřeba

minimálně 4 vodičů. Tento počet lze snížit na jeden, ovšem již není možné použít jednoduché ovládací tlačítko ale inteligentní. Inteligentním tlačítkem, či snímačem rozumíme takové, které datovou informaci sestaví do sériové zprávy a odešle řídicí jednotka po jednom vodiči. Tím je sice minimalizován počet použitých vodičů, ovšem enormním způsobem prodraží čidlo nebo snímač.

V automobilu nalezneme ale i čidla generující jednobitovou informaci, pro něž je jeden vodič postačující. Jsou to například dveřní nebo koncový kontakt. V takovém případě je připojení k digitálnímu vstupu řídicí jednotky zcela oprávněné.

Ke snímání těchto digitálních signálů je v modelu použit blok *C281x GPIO Digital Input* (*Obr. 6.*5). Signály s napěťovou úrovní 5 V jsou v emulátoru připojeny na bránu procesoru B a signály s napětím 12 V na bránu A. Pro každou bránu musí být v modelu vložen samostatný blok.



*Obr. 6.5: Blok **C281x GPIO Digital Input** – digitální vstup*

V modelu řídící ECU jsou využity pouze tři vstupy s napěťovou úrovní 12 V. Jak uvádí *Tab. 2.*4, jsou tyto vstupy emulátoru využity pouze k detekování krajních poloh okna a k detekci dovřených dveří. Jelikož se jedná pouze o 12 V vstupy, postačí pouze jediný blok *C281x GPIO Digital Input*, který je nastaven:

IO Port:	**GPIOA**
Bit:	**12, 14, 15**
Sample time:	**0,001**
Data type:	**boolean**

Použitá brána a její jednotlivé bity jsou určeny hardwarovým zapojením vstupů. Jelikož se jedná pouze o logický signál je zvolen datový typ *boolean*. Stejně jako v případě analogového vstupu, i zde není čas vzorkování nijak kritický a proto jsou vstupy opakovaně čteny

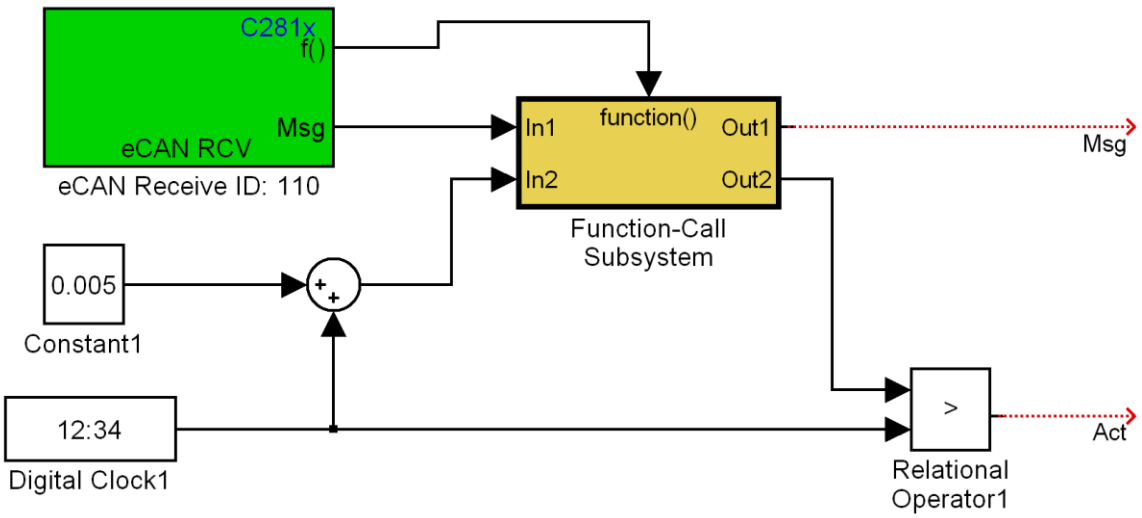
s periodou *0,001 s*. Perioda je úmyslně zvolena shodně u obou vstupů, analogového i digitálního, pro jednodušší práci s těmito signály a nebylo tak nutné vkládat bloky pro srovnání časování mezi jednotlivými signály. Máme tak zajištěnou stejnou periodu obou vstupů.

6.1.3 CAN zprávy - Receive

Posledním typem dat a řídících signálů vstupujících do ECU jsou přijímané zprávy ze sběrnice CAN. Tyto signály bychom mohli řadit na jedno z prvních míst stupnice důležitosti sledování okolí ECU a snímání vstupních podnětů potřebných pro řízení celého systému.

ŘJ dveří spolujezdce je jednou z mnoha v automobilu a každá z nich vysílá více než jeden typ zpráv s rozdílným určením, významem a důležitostí na jednu společnou sběrnici CAN. Je tedy nutné tyto zprávy sledovat a rozhodovat jaké mají být přijaty ke zpracování a které mají být ignorovány.

Příjem zpráv ze sběrnice CAN se v Simulinku pro konkrétně použitý typ procesoru realizuje blokem *C281x eCAN Receive*. Na následujícím obrázku je zobrazen tento blok zapojený s dalšími prvky související s příjmem zpráv.



*Obr. 6.6: Blok **C281x eCAN Receive** – zapojení bloku pro příjem zpráv*

Základní sestava pro příjem zprávy se skládá z bloků *eCAN Receive* a *Function-Call Subsystem*. Toto složení je jednoduché a snadno nastavitelné v případě, že není potřeba

přijímat velké množství zpráv s odlišnými ID.

Blok *eCAN Receive* umožňuje příjem zprávy s jedinečnou adresou. To plyne z nastavení jeho parametrů:

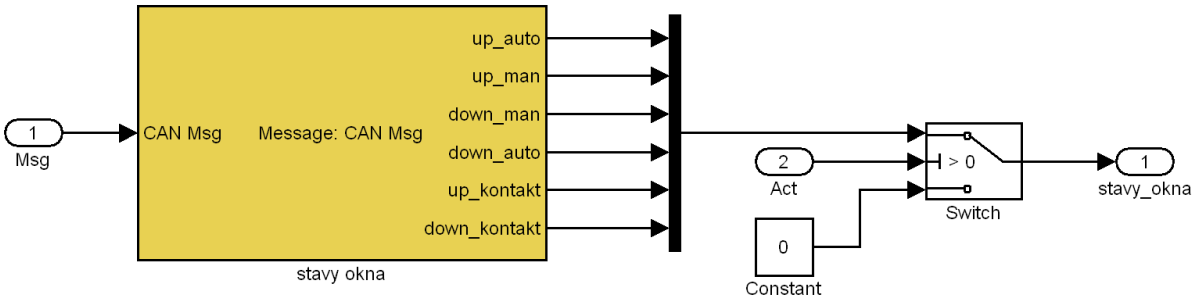
Mailbox number:	**0**
Message identifier:	**110**
Message type:	**Standard (11-bit identifier)**
Sample time:	**0,001**
Data type:	**CAN_MESSAGE_TYPE**
Initial output:	**0**

V procesoru je pro příjem i odesílání zpráv použita RAM paměť na 16 zpráv ve standardním módu a 32 zpráv v rozšířeném módu. Každý blok pro příjem *eCAN Receive* nebo pro vysílání *eCAN Transmit* zpráv musí mít parametr *Mailbox number* nastaven na jinou hodnotu. Výše uvedený identifikátor je v hexadecimálním tvaru ale jako parametr musí být zadán v decimálním v rozsahu 0 – 2047 pro standardní 11 bitový identifikátor ID. Pro zápis v binárním nebo hexadecimálním kódu je nutné užít předpis `bin2dec(' ')` nebo `hex2dec(' ')`. Vzorkovací frekvence zpráv je nastavena na 1 ms a datový výstup je ve speciálním formátu pro CAN zprávy.

Takto vypadá nastavení parametrů jedno bloku pro příjem zpráv s konkrétním identifikátorem ID 110 hex. Tento blok je uzpůsoben k příjmu jedné specifické ID adresy, a pokud je potřeba přijímat zprávy s jinou ID, musí být pro ni vložen další blok *eCAN Receive*. V případě ECU pro řízení dveří spolujezdce budou přijímány zprávy se třemi rozdílnými ID a do modelu je tedy nutné vložit tyto sestavy bloků tři a v každé uvést konkrétní identifikátor ID. Pokud by bylo nutné přijímat větší počet zpráv s rozdílnými ID, tak použití vždy jednoho bloku k jedné zprávě s autentickou ID je značně nepohodlné. Jednoduší je použit obecný blok, ve kterém se neuvádí jedna specifická ID adresa ale její rozsah. Tento blok se v knihovně Simuliku vyskytuje pod názvem *CAN Receive*. Zde je již možné nastavit filtr pro rozsah přijímaných zpráv s určitým rozmezím ID. Byl tím odstraněn problém se složitostí nastavení filtru pro příjem více zpráv s rozdílnými ID. Jelikož se tento blok nenachází v knihovně *Texas Instruments C2000 – C281x,* není tak jednoduché jej použít bez dalších bloků pro nastavení registru pro CAN v mikroprocesoru, který je použit v emulátoru.

V základní sestavě pro příjem zprávy složené z bloků *eCAN Receive* a *Function-Call Subsystem* zanechá na svém výstupu vždy poslední přijatou zprávu a ta je se zvolenou periodou čtena. Je to způsobeno tím, že každá zpráva je ukládána do paměti RAM v mikroprocesoru a pokud není přepsána novou, zůstává v paměti uložena poslední přijatá zpráva. Pro určení aktuálnosti zprávy je tato základní sestava doplněna o další bloky generující signál vypovídající o stáří přijaté zprávy (*Obr. 6.*6). Frekvence přijímaných zpráv je 1 kHz a každou 1 ms dojde k příjmu zprávy ze sběrnice CAN, pokud bude vysílající řídicí jednotka a přijímající řídicí jednotka v pořádku a nedojde ke ztrátě zprávy. Pokud ale komunikace selže a nedojde během 5 ms k příjmu nové zprávy, je signál Act nastaven do nuly a je tím signalizováno, že zpráva uložená v RAM paměti mikroprocesoru je starší než 5 ms. Při správném běh systému jsou zprávy přijímány každou 1 ms a signál Act je stále ve vysokém stavu. Ze základní sestavy pro příjem zpráv ze sběrnice CAN vedou dva signály Msg a Act.

Dále je přijatou zprávu nutné rozbalit a dekódovat. To má na starosti blok *CAN Unpack* (*Obr. 6.*7).



Obr. 6.7: Blok ***CAN Unpack.***

Přijatá zpráva je dekódována a jsou z ní vytaženy signály, které potřebujeme. Specifikace je umožněna volbou *manually specifik sinals* v nastavení výstupních dat tohoto bloku. Parametr *Identifier* je nastaven na hodnotu -1, která nerozlišuje ID identifikátor zpráv a dochází k dekódování všech, které přicházejí na vstup *CAN Msg*. Je také možné vložit tabulku definující výstupní signály, specifikující jejich umístění ve zprávě CAN a jejich velikost. Je možné definovat jedno i vícebitové signály a každému přiřadit název.

Signály je možné následně zavádět do logických bloků a nejrůznějším způsobem je

zpracovávat. Jelikož některé bloky modelu nereagují pouze na hrany signálu ale též na aktivní stav, je nutné zvolit, po jaké době mají být data nulována pro jejich neplatnost. V předchozím popisovaném bloku jsme generovali signál *Act*, který byl po 5 ms od poslední přijaté zprávy nastaven na logickou 0. Zde je tento signál přiveden ta vstup členu, který dekódovaný signál z bloku *CAN Unpack* odepne a výstup nastaví do definovaného stavu, většinou postačí nastavit nulový signál.

Spojením zapojení, která zobrazuje *Obr. 6.*6 a *Obr. 6.*7, dostaneme kompletní segment použitelný pro získání signálu z přijaté zprávy ze sběrnice CAN.

6.2 Hlavní Řídící logika

Data a vstupní signály, které byly přijaty vstupními obvody, přecházejí do další části řídícího modelu ECU, kterou je možné nazvat jako Řídící logika. Jedná se především o zapojení kombinačních a sekvenčních obvodů a pro sestavení této logiky je nutné znát chování systému a předpis, kterým má být řízen.

Okolím emulátoru ve funkci řídicí jednotky dveří spolujezdce rozumíme nejen části, které nalezneme na samotných dveřích, ale i ostatní řídicí jednotky připojené na společnou datovou sběrnici CAN. Tato sestava veškerých elektronických jednotek, snímacích členů, elektrických senzorů a nejrůznějších tlačítek tvoří celé okolí emulátoru a přitom nezohledňujeme, zda je konkrétní prvek připojen jako vstup či výstup. Jednoznačnými prvky připojených na výstup emulátoru, které je potřebné ovládat, jsou především akční členy:

- okno
- zpětné zrcátko
- zámek dveří
- blinkr

Jedná se o jednotlivé součásti dveří spolujezdce a jejich společnou vlastností je to, že jsou připojeny k výstupům řídicí jednotky. Všechny výše uvedené součásti, které je nutné ovládat, musí být připojeny k výkonovým výstupům.

Součásti umístěné přímo na dveřích spolujezdce a připojené na vstupy emulátoru slouží především ke snímání polohy motoricky ovládaných prvků nebo ke zjištění polohy ovládacích tlačítek a spínačů. Od nich je poloha nebo stav vyhodnocen a na základě konkrétního nastavení a funkčního uzpůsobení jsou tyto data odeslána ve zprávě CAN nebo vyhodnocena

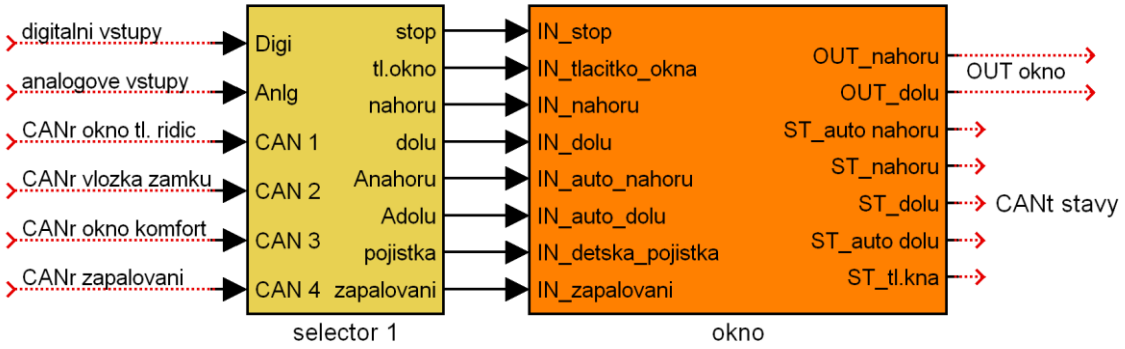
řídící logikou a použita k akci na konkrétním akčním členu. Do skupiny prvků připojených na vstup emulátoru patří:

- snímače polohy okna
- snímače polohy zpětného zrcátka
- snímač polohy zámku dveří
- vložka zámku
- tlačítko pro stahování okna

V celé oblasti součástí dveří připojených k emulátoru nalezneme čtyři základní funkční celky, které je nutné ovládat, sledovat nebo řídit. Jsou to okno, zpětné zrcátko, zámek a vložka zámku. V následujících kapitolách budou tyto jednotlivé části podrobněji popsány.

6.2.1 Okno

Základní struktura části modelu vykonávající funkce kolem okna je na následujícím obrázku:

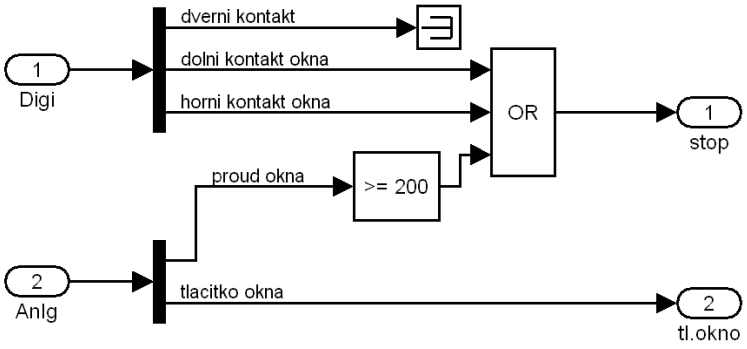


Obr. 6.8: Blok pro ovládání okna.

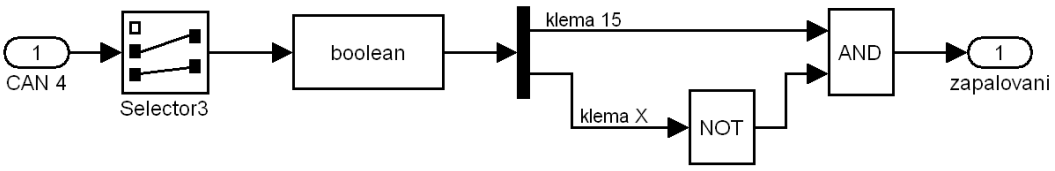
Blok *selector 1* se stará o výběr signálů ze vstupních vektorů a do bloku *okno* zavádí pouze ty, které jsou pro tento blok potřebné. Řízení okna je možné buď přímo tlačítkem umístěným ve dveřích spolujezdce, tlačítkem od řidiče nebo příkazem od *Centrální řídící jednotky komfortní elektroniky*. Signál pro ovládání pomocí tlačítka umístěného na dveřích spolujezdce vstupuje do bloku *selector 1* jako analogový signál (na *Obr. 6.8* je vstup označen jako *Anlg*). Další dva způsoby, jak zavřít nebo otevřít okno je možné dekódováním CAN zpráv od *Řídící jednotky dveří řidiče* a od *Centrální řídící jednotky komfortní elektroniky*.

V prvním případě je v CAN zprávě informace o stavu tlačítka umístěného ve dveřích řidiče určeného k ovládání pravého předního okna, tedy okna spolujezdce. Do bloku *selector 1* tato zpráva vstupuje pod názvem *CAN 1*. Druhý případ ovládání okna přes sběrnici CAN je od Komfortní jednotky, která zajistí úplné otevření nebo zavření okna. Od ostatních způsobů toto ovládání neumí hýbat oknem manuálně. Do bloku *selector 1* tato zpráva vstupuje pod názvem *CAN 3*.

K zastavení pohybu okna v případě, že je již zcela zavřené nebo otevřené slouží koncové kontakty označené v bloku *selector 1* jako *Digi*. Vygenerování signálu *stop* a tím tedy k zastavení okna nedochází jen při aktivních koncových kontaktech ale také při překročení povoleného proudu motoru ovládacího okno. Tyto tři způsoby, jak zastavit okno jsou naznačeny na následujícím obrázku.



Obr. 6.9: ***Selector*** *1, část – selekce z digitálních a analogových vstupů pro okno.*



Obr. 6.10: ***Selector*** *1, část – selekce signálu zapalování.*

Blok *selector 1* obsahuje ještě další část (*Obr. 6.*10), která se stará o reakci okna při různých polohách klíčku zapalování. Ten má celkem 4 stavy:

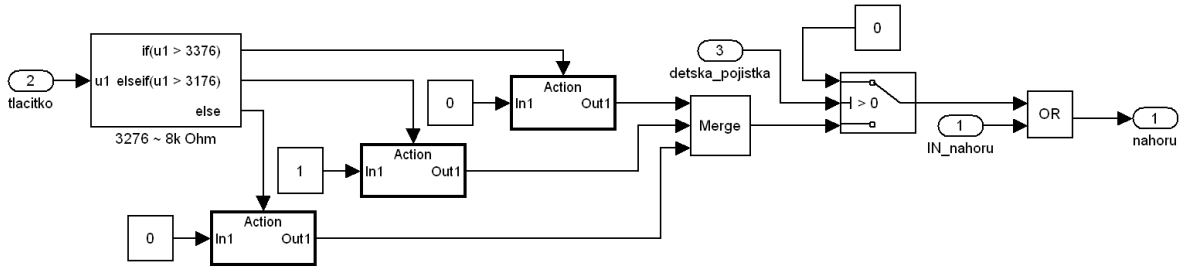
- rozepnutý kontakt *klema S* - klíč je vyjmutý
- sepnutý kontakt *klema S* - klíč v poloze 0*
- sepnutý kontakt *klema 15* - klíč v poloze 1
- sepnutý kontakt *klema X* - klíč v poloze 2

Reakce většiny obvodů na tyto signály je pouze zamezení funkce při neaktivním signálu *klema 15* nebo při aktivním signálu *klema X*.

Následující blok *okno* již připravené signály jen zpracuje a vyhodnotí. Jediný akční člen, který tento blok ovládá je zavírání a otevírání okna. K tomuto účelu má blok výstupy *OUT_nahoru* a *OUT_dolu*. Ostatní výstupy bloku okno jsou určené pouze pro určení stavu a informace o prováděné akci odesílané na sběrnici CAN. Uvnitř bloku *okno* nalezneme další blok pod názvem *Subsystem*. Tento blok generuje povely pro ovládání okna:

- Nahoru automaticky
- Nahoru manuálně
- Dolu manuálně
- Dolu automaticky

Vygenerování jednotlivých signálů je možné z přijaté zprávy nebo z výstupní hodnoty odporu ovládacího tlačítka ve dveřích spolujezdce, které je odporově kódované. Oba způsoby je možné vyvodit z následujícího obrázku.

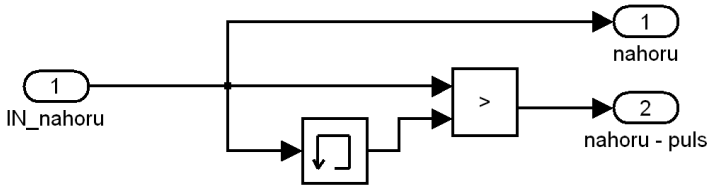


Obr. 6.11: Dekódování hodnoty odporového tlačítka.

Hodnota z ADC převodníku je přivedena na vstup 2. Pokud se hodnota z ADC převodníku nachází v udaném rozmezí, je na výstupu logická 1. Tento výstup je také možné zablokovat signálem *detska_pojistka*. Výsledný signál se sčítá se signálem na vstupu 1 přijatým ze zprávy CAN.

Takováto struktura se v bloku *Subsystem* nachází čtyřikrát pro všechny 4 polohy ovládacího tlačítka okna. Vždy se jen liší rozmezím hodnot v podmínkovém bloku *if* pro které generuje aktivní signál. Každý tento signál je z bloku *Subsystem* vyveden ve dvou variantách a to jako aktivní signál typu *latch* nebo jako *impuls* při vzestupné hraně.

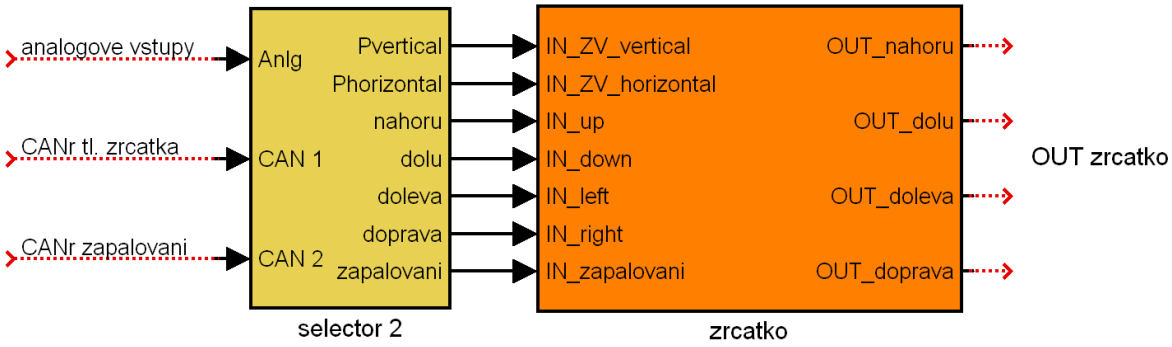
Způsob jakým je tento signál generován zobrazuje *Obr.* 6.12. Celkové vnitřní zapojení bloku *okno/Subsystem* nalezneme v příloze na straně 82.



Obr. 6.12: Detektor náběžné hrany.

6.2.2 Zpětné zrcátko

Další částí modelu je sestava ovládající zpětné zrcátko na pravé straně automobilu (*Obr. 6.*13).



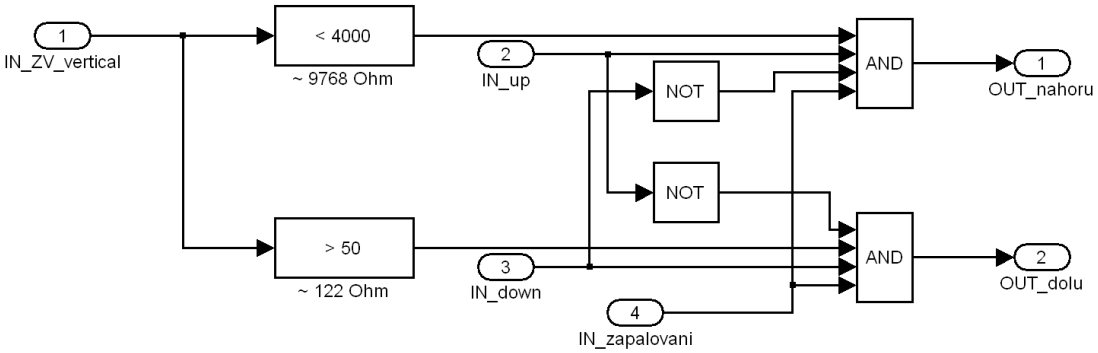
Obr. 6.13: Blok pro ovládání zpětného zrcátka.

Oproti části řešící problematiku okna, je zde ovládání zrcátka možné pouze z dat dekódovaných v přijaté zprávě CAN od *Řídící jednotky dveří řidiče*. Tento signál (*CANr tl. zrcatka*) opět nejprve vstupuje do bloku *selector 2*, kde jsou konkrétní použitelná data ze

zprávy vybrána. Dalším signálem vstupujícím do *selector 2* je informace o poloze zrcátka v obou rovinách, vertikální a horizontální. Tyto hodnoty jsou získané z potenciometrů mechanicky spojených se zrcátkem a vyhodnocené AD převodníkem. Pro vystavení funkce ovládání zrcátka při vypnutém klíčku zapalování nebo při startování, je navíc do tohoto bloku zaveden signál *CANr zapalovani* informující o těchto stavech.

Vnitřní struktura je obdobná jako je v bloku *selector 1* části modelu pro ovládání okna. Uvnitř bloku *zrcatko* je struktura, starající se o bezpečnost motorů v krajních polohách a zamezení aktivního signálu na obou výstupech jedné roviny najednou.

Krajní polohy zrcátka jsou detekovány ze zpětných vazeb potenciometrů s rozsahem 10 kΩ. Pokud je potenciometr na jednom z krajů svého rozsahu, dostáváme hodnoty blízké 10 kΩ, nebo naopak hodnoty blízké nulovému odporu. Minimální vzdálenost potenciometru od koncové polohy a tím krajová poloha pohybu zrcátka byla zvolena v místě kdy, potenciometr má mezi běžcem a krajem odporové dráhy odpor roven hodnotě 122 Ω. Při klasickém zapojení je jedna krajní poloha signalizována pokud je odpor menší než 122 Ω a druhá pokud je odpor větší než 9768 Ω. Realizace pro jednu rovinu je na obrázku níže.

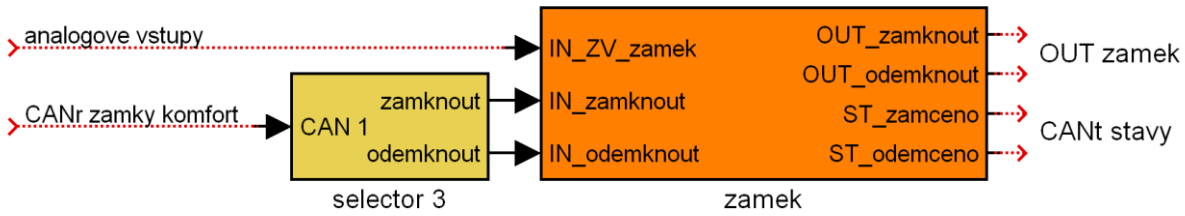


Obr. 6.14: Ovládání zpětného zrcátka ve vertikálním směru.

Na tomto obrázku je také vidět, jakým způsobem je realizováno zamezení vybuzení obou směrů pro motor zároveň. Stav je zabezpečen dvěma členy *NOT*, a pokud je například aktivní signál *IN_up*, pro signál *IN_down* je dolní člen *AND* nepropustný a není tak možné aktivovat výstup *OUT_dolu* dříve než vstup *IN_up* není nastaven do nízké úrovně. Takto je zapojeno ovládání motoru zpětného zrcátka ve vertikálním směru, ale zcela obdobný obvod je použitý pro motor ovládající zrcátko v horizontálním směru.

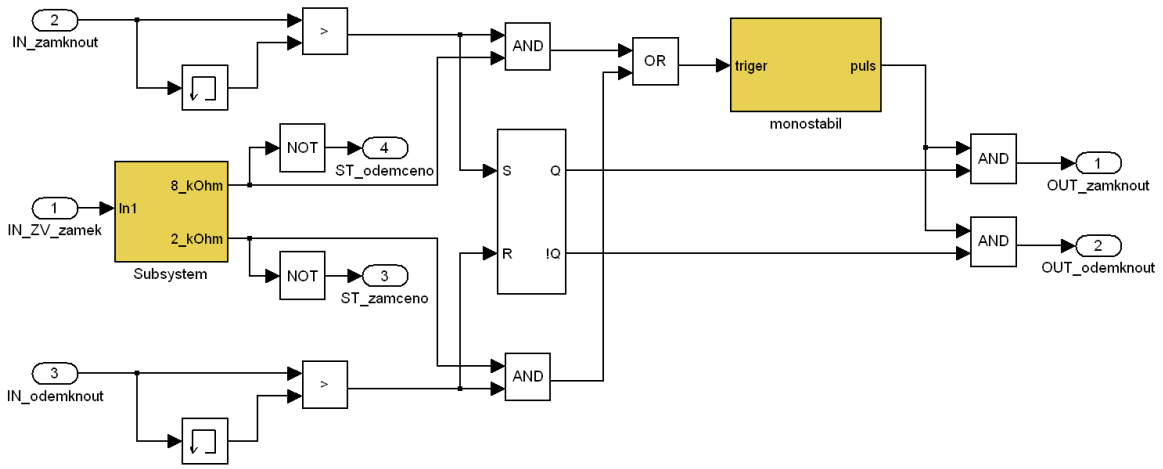
6.2.3 Zámek

Pro ovládání zámku dveří se stará následující sestava bloků (*Obr. 6.*15). Zde do bloku *selector 3* nevstupuje signál z AD převodníku, protože signál není vektorem a není tak nutné separovat jednotlivé kanály. Jediná analogová hodnota, která je snímána AD převodníkem, je signál o poloze zámku dveří, pro sledování polohy – stavu zámku (zamčeno nebo odemčeno).



Obr. 6.15: Blok pro ovládání zámku.

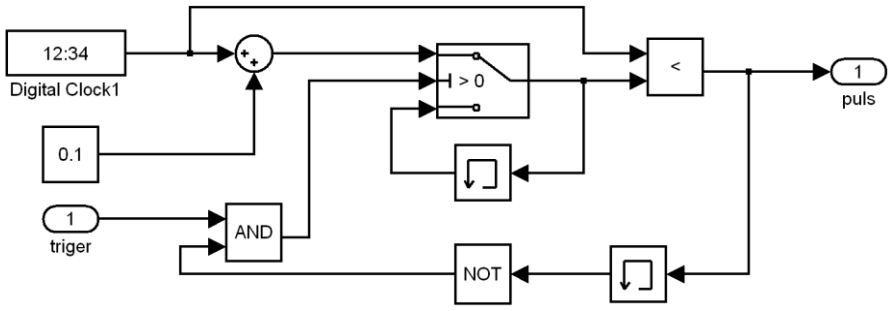
Veškeré příkazy k pohybu všech zámků v automobilu vydává pouze *Centrální řídící jednotky komfortní elektroniky*. Jednotka od ostatních obdrží pouze požadavek na uzamčení konkrétního zámku, ale samotný příkaz k této akci provádí pouze ona. Proto jsou data dekódována pouze ze zprávy od této řídící jednotky. Do bloku *selector 3* vstupuje pouze signál *CANr zamky komfort* ze kterých je dekódována informace týkající se zámku dveří spolujezdce.



Obr. 6.16: Vnitřní zapojení bloku ***zamek****.*

Jde o dva samostatné signály zamknout a odemknout, vstupující do následujícího bloku *zamek*, společně s informací o poloze zámku z AD převodníku. V nitřní zapojení můžeme vidět na *Obr. 6.*16. Zde jsou ze signálů pro zamčení a odemčení detekovány vzestupné hrany pomocí sestavy dvou bloků (paměť a komparátor). První vstup komparátoru je připojen přímo na vstupní signál a druhý je připojen přes paměť, která signál pouze zpozdí o jeden pracovní cyklus procesoru. Zvětší-li signál svoji hodnotu, je na prvním vstupu komparátoru větší hodnota a generuje signál True. Pokud se v dalším kroku procesoru vstupní hodnota signálu nemění, je na obou vstupech komparátoru shodná hodnota a je generován signál False. Dojde tak vygenerování pulzu, který nastaví nebo vynuluje RS klopný obvod, podle toho zda přijde signál k zamčení nebo odemčení.

K tomu, aby mohlo dojít k zamčení zámku, se nesmí nacházet v této poloze a naopak pokud je zámek odemčený, nesmí dojít k příkazu odemknout. Pro tuto kontrolu blok *Subsystem* generuje dvojici signálů ze zpětné vazby zámku. Oba signály jsou ve stavu True, pokud se zámek nenachází ani v jednom ze stavů zamčeno nebo odemčeno a je tedy někde na tomto rozmezí stavů, není tedy ani zamčeno ani odemčeno. Při správné funkci, kdy je zámek v jednom ze stavů, je příslušný výstupní signál bloku *Subsystem* ve stavu False. Tyto stavy, v jaké poloze se nachází zámek, jsou předány do bloku pro odesílání na sběrnici CAN pomocí signálů *ST_zamceno* a *ST_odemceno*.



Obr. 6.17: Vnitřní zapojení bloku ***monostabil****.*

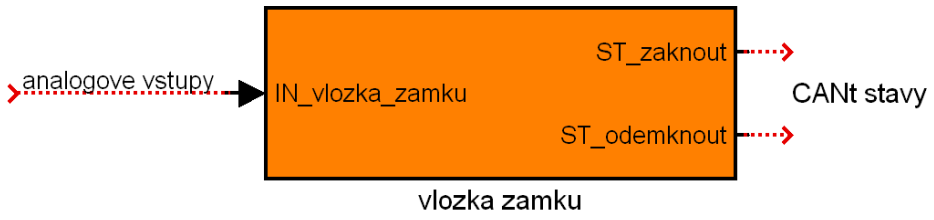
Obecně je zamčení, nebo odemčení zámku dveří definováno pouze stanovením doby, po kterou je nutné přivést napájení k ovládacímu motoru, aby došlo ke zvolené akci. Lze si to představit na příkladu, kdy přijde signál pro uzamčení zámku, který je v odemčeném stavu. Od okamžiku příjmu signálu pro zamčení je po stanovenou dobu aktivní výkonový výstup

k zamykání, bez ohledu na dobu trvání příchozího signálu pro zamčení. Tato doba je stanovena měřením na konkrétné použitých zámcích. V našem případě je tato doba nastavena na hodnotu 100 ms pro zamykání i odemykání. O generování této doby se stará blok *monostabil* (*Obr. 6.*17).

Principielně je tento blok podobný zapojení přijímacího bloku CAN zpráv na *Obr. 6.*6. Opět i zde je rozhodujícím členem komparátor, který porovnává hodnotu *Digital Clock1* se stejnou hodnotou zvětšenou o konstantu 0,1 ~ 100 ms. Po příchodu pulzu ze vstupu *triger* se přepínač přepne na malý okamžik do horní polohy a tím dojde k uložení hodnoty z *Digital Clock1* zvětšené o konstantu. V tuto chvíli se na dolním vstupu komparátoru nachází větší hodnota než na prvním a je generován signál True. Doba, než hodnota *Digital Clock1* dosáhne hodnoty uložené v paměti a tím dojde k překlopení komparátoru, je dána zvolenou konstantou. Aby nebylo možné monostabilní klopný obvod znovu spustit dřív, než dojde k překlopení do jeho výchozí polohy, je obvod doplněn o další bloky *AND, NOT* a *Memory*.

6.2.4 Vložka zámku

Blok vyhodnocuje stav vložky zámku, v níž je zasunutý klíč, kterým lze otočit doprava pro uzamčení dveří, nebo doleva pro odemčení dveří. Obě krajní polohy, do kterých je možné natočit klíč, můžeme brát jako samostatné dva stavy. Další stav je v základní poloze, kdy není klíč otočen ani na jedu ze stran, nebo klíč ve vložce zámku není zastrčen. Přestože přecházíme mezi třemi možnými stavy, blok vyhodnocující tyto stavy, signalizuje konkrétní polohu pouze, pokud je klíč otočen na jednu nebo na druhou stranu. Pro rozpoznání polohy je zde použito odporové kódování obdobně jako u tlačítka pro ovládání okna. Pro rozeznání poloh klíče je v této části modelu použito stejné zapojení bloků, jako je použito pro dekódování poloh tlačítka okna, které nalezneme na *Obr. 6.*11.



Obr. 6.18: Blok dekódování stavů vložky zámku.

Zapojení z obrázku *Obr. 6.*11 je v bloku *vlozka zamku* dvakrát. Jeho blokové schéma je zobrazeno na *Obr. 6.*18. Zapojení je zjednodušené, není zde dětská pojistka a proto dále směrem doprava od bloku *Merge* všechny následující bloky odpadají.

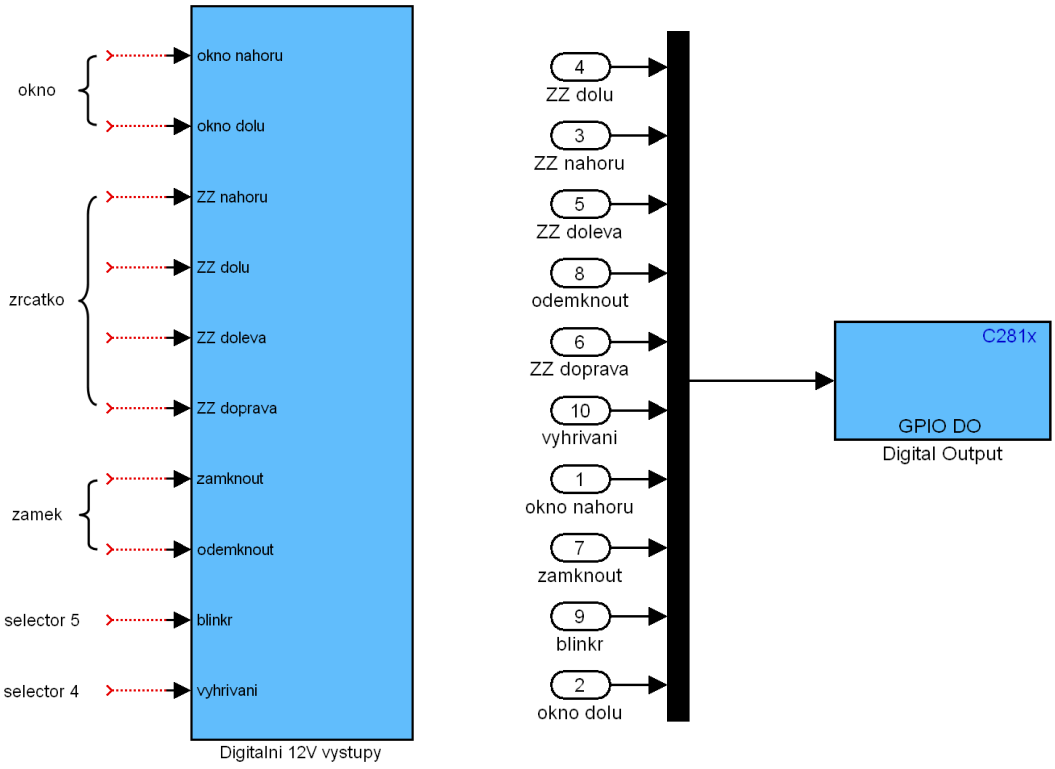
6.3 Výstupní bloky

Třetí částí modelu jsou výstupní bloky, kterými řídicí jednotka dveří řidiče ovládá a nastavuje jednotlivé akční členy na těchto dveřích a informuje ostatní řídicí jednotky o své činnosti, případně jim odesílá příkazy.

Emulátor disponuje několika typy výstupů, ovšem byly použity pouze dva z nich. V původní koncepci, patrné na *Obr. 6.*1, měl být využit i PWM výstup. Přestože je na obrázku uveden, nakonec jej nebylo využito pro jeho rychlost, jak je uvedeno v kapitole 2.1.2. Oproti tomu z naznačených bloků PWM, DigOUT a CAN má zdaleka největší zastoupení digitální výstup.

6.3.1 Digitální výstupy

Tento blok zahrnuje veškeré digitální výstupy, které jsou v modelu ECU použity.



Obr. 6.19: Blok digitálních výstupů (vpravo vnitřní struktura bloku).

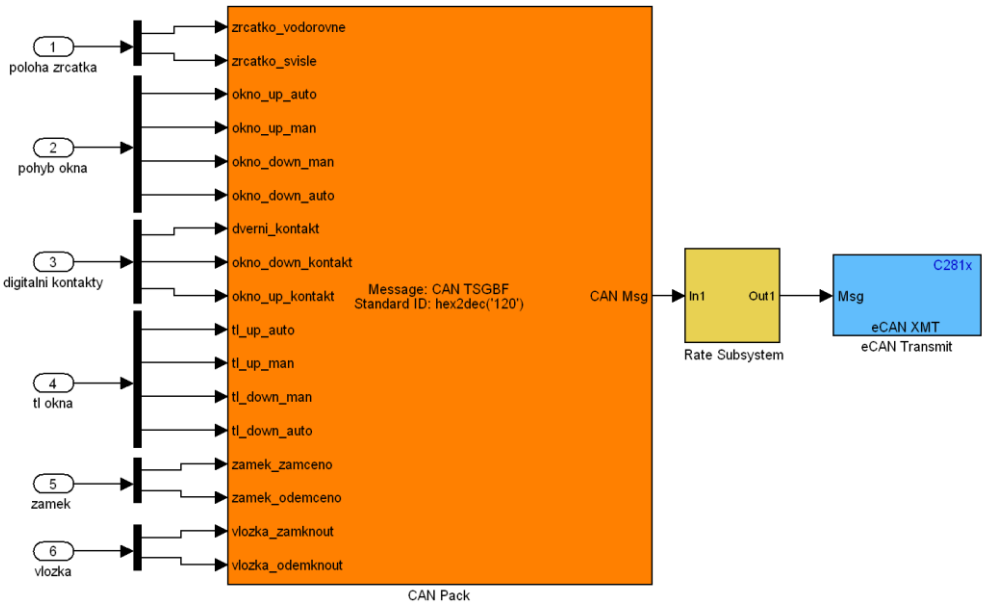
Na *Obr. 6.*1 je patrná velikost tohoto bloku a tím i důležitost v modelu. Je to dáno potřebou ovládat především motory, pohybující s oknem, zrcátkem nebo zámkem. Digitální výstupy jsou také použity k vyhřívání zpětného zrcátka a blinkru ve zpětném zrcátku. Jaké konkrétní signály vstupují do tohoto bloku, je patrné z *Obr. 6.*19.

Uvnitř bloku se nachází multiplexor skládající jednotlivé signály pro výstupní blok *Digital Output*. Z výběru je nastavena **brána B**, na které jsou umístěné digitální výkonové výstupy. Dále je v bloku nutné zaškrtnout, které bity z celého portu budou použity. V *Tab. 2.*3 je uvedeno, které piny procesoru jsou využívány a k jakému účelu v emulátoru slouží. Využity jsou GPIOB6 – GPIOB15, celkem tedy 10 výstupních pinů, které musí být zaškrtnuty v bloku *Digital Output*. Přivedení signálů do bloku musí být v pořadí od nejnižší hodnoty využitého bitu k té nejvyšší. V *Tab. 2.*3 je uvedeno, jaký signál odpovídá konkrétním bitům portu.

U tohoto bloku se nijak nenastavuje aktualizace nebo perioda nastavování výstupních hodnot. Tento proces je odvozen od signálů vstupujících do tohoto bloku.

6.3.2 CAN zprávy - Transmit

Neméně důležitým blokem ve výstupní části modelu řídicí jednotky je segment starající se o odesílání zpráv na sběrnici CAN. Všechny signály, které chceme odeslat ve zprávě na sběrnici CAN, vstupují do bloku *CAN Transmit*.



Obr. 6.20: Vnitřní zapojení bloku ***CAN Tansmit.***

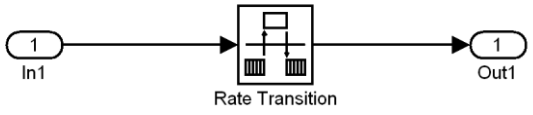
Na *Obr. 6.*20 je vidět vnitřní struktura zapojení bloku *CAN Trasmit*. Uvnitř se nachází další tři bloky, jejich funkce jsou zprávu zakódovat, synchronizovat a následně odeslat. První blok *CAN Pack* se stará o zakódování zprávy ze všech hodnot přicházejících v jednotlivých signálech. Blok má zcela opačnou funkci než blok *CAN Unpack* z *Obr. 6.*7, avšak jeho nastavení je velice podobné.

- Data is input as: manually specified sinals
- Identifier type: Standard (11-bit identifier)
- Identifier: 120

Takto jsou nastaveny základní parametry bloku *CAN Pack*. Opět je nastaven 11 bitový identifikátor, jehož uvedená hodnota 120 je v hexadecimálním tvaru. Na na sběrnici CAN budou odesílány zprávy s tímto identifikátorem, podle kterého ostatní jednotky zprávu identifikují.

V bloku *CAN Pack* je v případě volby manuální specifikace signálů nutné tyto signály definovat. Výstupem tohoto bloku je zabalená zpráva, kterou je již možné odeslat blokem *C281x eCAN Transmit*.

Blok odešle zprávu s každou změnou vstupního signálu. Pokud by se signál neměnil, *C281x eCAN Transmit* by žádnou zprávu neodesílal a ostatní jednotky by neměli žádnou informaci o chování dveří spolujezdce. Na sběrnici je potřeba pravidelně tyto informační zprávy odesílat a informovat tak ostatní řídicí jednotky. K tomu byl před *C281x eCAN Transmit* vložen další blok, zajišťující pravidelné buzení a tím odesílání zpráv na sběrnici CAN (*Obr. 6.21*).



Obr. 6.21: Vnitřní zapojení bloku ***Rate Subsystem*** *pro periodické buzení* ***C281x eCAN Transmit.***

Blok *Rate Transition* je pouze pro spojení mezi bloky s různým časováním. Aby byl blok *C281x eCAN Transmit* buzen v pravidelných intervalech a tím odesílal zprávy s předepsanou periodou, je parametr *Sample time*, výstupní svorky bloku *Rate Subsystem*, nastaven na hodnotu, kterou mají být zprávy odesílány. V našem případě je tato doba nastavena na dobu 10 ms.

6.4 Přeložení modelu ECU

Model vytvořený v Simulinku lze jedním kliknutím, případně klávesovou kombinací nahrát do procesoru emulátoru. To lze, pokud je model správně nastaven. Jak správně nastavit Target a model bylo popsáno v kapitole 4.2.1.

Pokud již tedy máme hotový a správně nastavený model, můžeme jej nechat přeložit a vygenerovaný kód nahrát do procesoru emulátoru. Tento proces je zcela automatický a je možné jej spustit několika způsoby[7].

- Stisknutím kláves ***Ctrl* + *B***.
- *Tools → Code Generation → Build Model*

Nejprve je model simulací zkontrolován. Simulink následně spustí CCStudio, připojí se k emulátoru a nahraje program do paměti procesoru. Tento proces spojování Simulinku s CCStudiem není vždy zaručen a může dojít k chybě. V tom případě je bezpečnější program Code Composer Studio spustit ručně a k emulátoru se manuálně připojit. To provedeme podle posledního bodu v kapitole 2.6.

[7] Vždy je nutné mít aktivní okno překládaného modelu.

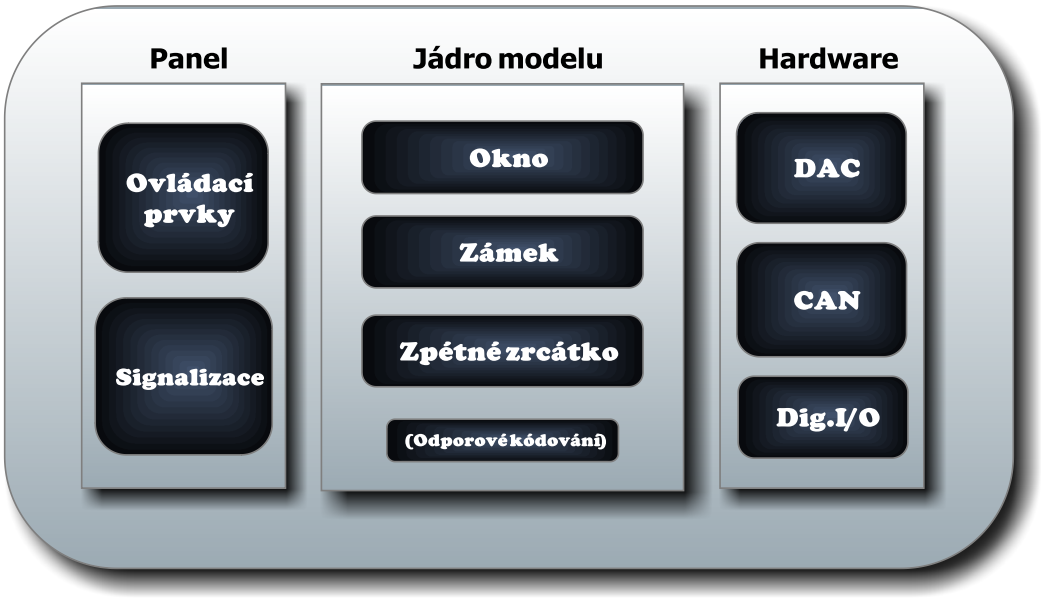
7 Model okolí řídicí jednotky

Vytvořit model okolního prostředí řídicí jednotky obecně není jednoduché a ve většině případů je daleko složitější než samotný model řídicí jednotky. Rozdíl ve složitosti může být v některých případech dosti velký. Záleží na systému, pro který model tvoříme.

Téměř všechny systémy, se kterými je možné se setkat, jsou nelineární. V reálném světě neexistuje systém, který by se projevoval jakýmkoli způsobem dokonale lineárně. Tvoříme-li model těchto systémů, cílem je dosáhnout shodného chování. Ovšem matematický popis nelineárních systémů je složen převážně z diferenciálních rovnic. Dnešní výpočetní systémy zvládají určitou složitost těchto soustav, ta se ale podepisuje na výpočetním čase. I kdyby bylo možné systém dokonale popsat a chování modelu by se téměř neodlišovalo od reality, vytvořený model by byl tak složitý, že by nebylo prostředků (počítače), který by dokázal reagovat v reálném čase na podněty okolí. Reálné systémy je tedy nutné alespoň částečně linearizovat, a tím zjednodušit vytvářený model.

Podíváme-li se na prostor kolem řídící jednotky dveří spolujezdce, nalezneme zde jen několik základních prvků. Dveře nesou motory, odporově kódované senzory a jednoduché spínací kontakty a všechny tyto části lze v Simulinku vytvořit jednoduchými bloky.

Podobně jako u ECU, je i zde celý model blokově rozdělen do třech funkčních částí, jejichž strukturu je možné vidět na *Obr. 7.1*.

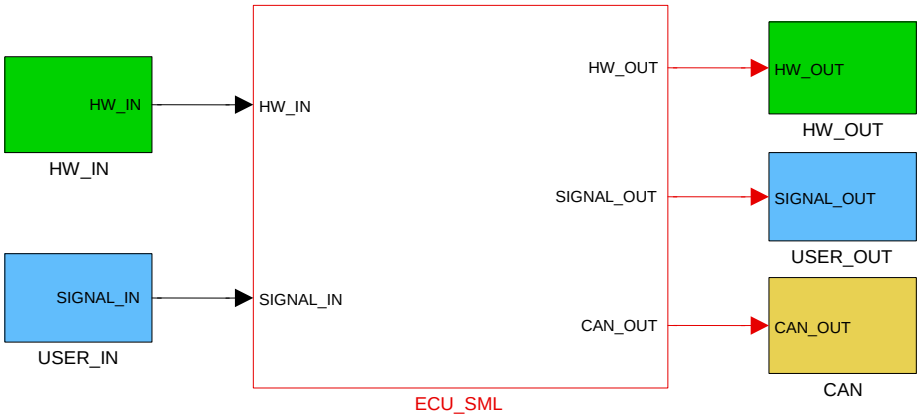


Obr. 7.1: Struktura modelu okolního prostředí ECU.

V průběhu tvorby modelu pro µHiL nebylo možné průběžně testovat vytvořené soubory, což zákonitě vedlo k velkému předělávání před koncem tvorby této práce. Poskytnutí licence pro spuštění programu umožnilo testovat na simulátoru předem vytvořené modely. Bylo vytvořeno mnoho variant, aby bylo dosaženo vhodného modelu, který bude mít v programu viditelné všechny signály.

V následujících části budou naznačeny nejdůležitější bloky modelu, jejichž struktura se v průběhu jejich tvorby několikrát změnila. Především se jednalo o doplňování podpůrných bloků, zajišťující komfortnější chování simulačního modelu.

7.1 Celková struktura modelu



Obr. 7.2: Celková struktura modelu okolního prostředí ECU.

Zde je možné všimnout podobné struktury, jako je na obrázku na předchozí stránce. Do jádra modelu *ECU_SML* vstupují signály od uživatele nebo výstupní signály testované ECU. Tato data jsou vyhodnocena algoritmem v *ECU_SML* a odeslána na správný port modelu.

Bloky označené modrou barvou označují uživatelské rozhraní a opticky jej oddělují od zbývajících částí modelu. Signály vedoucí do bloků zelených slouží ke spojení mezi testovanou ECU a modelem jejího okolí. Jedná se tedy o fyzické propojení.

7.2 Panel s prvky pro uživatele – bloky *USER_xx*

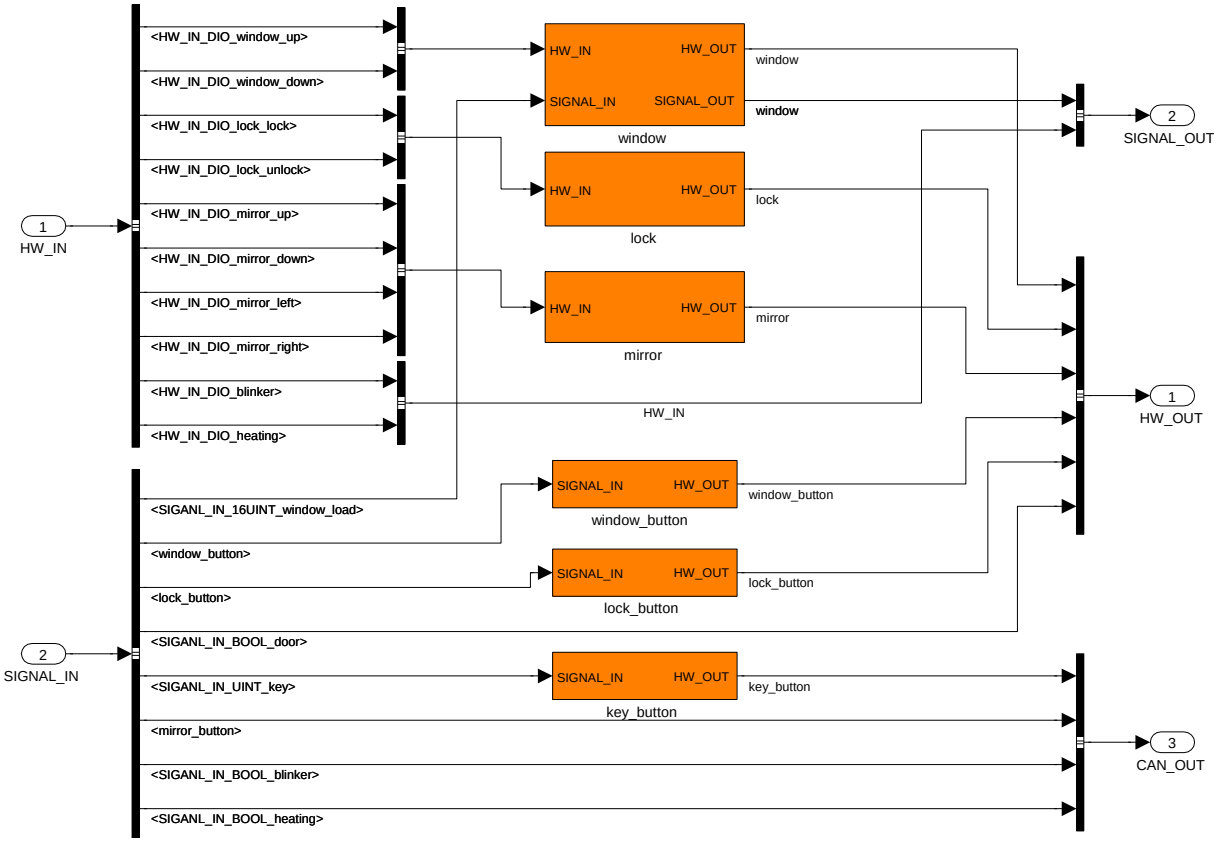
Část modelu obsahující odkazy na signály, které mají být monitorovány nebo nastavovány ve Workpace programu *PROVEtech:TA*. Jakým způsobem je možné jednotlivé

signály zobrazit nebo jiným způsobem s nimi pracovat popisuje kapitola 5.1. Převážná část signálů v tomto bloku je vstupních. Jedná se především o signály typu:

- poloha klíčku zapalování
- ovládací tlačítko zpětného zrcátka
- tlačítka pro ovládání okna
- kontakt signalizující otevřené dveře
- blinkr
- vložka zámku

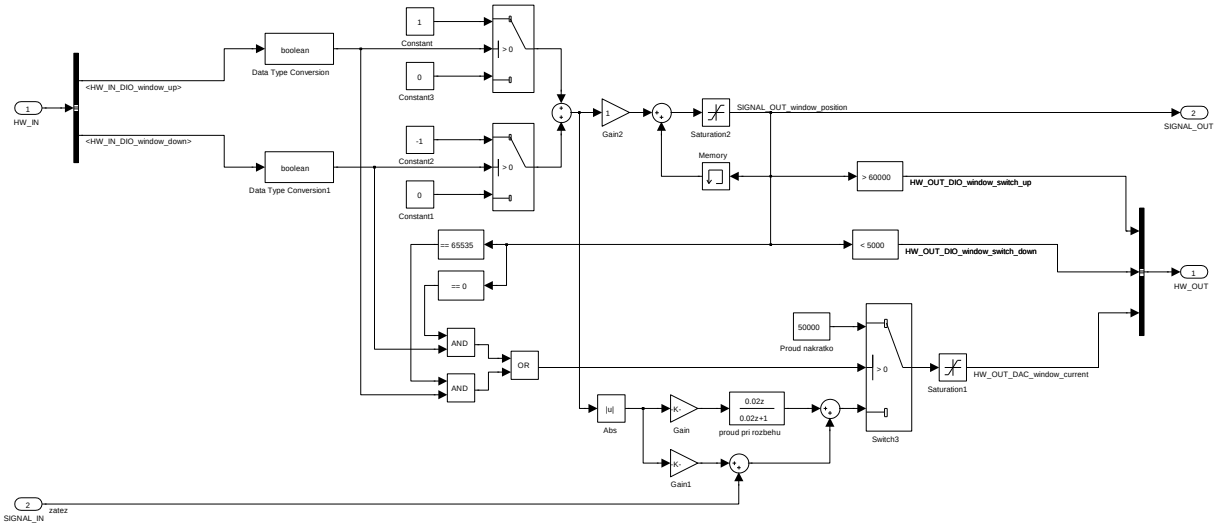
7.3 Jádro modelu – blok *ECU_SML*

V této části modelu se nacházejí logické členy, které se starají o generování výstupních signálů, na základě vstupních podnětů. Podobně jako v modelu ECU se i zde pracuje se třemi základními prvky dveří (Okno, zámek a zpětné zrcátko).



*Obr. 7.3: Vnitřní zapojení bloku **ECU_SML** (jádro modelu).*

7.3.1 Motor okna – blok *window*



Obr. 7.4: Model okna dveří.

Signál *1 – HW_IN* vstupující do tohoto bloku modelu určuje, jakým směrem se má okno pohybovat. Podle toho zda je vysoká úroveň na jednom nebo na druhém vstupu dostáváme na výstupu součtového bloku hodnotu 1 nebo -1, případně 0, není-li aktivní ani jeden signál. Následně tento součet vstupuje do bloku *Gain2*, kde je přenásoben a dále zaveden do diskrétního integrátoru složeného z bloků *Suma, Saturation2* a *Memory*. Aktuální hodnota v bloku *Memory* a tím i hodnota na výstupní svorce *2* určuje polohu okna.

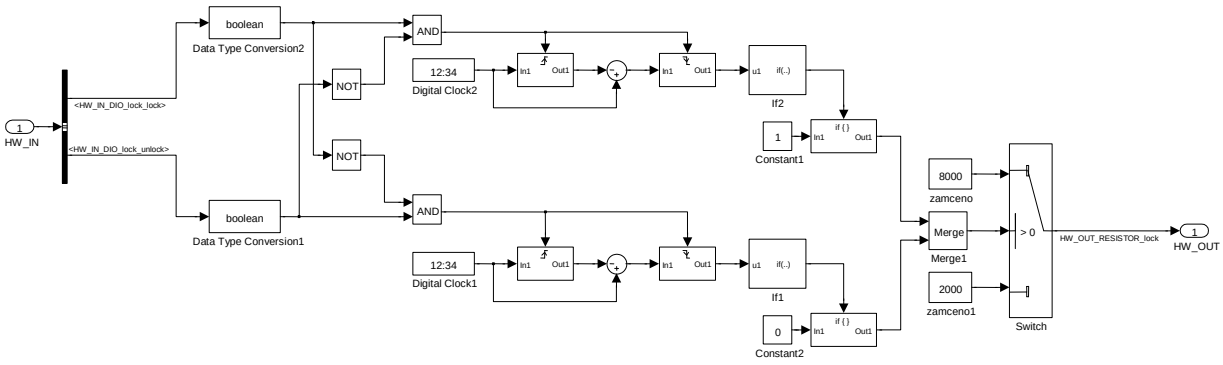
O detekci a generování logických signálů při zavřeném a zcela otevřeném okně se starají dva bloky porovnávající aktuální polohu okna s nastavenou konstantou. Hodnot jakých může poloha okna nabývat je v rozmezí 0 – 65535, přičemž 0 vyjadřuje zcela otevřené okno a 65535 zavřené okno.

ECU snímá i proud motoru okna a podle jeho velikosti vyhodnocuje, zda nedošlo k nějakému zablokování nežádoucím předmětem jako například ruka dítěte. Velikost proudu je při běžném zavírání či otvírání konstantní. Jak je vidět na *Obr. 7.4*, tak o generování velikosti proudu se starají veškeré bloky připojené za blokem *Abs*. Základní proud je vytvořen blokem *Gain1*, starající se o konstantní hodnotu proudu, a blokem přenosové funkce pro generování impulzu při rozběhu motoru. Vstupní signál *2 – SIGNAL_IN* zvyšuje konstantní proud, v závislosti mechanickém zatížení. Pokud je jeden ze vstupních signálů pro pohyb nahoru nebo dolu aktivní a okno je již v krajní poloze, blok *Switch3* přepne výstup na hodnotu proudu, kterou by měl motor nakrátko.

7.3.2 Motor zámku – blok *lock*

Zámek dveří je také ovládán motorem. Opět máme dva vstupní signály, jeden pro zamčení a druhý pro odemčení zámku. Vnitřní zapojení modelu zámku je na

Obr. 7.5.

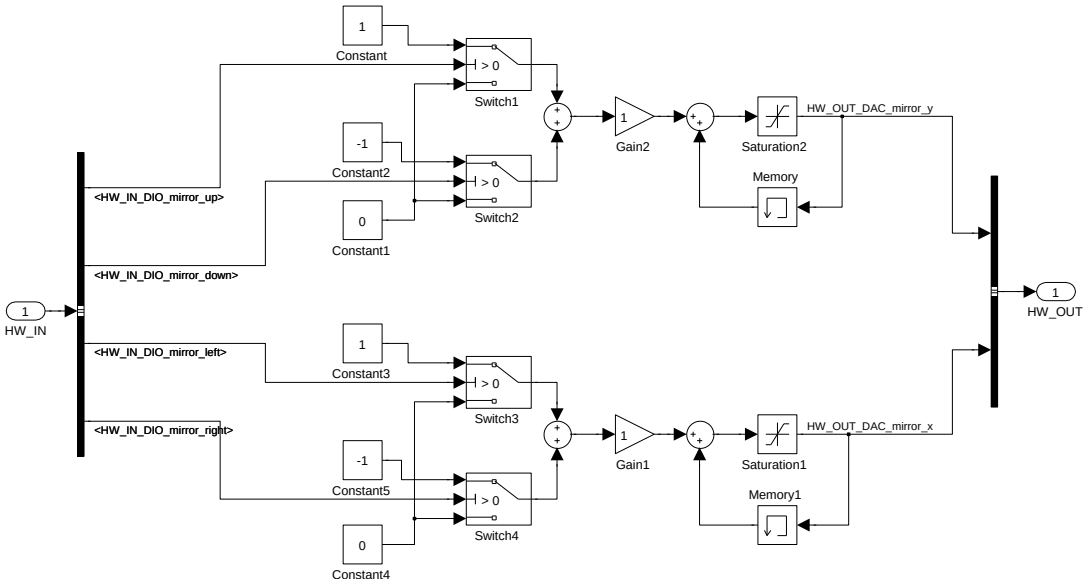


Obr. 7.5: Model zámku dveří

Zde se pouze snímá doba aktivního signálu, která by měla být 100±10 ms. Ta je potřebná pro zamčení nebo odemčení zámku, a proto ECU musí generovat signál po tuto dobu.

Při vzestupné hraně vstupního signálu se uloží hodnota *Digital Clock* do prvního bloku reagujícího na hranu. Druhý blok při sestupné hraně, uloží rozdíl odpovídající délce impulsu. Pokud doba vyhovuje podmínce, je generován příslušný odpor odpovídající poloze zámku.

7.3.3 Zpětné zrcátko – blok *mirror*



Obr. 7.6:Model generující polohu zpětného zrcátka.

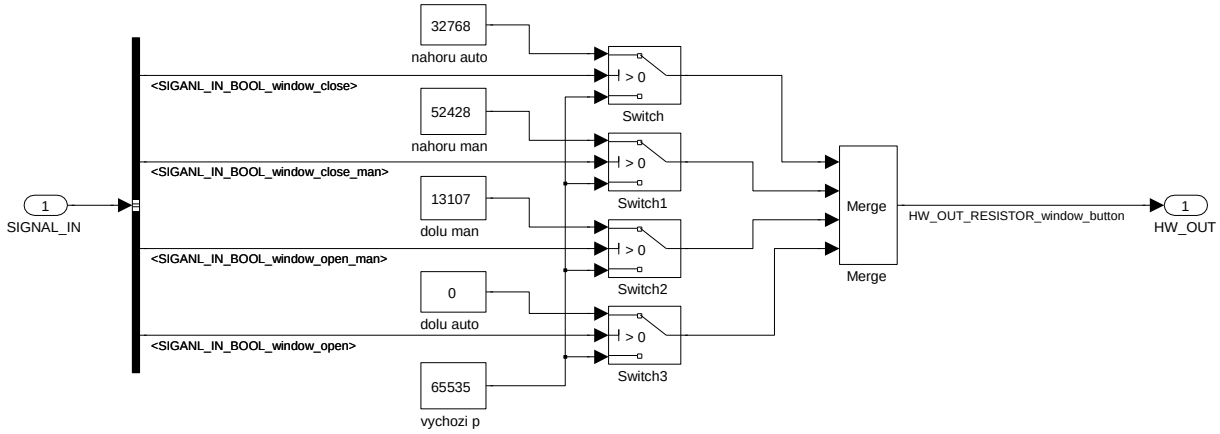
Motory zpětného zrcátka musí být dva, jeden pro ovládání vertikální roviny a druhý pro ovládání horizontální roviny. Zapojení bloků je analogicky stejné jako na

Obr. 7.5. Protože zde není nutné sledovat proud motorem, odpadá veliká část bloků, a zbývá pouze struktura diskrétního integrátoru, určující polohu zpětného zrcátko a dané rovině.

7.3.4 Odporové kódování ovládacích prvků

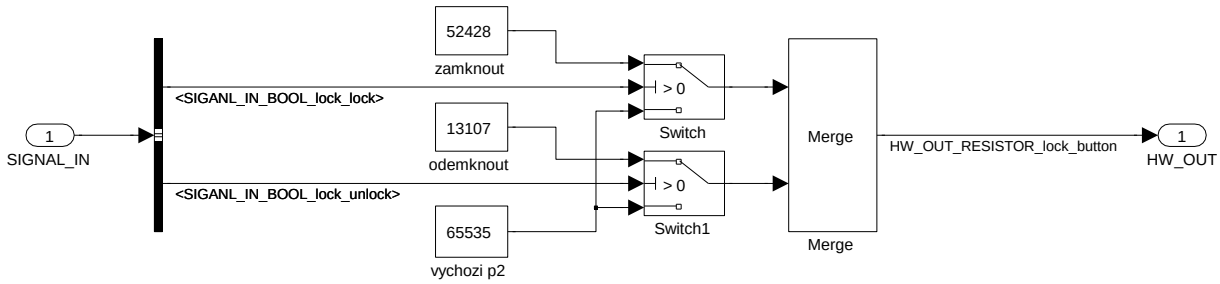
Ke zjištění stavu polohy tlačítka ovládacího stahování okna je použito odporového kódování. Stejně tak i polohy klíče ve vložce zámku je zastoupeny konkrétním odporem vůči zemi. Na

Obr. 7.7 je znázorněn princip generování pěti možných odporů představujících čtyři akční stavy a jeden výchozí od tlačítka okna.



Obr. 7.7: Odporové kódování tlačítka okna.

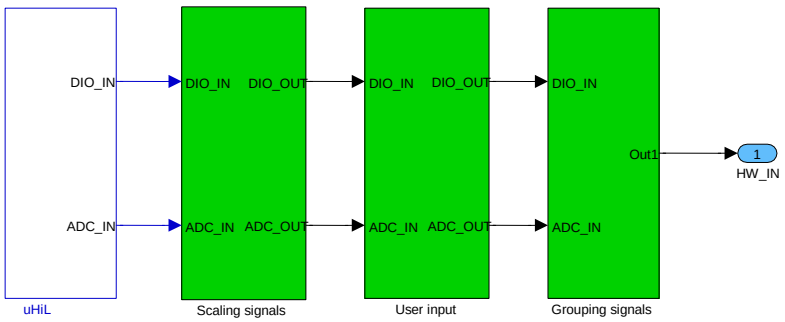
Vložka zámku je vytvořena stejným způsobem, jen se liší v počtu stavů.



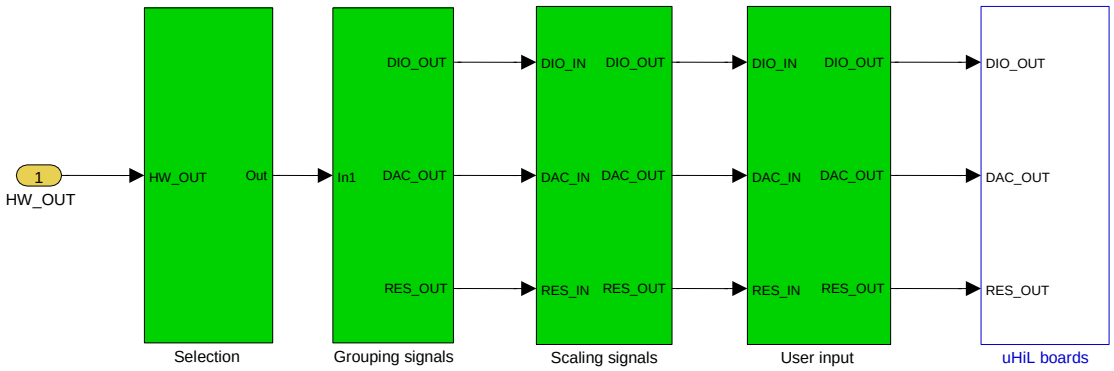
Obr. 7.8: Odporové kódování vložky zámku.

7.4 Hardwarové propojení – bloky *HW_xx*

Model okolního prostředí řídicí jednotky má signály vstupní a výstupní, z nichž některé nastavuje nebo sleduje přímo uživatel. Tyto signály odkazují na prvky umístěné ve Workpage, sloužící k vizualizaci a parametrizaci. Ostatní signály jsou napojeny na vstupy a výstupy simulátoru. Na *Obr. 7.9* jsou bloky, které zajišťují vstup signálů do simulátoru vycházející z ECU. Na *Obr. 7.10*.



Obr. 7.9: Vstupní bloky pro hardwarové signály.



Obr. 7.10: Výstupní bloky pro hardwarové signály.

V obou případech je možné signály upravit do jiných formátů, rozdělit na skupiny, nebo omezit rozsah.

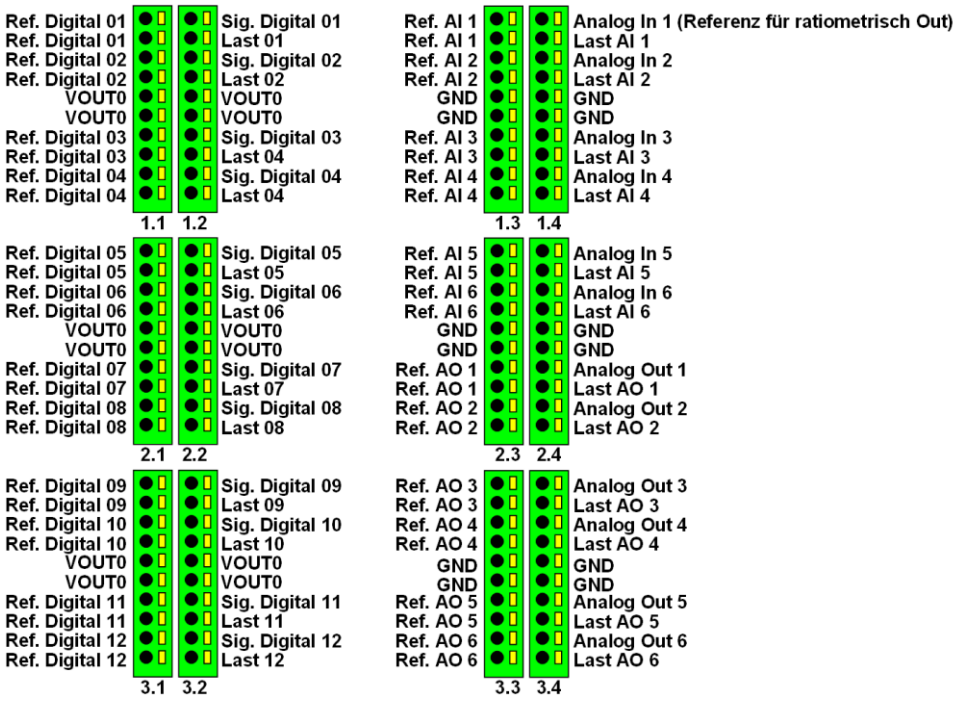
Bloky odkazující na signály, sloužící k propojení simulátoru s emulátorem, byly v prvních fázích návrhu zcela odlišné od výše uvedených. Matlab-Simulink totiž neobsahuje konkrétní knihovnu pro *PROVEtech:µHiL* a určit, který blok z obecné knihovny Simulinku, odkazuje na vhodné signály, bylo velice obtížné.

8 Spojení Emulátoru a *PROVEtech:µHiL*

Testovací platforma, se kterou budou simulace na ECU prováděny, je *PROVEtech:µHiL*. Jedná se o HiL simulátor, jehož základ je postaven na technologii FPGA umožňující zpracování časově kritických operací. Jedná se o kompaktní zařízení, které může být umístěno přímo na stole na vývojářském pracovišti a tím uživatel může jednoduše při testování ECU zasahovat do této sestavy. Simulátor, jež můžeme vidět na *Obr. 8.2*, se skládá ze tří částí:

- Power-box 
- FPGA-box 
- SC-box 

Power-box a FPGA-box tvoří základ testovacího systému, do kterého je možné zasunout až čtyři SC-boxy. Power-box poskytuje napájecí napětí pro všechny vnitřní funkce µHiL. Slouží tedy jako zdroj pro FPGA-box a čtyři volitelné SC-boxy. Aby bylo možné jej připojit µHiL k počítači, musí být použito CAN/USB převodníku, protože simulátor je řízen přes sběrnici CAN. Každý SC-box je složen ze dvou desek, jedné primární a druhé sekundární. V našem případě jsou hlavní deska MBT-B10203 a sekundární deska MBT-B10204.



Obr. 8.1: Zapojení pinů SC-boxu (vlevo – sekundární deska, vpravo – hlavní deska).

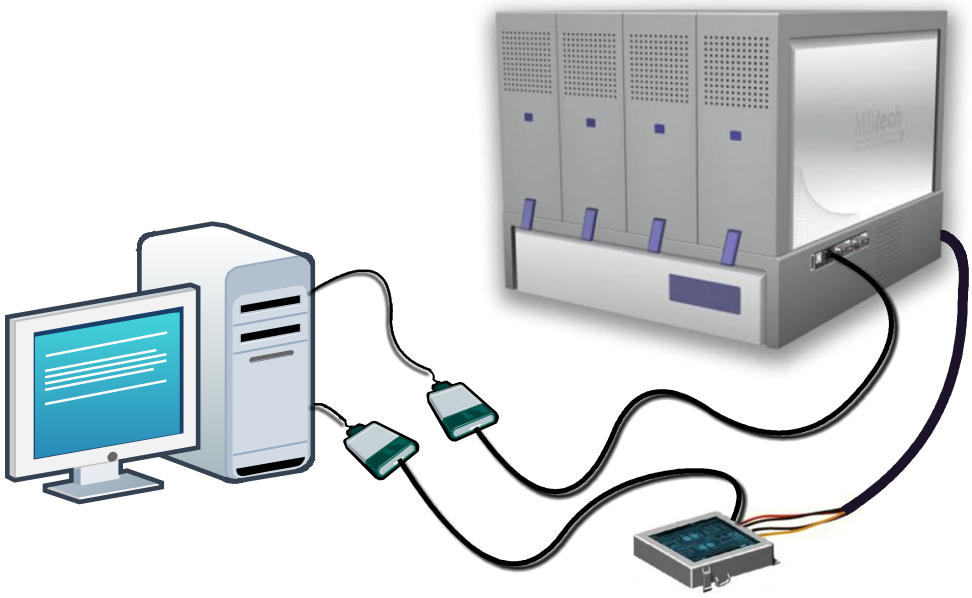
Spojením těchto dvou hlavních desek dostáváme SC-box umožňující připojení:

- 6x Analog-OUT
- 6x Analog-IN

} MBT-B10203

- 12x Digital-IO (PWM-IO) MBT-B10204

Zapojení pinů jednotlivých vstupů a výstupů je na obrázku *Obr. 8.1*. Testovaná řídicí jednotka je připojena právě k těmto pinům. Podle toho zda se jedná o analogový nebo digitální rozhraní zapojíme signál do jedné nebo do druhé desky. Deska zpracovávající analogový signál má rozlišeny jednotlivé piny pro vstup a výstup. U druhé digitální desky není vstup a výstup rozlišen a navíc se zde pod každým pinem spojují funkce PWM a Digital. Jak by mohl celý testovací systém vypadat je znázorněno na *Obr. 8.2*.



Obr. 8.2: Spojení testovacího systému PROVEtech:µHiL, ECU a počítače.

Počítač, v němž je spuštěn řídící program *PROVEtech:TA* je připojen ke sběrnici CAN jednotky ECU, podobně jako je připojen k µHiL. Je tedy zapotřebí dvou převodníků. Jeden pro komunikaci s µHiL a druhý pro komunikaci s řídicí jednotkou.

Jelikož je µHiL vybaven i inteligentním napájecím zdrojem, umístěným pod SC-boxy, můžeme testované zařízení připojit rovnou k simulátoru a není potřeba dalších komponent.

V kapitole 5.3 je popsaná konfigurace modelu, včetně postupu, jak vytvořit soubor *.xml* definující porty, konstanty, signály a další prvky modelu vytvořeného v Simulinku.

Tento soubor ještě společně se souborem *.dll,* vygenerovaný Simulinkem, je potřeba zahrnout do konfiguračního souboru *.xml* [8]. Konfigurační soubor je nutné otevřít a do místa označeného `<ModelConfig>` *Signal_JmenoModelu.xml* `</ModelConfig>` zapsat umístění modelového souboru . *xml* včetně cesty. Podobně se musí zadat i umístění modelového souboru `<ModelDll>` *JmenoModelu.dll* `</ModelDll>`.

```
…

<Peer>
      <Name>Model</Name>
      <Class>RC_CModel</Class>
      <Config>
            <ModelConfig>.\Signal.xml</ModelConfig>
            <ModelDll>.\sample.dll</ModelDll>
            <tick>0.01</tick>
      </Config>
</Peer>

…
```

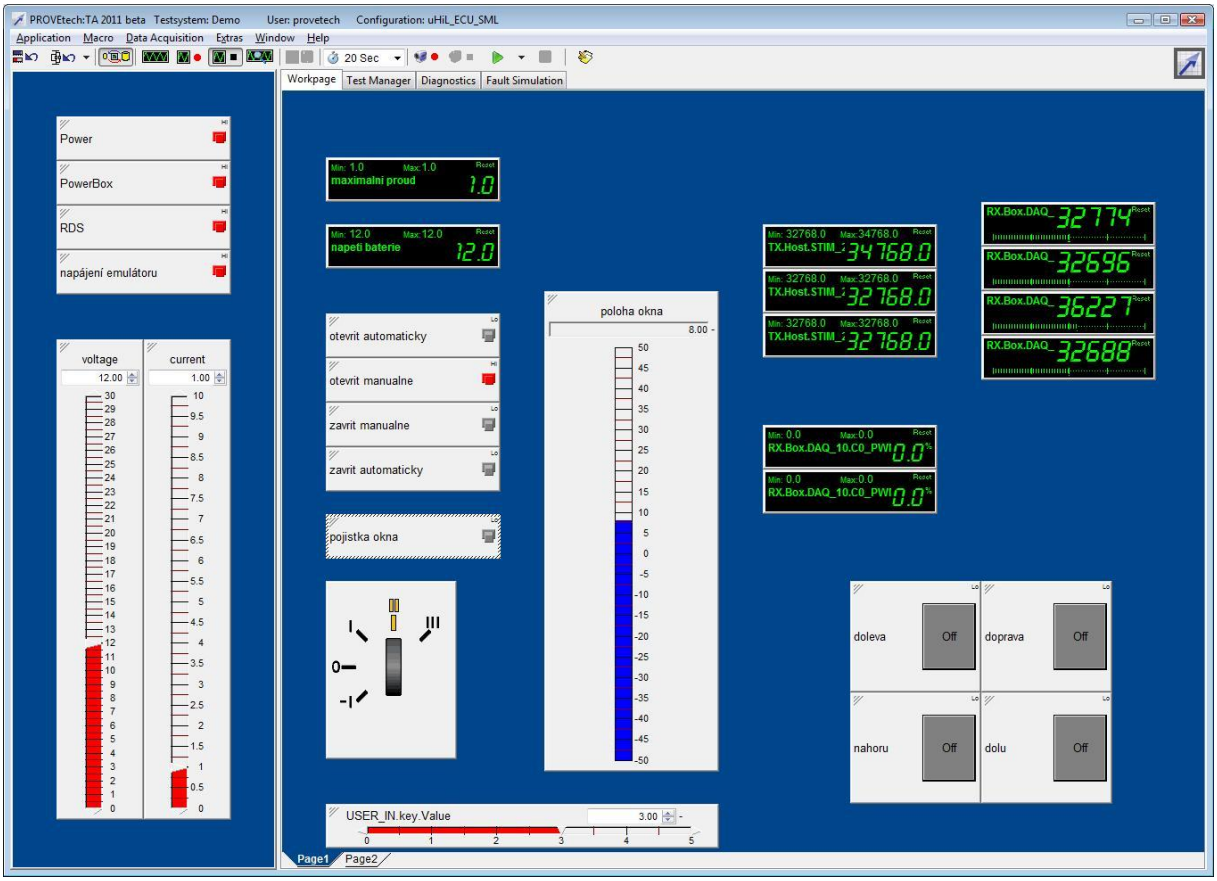
Takto jsou do prostředí přidány signály obsažené v modelovém *.xml* souboru. Obecně je možné tyto soubory editovat. Je nutné dodržovat základní zásady. Například se nesmí měnit pořadí signálů a obecně všech prvků. Při změně by vznikla nekonzistence mezi modelovými soubory *.dll* a *.xml*.

Pokud jsou již signály do programu přidány, je možné s nimi pracovat a vytvořit pracovní prostředí s ovládacími a zobrazovacími prvky. Signály přidáme z okna *Signal Selection*, kde jsou umístěny veškeré signály, se kterými je možné pracovat. Toto okno vyvoláme *Extras* → *Signals.* Na výběr je z několika přednastavených ovládacích a monitorovacích tlačítek a obrazovek. Stačí pouze zvolit typ, a myší požadovaný signál přesunout do okna *Workplage*, případně do okna *Cocpit*.

Pro spojení s ECU bylo vytvořeno pracovní zkušební prostředí, v němž byly umístěny základní nastavovací prvky, uvádějící jednotku do provozu. Náhled tohoto okna je vidět na *Obr. 8.3*.

[8] Jde o konfigurační soubor ***microHil.xml*** *zmíněný v kapitole 5.3.*

Jak je na obrázku vidět, v okně *Cocpit* jsou umístěny tlačítka pro spuštění napájecího napětí, včetně možnosti přesného nastavení napětí a omezovacího proudu. V obrazovce *Workpage* je příklad rozmístění tlačítek pro testování okna a zpětného zrcátka.



Obr. 8.3: Příklad Layoutu pro testování jednotlivých funkcí modelu.

Takto lze testovat ručně a jednotlivé akce se signály provádět manuálně. Pokud neodpovídá jedna malá část ze systému, je potřeba jí věnovat větší pozornost a zabývat se jí konkrétně. Takto lze efektivně a flexibilně přistupovat k určitým částem systému.

Pro komplexnější testování lze použít automatizované testy. V prostředí *PROVEtech:TA*, existuje aplikace *Test Manager*, v níž je možné programovat skripty a mít celkovou správu nad testy. Je to tedy klíčový komponent pro automatizované testování. Jednotlivé případy jsou implementovány jako makra v integrovaném programovacím prostředí. Testovací skrypty mohou být prováděny odděleně nebo ve skupinách.

K vytvoření nového testu je nutné nejprve vytvořit nový testovací objekt a v něm definovat obecná data. Dále jen stačí napsat samotný testovací skript.

Zde je ukázka jak může takový testovací skript vypadat.

```
Sub Main

  Dim Up, Down, pos
  System.Remark("<h2>Test of window closing</h2>")
  System.Remark("<hr width=100% size=2 color=black>")
  System.Remark("<h4>Preconditions...</h4>")
  System.LoadModel("")
  Wait(0.25)
  System.SetSignal("window.Up.Value", 0)
  System.SetSignal("Window.Down.Value", 0)
  System.SetSignal("window.load.Value", 0)
  System.SetSignal("window.position.Value", 50)
  System.Remark("<b>Model initialized</b>")
  System.Remark("Button Up        released")
  System.Remark("Button down      released")
  System.Remark("Load             released")
  System.Remark("Position win.    open")
  Wait(1)
  m_lib.PowerSys()
  Wait(1)
  System.Remark("<hr width=100% size=2 color=black>")
  System.Remark("<h4>Test case...</h4>")
  System.SetSignal("window.Up.Value", 1)
  System.Remark("Push a button Up")
  Wait(3)
  System.Remark("Wait for 3 s")
  System.Remark("Window goes up<br>")
  pos = System.GetSignal("window.position.value")
  If (position > 99) Then
  System.Remark("Window is closed <font color=#00FF00>PASSED</font><br>")
  Else
  System.Remark("Window is not closed <font color=#FF0000>FAILED</font><br>")
  End If

End Sub
```

Po spuštění skriptu se provede sekvence příkazů a na následujícím obrázku je vidět výsledek testu. Pro ten jsou použity standardní html tagy.

Test of window closing

Preconditions...

Model initialized
Button Up released
Button down released
Load released
Position of win. open

Power on

Test case...

Push a button Up
Wait for 3 s
Window goes up

Window is not closed FAILED

Obr. 8.4: Protokol testu zavření okna.

Závěr

Testování elektronických řídících jednotek, především jejich řídícího software, dnes neodmyslitelně patří do vývojového procesu automobilové elektroniky. Software se stal klíčovým prvkem inovací ve vozidle, což se projevuje velikou složitostí a rozsahem kódu. Využívání programů s interaktivním rozhraním rozšiřuje a především urychluje proces jeho vývoje.

V této práci je využito modelovacího prostředí Matlab-Simulink k vytvoření kódu pro emulátor řídící jednotky, který vznikl pro laboratorní simulace a diagnostiku. S ohledem na jeho možnosti a dostupná rozhraní, byla pro tvorbu modelu vybrána řídící jednotka dveří spolujezdce. Dále byl vytvořen model virtuálního fyzikálního prostředí, ve kterém řídící jednotka bude pracovat. Model byl vytvořen pro testovací platformu *PROVEtech:µHiL* vyvinutý společností MBtech Group. Pro simulaci prostředí, tedy provoz modelu v reálném čase, bylo použito běhové prostředí *PROVEtech:RE*. Spojením HiL testovací platformy s příslušnými modely a testované řídící jednotky, dostáváme uzavřenou regulační smyčku systému a to v laboratorním prostředí.

V této práci jsou uvedeny základní popisy software, včetně jejich nastavení, použitých pro vytvoření modelů a následně jejich implementace v HiL testovacích sestavách. Velká část práce je věnována vlastnímu popisu realizovaných modelů. Na závěr práce měly být vytvořeny sady testů pro oživení řídicí jednotky. Licence do software používaného pro tento simulátor, byla poskytnuta až ke konci tvorby této práce, a nebylo tak možné vytvořit kvalitní sestavu testů.

Práce s tímto software není tak hluboce podporována jako například od společností National Instruments, ETAS nebo dSPACE, kde nejen podpora pro Matlab-Simulink je mnohem vyšší, ale i samotné simulátory poskytují širší možnosti. Na druhou stranu se jedná o levnější variantu, těchto konkurenčních systémů.

Literatura a použité internetové zdroje

[1] HYNEK, Jan. Použití jednočipového mikropočítače pro simulaci řídící jednotky. Plzeň, 2011. Diplomová práce. Západočeská univerzita, Fakulta elektrotechnická.

[2] VEVERKA, Stanislav. Simulace dynamického systému v prostředí Matlab-Simulink a PROVEtech:TA. Plzeň, 2010. Diplomová práce. Západočeská univerzita, Fakulta elektrotechnická.

[3] HOUSAR, Pavel. Model okolí pro HIL testování dveřních řídících jednotek automobilu Škoda. Plzeň, 2007. Diplomová práce. Západočeská univerzita, Fakulta elektrotechnická.

[4] JELÍNEK, Pavel. Simulace Processor In the Loop a Hardware In the Loop. *Automa* [online]. May/June 2007, 5(25) [cit. 8.3.2012]. Dostupné z: http://www.odbornecasopisy.cz/index.php?id_document=34311

[5] *Embedded Coder. Generate C and C++ code optimized for embedded systems* [online]. © 1994-2012 The MathWorks, Inc. [cit. 16.3.2012]. Dostupné z: http://www.mathworks.com/products/embedded-coder/

[6] *Creating and Working with Models, Simulink* [online]. 1994-2012 The MathWorks, Inc. [cit. 20.3.2012]. Dostupné z: http://www.mathworks.com/products/simulink/description2.html

[7] *Generování kódu v jazyce C a C++ z modelů v Simulinku a Stateflow, Simulink Coder™* [online]. © HUMUSOFT 1991 - 2012 [cit. 21.3.2012]. Dostupné z: http://www.humusoft.cz/produkty/matlab/aknihovny/simulink-coder/

[8] *Simulace a modelování dynamických systémů, Model-Based Design, Simulink* [online]. © HUMUSOFT 1991 - 2012 [cit. 18.3.2012]. Dostupné z: http://www.humusoft.cz/produkty/matlab/simulink/

[9] A comprehensive tool for test automation [online]. MBtech Group GmbH & Co. KGaA [cit. 20.4.2012]. Dostupné z: http://www.mbtech-group.com/fileadmin/media/pdf/electronics_solutions/PROVEtech_TA_EN.pdf

[10] *Texas Instruments. Data Manual, TMS320F281x, TMS320C281x DSPs* [online]. *Dallas, Texas. April 2001*. Poslední změna March 2011 [cit. 11.3.2012]. 1149.1-1990 IEEE. Dostupné z: http://www.ti.com/lit/ds/symlink/tms320f2812.pdf

[11] *Texas Instruments. TMS320x281x DSP Analog-to-Digital Converter (ADC) Reference*

Guide [online]. *Dallas, Texas. June 2002*. Poslední změna July 2005 [cit. 22.3.2012]. Lit.N.: SPRU060D. Dostupné z: http://www.ti.com/lit/ug/spru060d/spru060d.pdf

[12] XDS 100 SW [software]. Verze 2. © Copyright 1995-2012 Texas Instruments Incorporated. [cit. 3.3.2012]. Velikost 7,32 MB. Download dostupný z: http://software-dl.ti.com/dsps/dsps_registered_sw/sdo_ccstudio/CCSv3/Drivers/XDS100_v1_1_RTM.exe

[13] Virtual COM Port Drivers [software]. Verze 2.08.24. © Future Technology Devices International Ltd. 2012. Aktualizováno 26.2.2012. [cit. 3.3.2012]. Download dostupný z: http://www.ftdichip.com/Drivers/VCP.htm

[14] MProg [software]. Verze 3.5. © Future Technology Devices International Ltd. 2012 [cit. 3.3.2012]. Velikost 1,6 MB. Download dostupný z: http://www.ftdichip.com/Support/Utilities/MProg3.5.zip

Seznam příloh

Příloha A. Schémata modelů

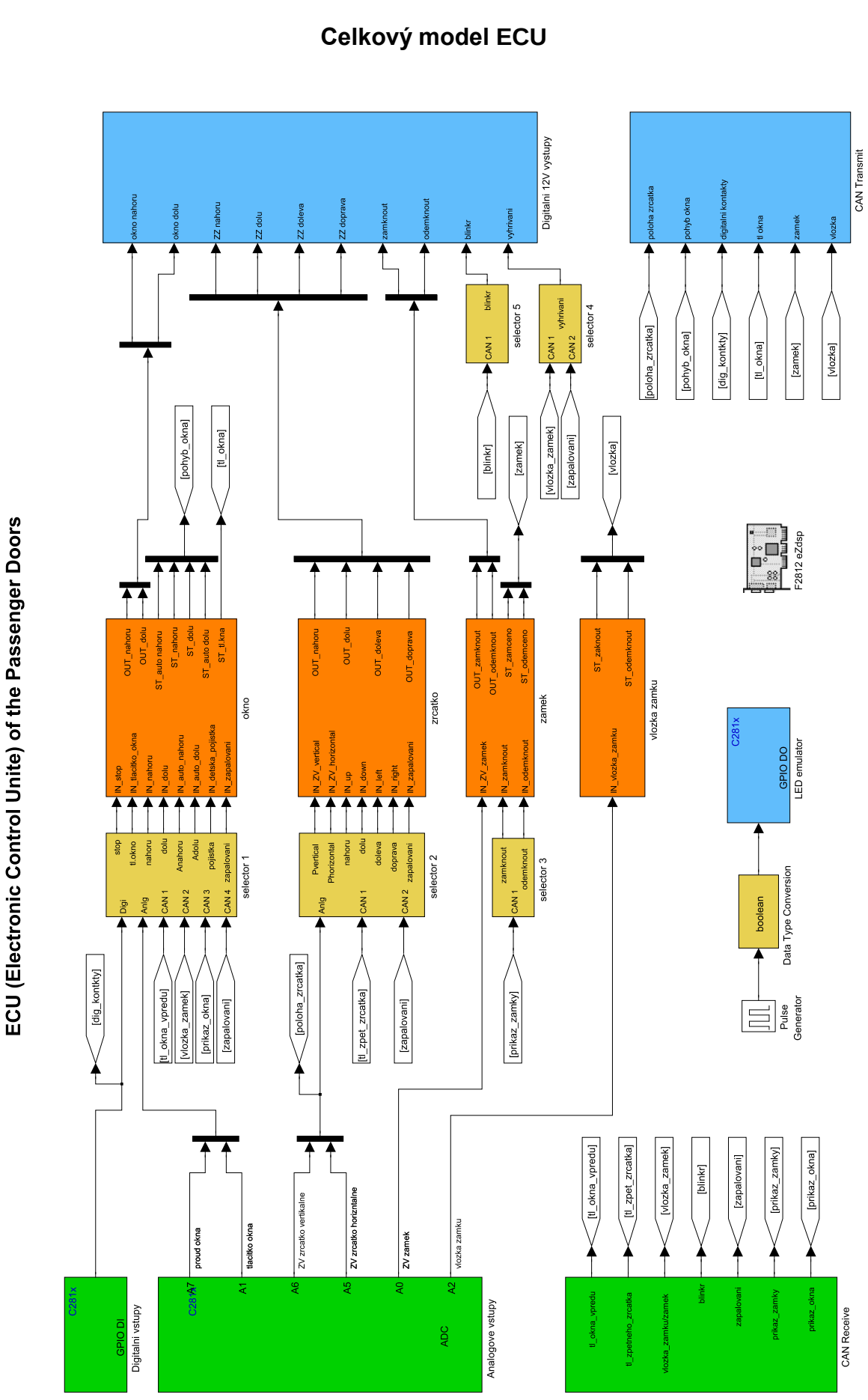
- Celkový model ECU
- CAN Receive
    - RJ ridice
    - centralni RJ
    - komfortni RJ
- okno
    - Subsystem
- zrcatko
- zamek
    - subsystem
    - monostabil
- vlozka zamku
- selector 1
- selector 2
- selector 3
- selector 4
- selector 5
- Digitalni 12V vystupy
- CAN Transmit
    - Rate Subsystem

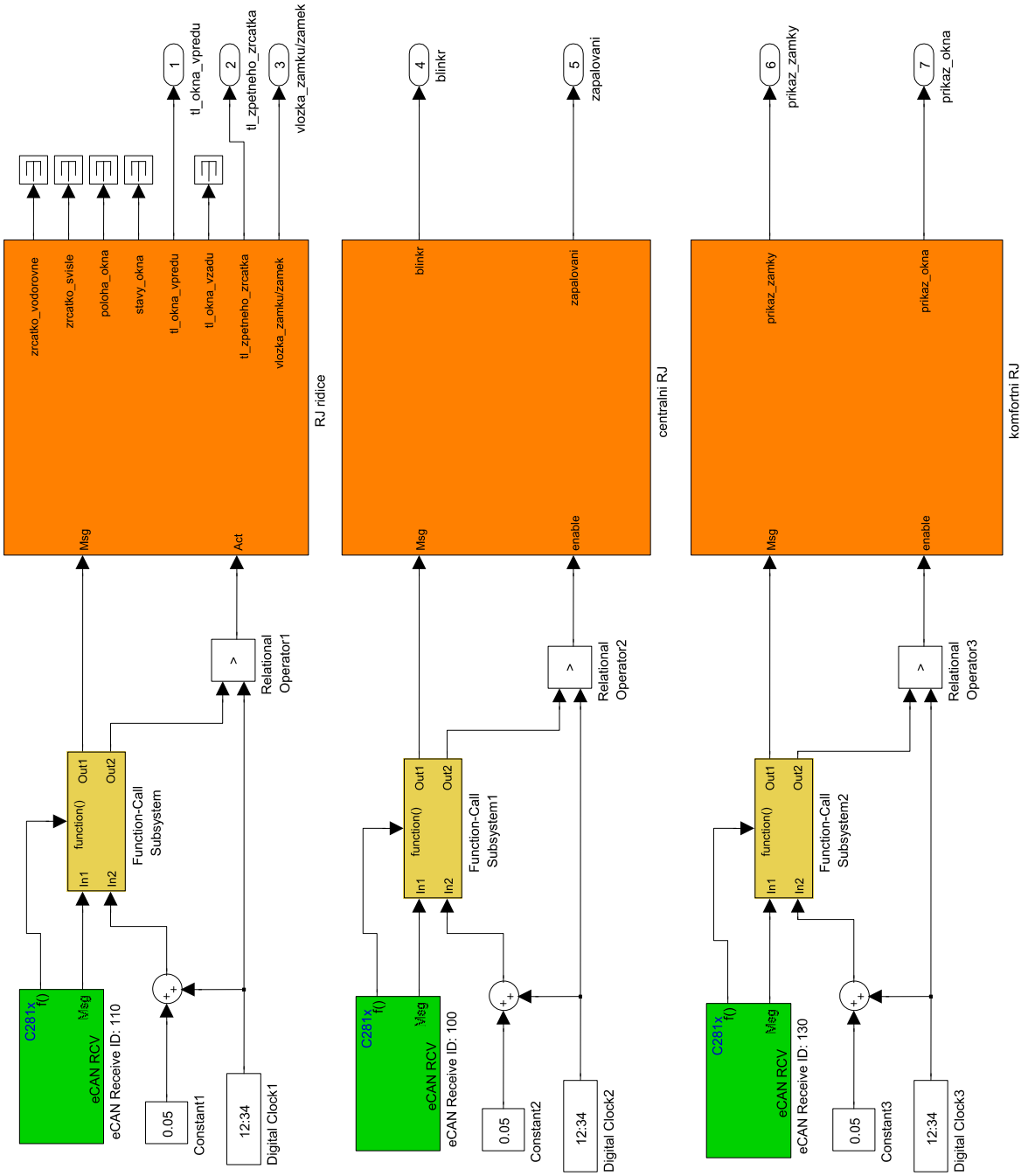
Příloha B. Tabulky CAN zpráv

- Centrální řídící jednotka
- Centrální řídící jednotka komfortní elektroniky
- Řídící jednotka dveří řidiče
- Řídící jednotka dveří spolujezdce

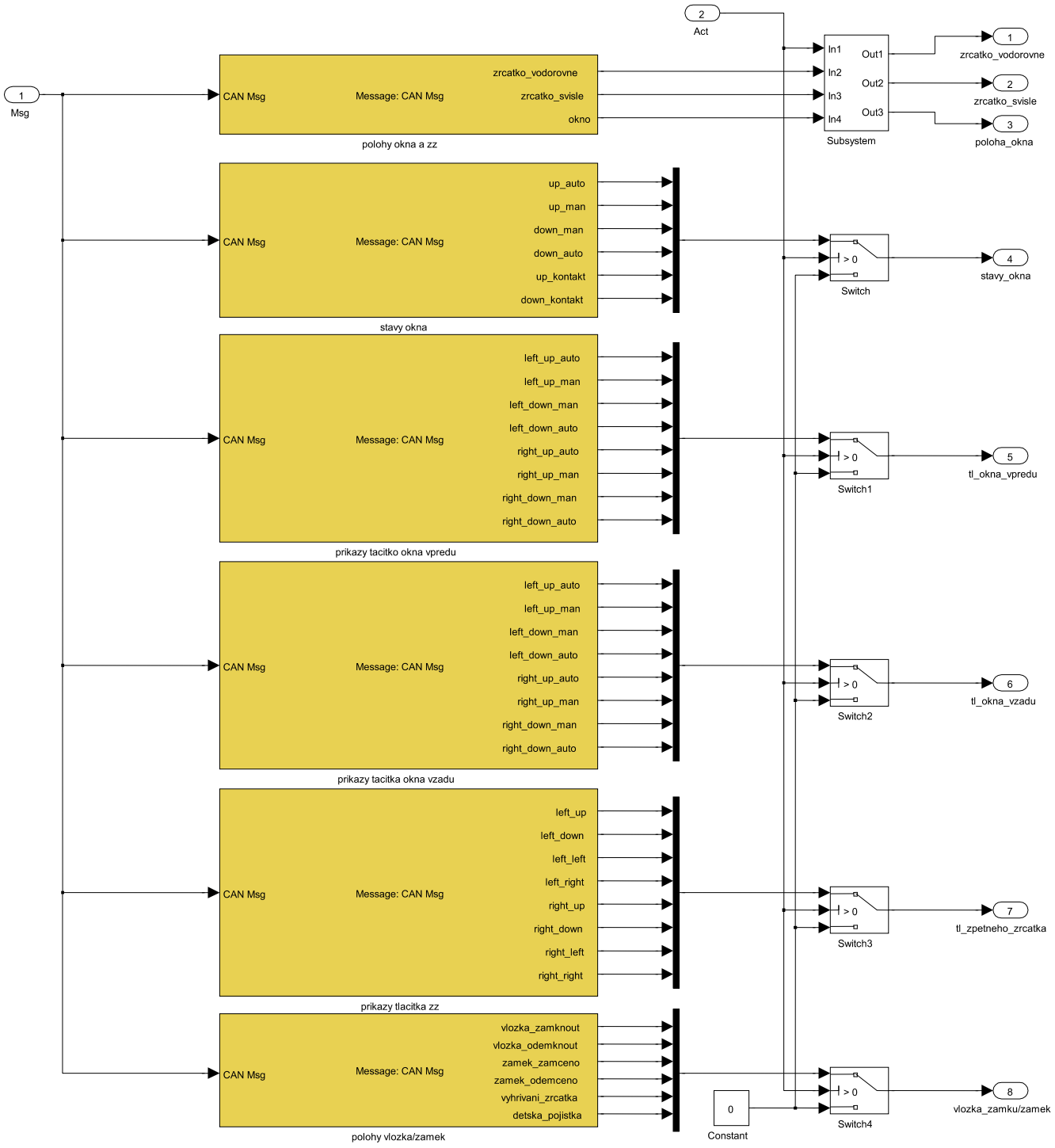
Příloha na CD

- model ECU
- model ECU_SML

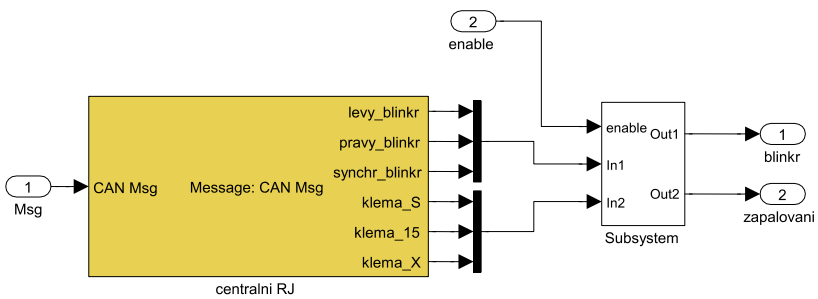

Celkový model ECU
ECU (Electronic Control Unite) of the Passenger Doors
okno nahoru
okno dolu
ZZ nahoru
ZZ dolu
ZZ doleva
ZZ doprava
zamknout
odemknout
blinkr
vyhrivani
Digitalni 12V vystupy
poloha zrcatka
pohyb okna
digitalni kontakty
tl okna
zamek
vlozka
CAN Transmit
selector 5
selector 4
[blinkr]
[zamek]
[vlozka_zamek]
[zapalovani]
[vlozka]
[poloha_zrcatka]
[pohyb_okna]
[dig_kontkty]
[tl_okna]
okno
zrcatko
zamek
vlozka zamku
F2812 eZdsp
C281x
GPIO DO
LED emulator
boolean
Data Type Conversion
Pulse Generator
selector 1
selector 2
selector 3
[tl_okna_vpredu]
[vlozka_zamek]
[prikaz_okna]
[zapalovani]
[poloha_zrcatka]
[tl_zpet_zrcatka]
[zapalovani]
[prikaz_zamky]
C281x
GPIO DI
Digitalni vstupy
C281x
ADC
Analogove vstupy
proud okna
tlacitko okna
ZV zrcatko vertikalne
ZV zrcatko horizntalne
ZV zamek
vlozka zamku
CAN Receive
tl_okna_vpredu
tl_zpeneho_zrcatka
vlozka_zamku/zamek
blinkr
zapalovani
prikaz_zamky
prikaz_okna


CAN Receive
zrcatko_vodorovne
zrcatko_svisle
poloha_okna
stavy_okna
tl_okna_vpredu
tl_okna_vzadu
tl_zpetneho_zrcatka
vlozka_zamku/zamek
1
tl_okna_vpredu
2
tl_zpetneho_zrcatka
3
vlozka_zamku/zamek
RJ ridice
Msg
Act
C281x
f()
Msg
eCAN RCV
eCAN Receive ID: 110
function()
Out1
Out2
In1
In2
Function-Call
Subsystem
0.05
Constant1
12:34
Digital Clock1
>
Relational
Operator1
blinkr
zapalovani
4
blinkr
5
zapalovani
centralni RJ
Msg
enable
C281x
f()
Msg
eCAN RCV
eCAN Receive ID: 100
function()
Out1
Out2
In1
In2
Function-Call
Subsystem1
0.05
Constant2
12:34
Digital Clock2
>
Relational
Operator2
prikaz_zamky
prikaz_okna
6
prikaz_zamky
7
prikaz_okna
komfortni RJ
Msg
enable
C281x
f()
Msg
eCAN RCV
eCAN Receive ID: 130
function()
Out1
Out2
In1
In2
Function-Call
Subsystem2
0.05
Constant3
12:34
Digital Clock3
>
Relational
Operator3

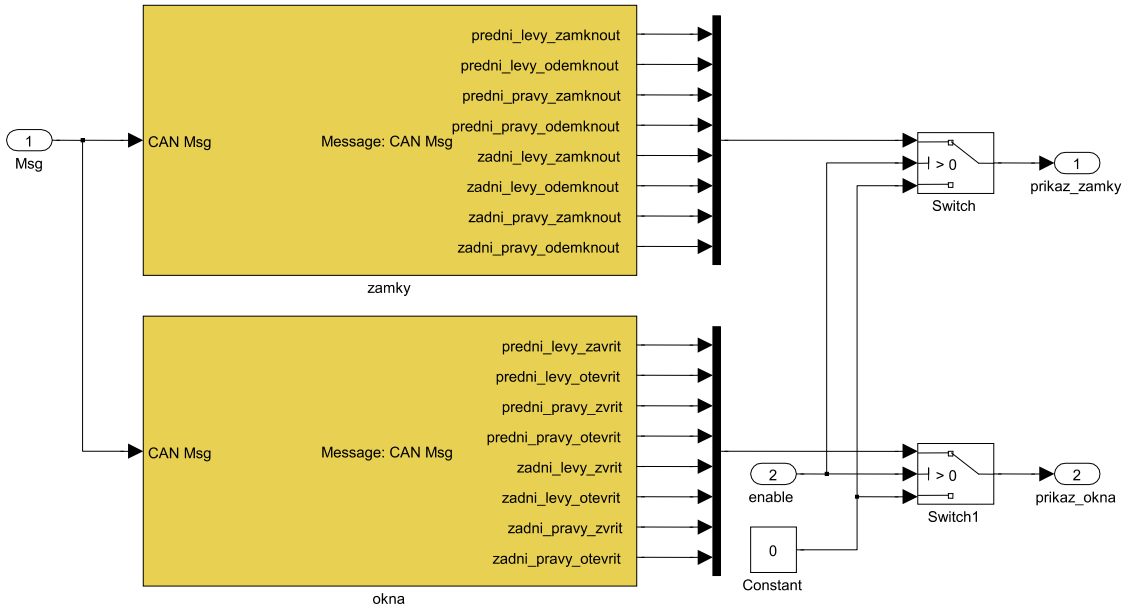
CAN Receive/RJ ridice



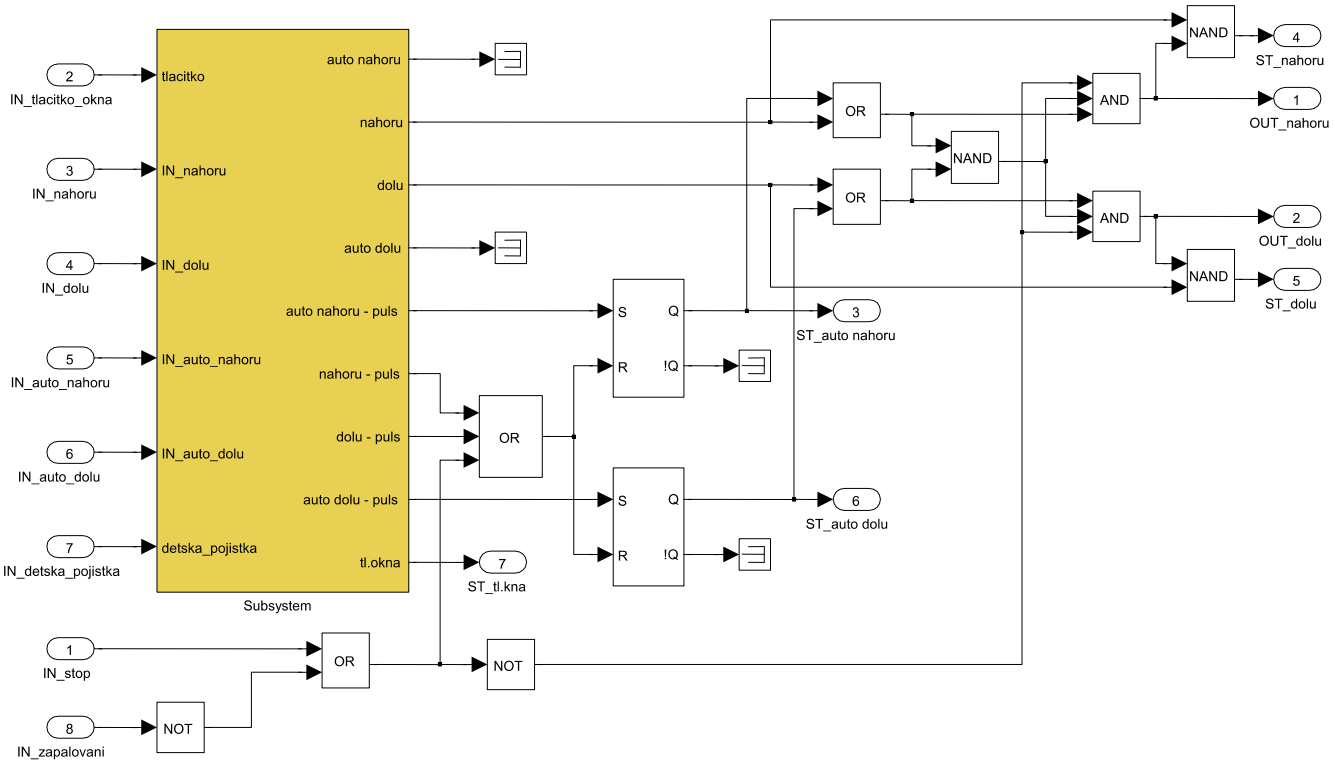
CAN Receive/centralni RJ

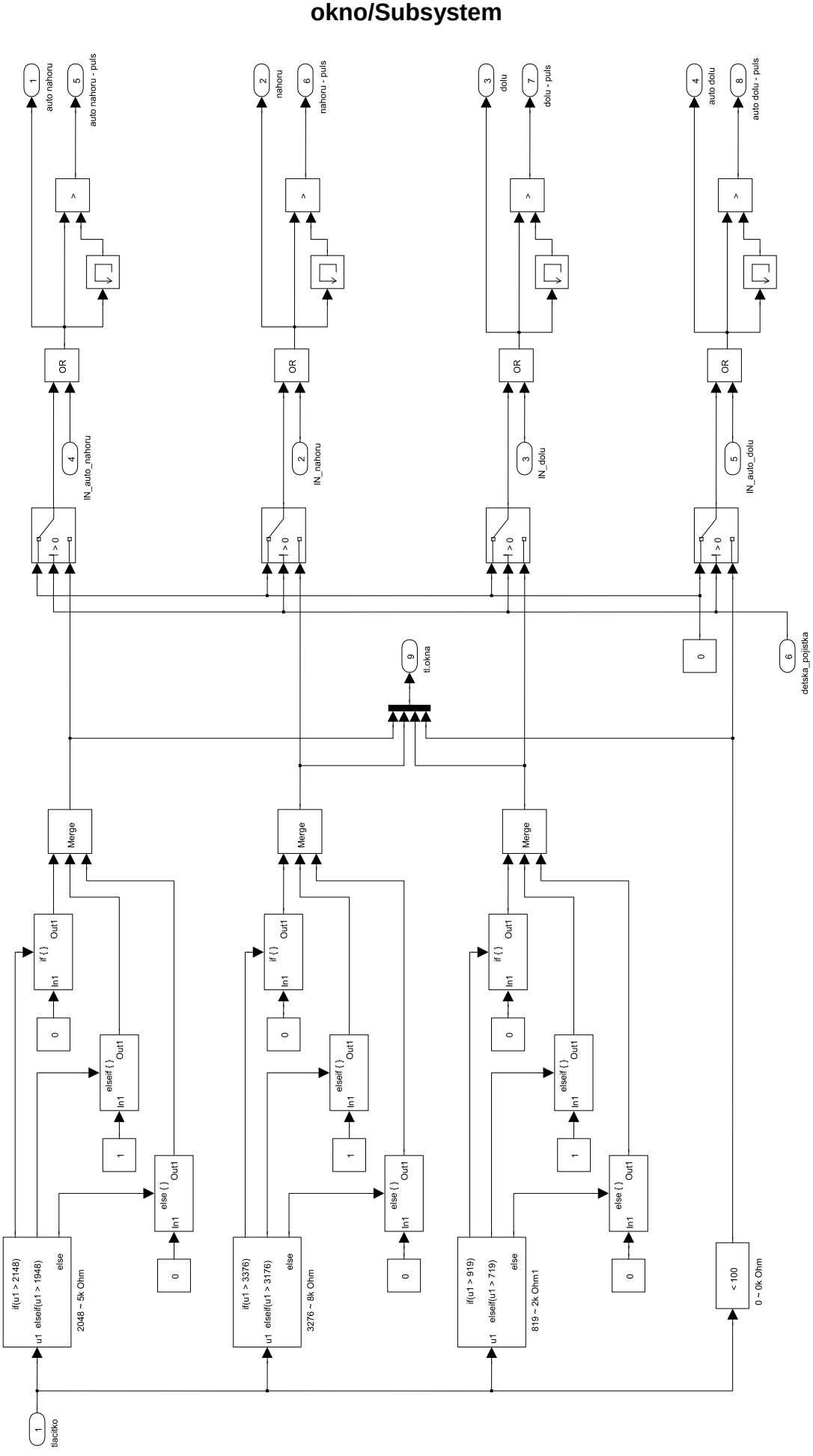


CEN Receive/ komfortni RJ

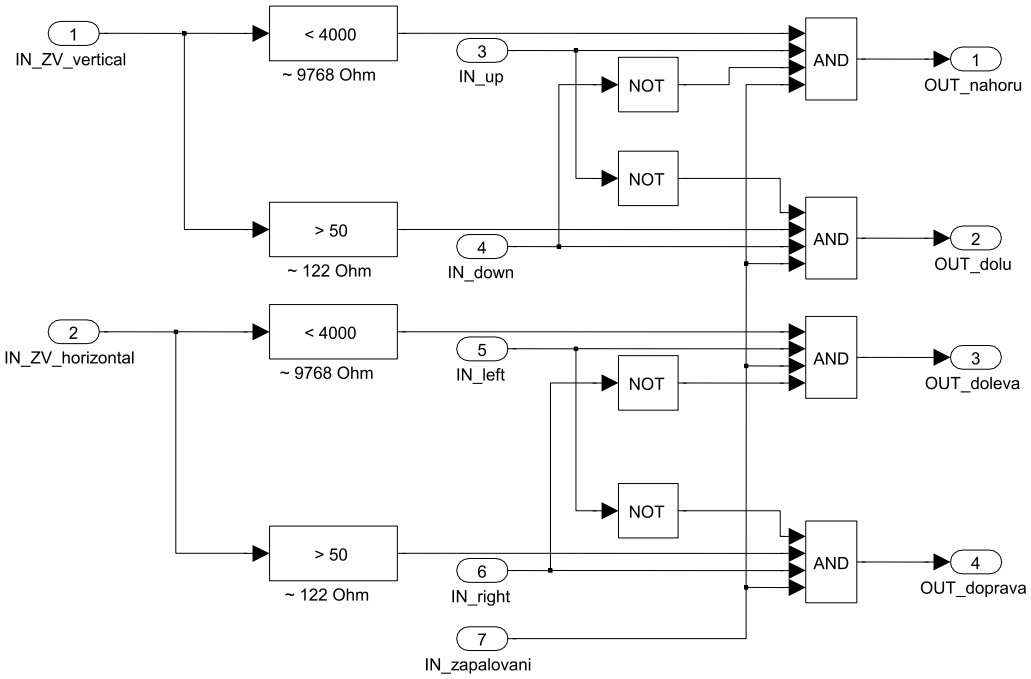


okno

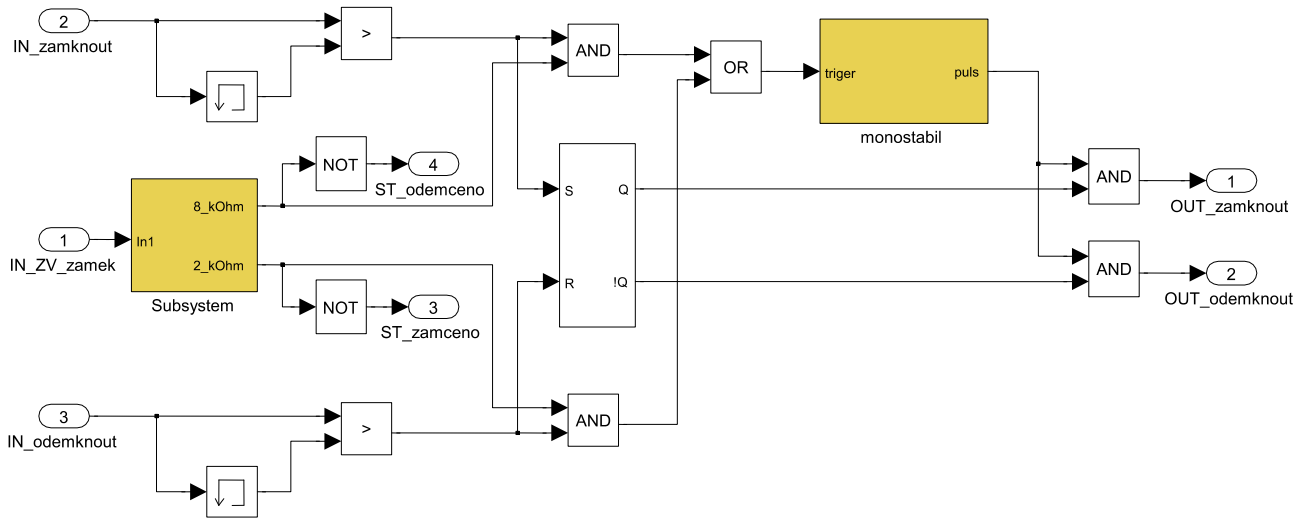



okno/Subsystem
1
tlaciitko
if(u1 > 2148)
u1 elseif(u1 > 1948)
else
2048 ~ 5k Ohm
if(u1 > 3376)
u1 elseif(u1 > 3176)
else
3276 ~ 8k Ohm
if(u1 > 919)
u1 elseif(u1 > 719)
else
819 ~ 2k Ohm1
< 100
0 ~ 0k Ohm
if { }
elseif { }
else { }
In1
Out1
Merge
0
1
9
tl.okna
6
detska_pojistka
> 0
OR
4
IN_auto_nahoru
2
IN_nahoru
3
IN_dolu
5
IN_auto_dolu
1
auto nahoru
5
auto nahoru - puls
2
nahoru
6
nahoru - puls
3
dolu
7
dolu - puls
4
auto dolu
8
auto dolu - puls
^

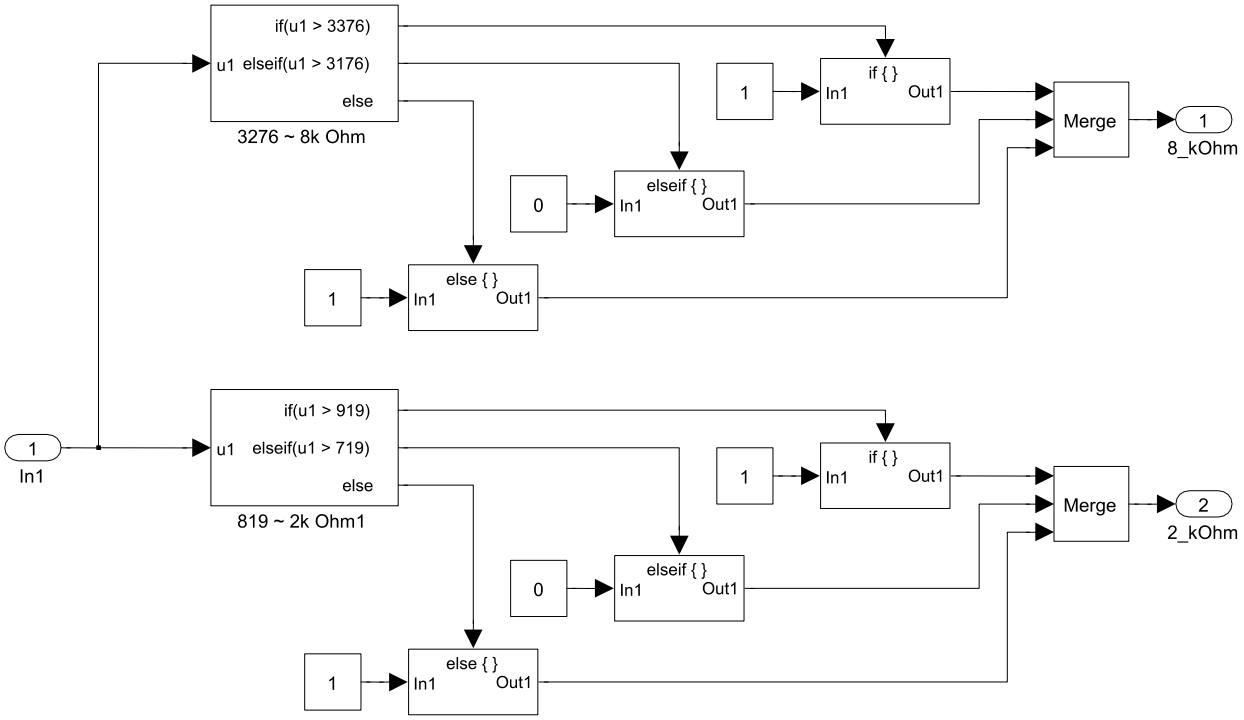
zrcatko



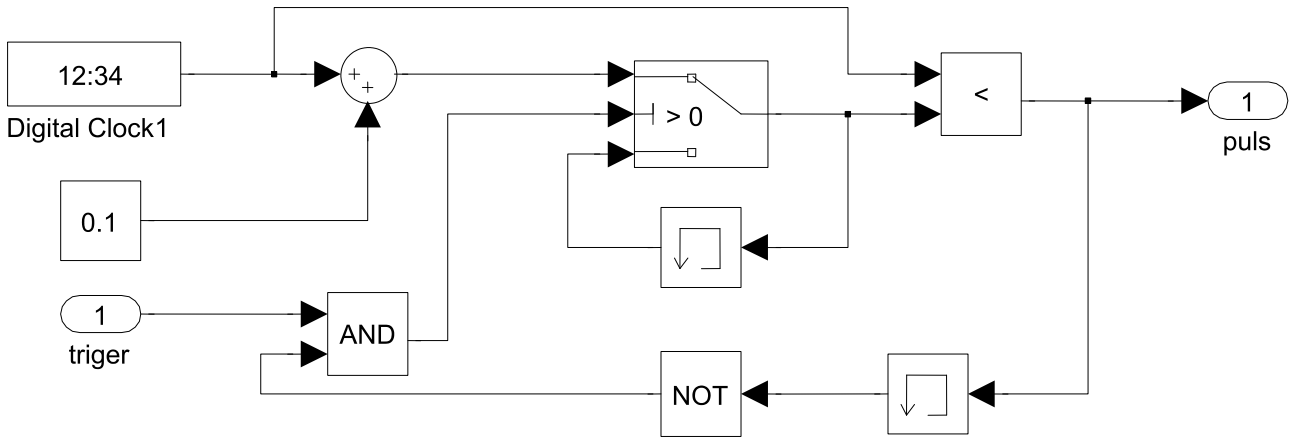
zamek



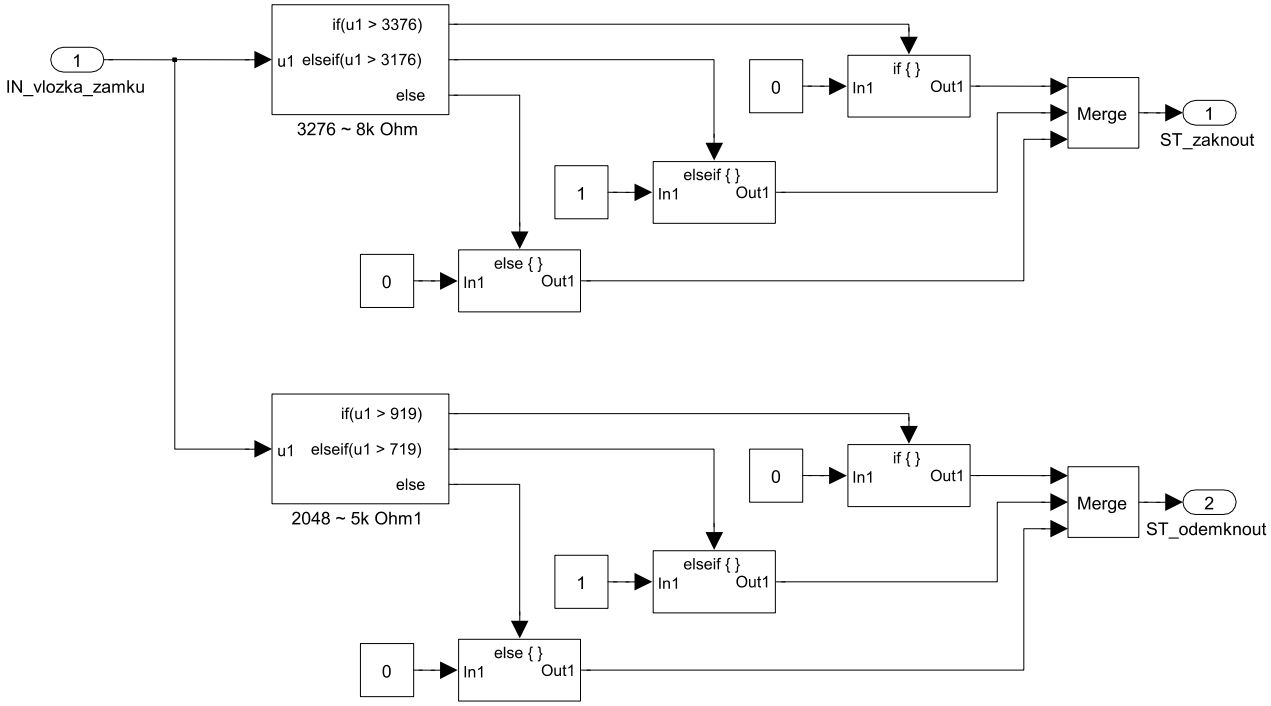
zamek/Subsystem



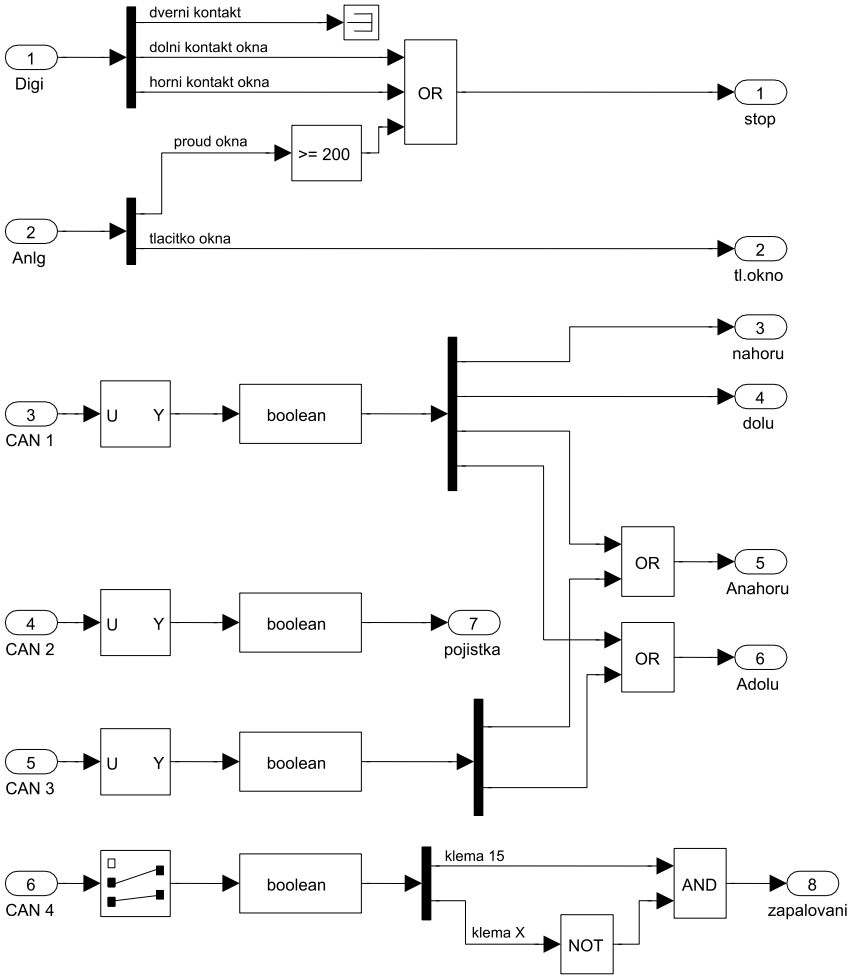
zamek/monostabil



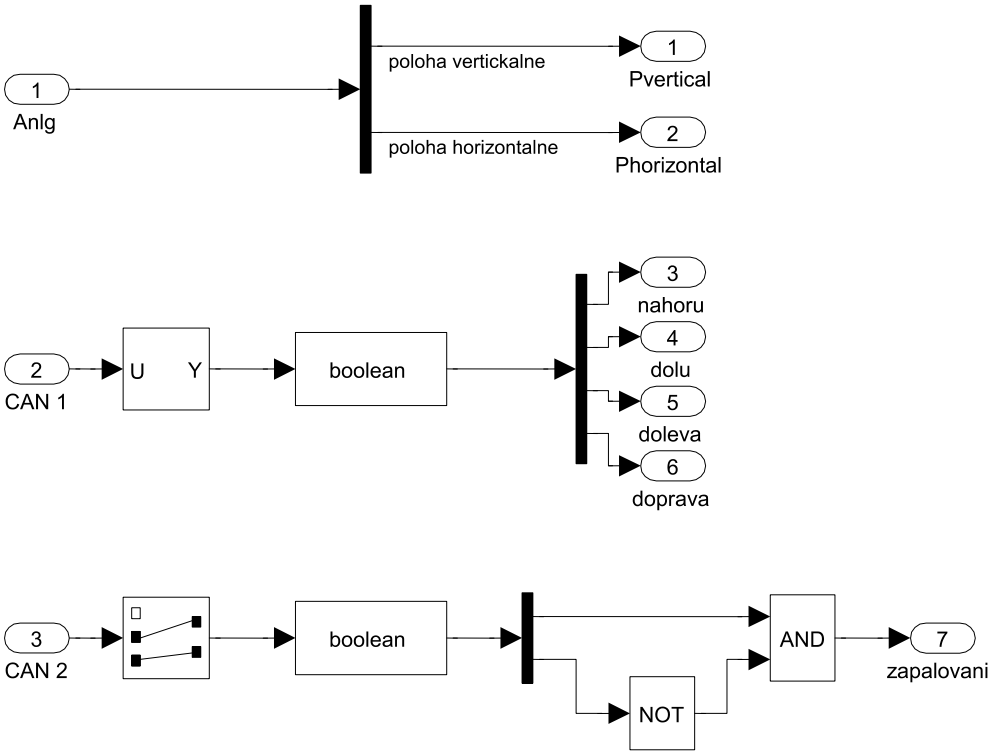
vlozka zamku



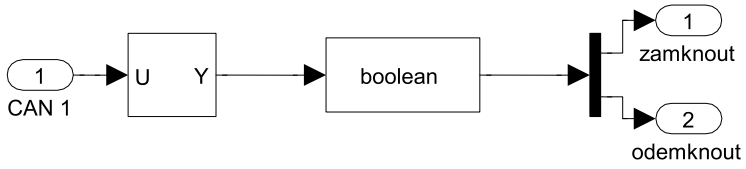
selektor1



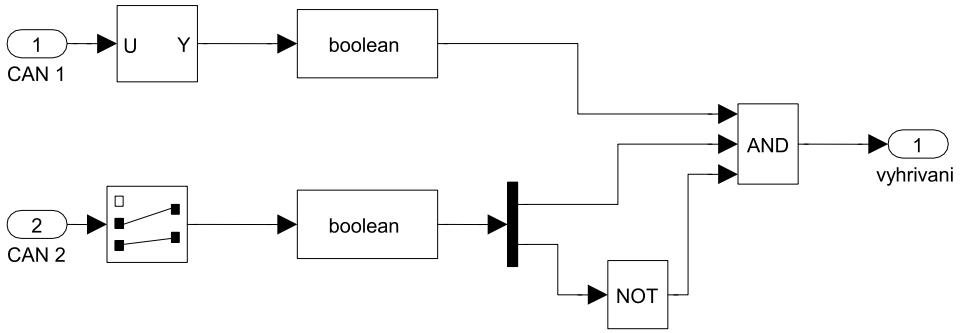
selector2

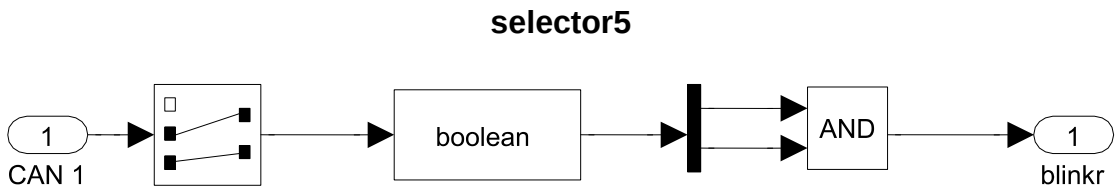


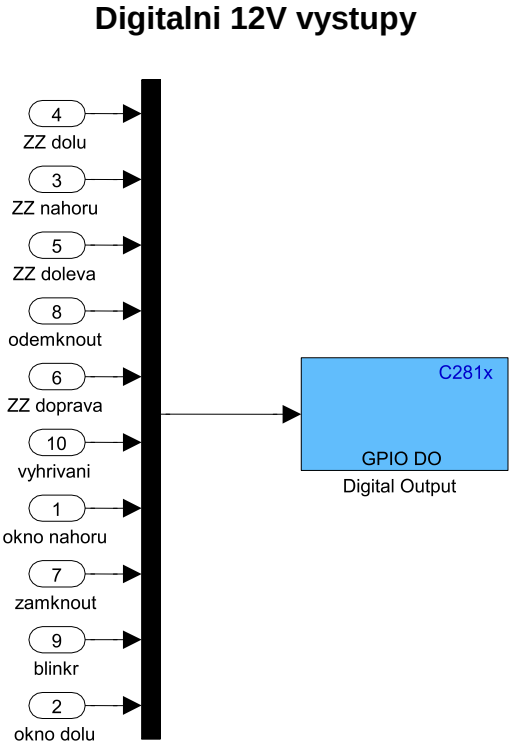
selector3

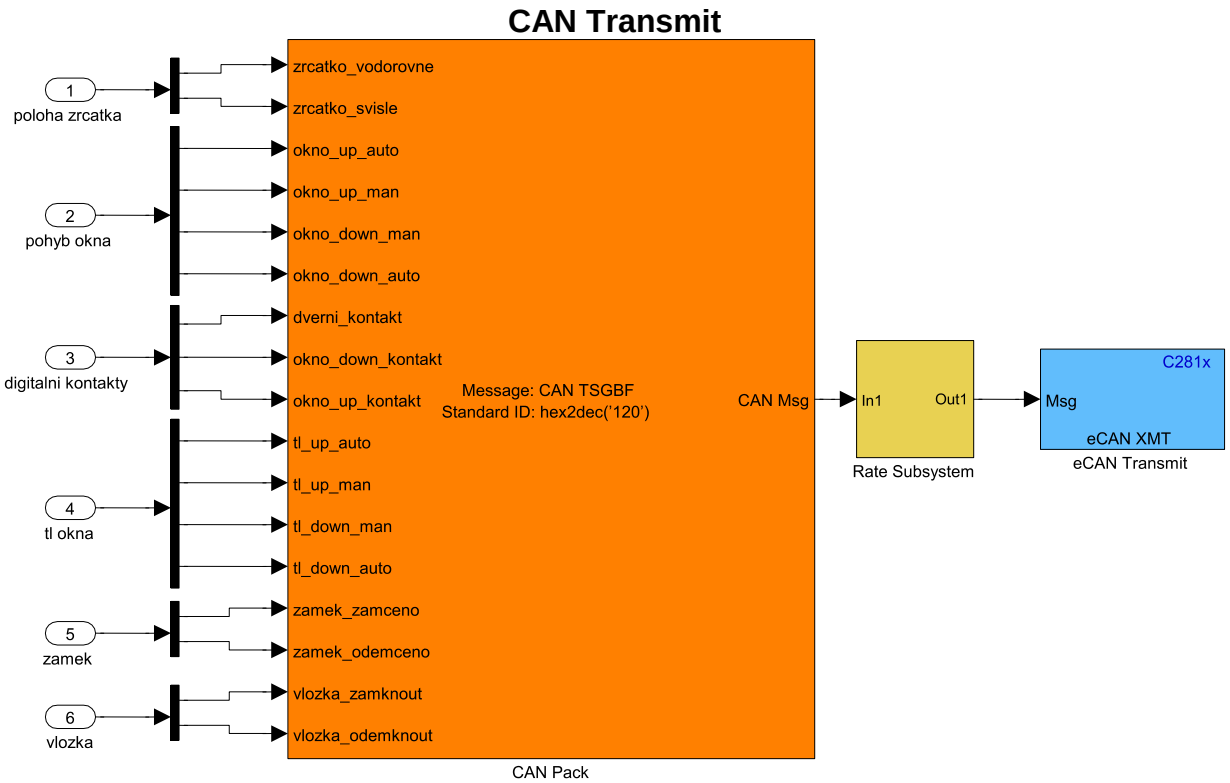


selector4

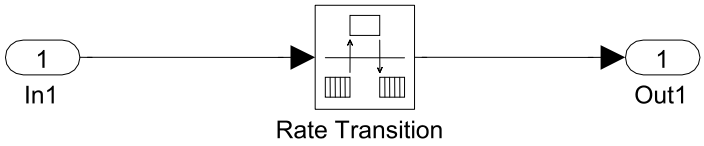



selector5
1
CAN 1
boolean
AND
1
blinkr


Digitalni 12V vystupy
4
ZZ dolu
3
ZZ nahoru
5
ZZ doleva
8
odemknout
6
ZZ doprava
10
vyhrivani
1
okno nahoru
7
zamknout
9
blinkr
2
okno dolu
C281x
GPIO DO
Digital Output


CAN Transmit
1
poloha zrcatka
2
pohyb okna
3
digitalni kontakty
4
tl okna
5
zamek
6
vlozka
zrcatko_vodorovne
zrcatko_svisle
okno_up_auto
okno_up_man
okno_down_man
okno_down_auto
dverni_kontakt
okno_down_kontakt
okno_up_kontakt
tl_up_auto
tl_up_man
tl_down_man
tl_down_auto
zamek_zamceno
zamek_odemceno
vlozka_zamknout
vlozka_odemknout
Message: CAN TSGBF
Standard ID: hex2dec('120')
CAN Pack
CAN Msg
In1
Out1
Rate Subsystem
C281x
Msg
eCAN XMT
eCAN Transmit

CAN Transmit/Rate Subsystem



Centrální řídící jednotka (ID: 100)

byte	bit	CAN bit	skupina	funkce
0	**0**	0	blinkr	**levy**
	1	1		**pravy**
	2	2		**synchro**
	3	3	zapalovani	**klema S (vlozeny klic)**
	4	4		**klema 15 (bezi motr)**
	5	5		**klema X (startovani)**
	6	6		
	7	7		

Centrální řídící jednotka komfortní elektroniky (ID: 130)

byte	bit	CAN bit	skupina	funkce
0	**0**	0	predni zamky	**zamknout - levy**
	1	1		**odemknout - levy**
	2	2		**zamknout - pravy**
	3	3		**odemknout - pravy**
	4	4	zadni zamky	**zamknout - levy**
	5	5		**odemknout - levy**
	6	6		**zamknout - pravy**
	7	7		**odemknout - pravy**
1	**0**	8	predni okna	**zavrit - levy**
	1	9		**otevrit - levy**
	2	10		**zavrit - pravy**
	3	11		**otevrit - pravy**
	4	12	zadni okna	**zavrit - levy**
	5	13		**otevrit - levy**
	6	14		**zavrit - pravy**
	7	15		**otevrit - pravy**

Řídící jednotka dveří řidiče (ID: 110)

byte	bit	CAN bit	skupina	funkce
0	0	0		poloha zrcatka - vodorovne
	1	1		
	2	2		
	3	3		
	4	4		
	5	5		
	6	6		
	7	7		
1	0	8		poloha zrcatka - svisle
	1	9		
	2	10		
	3	11		
	4	12		
	5	13		
	6	14		
	7	15		
2	0	16		
	1	17		
	2	18		
	3	19		
	4	20		
	5	21		
	6	22		
	7	23		
3	0	24	stavy okna	jede nahoru - Auto
	1	25		jede nahoru - Man
	2	26		jede dolu - Man
	3	27		jede dolu - Auto
	4	28		horni kontakt
	5	29		dolni kontakt
	6	30	stav dveri	dverni kontat
	7	31		
4	0	32	tl. pro okno vpredu	leve - nahoru - Auto
	1	33		leve - nahoru - Man
	2	34		leve - dolu - Man
	3	35		leve - dolu - Auto
	4	36		prave - nahoru - Auto
	5	37		prave - nahoru - Man
	6	38		prave - dolu - Man
	7	39		prave - dolu - Auto
5	0	40	tl. pro okno vzadu	leve - nahoru - Auto
	1	41		leve - nahoru - Man
	2	42		leve - dolu - Man
	3	43		leve - dolu - Auto
	4	44		prave - nahoru - Auto
	5	45		prave - nahoru - Man
	6	46		prave - dolu - Man
	7	47		prave - dolu - Auto
6	0	48	tl. pro zrcatko vlevo	nahoru
	1	49		dolu
	2	50		doleva
	3	51		doprava
	4	52	tl. pro zrcatko vpravo	nahoru
	5	53		dolu
	6	54		doleva
	7	55		doprava
7	0	56	vlozka zamku	zamknout
	1	57		odemknout
	2	58	stav zamku	zamceno
	3	59		odemceno
	4	60		vyhrivani zrcatka
	5	61	tl. pro detskou pojistku	detska pojistka
	6	62		
	7	63		

Řídící jednotka dveří spolujezdce (ID: 120)

byte	bit	CAN bit	skupina	funkce
0	0	0		poloha zrcatka - vodorovne
	1	1		
	2	2		
	3	3		
	4	4		
	5	5		
	6	6		
	7	7		
1	0	8		poloha zrcatka - svisle
	1	9		
	2	10		
	3	11		
	4	12		
	5	13		
	6	14		
	7	15		
2	0	16		
	1	17		
	2	18		
	3	19		
	4	20		
	5	21		
	6	22		
	7	23		
3	0	24	stavy okna	jede nahoru - Auto
	1	25		jede nahoru - Man
	2	26		jede dolu - Man
	3	27		jede dolu - Auto
	4	28		horni kontakt
	5	29		dolni kontakt
	6	30	stav dveri	dverni kontat
	7	31		
4	0	32		
	1	33		
	2	34		
	3	35		
	4	36	tl. pro okno vpredu	prave - nahoru - Auto
	5	37		prave - nahoru - Man
	6	38		prave - dolu - Man
	7	39		prave - dolu - Auto
5	0	40		
	1	41		
	2	42		
	3	43		
	4	44		
	5	45		
	6	46		
	7	47		
6	0	48		
	1	49		
	2	50		
	3	51		
	4	52		
	5	53		
	6	54		
	7	55		
7	0	56	vlozka zamku	zamknout
	1	57		odemknout
	2	58	stav zamku	zamceno
	3	59		odemceno
	4	60		
	5	61		
	6	62		
	7	63		