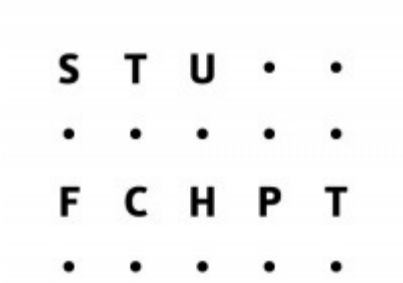
SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE

FAKULTA CHEMICKEJ A POTRAVINÁRSKEJ TECHNOLÓGIE

Ústav informatizácie, automatizácie a matematiky



RIADENIE LEGO ROBOTOV

BAKALÁRSKA PRÁCA

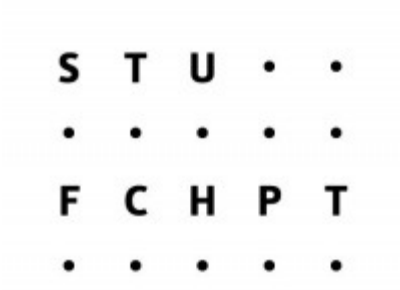
FCHPT-5415-50950

2010 **Bálint Takács**

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE

FAKULTA CHEMICKEJ A POTRAVINÁRSKEJ TECHNOLÓGIE

Ústav informatizácie, automatizácie a matematiky



RIADENIE LEGO ROBOTOV

BAKALÁRSKA PRÁCA

FCHPT-5415-50950

Študijný program: Automatizácia, informatizácia a manažment v chémii a potravinárstve
Číslo a názov študijného odboru: 5.2.14 automatizácia, 5.2.25 priemyselné inžinierstvo
Školiace pracovisko: Radlinského 9, 812 37 Bratislava
Vedúci záverečnej práce/školiteľ: Ing. Michal Kvasnica, PhD.

2010 **Bálint Takács**

Čestné vyhlásenie

Týmto podpisom potvrdzujem, že som túto prácu vypracoval v spolupráci so spolužiakom Jurajom Holazom. Mojím majoritným prínosom je pritom kapitola č. 5.

. . . . . .

Bálint Takács

Slovenská technická univerzita v Bratislave
Oddelenie informatizácie a riadenia procesov

Fakulta chemickej a potravinárskej technológie
Akademický rok: 2009/2010
Evidenčné číslo: FCHPT-5415-50950

S T U
F C H P T

ZADANIE BAKALÁRSKEJ PRÁCE

Študent:	**Bálint Takács**
ID študenta:	50950
Študijný program:	automatizácia, informatizácia a manažment v chémii a potravinárstve
Kombinácia študijných odborov:	5.2.14 automatizácia, 5.2.52 priemyselné inžinierstvo
Vedúci práce:	Ing. Michal Kvasnica, PhD.

Názov práce: **Riadenie LEGO robotov**

Špecifikácia zadania:

LEGO Mindstorms NXT Stavebnica s programovateľnou NXT kockou a rozličnými senzormi je zaujímavý nástroj na programovanie a navrhovanie rôznych druhov robotov na viaceré predmety – riadenie procesov, projektovanie riadiacich systémov a iné. Študenti môžu aplikovať svoje znalosti pri stavaní múdrych robotov, zbieraní dát a prezentácii nameraných výsledkov. Takto sú motivovaní riešiť praktické problémy či vytvárať a overovať vlastné námety.

Pomocou tejto stavebnice sa dajú jednoducho vytvárať robotov či vozidlá, ktoré si samé dokážu nájsť cestu, zaparkovať, prekonať prekážku či zbierať predmety roztrúsené po miestnosti. Mechanizmy tiež dokážu reagovať na hlasové povely, čo sa dá využiť napríklad pre vzájomnú komunikáciu a koordináciu viacerých robotov.

Ciele práce: preskúmanie možností riadenia robotov, vytvorenie algoritmu na ovládanie robotického auta.

Rozsah práce: 30

Riešenie zadania práce od: 15. 02. 2010

Dátum odovzdania práce: 22. 05. 2010

L. S.

Bálint Takács
Študent

prof. Ing. Miroslav Fikar, DrSc.
Vedúci pracoviska

prof. Ing. Miroslav Fikar, DrSc.
Garant študijného programu

ABSTRAKT

Hlavným cieľom prace je overiť možnosti využitia softvéru Stateflow, ako aj jeho následnou aplikáciou na reálnych objektoch zhotovených z lega a použitím techniky LEGO Mindstorms NXT. V prvých dvoch kapitolách sa oboznámime s možnosťami, ktoré nám ponúka LEGO Mindstorms NXT, ako aj s prácou v užívateľskom prostredím softvéru Stateflow. Princíp stavového riadenia je vysvetlené od základov, pomocou zrozumiteľných príkladov. Výsledkom práce je otestovať a dokázať správnosť navrhnutého algoritmu na riadenie vopred určeného objektu.

Kľúčové slová: MATLAB, Simulink, Stateflow, LEGO Mindstorms NXT.

ABSTRACT

The aim of this bachelor thesis is to verify the possibilities of using Stateflow software, as well as its subsequent application of real objects made of Lego using the LEGO Mindstorms NXT technology. The first two chapters are dealing with options that the LEGO Mindstorms NXT offers, as well as work in the user environment Stateflow software. The principle of state control is explained from the basics, with simple examples. The result of the work is to test and prove the correctness of the proposed algorithm to control a predetermined object.

Key words: MATLAB, Simulink, Stateflow, LEGO Mindstorms NXT.

Poďakovanie

Chcel by som poďakovať vedúcemu bakalárskeho projektu Ing. Michalovi Kvasnicovi, PhD. za odborné vedenie pripomienky a cenné rady.

Obsah

Úvod

Automatizácia ma čoraz väčšiu úlohu v našom živote. Jej cieľom je vytvárať riadiace systémy, ktoré uľahčujú prácu človeka. Tento vedný odpor sa uplatňuje všade, kde je potrebné zabezpečiť prácu so strojmi, robotmi a inými zariadeniami. Dobrým príkladom je automobilový priemysel, ktorý využíva mechanické ramená na urýchlenie výrobného procesu. Takto dokážeme odbremeniť ľudskú pracovnú silu od náročných a nebezpečných úkonov.

Na riešenie úlohy použijeme princíp stavového riadenia, ktoré má široké využitie. Na základe tohto typu riadenia sa môžu riadiť jednoduché systémy, s ktorými sa môžeme stretnúť v každodennom živote (semafory, rampy, výťahy, bojlery, ...), ako aj zložitejšie priemyselné procesy (plnenie fliaš pohybujúcich sa po páse, zváracie roboty v automobilovom priemysle, ...). Jeden zo softvérov, ktorý je vhodný na tvorbu algoritmov založených na báze stavového riadenia je Stateflow, ktorý je súčasťou programu MATLAB. Jeho užívateľské prostredie je založené na grafickom zobrazovaní stavov, čím umožňuje ľahkú prácu vhodnú aj pre začiatočníkov.

Lego Mindstorms NXT predstavuje novú generáciu vo svete legorobotov. Poskytuje nám široký výber senzorov a akčných členov, pomocou ktorých sa dajú vytvoriť modely na vykonanie požadovaných úkonov. Ich riadiacim „mozgom“ je 32 bitový mikroprocesor, zabudovaný v inteligentnej kocke NXT.

1 Lego Mindstorms NXT

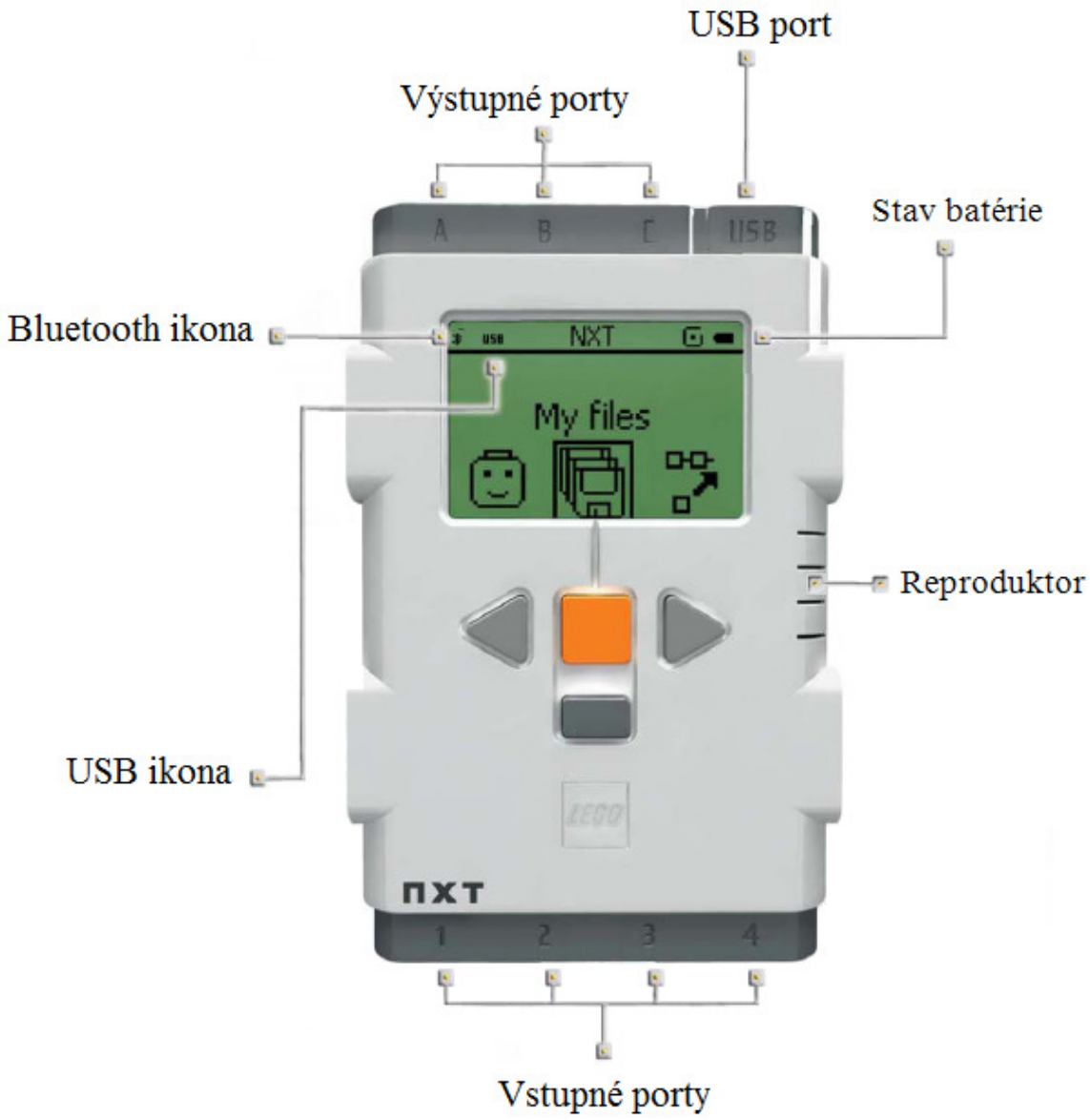
Lego Mindstorms NXT predstavuje novú generáciu v robotickom odvetví Lego. Je vhodné pre edukačné účely vo svete vedy, techniky a matematiky. Vedie k tvorivému mysleniu a riešeniu jednoduchých, ako aj zložitých úloh. [1]

Súprava, ktorú máme k dispozícii, pozostáva z dvoch oddelení. Na prvom sa nachádzajú Lego súčiastky, pomocou ktorých sme schopný zostrojiť reálny model. V druhej časti vieme nájsť okrem kocky NXT aj senzory, akčné členy a prepájacie káble. V prípade potreby môžeme potrebné súčiastky, ktoré táto súprava neobsahuje, samostatne dokúpiť.

Obr. 1 Stavebnica lego

1.1 Kocka NXT

Kocka je samostatnou riadiacou jednotkou, ktorá je vybavená s vlastným zdrojom energie. Ako zdroj energie pre kocku NXT sa používa 6 kusov batérii typu AA, alebo akumulátor, ktorý je možne dobiť pomocou adaptéra. Na pripojenie senzorov slúžia 4 porty na spodnej strane kocky, ktoré sú očíslované od jedna po štyri. Na vrchnej časti kocky sa nachádzajú výstupy pre akčné členy (označené písmenami A,B a C) ako aj pre USB kábel. Na pravej strane sa nachádza reproduktor. Pod displejom sú tlačidlá, ktorými dokážeme ovládať kocku. Na zapnutie slúži oranžové tlačidlo. Na orientáciu v menu slúžia bočné šípky. Vypnúť sa dá pomocou spodného obdĺžnika. [1]



Obr. 2 Kocka NXT

1.2 Senzory

Dotykový senzor (tlačidlo) - vracia nám hodnotu 0 ak tlačidlo nie je stlačené a 1 keď tlačidlo je stlačené.

Ultrazvukový senzor - Používa sa na meranie vzdialenosti na základe ultrazvukových vĺn.

Svetelný senzor - Slúži na rozoznávanie farieb pomocou rozličnej intenzity svetla.

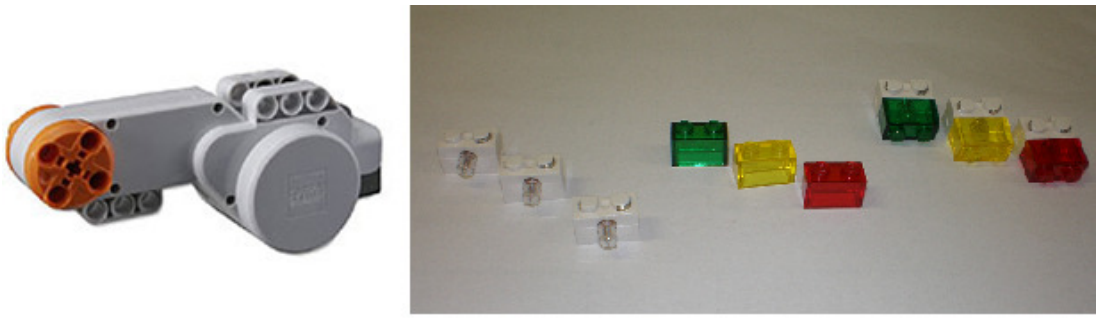
Zvukový senzor - Zaznamenáva intenzitu zvuku.

Obr. 3 Senzory

1.3 Akčné členy

Motor - môže sa točiť vo dvoch smeroch, jeden je kladný a druhý je záporný. Na rýchlosť otáčanie motora sa zadávajú čísla z intervalu 0 až 100. Môžeme merať aj uhol motora, ktorý sa využíva v podmienkach stavového riadenia.

Žiarovka – slúži na vydávanie svetelných signálov.



Obr. 4 Akčné členy

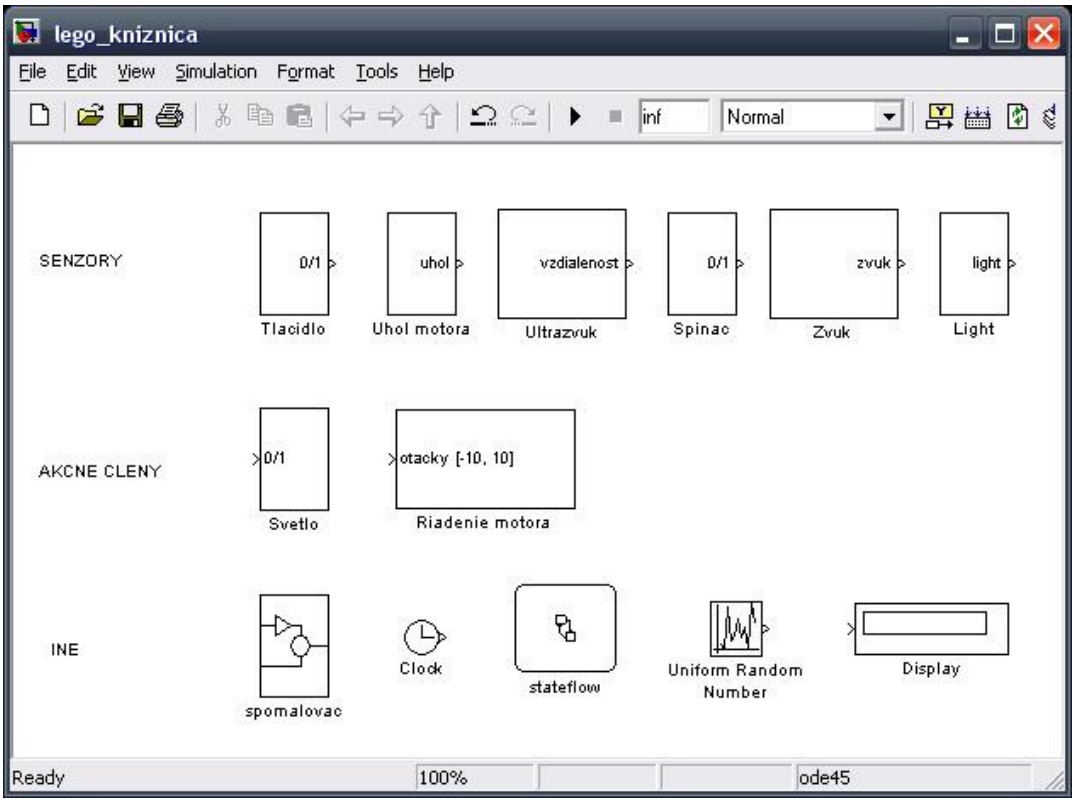
2 Software

MATLAB je neustále sa vyvíjajúci program, ktorý sa využíva aj pri riešení náročných technických problémov. Jeho priateľské prostredie nám uľahčuje prácu pri vytváraní algoritmov, vizualizácie, simulácie a numerických výpočtov. Nadstavbou MATLAB-u je Simulink. [2]

2.1 Simulink

Simulink slúži na simuláciu a modelovanie dynamických systémov. Pozostáva zo súboru knižníc, ktoré sa dajú v prípade potreby rozširovať (lego_kniznica). Každá z nich obsahuje špecifické bloky, pomocou ktorých vieme vytvoriť blokové schémy. Jednotlivé bloky, ktoré obsahuje lego knižnica, môžeme rozdeliť do troch skupín.

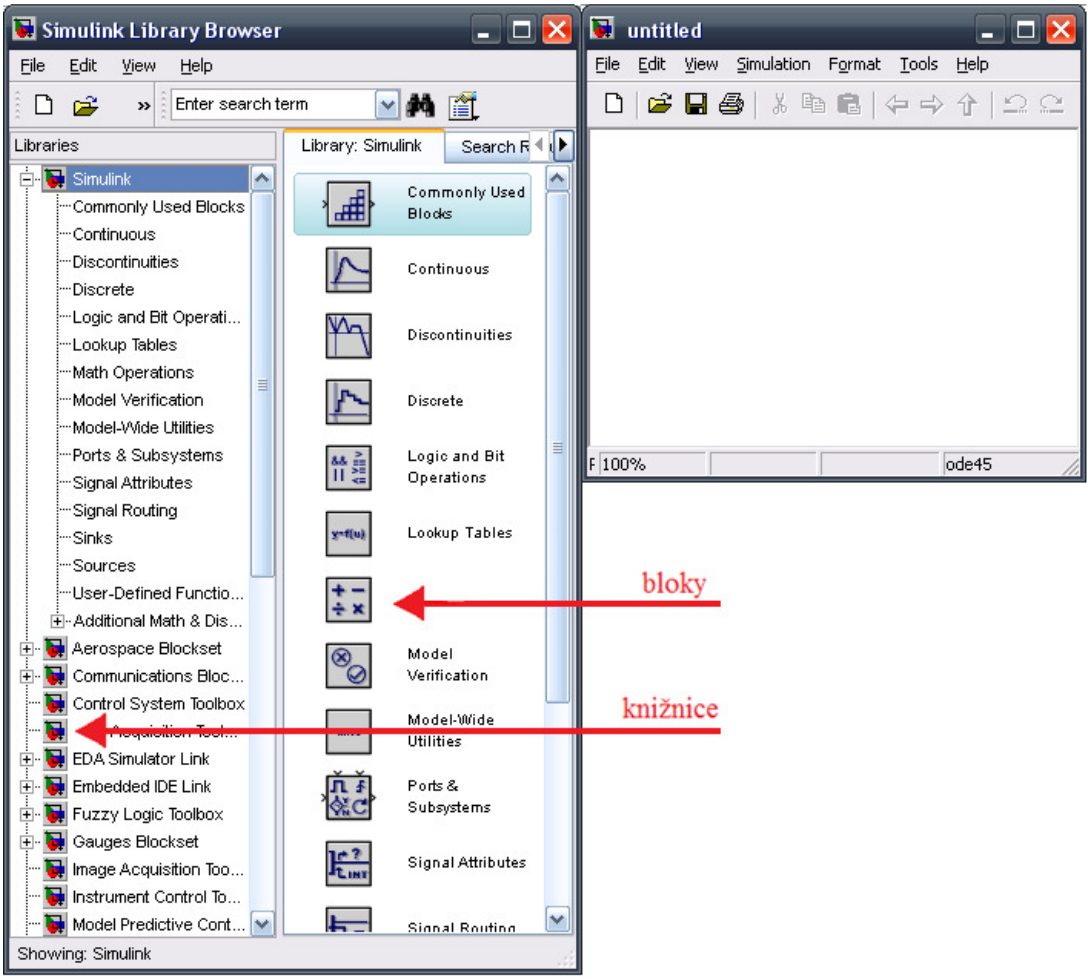
- **Senzory**: tlačidlo, uhol motora, ultrazvuk, light, spínač, zvuk.
- **Akčné členy**: riadenie motora, svetlo
- **Iné**: spomaľovač, stateflow, clock, display, uniform random number.



Obr. 5 Lego knižnica

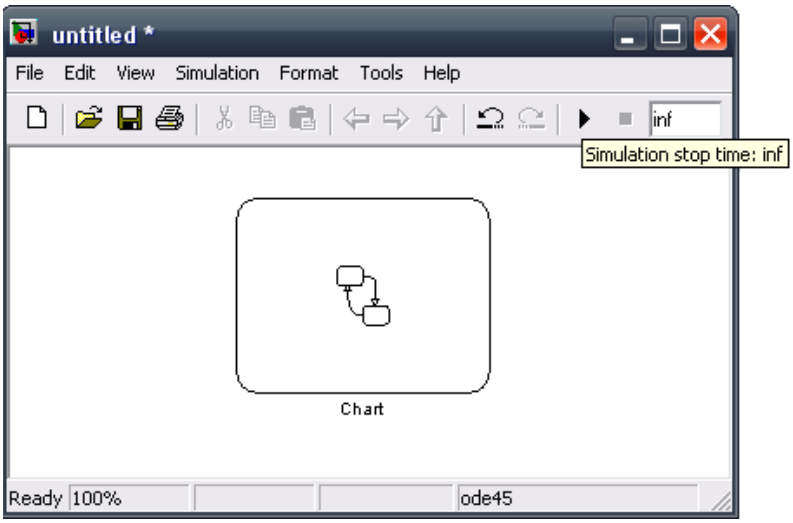
2.1.1 Práca v Simulinku

Nový model vytvoríme následne. V MATLAB-e do príkazového okna „command window" nepíšeme Simulink, alebo jednoducho kliknutím na tlačidlo Simulink. Otvorí sa nám Simulinková knižnica a v nej klikneme na ikonu „Create a new model", alebo stlačením kombinácie „Ctrl+N". Teraz do novo vytvoreného modelu môžeme vkladať bloky z knižníc a vytvoriť tak požadovanú blokovú schému.



Obr. 6 Simulinkový prehliadač

Z lego knižnice sa vyberie blok „stateflow", ktorý sa ťahavým pohybom prenesie do okna nového modelu.(Dvojitým kliknutím na tento blok sa otvorí Stateflow). Čas simulácie je pôvodne nastavený na 10 sekúnd. Po uplynutí tohto času od spustenia sa simulácia automaticky vypne. Preto je vhodne nastaviť tuto hodnotu na nekonečno(„inf").

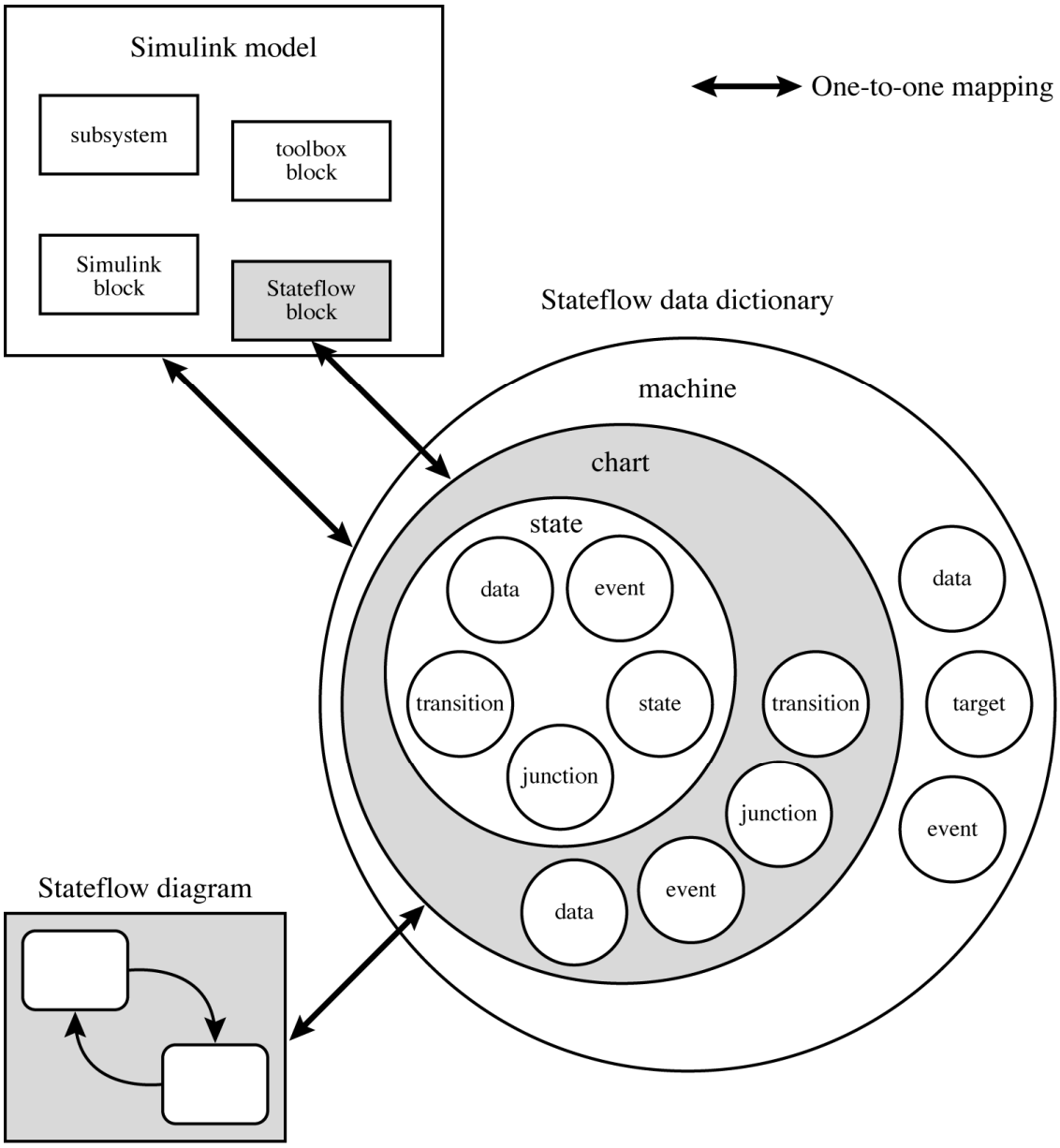


Obr. 7 Nový model

2.2 Stateflow

Stateflow je interaktívny dizajnový a simulačný nástroj, ktorý poskytuje jazykové prvky, pomocou ktorých sme schopný opísať aj zložite logické úlohy. Používa teóriu automatu s konečným počtom stavu, ktorá sa zakladá na prechode medzi stavmi na základe splnenia preddefinovanej podmienky. [3] Príkladom stavového automatu môže byť princíp fungovania výťahu. Predpokladajme, že dom má 5 poschodí (5 stavov). Akčnými členmi sú tlačidla s hodnotami od jedna po päť (aj poschodia majú také isté hodnoty). V prípade, že by sme sa chceli dostať na 4.poschodie (4.stav), stlačíme tlačidlo s hodnotou štyri. Splníme tak prvú podmienku pre prestup z prízemia na prvé poschodie, pretože hodnota stlačeného tlačidla je väčšia ako jedna. Analogicky sme splnili podmienky k prestupu na druhé, tretie až štvrté poschodie. Na piate sa už nedostaneme, lebo hodnota stlačeného tlačidla je menšia ako hodnota poschodia.

Medzi stavmi existuje prísna hierarchia (obr.8). V simulinku vieme pripojiť bloky k príslušným udalostiam (vstupy a výstupy), čím zabezpečíme prepojenie medzi reálnym objektom a jeho riadiacim systémom (stateflow). Na nižšej úrovni sa nachádza schéma stateflowu (Stateflow diagram), ktorá je súborom stavov, uzlov ako aj prepojenia medzi nimi. Základným kmeňom sú samotne stavy a funkcie v nich. [3]



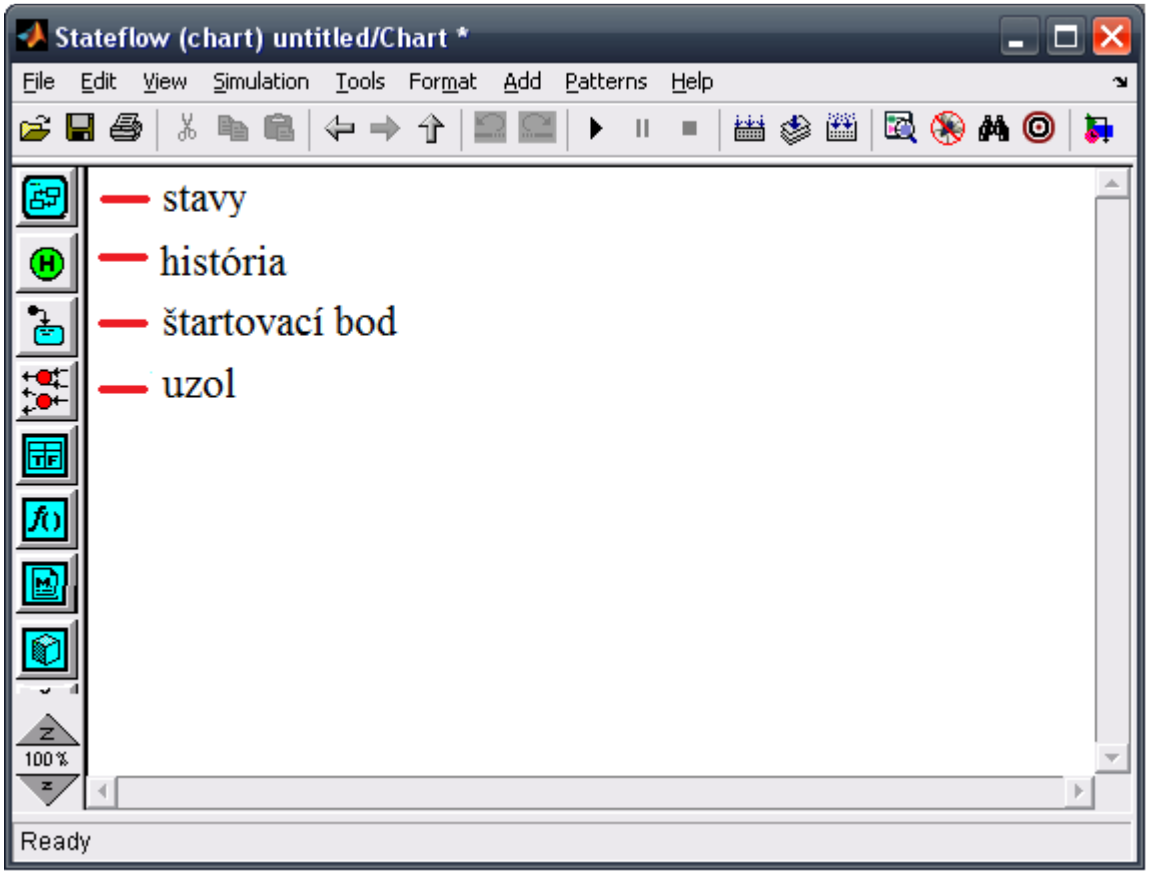
Obr. 8 Anatómia stavového automatu

Informácie získane v programe Stateflow sa využívajú aj v nasledovných odvetviach [3]:

- Programovanie a nastavovanie PLC zariadení.
- Pri vytváraní smerových algoritmov, ktoré sa používajú v telekomunikácii
- V automobilovom priemysle (automatická prevodovka)
- Rôznych spotrebičoch (bojler), a priemyselných zariadeniach
- Systémy riadenia letovej prevádzky (Digital Signal Processing)

2.2.1 Práca so Stateflow

V užívateľskom prostredí Stateflowu sa na ľavej strane nachádza paleta, v ktorej si možno vybrať z ponuky: stav, história, štartovací bod, uzol, pravdivostnú tabuľku, funkciu, vloženú MATLAB funkciu a krabicu.



Obr. 9 Stateflow paleta

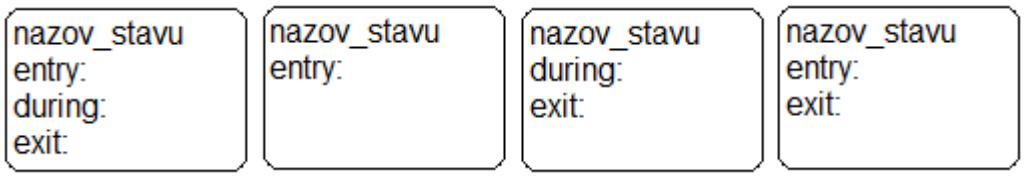
2.2.1.1 Stav

Na vytvorenie stavov sa používa prvé tlačidlo na ľavej strane. Po kliknutí na tlačidlo vieme umiestniť čistý stav na pracovnú plochu.

Obr. 10 Nový stav

1. Štruktúra stavu

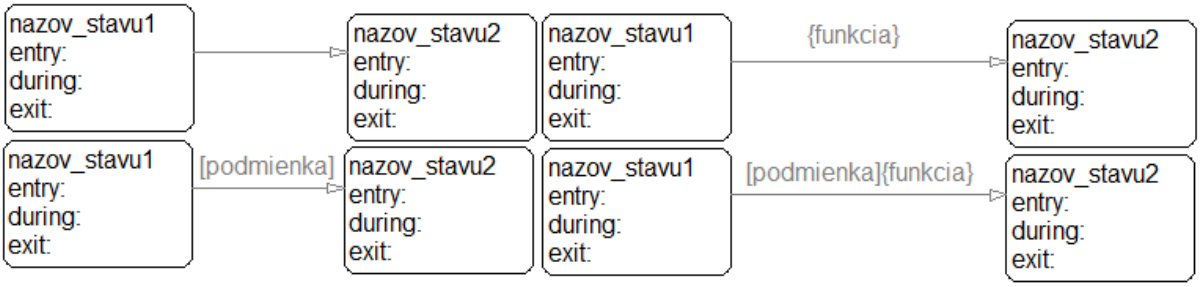
Každý stav ma svoju anatómiu. Do prvého riadku sa zadáva názov stavu, ktorý musí byť jedinečný, nemôže obsahovať diakritiku, medzeru medzi slovami a nesmie sa začínať číslom. V druhom sa zadáva „entry:", v treťom „during:" a v štvrtom „exit:" príkaz. Nemusia byť všetky definované, ale poradie musí byť dodržané.



Obr. 11 Štruktúra stavu

2. Prepájanie stavov

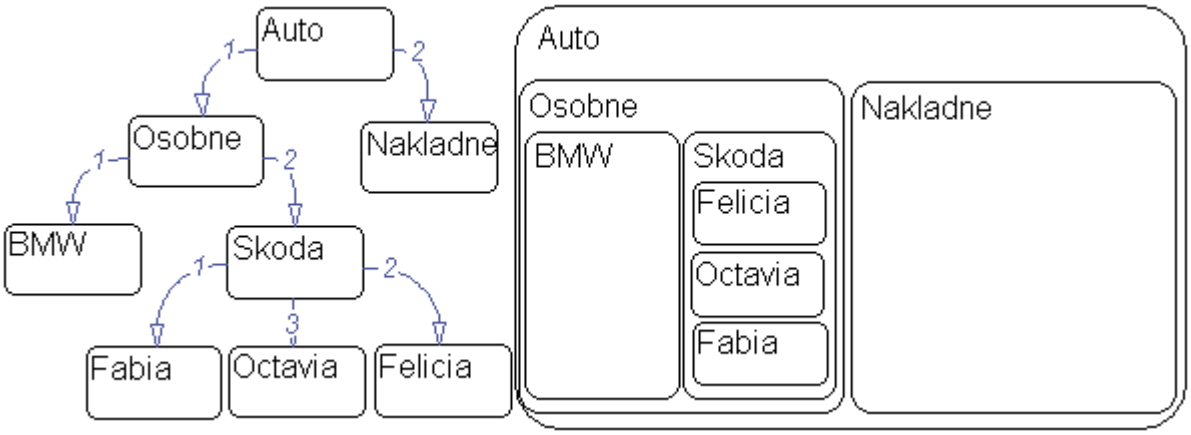
Aktivita stavov je dynamická závislosť, ktorá sa mení na základe splnenia podmienok. Prechod medzi stavmi zabezpečujú prepojenia, ktoré ukazujú smer prestupu a môžu obsahovať funkcie ({funkcia}) a podmienky na opustenie stavu ([podmienka]).



Obr. 12 Prepájanie stavov

3. Superstav

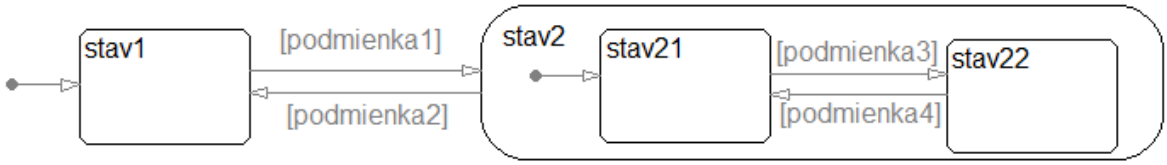
Stav, ktorý okrem príkazov obsahuje aj podradený stav (stavy) nazývame superstav. Ako príklad si môžeme uviesť hierarchiu aut.



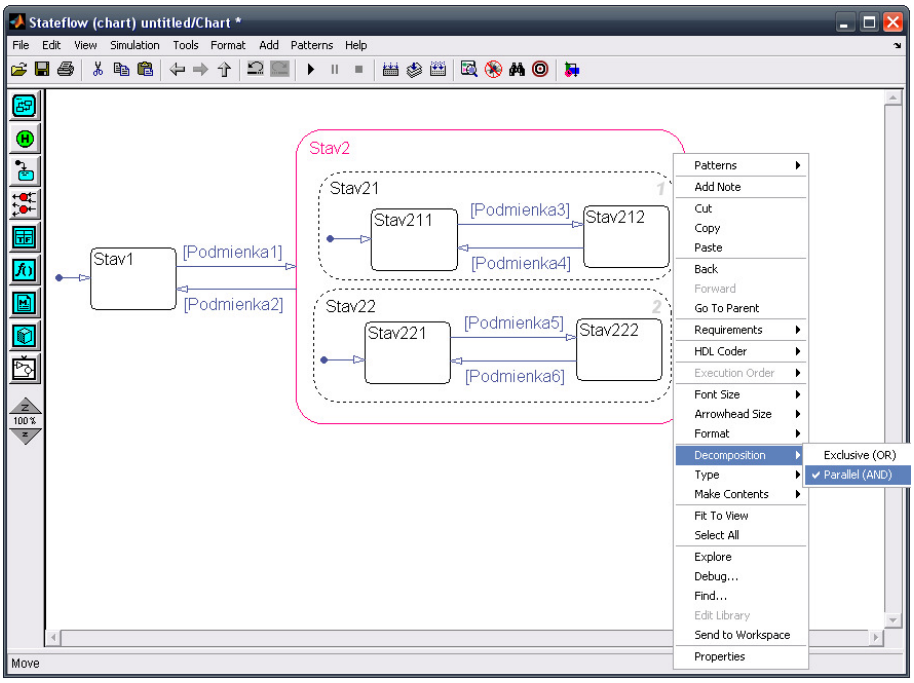
Obr. 13 Superstav hierarchie aut

4. Dekompozícia stavov

Stavy môžeme zapájať do dvoch rôznych dekompozícií. Východisková dekompozícia sa nazýva uzavretá (OR), ktorá nám dovoľuje nachádzať sa iba v jednom stave. Druhým typom je paralelná dekompozícia (AND). Toto zapojenie nám umožňuje, aby v stavovom diagrame mohlo byť aktivovaných viacej stavov súbežne.



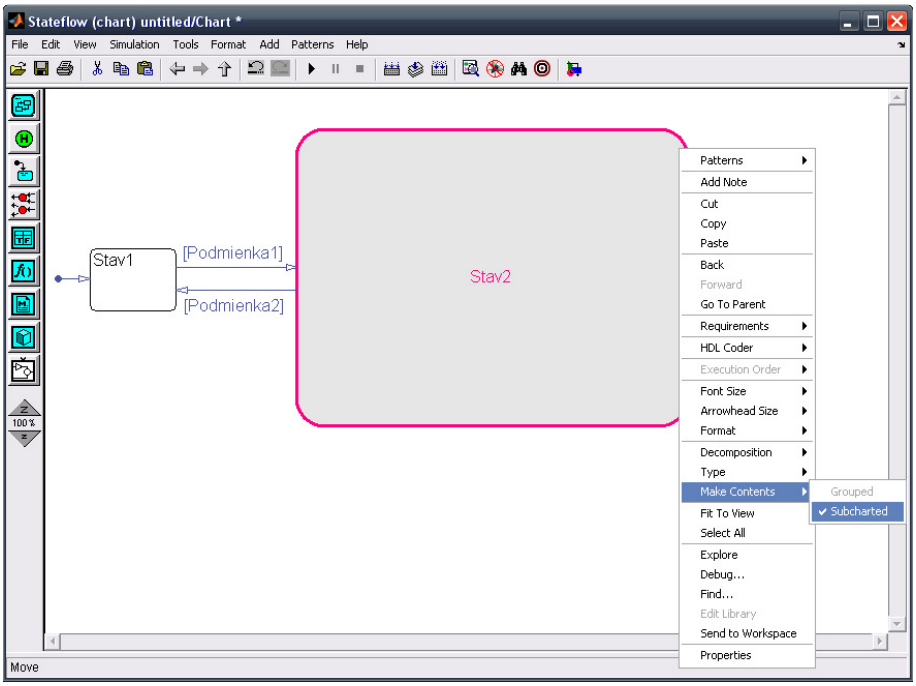
Obr. 14 Uzavretá dekompozícia



Obr. 15 Paralelná dekompozícia

5. Zakrytie obsahu stavov

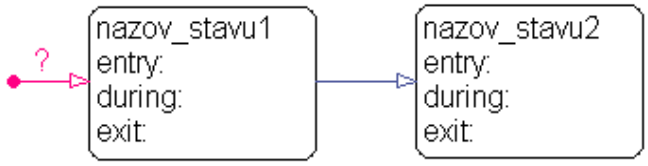
Pri vytváraní algoritmov pre zložitejšie systémy sa diagram môže stať nepriehľadným. Za pomoci funkcii „subcharted" sa tomuto problému môžeme vyhnúť.



Obr. 16 Zakrývanie obsahu stavov

2.2.1.2 Štartovací bod

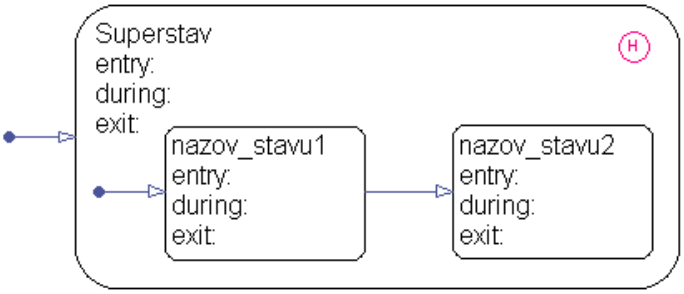
Štartovací bod slúži na určenie stavu, ktorý sa po spustení simulácie aktivuje ako prvý. Ide o prepojenie so smerom, ale bez zdroja. Prvý štartovací bod nesmie obsahovať podmienku.



Obr. 17 Štartovací bod

2.2.1.3 História

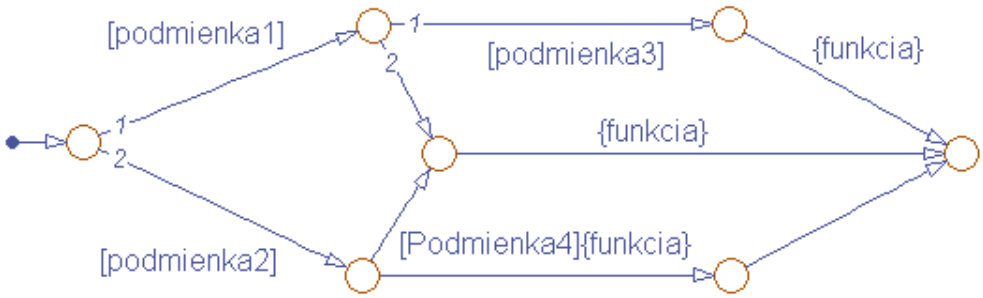
História si zapisuje do pamäti stav, ktorý bol aktívny ako posledný pred opusteným superstavu. Vzťahuje sa len na úroveň v ktorom sa nachádza.



Obr. 18 Použitie histórii v superstave

2.2.1.4 Uzol

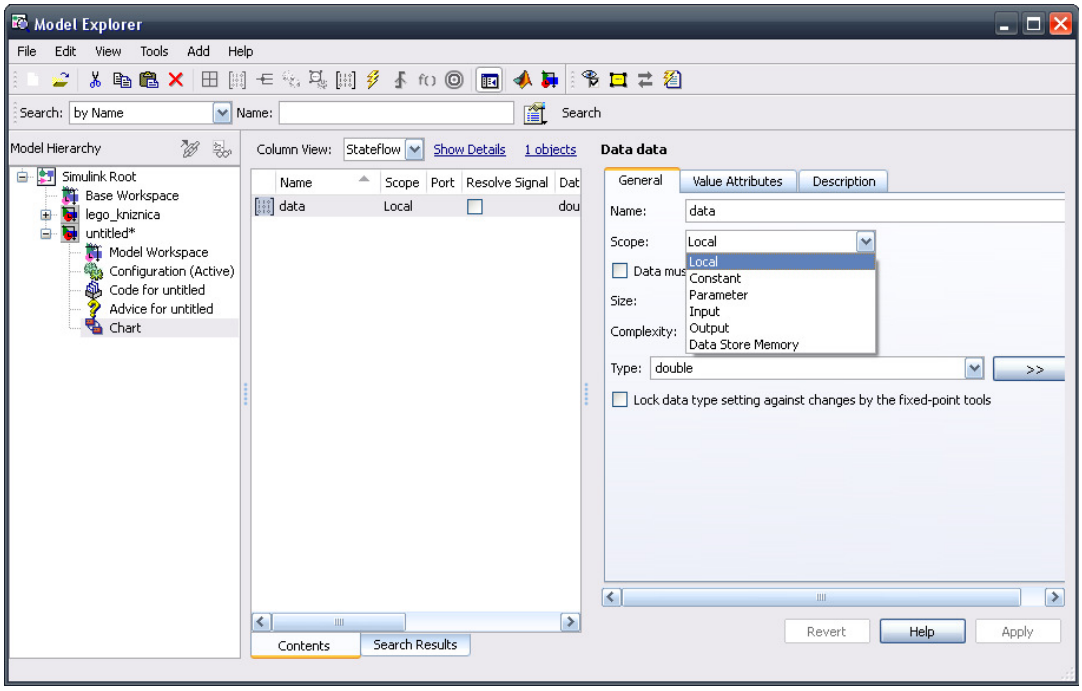
Predstavuje prázdny stav (bez stavovej štruktúry). Slúži na rozdeľovanie (spájanie) prepojení.



Obr. 19 Využitie uzlov v stavovom riadení

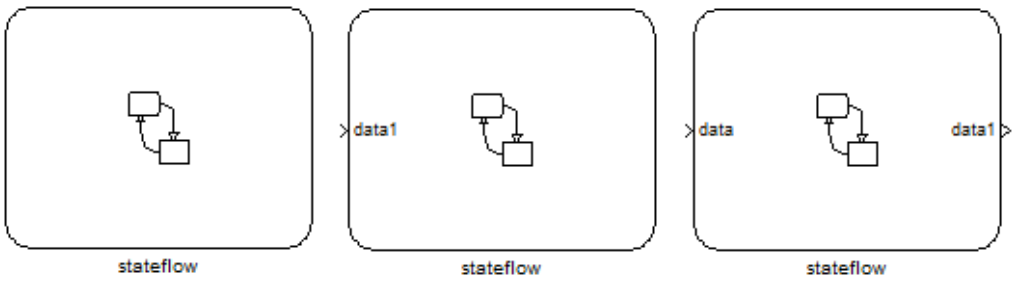
2.2.1.5 Pridávanie udalostí

Poznáme viac druhov udalosti, najčastejšie sa používajú: vstupy, výstupy a lokálne premenné. Pridávajú sa v prostredí Stateflow za pomoci nástroja „Model Explorer“. Nájdeme ho v ponuke „View“, alebo stlačením kombinácie „Ctrl+R“. Na hornej lište po kliknutí na „Add“, kde si vyberieme možnosť „Data“, sa vytvorí udalosť s názvom „data”. V riadku „scope“ si môžeme nastaviť vlastnosti v závislosti od toho či to bude vstup, výstup či lokálna premenná.



Obr. 20 Model Explorer

Takýmto spôsobom sa k pôvodnému bloku Stateflow pridávajú udalosti. Po pridaní sa nám zmení blok Stateflow, v ktorom budeme môcť pripojiť bloky zo Simulinku. Po pripojení môžeme spustiť simuláciu na reálnom objekte.



Obr. 21 Pridávanie udalostí v bloku Stateflow

3 Samoparkujúce auto

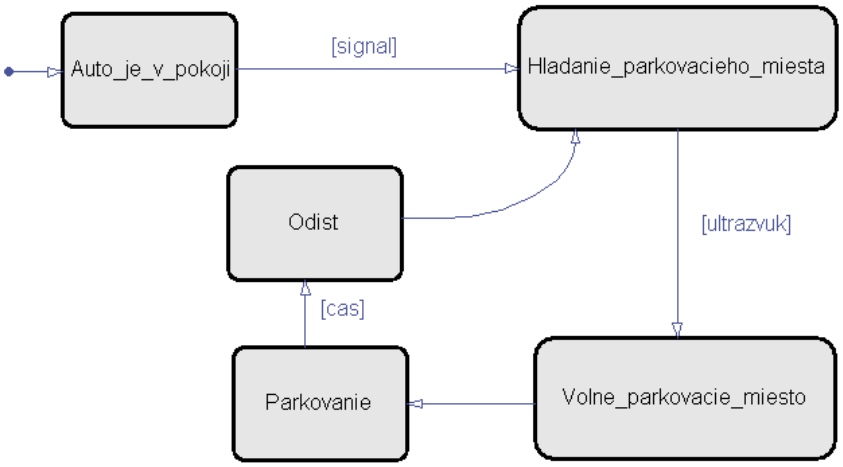
Cieľom úlohy je zostrojiť auto a naprogramovať ho tak, aby dokázalo samé zaparkovať. K dispozícii máme stavebnicu Lego Mindstorms NXT, pomocou ktorej zostavíme model auta. Na napísanie algoritmu použijeme program Stateflow. Základné informácie o komponentoch, ktoré obsahuje Lego Mindstorms NXT, ako aj práca v prostredí Stateflow boli už spomenuté v prvých kapitolách. Pri riešení úlohy je dôležité zamyslieť sa nad problematikou a rozdeliť si ju na viacej menších častí.

3.1 Analýza problematiky

Cieľom analýzy je zostrojenie osnovy programu podľa ktorej budeme vedieť aké senzory a akčné členy budeme potrebovať. Všetky poznatky, ktoré nadobudneme v tejto časti použijeme v ďalších krokoch našej úlohy.

Ako štartovací bod si zvolíme, že auto bude v pokoji a bude situované pred parkoviskom. Budeme mu musieť dať signál, aby sa naštartoval. K tomu nám pomôže zvukový senzor. Auto po aktivovaní pôjde hľadať svoje parkovacie miesto. Bude teda potrebovať pohon dopredu. Keďže nevieme aký bude náš model ťažký použijeme dva motory. Jeden nebude vhodný z toho dôvodu, aby sme ho nepreťažili a nezhodnotili. Pomocou ultrazvuku budeme vedieť odmerať voľný priestor a tým zistíme voľné parkovacie miesto, kde budeme môcť zaparkovať. Pri parkovaní treba umožniť autu aj pohyb do strán. Tento problém zabezpečíme vhodne umiestneným tretím motorom. Odparkovanie analogicky zodpovedá parkovaniu a teda použijeme tie isté senzory a akčné členy.

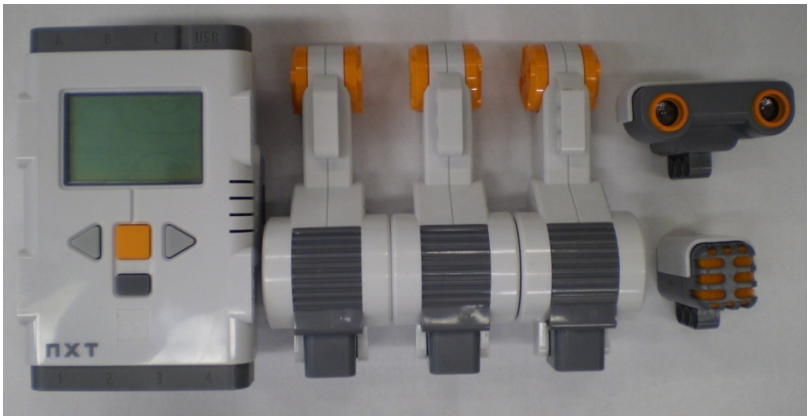
Takouto analýzou sme zistili, že náš model bude potrebovať zvukový senzor, ultrazvukový senzor a tri motory. Kocka NXT má porty pre štyri senzory a tri akčné členy. Keďže túto kapacitu sme neprekročili, bude nám postačovať iba jedna.



Obr. 22 Základná osnova auta

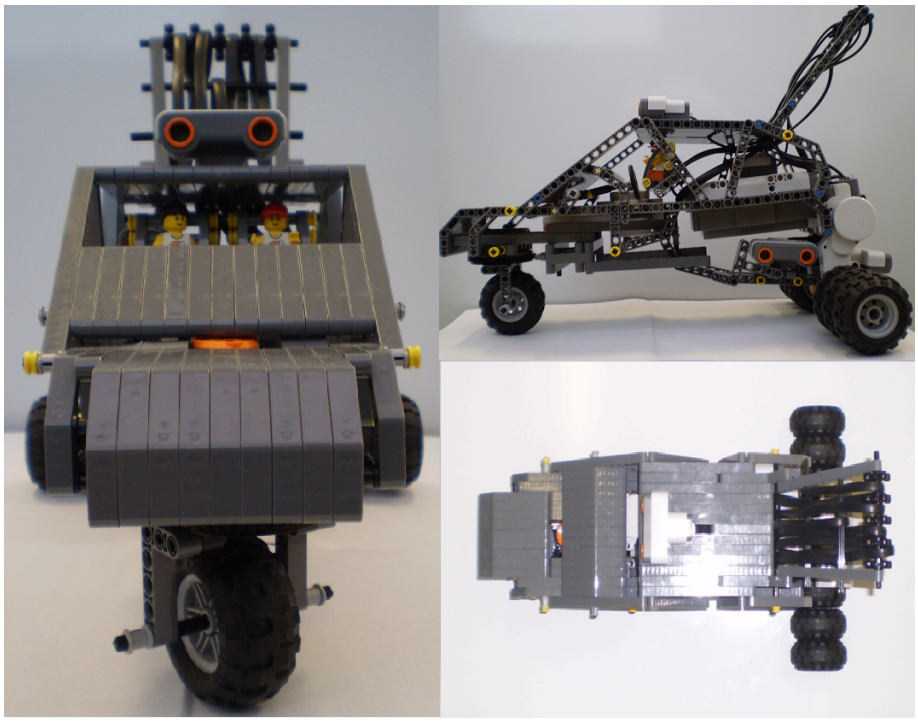
3.2 Konštrukcia

Z prvotnej analýzy vieme, že v našom modelu musia byť zakomponované nasledujúce komponenty: kocka NXT, zvukový senzor, ultrazvukový senzor a tri motory.



Obr. 23 Komponenty potrebné ku konštrukcii modelu

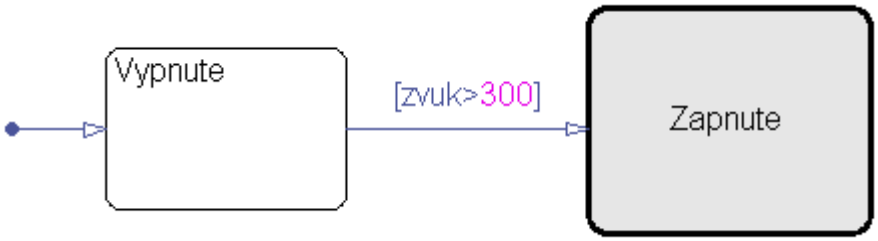
Pri konštrukcii bol kladený dôraz, aby nás model bol stabilný, mal vhodne umiestnené všetky senzory a motory. Pozor sme si museli dávať aj na to, či sa jednotlivé zložky nerušia a nezasekávajú. Všetky senzory a motory sme popripájali káblami k inteligentnej kocke k daným portom A,B,C, respektíve 1,2,3,4. Bolo veľmi dôležité si poznačiť aký príslušný senzor alebo motor sme zapojili k danému portu.



Obr. 24 Model auta

3.3 Stateflow

Celý algoritmus je dosť rozsiahly, preto ho rozdelíme na viacej častí. Každému materskému stavu pridelíme číslicu a ostatné stavy, ktoré sa nachádzajú v nich, samostatne oddelíme.



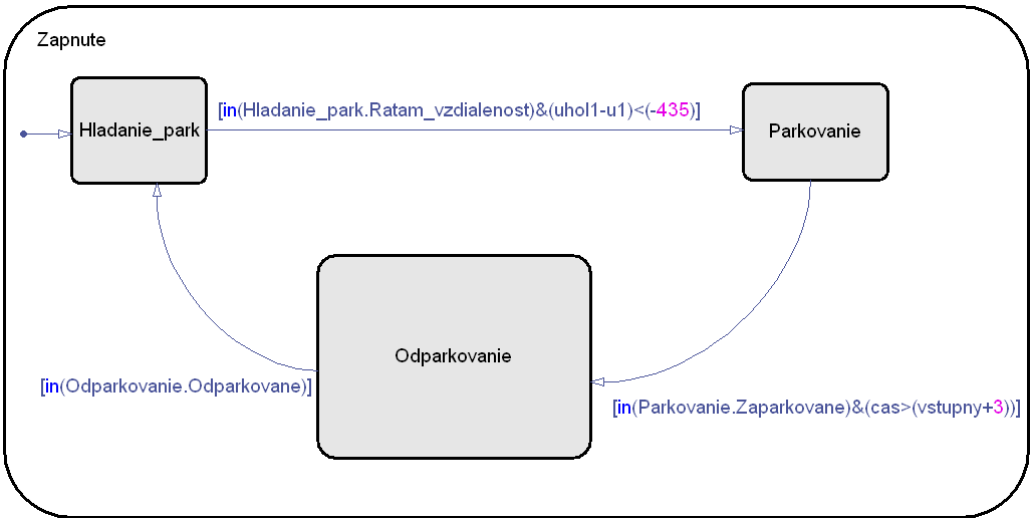
Obr. 25 Stavový diagram samoparkujúceho auta

3.3.1 Stav - Vypnute

Do stavu vstúpime hneď po spustení simulácie. Zostávame v ňom pokiaľ zvukový senzor zaznamená hodnotu väčšiu ako 300, kedy prejdeme do stavu „ Zapnute“.

3.3.2 Stav - Zapnute

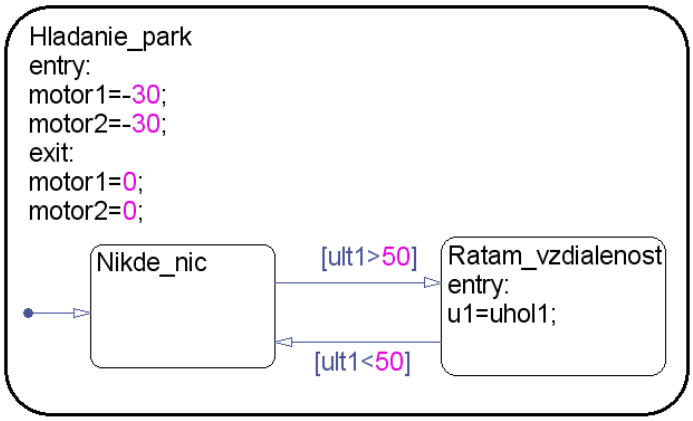
V tomto stave sa nevykoná žiadny „entry“ príkaz, hneď prejdeme do stavu „Hladanie_park“.



Obr. 26 Stavový diagram stavu Zapnute

3.3.2.1 Stav - Hladanie_park

Pri vstupe sa vykoná príkaz „entry“ a zapne sa motor1 a motor2 na hodnotu -30. Auto sa začne pohybovať smerom dopredu. V tejto fáze sa auto snaží nájsť voľné parkovacie miesto, aby mohlo zaparkovať. Podarí sa mu to, ak ultrazvukový senzor zaregistruje vzdialenosť väčšiu ako 50 a súčastne zotrvá v stave „Ratam_vzdialenost“ po dobu pokiaľ sa uhol motora1 otočí o -435 stupňov. V okamihu, keď bude táto podmienka splnená prejdeme do stavu „Parkovanie“. Ak odídeme zo stavu vykoná sa príkaz „exit“ a motory sa vypnú.

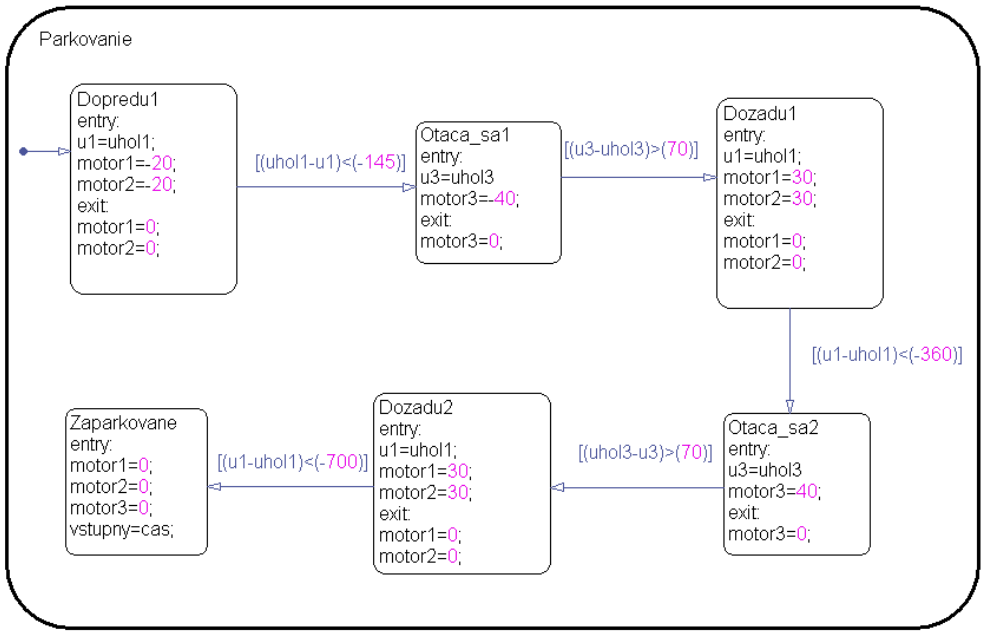


Obr. 27 Stavový diagram stavu Hladanie_park

- **Nikde_nic** – V stave sa nič nevykoná. Budeme sa v ňom nachádzať pokiaľ ultrazvuk (ult1) nadobudne hodnotu väčšiu ako 50.
- **Ratam_vzdialenost** - Do lokálnej premennej „u1“ sa zapíše (prepíše) hodnota uhla, ktorú mal motor1 hneď pri vstupe do stavu. Ak ultrazvuk zachytí menšiu vzdialenosť ako 50, tak sa vrátime naspäť do stavu „Nikde_nic“.

3.3.2.2 Stav - Parkovanie

Auto si už našlo voľné miesto a chystá sa zaparkovať. Experimentálne sme zistili vzdialenosti (vyjadrené v uhloch motorov), ktoré musí model auta prejsť pre jednotlivé kroky parkovania. Pri vstupe do stavu sa nič nevykoná, hneď sa spustí stav „Dopredu1“. Postupným plnením podmienok prejdeme až do posledného stavu „Zaparkovane“, v ktorom naše auto bude zaparkované. Ak v tomto stave zotrváme tri sekundy, tak sa splní podmienka pre prechod do stavu „Odparkovanie“.



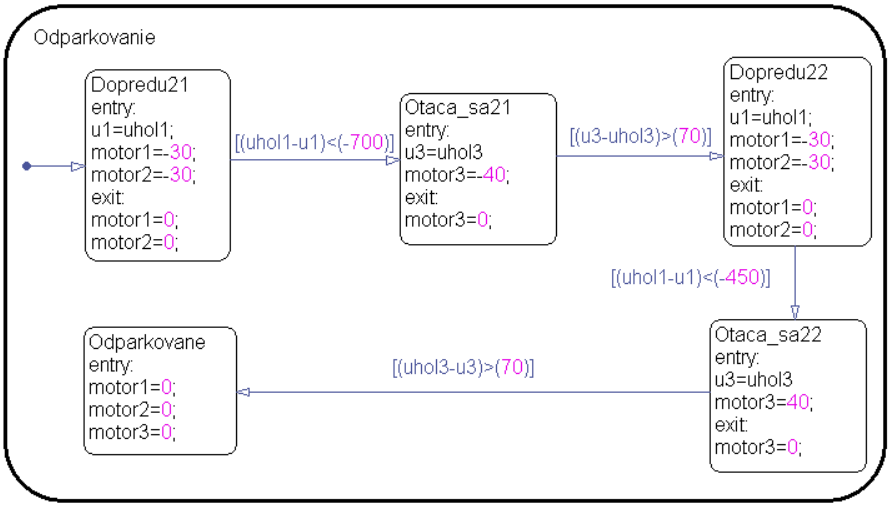
Obr. 28 Stavový diagram stavu Parkovanie

- **Dopredu1** - Pri vstupe sa nastavia motory na -20 a teda pôjde dopredu. V lokálnej premennej „u1“ sa zaznamená uhol motora1. Zo stavu odídeme, ak sa „uhol1“ otočí minimálne o ďalších -145 stupňov. Pri splnení podmienky sa vypnú oba motory. Auto si takto nadbehlo, aby dokázalo zaparkovať.

- **Otaca_sa1** - V tomto stave sa otočí predné koleso o 70 stupňov doľava. Dosiahneme to ovládaním motora3.
- **Dozadu1** - Zadné motory sú tu nastavené na kladnú hodnotu 30, čo predstavuje pohyb dozadu. Auto začne cúvať s otočeným predným kolesom po dobu, kým sa uhol motora1 otočí o 360 stupňov. Takto sa celé auto otočí o 90 stupňov a dostaneme na zarovno parkovacieho miesta.
- **Otaca_sa2** - Predné koleso vrátime do pôvodného stavu. Otočíme ho o 70 stupňov doľava.
- **Dozadu2** - Tento stav zabezpečí docúvanie auta na parkovacie miesto, aby nevyčnievalo z cesty. Docielime to otočením uhla1 o 700 stupňov.
- **Zaparkovane** - Predstavuje posledný stav materského stavu „Parkovanie“. Pri vstupe sa vypnú všetky motory a do premennej „vstupný“ sa zapíše aktuálny čas. Túto premennú budeme potrebovať na to, aby sme prešli do ďalšieho stavu „Odparkovanie“.

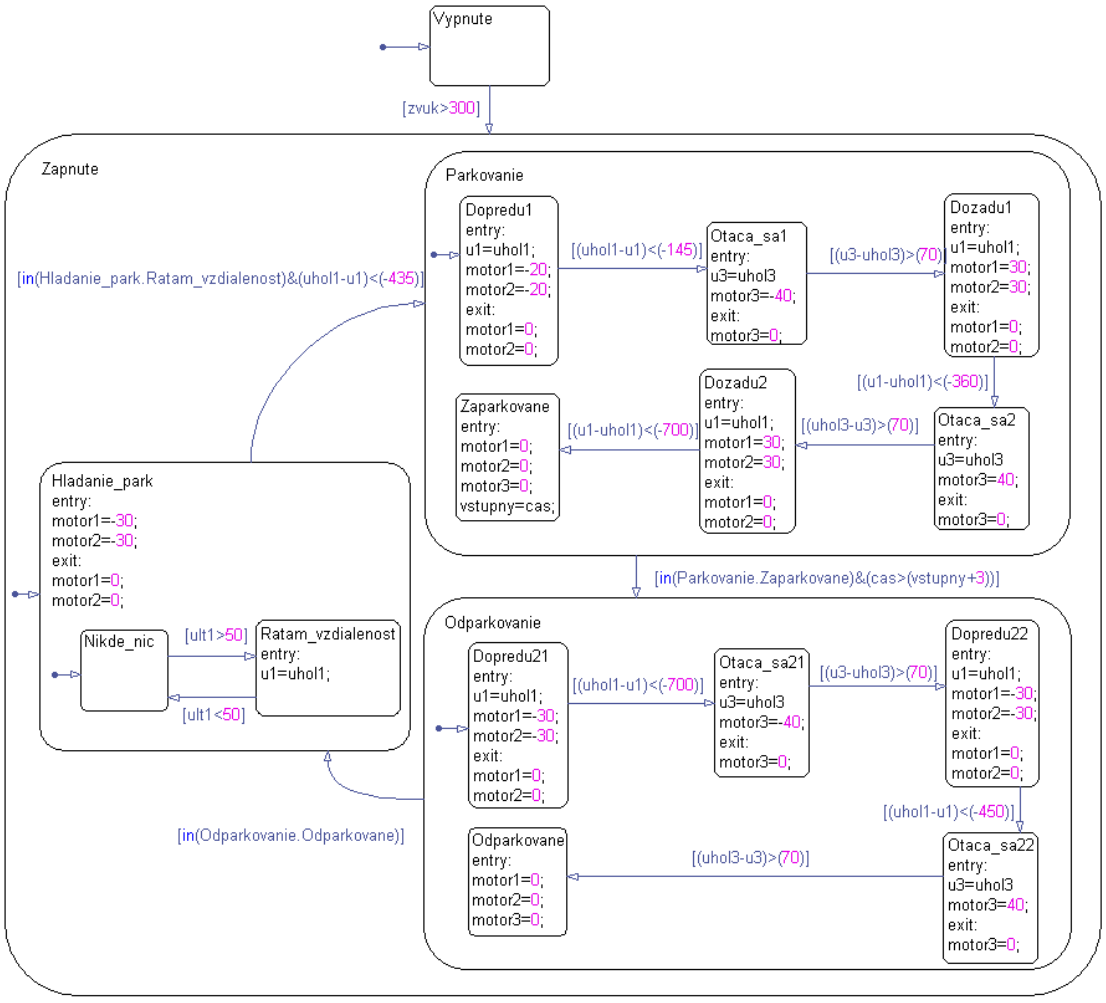
3.3.2.3 Stav - Odparkovanie

Po úspešnom zaparkovaní a ubehnutí troch sekúnd naše auto odparkujeme. Postup je analogicky ako pri parkovaní, len v opačnom poradí. Ak sa budeme nachádzať v poslednom stave „Odparkovane“ prejdeme do stavu „Hladanie_park“.



Obr. 29 Stavový diagram stavu Odparkovanie

- **Dopredu21** - V tomto stave auto vyjde z parkovacieho miesta o vzdialenosť, kým sa motor1 otočí o -700 stupňov (dopredu).
- **Otaca_sa21** - Motorom3 otočíme predné koleso o -70 stupňov (doľava).
- **Dopredu22** - S otočeným predným kolesom a zadnými motormi nastavenými na chod dopredu (záporná hodnota -30) sa auto vytočí o 90 stupňov.
- **Otaca_sa22** - Predné koleso otočíme do pôvodného stavu, aby osi všetkých kolies boli rovnobežné.
- **Odparkovane** - V stave vypíname všetky motory. Teraz sa auto nachádza na ceste a je pripravené opäť hľadať parkovacie miesto. Podmienka pre prechod do nového stavu „Hladanie_park“ je splnená v okamihu, keď sme vošli do stavu „Odparkovane“.

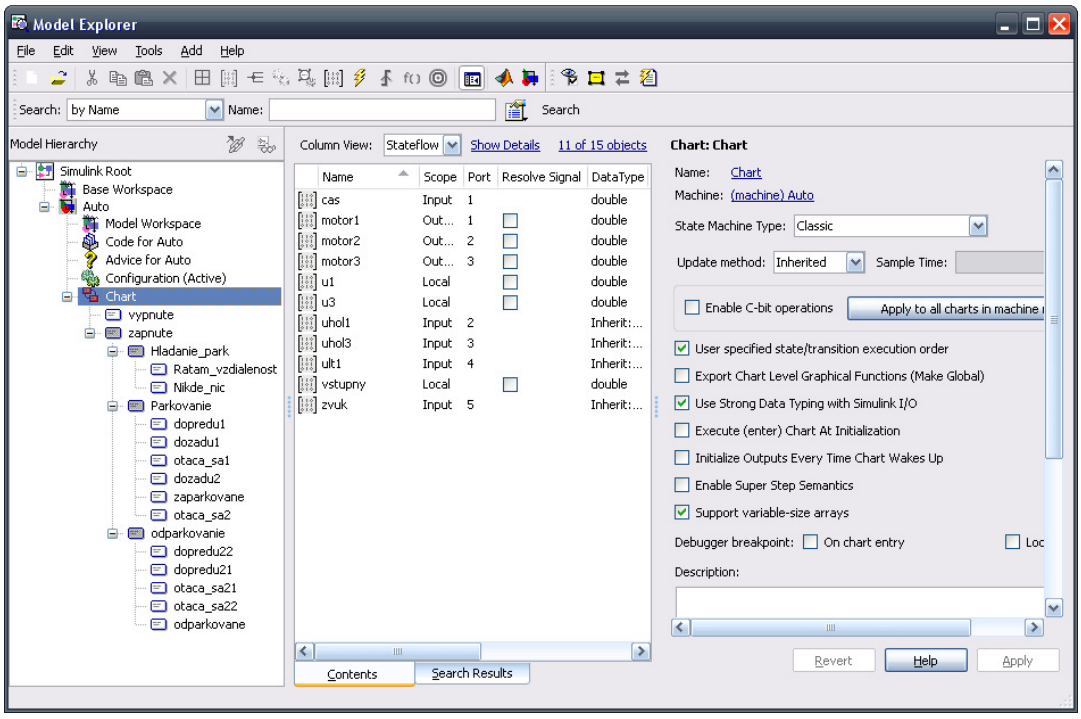


Obr. 30 Stavový diagram celého algoritmu úlohy 1

3.4 Simulink

Do Similinku sme definovali všetky udalosti:

- Vstupy: zvuk, uhol1, uhol3, cas, ult1
- Výstupy: motor1, motor2, motor3
- Lokálne premenné: u1, u3, vstupny

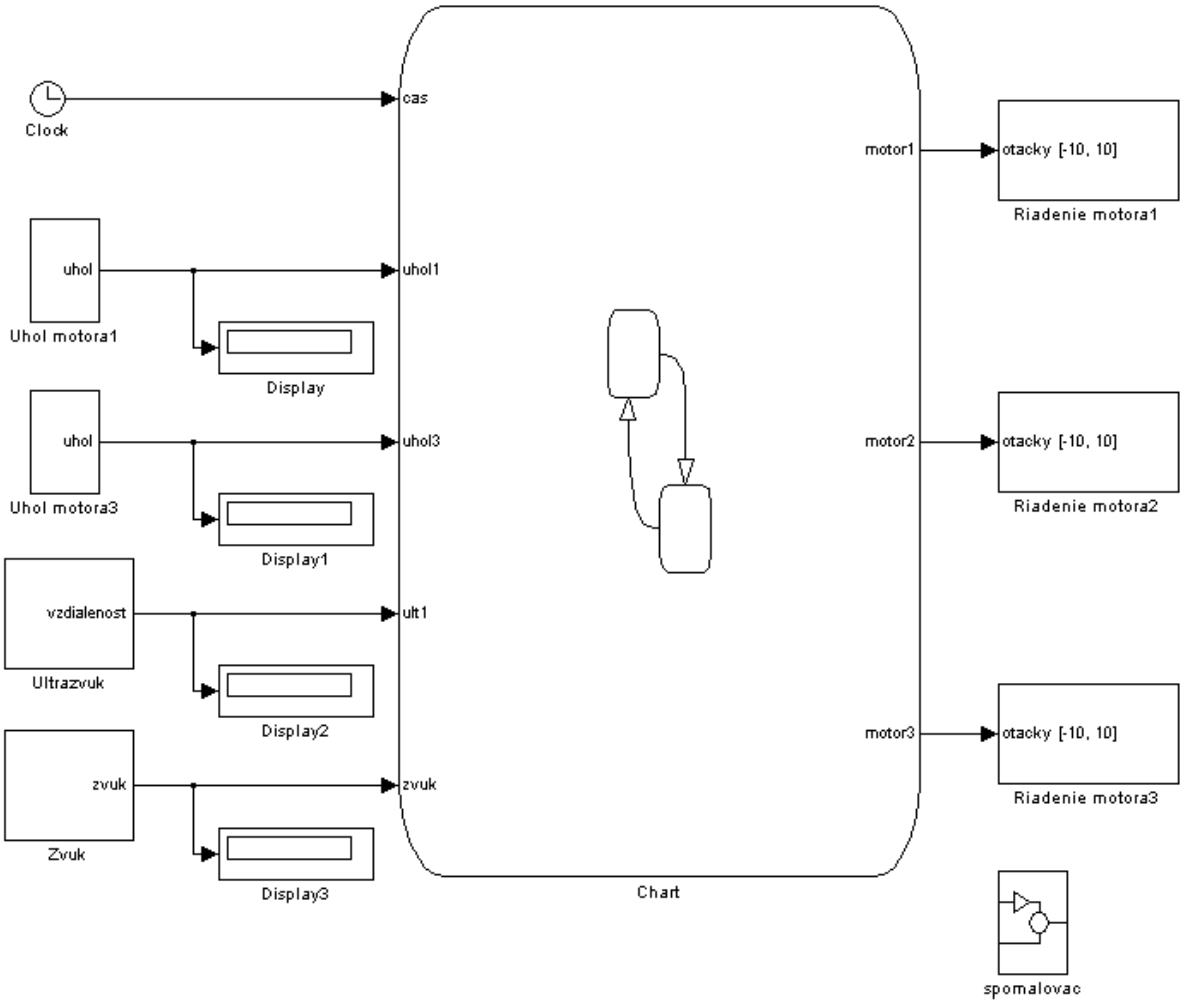


Obr. 31 Model Explorer úlohy 1

Z knižnice sme vybrali bloky, ktoré sme potrebovali:

- Riadenie motora
- Uhol motora
- Ultrazvuk
- Clock
- Zvuk
- Display
- Spomalovac

Na záver sme všetky pridané udalosti a bloky prepojili. Bolo veľmi dôležité zadať každému bloku správne číslo kocky a portu. Keby sme tak neurobili určite by model nevykonával to čo mal. Pri analýze sme si poznačili všetky porty, ktoré sme pripojili ku kocke, čím sme sa vyvarovali problémom.



Obr. 32 Simulinková schéma úlohy 1

4 Komplexné parkovisko

Cieľom úlohy je zostrojiť a naprogramovať parkovisko, ktoré bude plne automatizované. Jedná sa o rozšírenie prvej úlohy. K dispozícii máme naprogramované samoparkujúce auto. Pridáme k nemu model parkoviska, ktoré zostavíme zo stavebnice Lego Mindstorms NXT. Algoritmus napíšeme pomocou programu Stateflow.

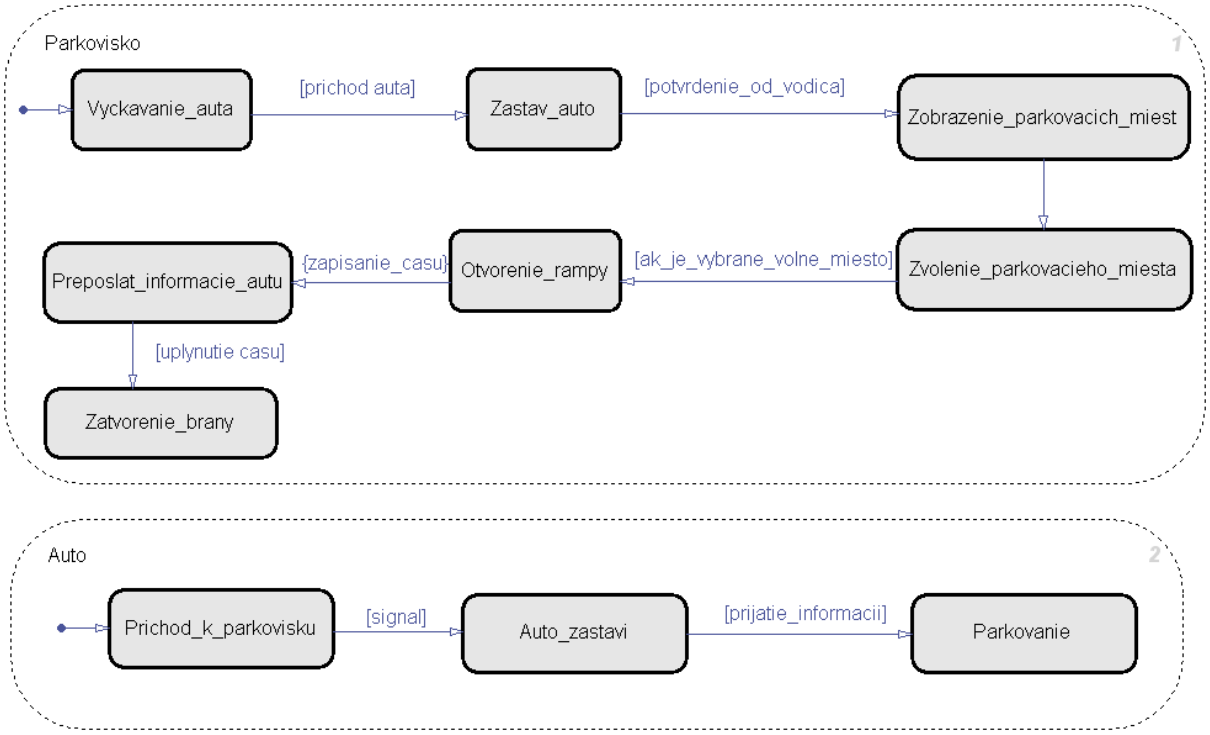
4.1 Analýza problematiky

Postupnými krokmi si načrtneme osnovu, ktorú použijeme ako predlohu pri tvorení výsledného algoritmu. Takto zistíme aj koľko senzorov, akčných členov a kociek budeme potrebovať.

Na začiatku bude auto v pokoji na ceste. Po prijatí zvukového signálu mu dáme povel, aby prišlo k parkovisku. Pred rampu umiestnime ultrazvukový senzor, ktorý zaregistruje prichádzajúce auto a prikáže mu, aby zastavilo. V tejto fáze vodič príde ku kontrolnému panelu. Po stlačení prvého tlačidla sa mu na displeji vysvietia všetky voľné parkovacie miesta. Opätovným stlačením si vodič vyberie pozíciu z danej ponuky, kde bude chcieť zaparkovať svoje auto. Po korektnom zadaní pozície parkovacieho miesta sa rampa otvorí a auto zaparkuje na zvolené miesto. Rampu zatvoríme po uplynutí niekoľkých sekúnd.

Použijeme nasledovné komponenty, z ktorých sú zložené auto, rampa, kontrolý panel a parkovacie miesta:

- **Auto**: jedná kocka NXT, tri motory, jeden zvukový senzor
- **Rampa**: jedná kocka NXT, jeden motor, jeden ultrazvukový senzor
- **Kontrolný panel**: dve kocky NXT, šesť žiaroviek, dva dotykové senzory
- **Parkovacie miesta**: jedná kocka NXT, tri ultrazvukové

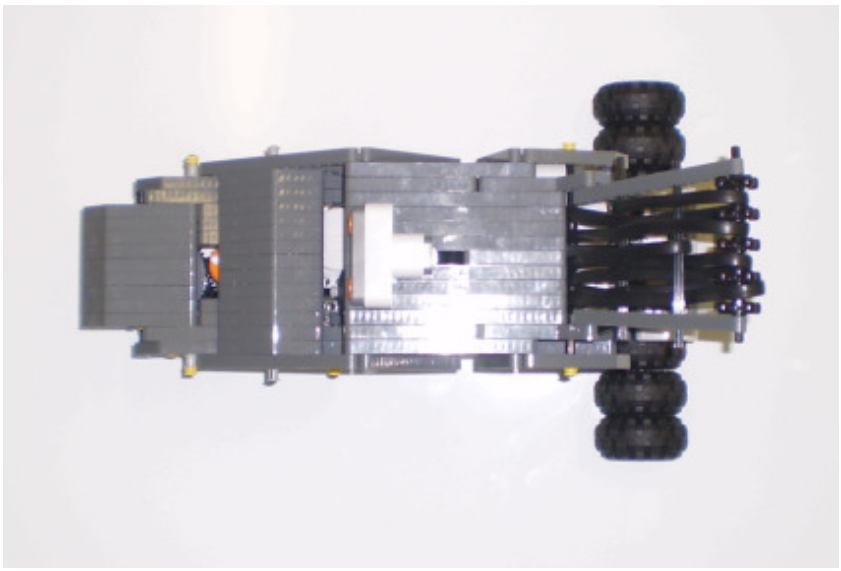


Obr. 33 Základná osnova komplexného parkoviska

4.2 Konštrukcia

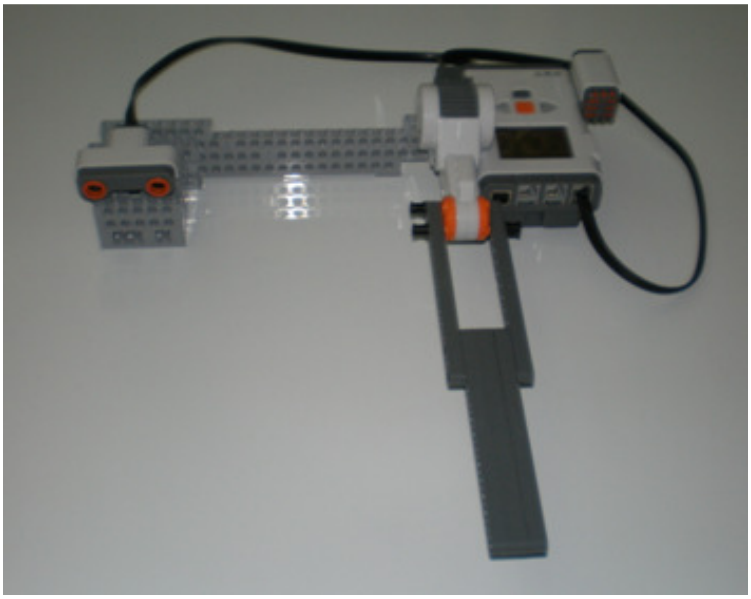
Z analýzy sme získali základné informácie, ktoré sme v tejto časti zužitkovali. Pri stavaní jednotlivých blokov sme si zapisovali aký akčný člen, motor alebo žiarovku sme pripojili k jednotlivým portom. Bolo to veľmi dôležité pri pridávaní a spájaní udalostí k blokovým schémam v simulinku.

- **Auto -** Model auta sme už zhotovili v prvej úlohe. Má zakomponované tri motory, dva ultrazvukové a jeden zvukový senzor. Pre túto úlohu nám bude viac ako postačujúce (auto nebude hľadať parkovacie miesto pomocou ultrazvuku, ale bude ho mať zadané pomocou kontrolného panela).



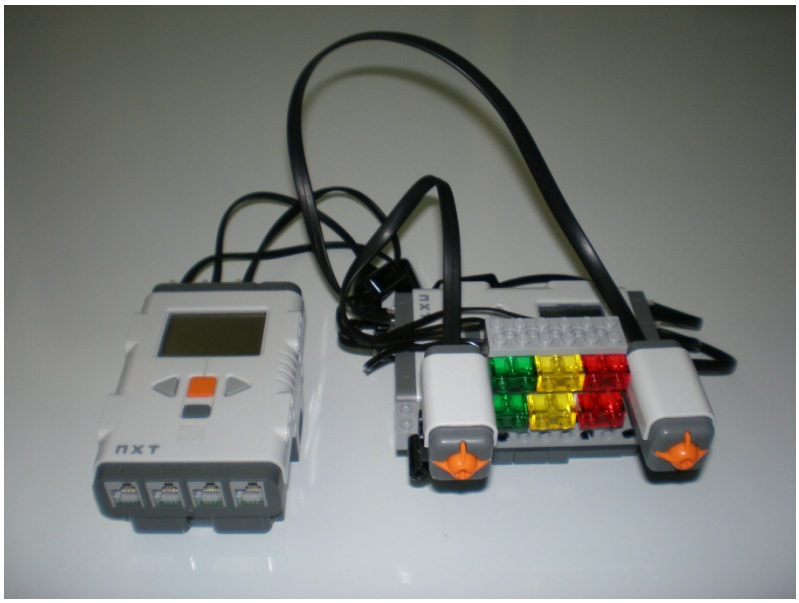
Obr. 34 Model auta

- **Rampa -** Na zabezpečenie pohybu ramena rampy použijeme jeden motor, ktorým budeme otáčať o 90 stupňov. Ultrazvukový senzor musí byť umiestnený aspoň 5cm pred ramenom, aby stihol zachytiť signál a odoslal ho autu. Takto sa vyhneme rozbitiu brány.



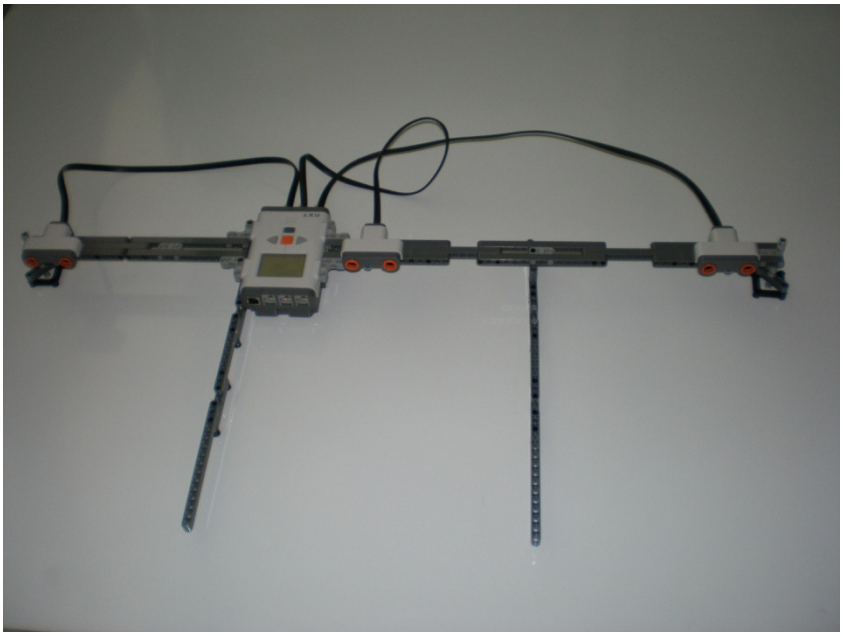
Obr. 35 Model rampy

- **Kontrolný panel -** Ako displej sme použili šesť svetelných žiarovky, ktoré sme zapojili vedľa seba. Tlačidlá sme pripevnili na ľavú a pravú stranu kocky, aby sme vedeli lepšie vysvetliť ich funkciu.



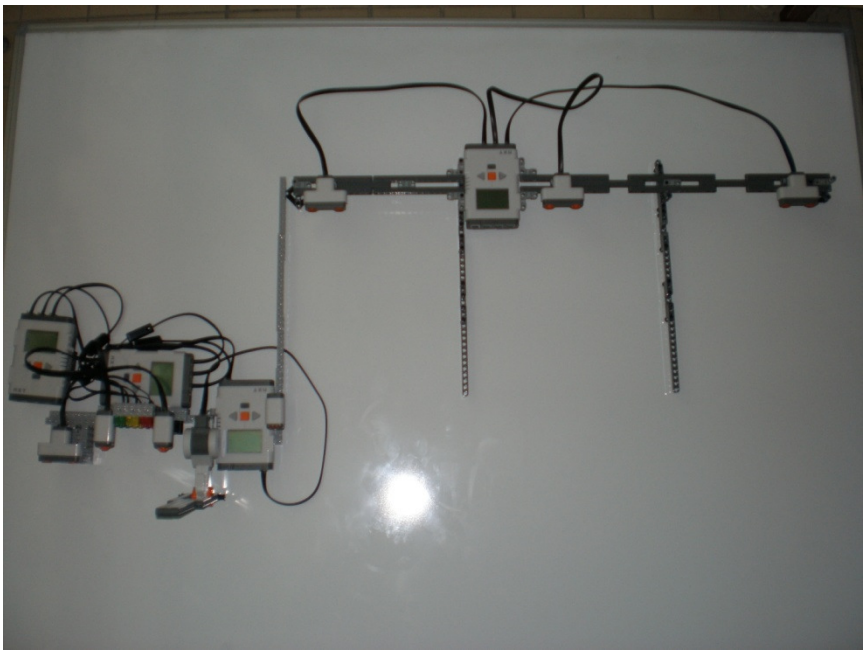
Obr. 36 Model kontrolného panela

- **Parkovacie miesta -** Z lega sme vytvorili tri parkovacie miesta. Na koniec každého sme pripevnili ultrazvukový senzor. Dbali sme na to, aby všetky parkovacie miesta boli rovnaké a rozmerovo vyhovovali modelu auta.



Obr. 37 Model parkovacích miest

Po spojení všetkých častí sme dostali model parkoviska s troma parkovacími miestami, rampou a kontrolným panelom.



Obr. 38 Model komplexného parkoviska

4.3 Stateflow

Algoritmus sa skladá z viacerých hlavných stavov (superstavov), ktoré majú v sebe vnorené dcérske stavy (prípadne ďalšie superstavy). Všetky si ich podľa nadradenosti (poradia v hierarchií) postupne predstavíme a vysvetlíme ich význam.

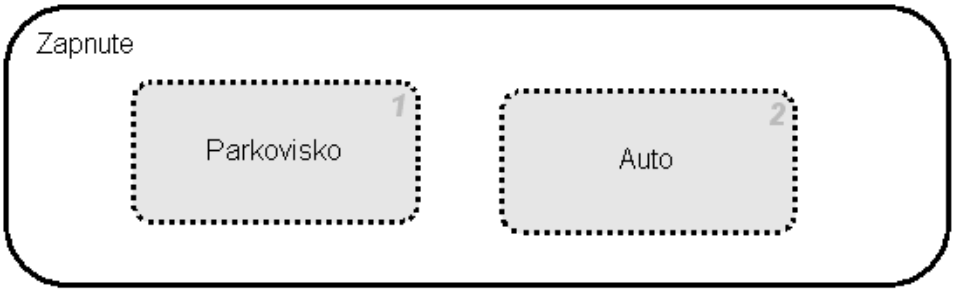
Obr. 39 Stavový diagram komplexného parkoviska

4.3.1 Stav - Vypnute

Do stavu sa dostaneme hneď po spustení simulácie. Ak zvukový senzor zaregistruje silnejší zvukový signál (väčší ako 300) prejdeme do nového stavu „Zapnute“. Spätná podmienka pre návrat do stavu „Vypnute“ bude splnená keď auto úspešne zaparkuje a súčastne rampa bude zatvorená, alebo druhou možnosťou je ak v lokálnej premennej „error“ bude hodnota 1.

4.3.2 Stav - Zapnute

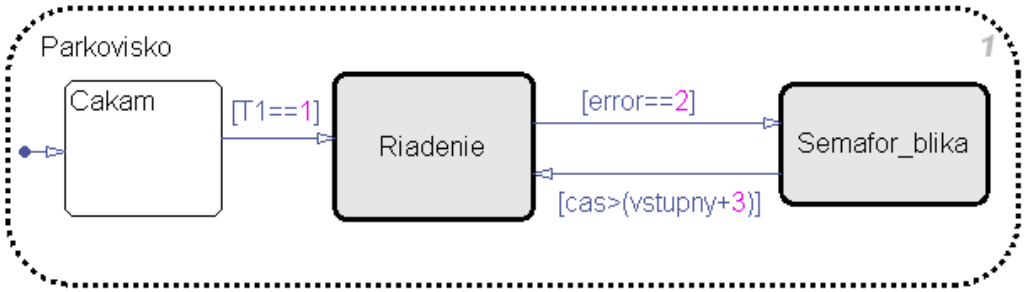
Stav sa skladá z dvoch podradených superstavov, ktoré sú zapojené paralelne. Týmto spôsobom budeme riadiť parkovisko, ako aj auto súbežne.



Obr. 40 Stavový diagram stavu Zapnute

4.3.2.1 Stav - Parkovisko

Pomocou tohto stavu riadime celé parkovisko (kontrolný panel, rampu, parkovacie miesta).



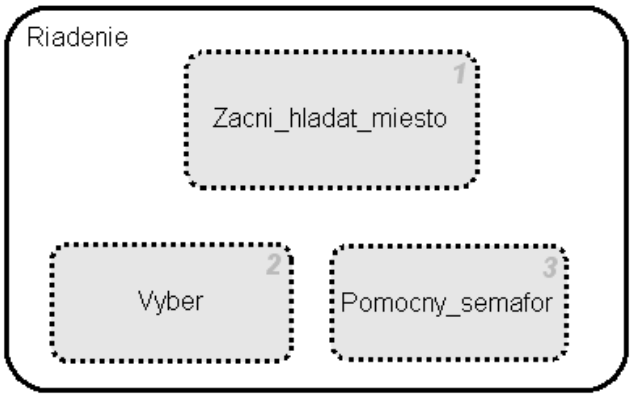
Obr. 41 Stavový diagram stavu Parkovisko

4.3.2.1.1 Stav - Cakam

V stave sa nič nevykonáva. Slúži iba ako pomocný, keďže v paralelnom zapojení potrebujeme aspoň jeden aktívny stav. Po stlačení tlačidla 1 (T1==1) stav opúšťame a aktivuje sa stav „Riadenie“.

4.3.2.1.2 Stav - Riadenie

V tejto fáze sa zastavilo auto pred rampou (viac v stave „Auto“). Vodič prišiel ku kontrolnému panelu a stlačil pre tlačidlo, čím aktivoval tri nové stavy „Zacni_hladat_miesto“, „vyber“ a „Pomocny_semafor“.

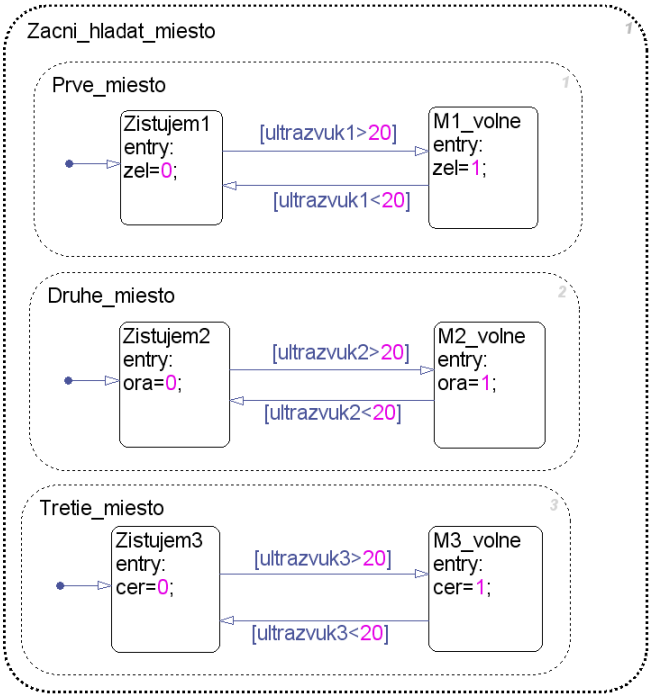


Obr. 42 Stavový diagram stavu Riadenie

4.3.2.1.2.1 Stav - Zacni_hladat_miesto

Za pomoci ultrazvukových senzorov, ktoré sa nachádzajú na každom parkovacom mieste, vieme zistiť všetky voľné parkovacie miesta. Ak senzor (po stlačení prvého tlačidla) zaregistruje pred sebou voľný priestor odovzdá povel k rozsvieteniu príslušného svetla. Stavy sú paralelne zapojené, preto aby bolo možné signalizovať všetky voľné miesta.

Bude nám stačiť, keď si vysvetlíme jeden stav „Prve_miesto“. Ostatné si analogicky budeme vedieť odvodiť. Meníme v nich len farbu svetelnej žiarovky.

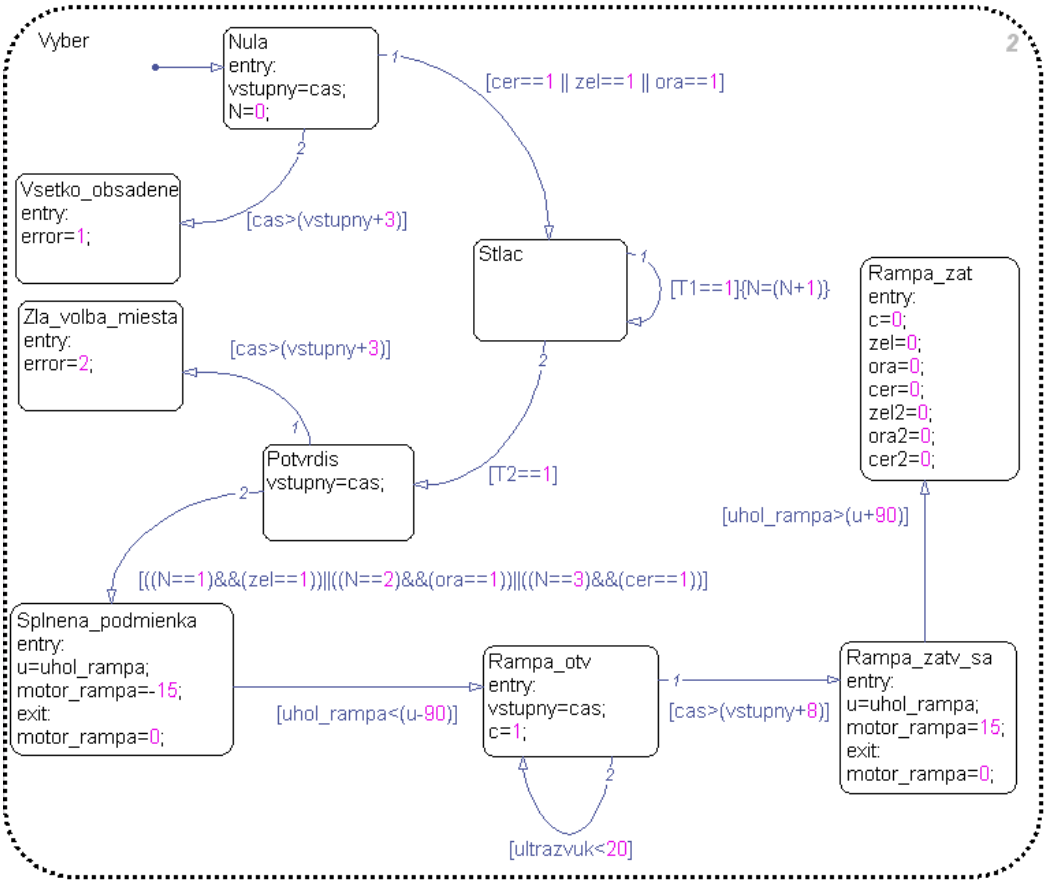


Obr. 43 Stavový diagram stavu Zacni_hladat_miesto

- **Prve_miesto** - Po aktivovaní sa dostávame do stavu „Zistujem1“, kde vypíname zelenú žiarovku (zel=0). K prestupu do stavu „M1_volne“ nám musí dať ultrazvukový senzor (ultrazvuk1) namerať väčšiu vzdialenosť ako 20, čo signalizuje voľné parkovacie miesto na prvej pozícii. Po splnení podmienky sa vykoná „entry“ príkaz a zelená žiarovka sa rozsvieti (zel=1). Aby bol vplyv poruchy ultrazvukového senzora (zriedkavé kmity) odstránený, zo stavu „M1_volne“ po nesplnení podmienky prejdeme naspäť do stavu „Zistujem1“.

4.3.2.1.2.2 Stav - Vyber

Stav sa aktivuje súbežne so stavom „Zacni_hladat_miesto“, kde ultrazvukové senzory zistia všetky voľné miesta (do jednej sekundy) a rozsvietia príslušnú žiarovku (cer=1, ora=1 alebo zel=1).

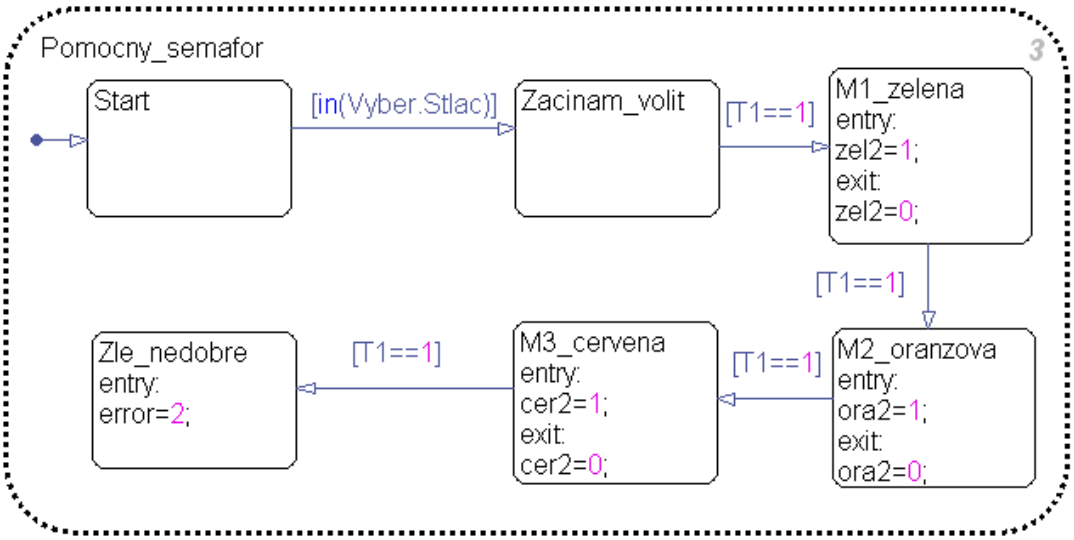


Obr. 44 Stavový diagram stavu Vyber

V stave „Nula“ zadáme do premennej nulu a začneme odpočítavať čas. Ak sa v stave nachádzame viac ako tri sekundy, znamená to, že nie je voľné ani jedno parkovacie miesto a prejdeme do stavu „Vsetko_obsadene“. Tu do premennej „error“ zapíšeme jednotku, čím sa splní podmienka a vrátime sa do stavu „Vypnute“. V prípade, keď bude aspoň jedno miesto voľné (zel=1, ora=1 alebo cer=1) aktivuje sa stav „Stlac“. Prvým tlačidlom tu navolíme parkovacie miesto, kde chceme zaparkovať. Druhým tlačidlom našu voľbu potvrdíme a prejdeme do stavu „Potvrdis“. Tu skontrolujeme, či zvolené miesto je voľné. Ak nebude (ubehnú tri sekundy) prejdeme do stavu „Zla_volba_miesta“. Premenná „error“ nadobudne hodnotu 2, čím odídeme zo stavu „Riadenie“ do stavu „Semafor_blika“. V prípade správneho zvolenia voľného parkovacieho miesta (hodnota v premennej „N“ sa musí zhodovať s pozíciou voľného miesta) prejdeme do stavu „Splnena_podmienka“, kde ovládaním motora otvoríme rampu o 90 stupňov. V stave „Rampa_otv“ dáme autu signál (c=1) aby išlo zaparkovať. Rampu zatvoríme až po uplynutí ôsmych sekúnd. Čas sa začne odpočítavať až keď auto prejde okolo rampy a teda ho už ultrazvukový senzor nezachytí. Nakoniec v stave „Rampa_zat“ vypneme všetky žiarovky, ako aj hodnotu premennej „c“ prepíšeme na 0.

4.3.2.1.2.3 Stav - Pomocny_semafor

Pomocný semafor slúži ako vizuálna pomôcka (na displeji) pri volení parkovacieho miesta.

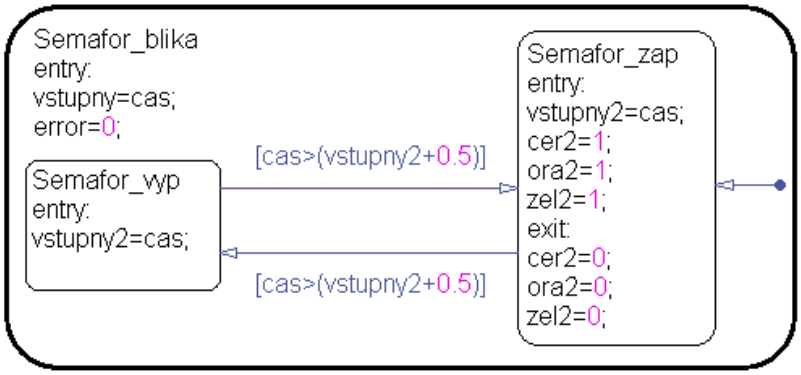


Obr. 45 Stavový diagram stavu Pomocny_semafor

V stave „Start“ zotrváme po dobu, kým sa aktivuje stav „Stlac“. Podmienkami pre prestup do ďalších stavov, kde vždy zasvietime zodpovedajúcu žiarovku, je stlačenie prvého tlačidla. Do posledného stavu „Zle_nedobre“ sa dostaneme až po štvrtom stlačení. Premenná „error“ nadobudne hodnotu 2 a prejdeme do stavu „Semafor_blika“.

4.3.2.1.3 Stav - Semafor_blika

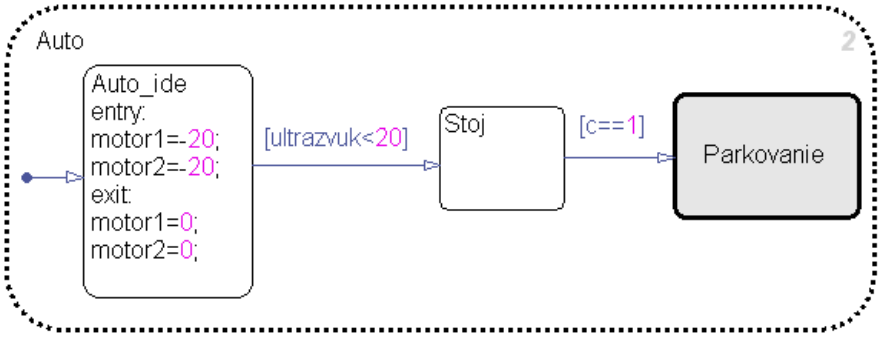
Stav reprezentuje chybnú správu (blikanie semafora), ktorá sa zobrazí na displeji. Pri vstupe prepíšeme hodnotu v premennej „error“ na 0 a v premennej „vstupny“ sa zaznamená aktuálny čas, pomocou ktorého zo stavu odídeme. V stave „Semafor_zap“ riadime všetky tri žiarovky pomocného semafora. Stav „Semafor_vyp“ slúži na vytvorenie 0,5 sekundového oneskorenia, ktoré je potrebné aby sa vytvorilo blikanie.



Obr. 46 Stavový diagram stavu Semafor_blika

4.3.2.2 Stav - Auto

Stav slúži na riadenie auta. Aktivuje sa po spustení simulácie a zaznamenania väčšej hodnoty zvuku ako 300.

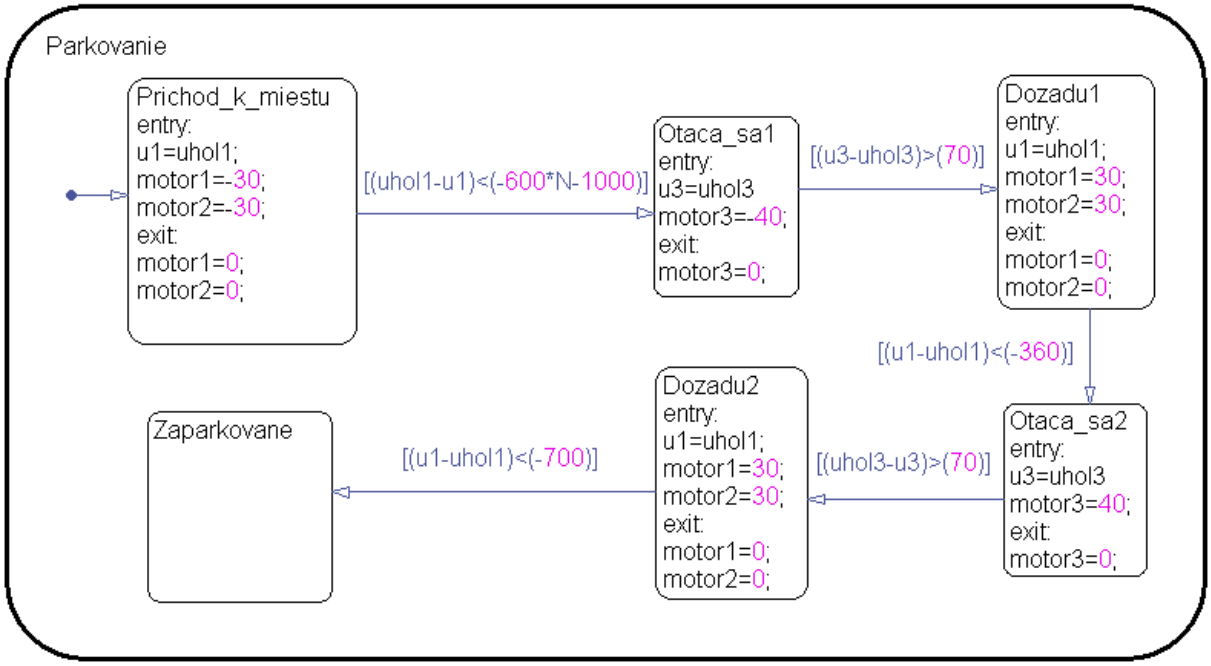


Obr. 47 Stavový diagram stavu Auto

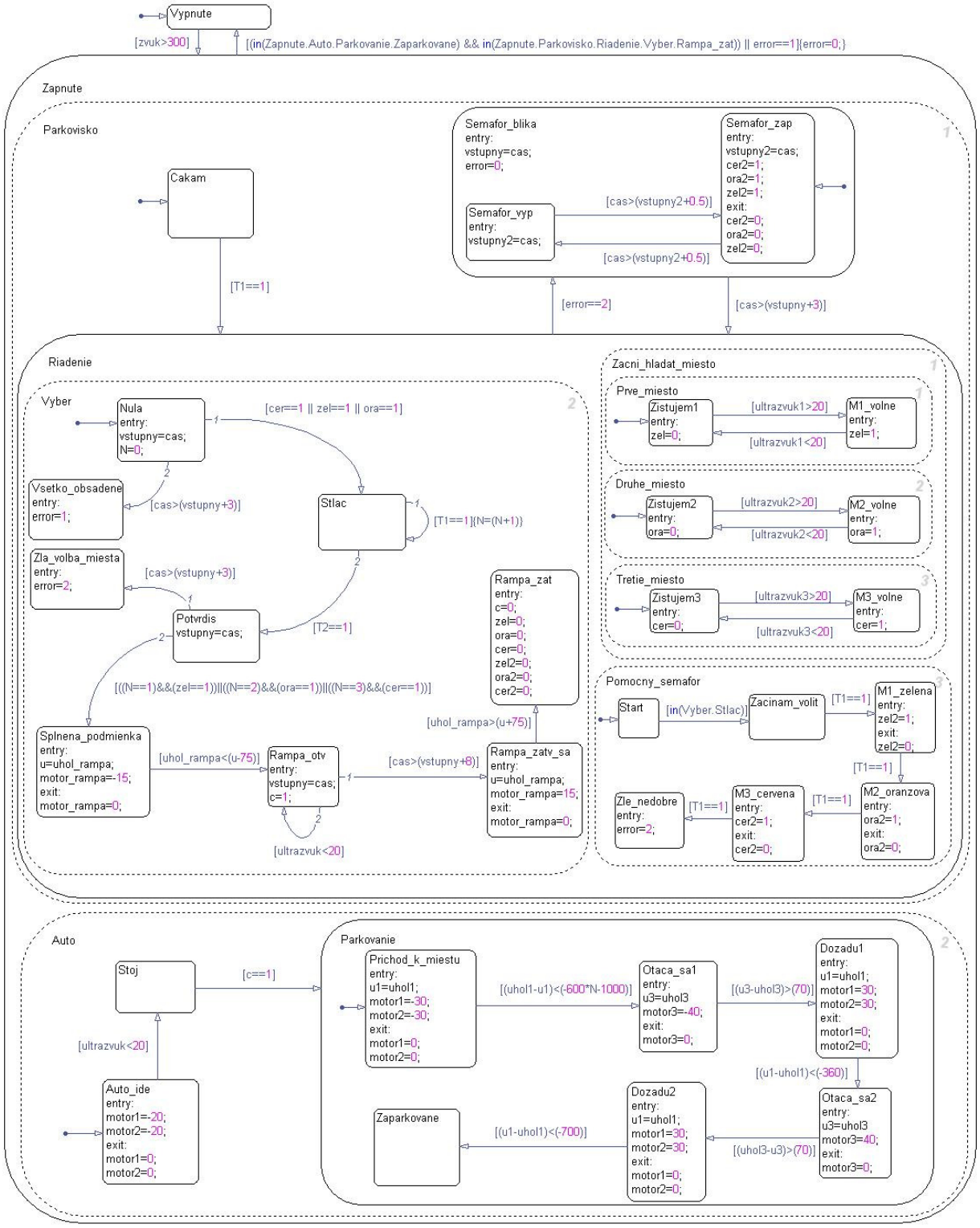
Po vstupe do stavu „Auto_ide“ príkazom „entry“ prikážeme autu, aby sa vydalo smerovalo dopredu. Ak ultrazvukový senzor pred rampou zaregistruje príchod auta (ultrazvuk<20), dá mu povel, aby zastavilo. Dostaneme sa do stavu „Stoj“, kde zostaneme pokiaľ si vodič navolí parkovacie miesto. Vtedy sa rampa otvorí a v lokálnej premennej „c“ bude hodnota1, čím splníme podmienku pre prechod do stavu „Parkovanie“.

4.3.2.2.1 Stav - Parkovanie

Auto (situované pred rampou) dostalo povel k zaparkovaniu. Musí prejsť okolo rampy k najbližšiemu parkovaciemu miestu (N=1), čo predstavuje otočenie motora 1 a 2 o 1600 stupňov (čo zabezpečuje stav „Prichod_k_miestu“). Každé ďalšie miesto je vzdialené o ďalších 600 stupňov. Celý algoritmus stavu už bol vysvetlený v prvej praktickej úlohe.



Obr. 48 Stavový diagram stavu Parkovanie

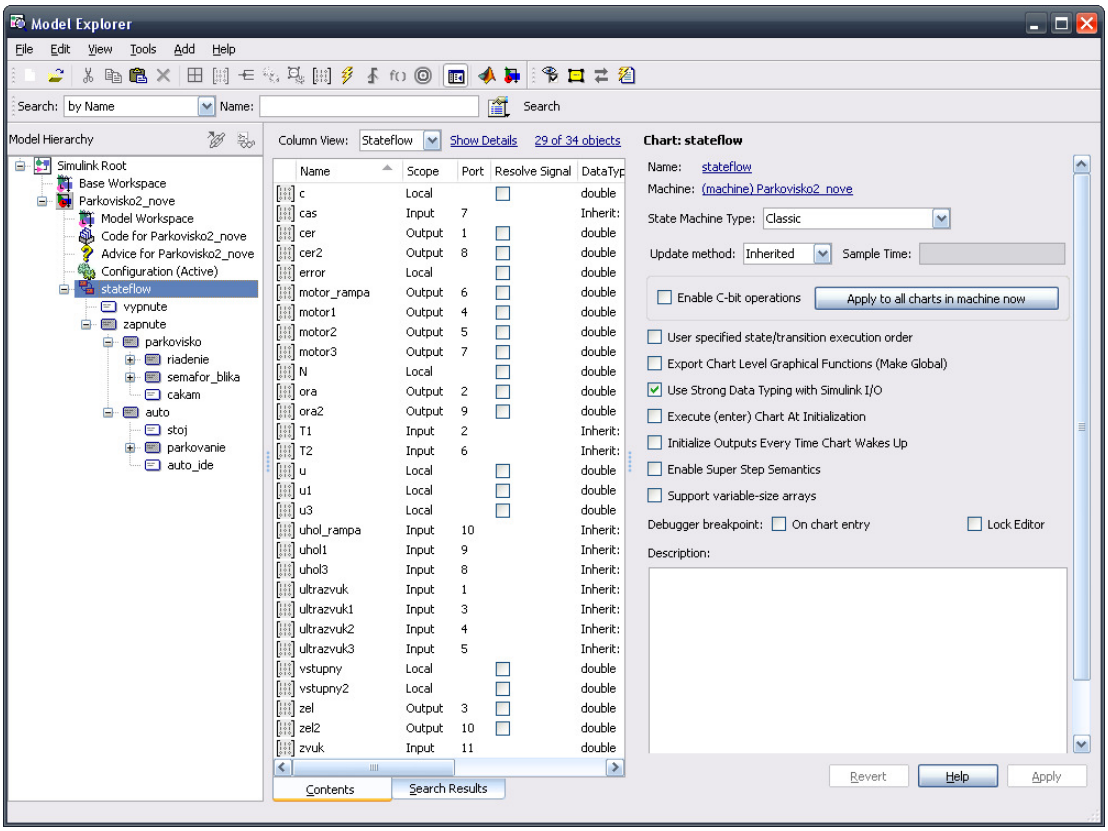


Obr. 49 Stavový diagram celého algoritmu úlohy 2

4.4 Simulink

Do Similinku sme definovali všetky udalosti:

- Vstupy: cas, T1, T2, uhol_rampa, uhol1, uhol3, ultrazvuk, ultrazvuk1, ultrazvuk2, ultrazvuk3, zvuk
- Výstupy: cer, cer2, ora, ora2, zel, zel2, motor_rampa, motor1, motor2, motor3,
- Lokálne premenné: c, error, N, u, u1, u3, vstupny, vstupny2



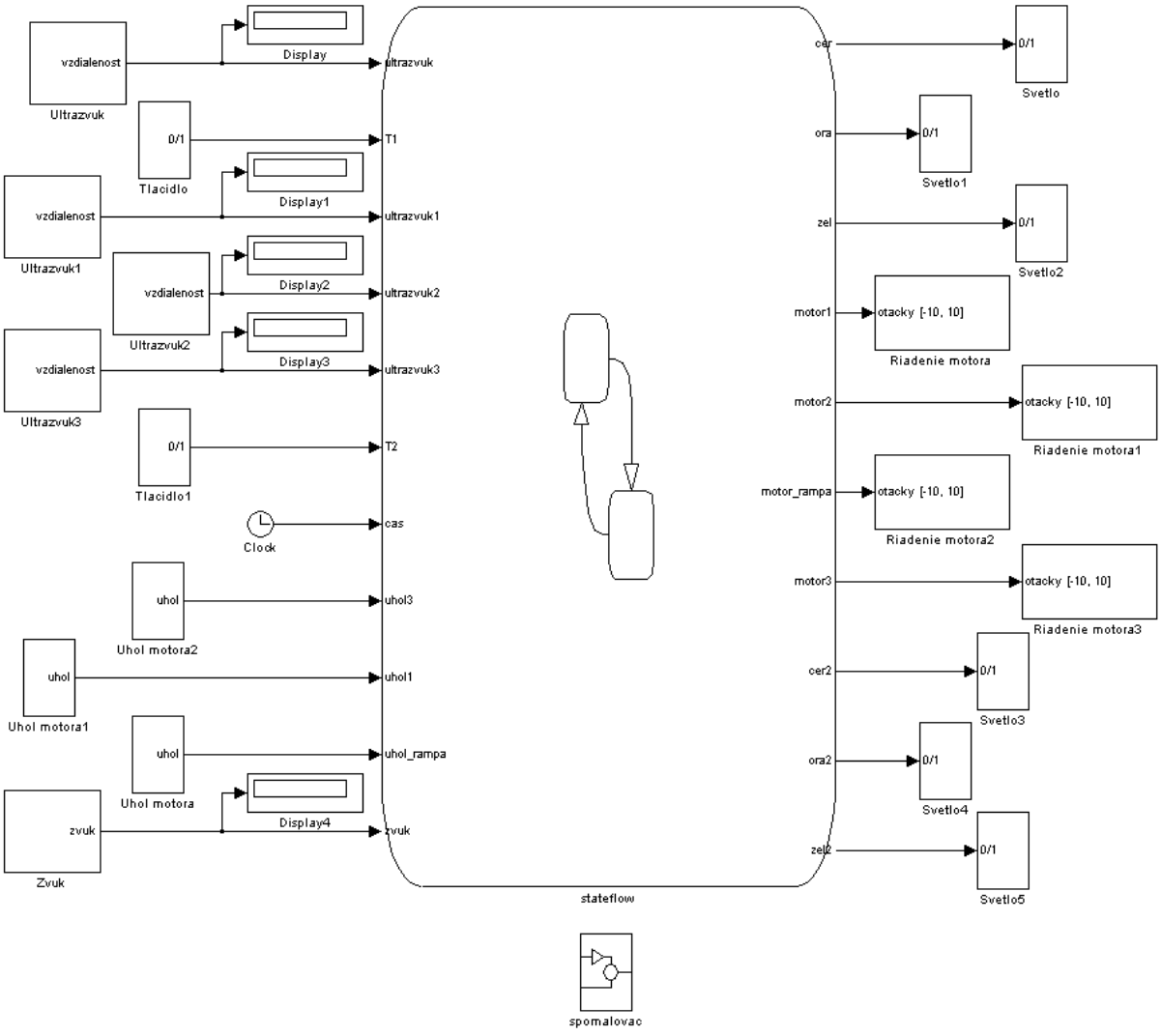
Obr. 50 Model Explorer úlohy 2

Z knižnice sme vybrali bloky, ktoré sme potrebovali:

- Riadenie motora
- Uhol motora
- Ultrazvuk
- Clock

- Zvuk
- Display
- Spomalovac
- Tlačidlo
- Svetlo

Všetky bloky a udalosti prepojíme a nastavíme v nich príslušné číslo (písmeno) portu, podľa informácii zapísaných pri konštrukcii.



Obr. 51 Simulinková schéma úlohy 2

5 Roboti

Úloha spočíva v zostrojení dvoch robotov, ktorí si budú medzi sebou vymieňať informácie prostredníctvom svetiel. Jeden robot bude mat napevno zadefinované rôzne pohybové kombinácie a druhý robot ho bude kopírovať.

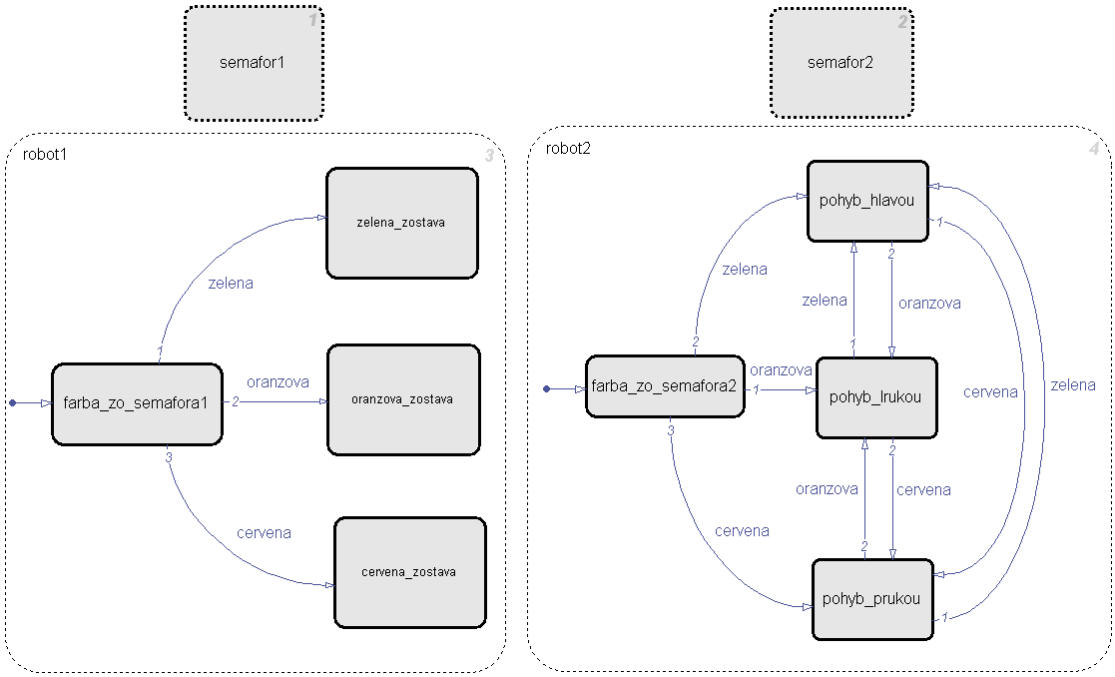
5.1 Analýza problematiky

K splneníí tejto úlohy budeme potrebovať dvoch robotov. Keďže majú po sebe opakovať pohyby, bude vhodné aby boli identickí. Signály im budú odovzdávať dva semafory otočené chrbtom k sebe.

V simulinku vygenerujeme náhodnú číslicu, ktorou dáme prvému semaforu príkaz aby zasvietil príslušnú farbu. Tu prečíta prvý robot pomocou svetelného senzora a na základe toho začne vykonávať priradenú pohybovú kombináciu. Tento pohyb rozozná druhý semafor pomocou stavov (v ktorých sa bude prvý robot nachádzať) a zasvieti zodpovedajúcu farbu. Druhý robot ju rozozná a aktivuje príslušný stav .

Komponenty, ktoré budeme potrebovať, sú rozdelené medzi dvoma (identickými) robotmi a semaformi.

- **Robot**: jedna kocka NXT, tri motory, jeden svetelný senzor
- **Semafor**: jedna kocka NXT, tri žiarovky

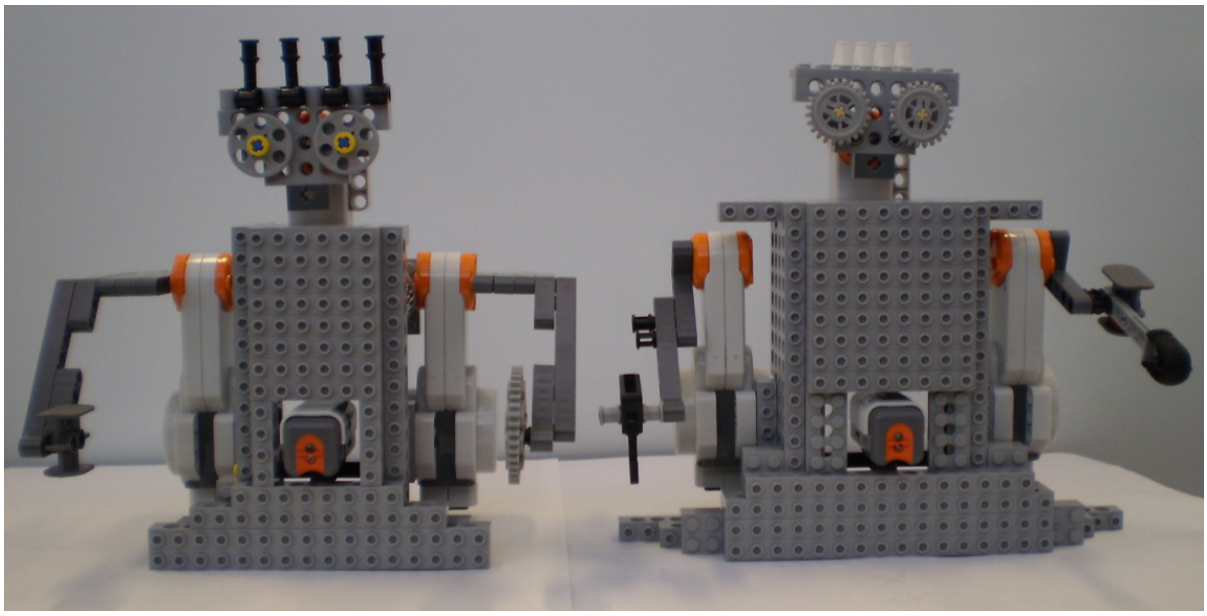


Obr. 52 Základná osnova robotov

5.2 Konštrukcia

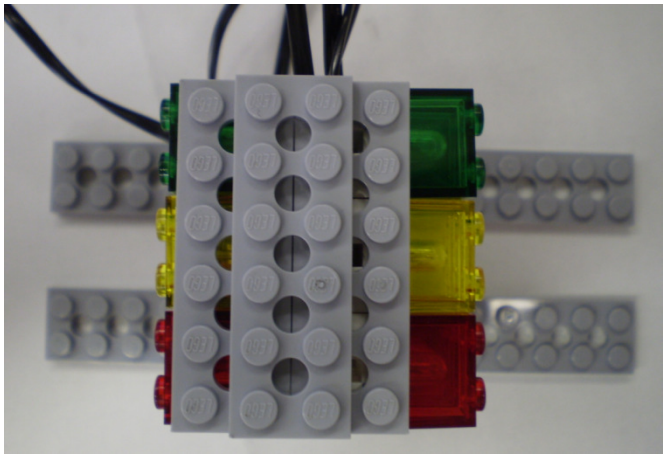
Z analýzy vyplynulo, že sme museli sme skonštruovať dvoch robotov a dva semafory. Stavebnica lega nám už neponúkla široký výber súčiastok. Museli z väčšej časti improvizovať a zostrojiť robotov a semafory z dostupných častíc.

- **Roboti** - Z lega sme postavili základnú kostru, ku ktorej sme pripevnili svetelný senzor (zabezpečenie rozoznávanie farieb semafora) a tri motory (umožňujú vykonávať pohyb).



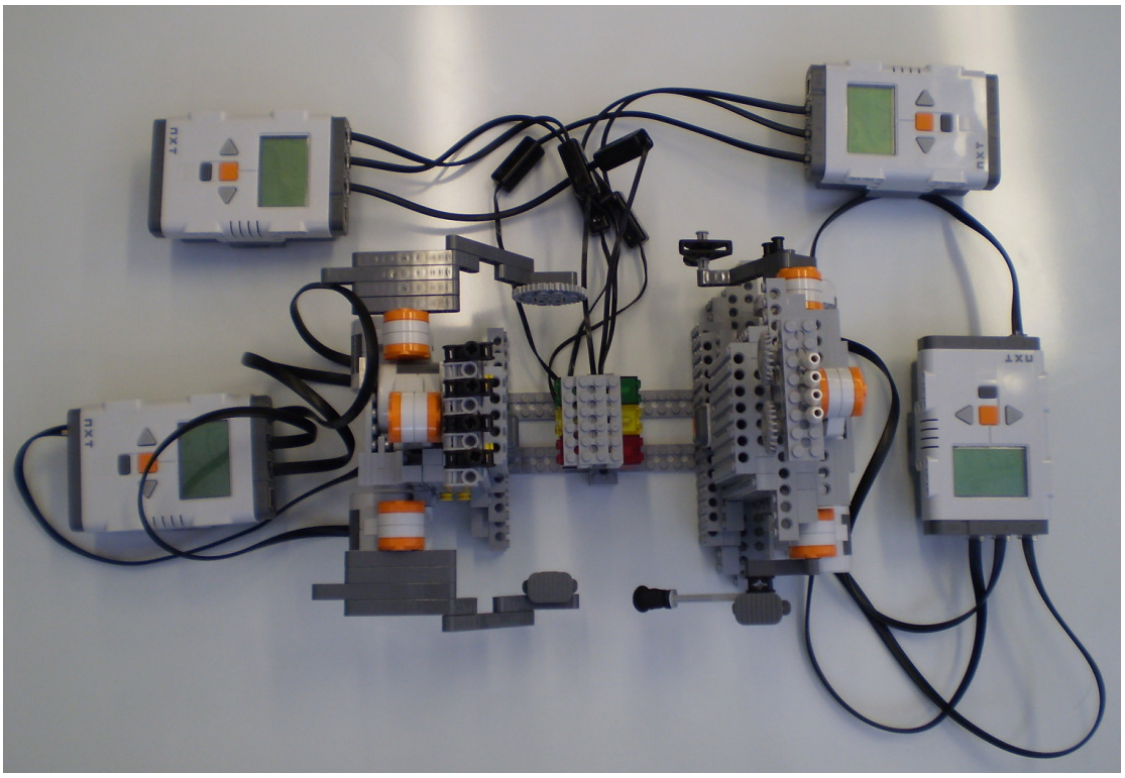
Obr. 53 Model robota1 a robota2

- **Semafory** - Každý semafor je zostrojený z deviatich svetelných žiaroviek. Signál, ktorý zaznamená svetelný senzor, závisí (okrem farby) aj od vzdialenosti. Preto sme k semaforu vytvorili spodné koľaje, pomocou ktorých spojíme semafory a roboty. Takto udržíme konštantnú vzdialenosť medzi nimi.



Obr. 54 Model semaforov

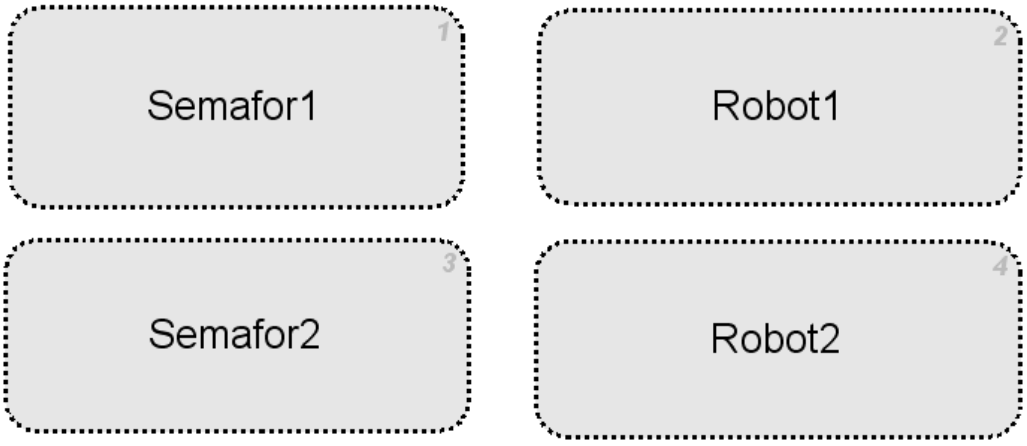
Nakoniec sme všetky časti spojili. Pri pripájaní akčných členov a motorov sme si zapísali číslo (resp. písmeno) portov a kociek. Tieto údaje budeme zadávať blokovým schémam v Simulinku.



Obr. 55 Model robotov

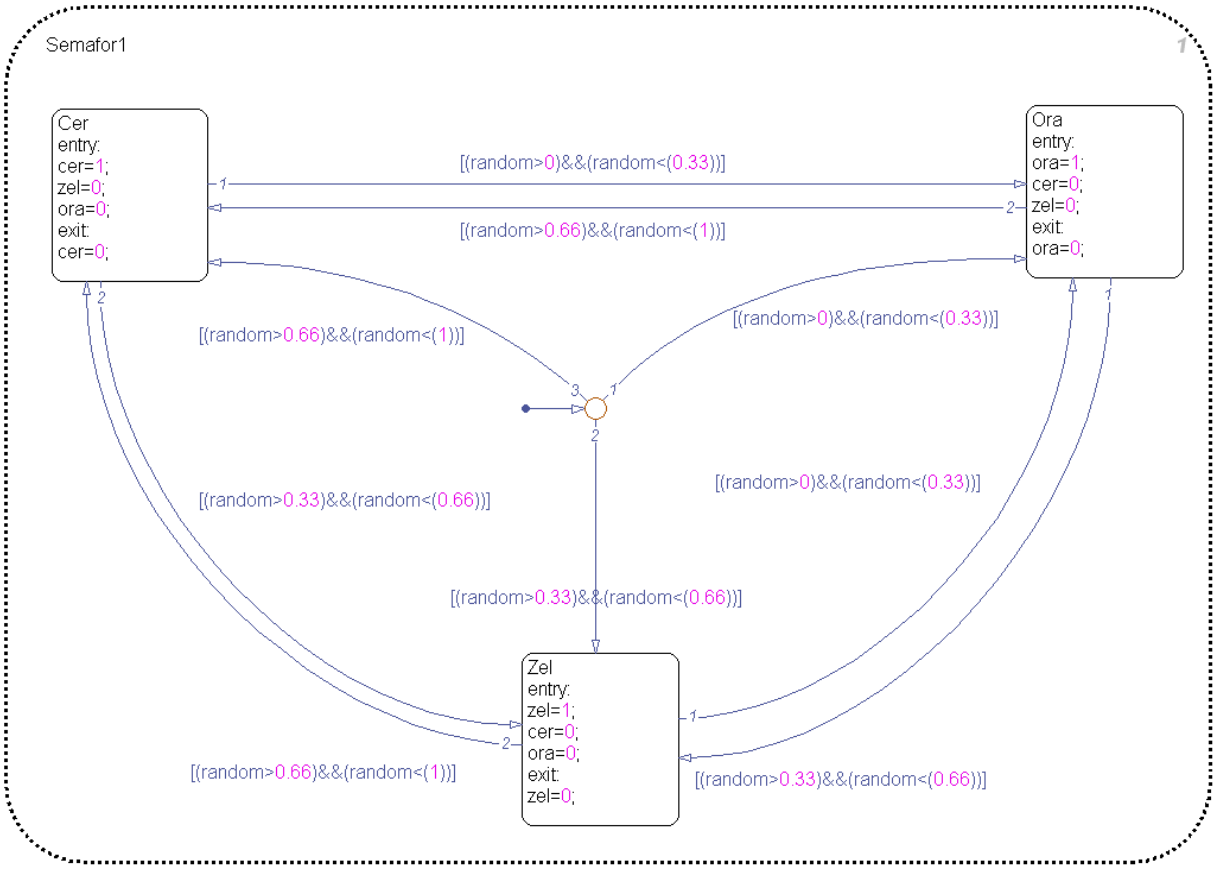
5.3 Stateflow

Podľa informácii z analýzy sme vedeli, že budeme potrebovať štyri základné stavy. Všetky budú aktívne súčastne, preto ich zapojenie bude paralelné. Každý materský stav bude obsahovať viacej dcérskych stavov, preto ich obsah zakryjeme pomocou možnosti „subchart“. Takto sme dosiahli prehľadnejší stavový diagram.



Obr. 56 Stavový diagram robotov

5.3.1 Stav - Semafor1



Obr. 57 Stavový diagram stavu Semafor1

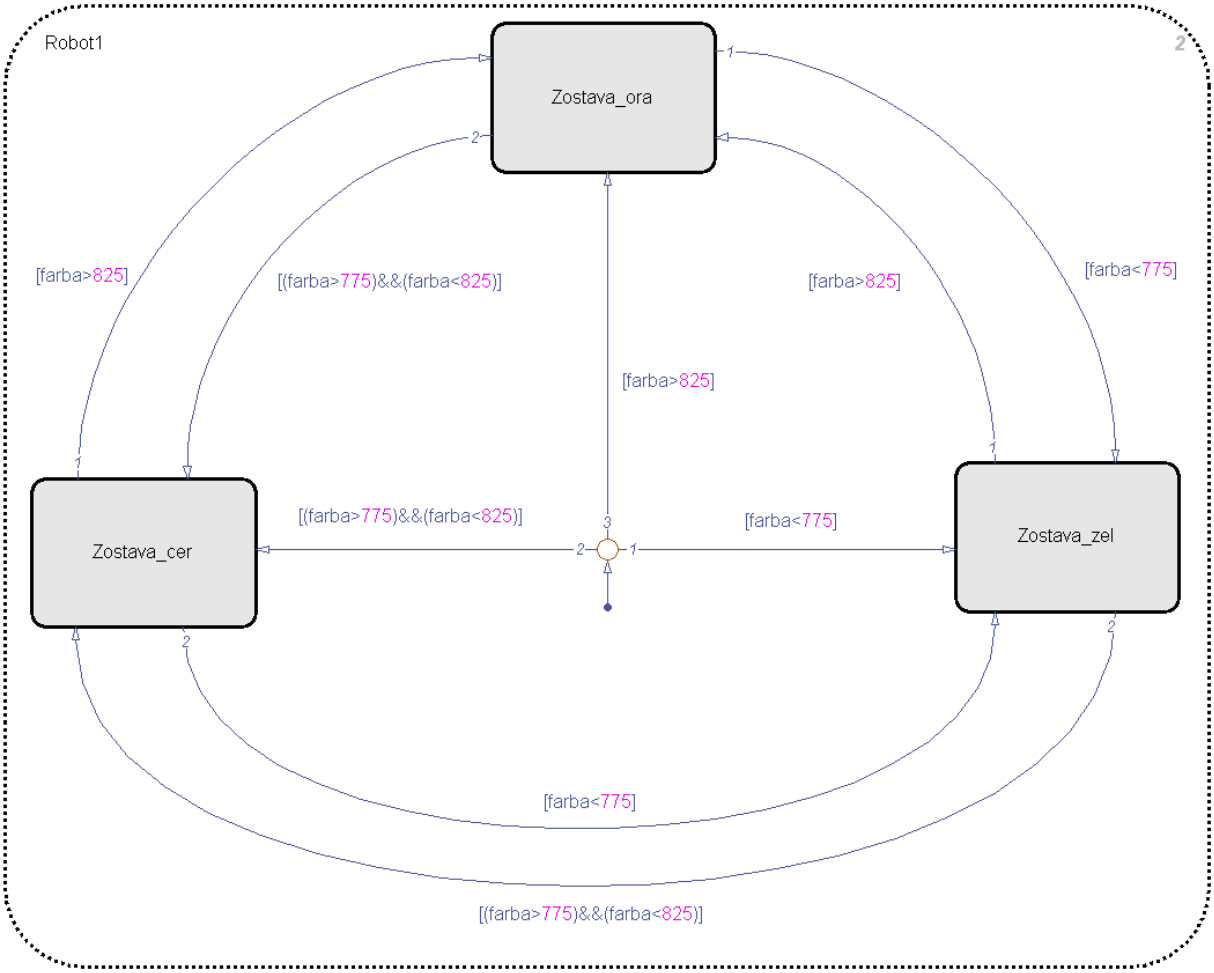
V Simulinku vygenerujeme (každých dvadsať sekúnd) náhodné číslo „random“ z intervalu <0;1>, pomocou ktorého semafor1 zasvieti farbu. Na začiatku sa nachádzame na spojovacej križovatke. Ak náhodné číslo padne do intervalu <0;0,33> prejdeme do stavu „Ora“, kde sa rozsvieti oranžové svetlo. Keby číslo bolo z intervalu <0,33;0,66> vstúpili by sme do stavu „Zel“ a zasvietilo by sa zelené svetlo. Do stavu „Cer“ sa dostaneme, ak naše náhodné číslo bude z intervalu <0,66;1>. V tomto prípade sa rozsvieti červené svetlo. Všetky stavy sú navzájom prepojené, aby sa zabezpečil prechod z každého stavu.

5.3.2 Stav - Robot1

Stav „Semafor1“ nám každých dvadsať sekúnd rozsvieti náhodnú farbu, ktorú vieme rozoznať pomocou svetelného senzora. Do premennej „farba“ nám zapíše hodnotu zodpovedajúcej farby.

Z výsledkov merania svetelného senzora1 sme ku každému stavu (farbe) priradili interval:

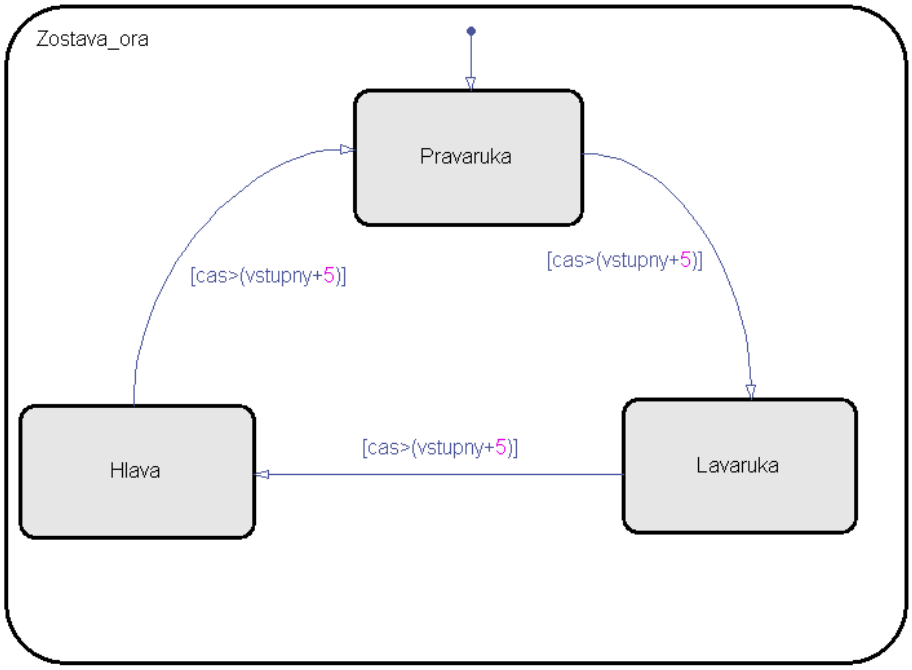
- Zostava_ora – (825;∞)
- Zostava_cer – (775;825)
- Zostava_zel – (0;775)



Obr. 58 Stavový diagram stavu Robot1

5.3.2.1 Stav - Zostava_ora

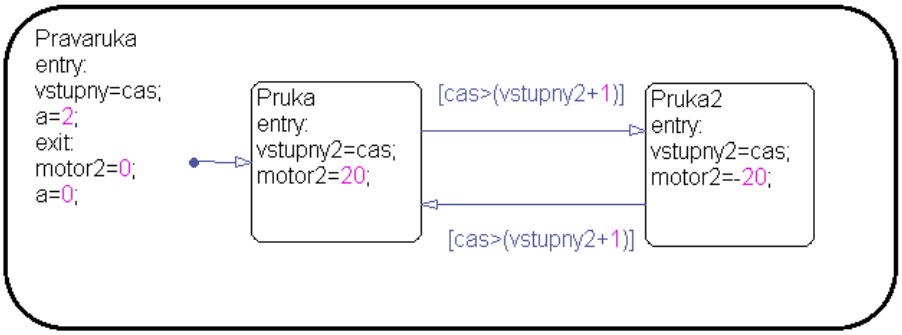
Stav sa aktivuje ak hodnota v premennej „farba“ bude väčšia ako 825. Ak podmienka nebude splnená (čo bude znamenať, že sa v Simulinku vygenerovalo nové číslo) prejdeme do ďalšieho stavu.



Obr. 59 Stavový diagram stavu Oranzova_zostava

5.3.2.1.1 Stav - Pravaruka

Pri vstupe do stavu si zapíšeme do lokálnej premennej „a“ hodnotu 2 (ktorú budeme potrebovať pri stave „Semafor2“) a do premennej „vstupný“ aktuálny čas, pomocou ktorej budeme môcť po piatich sekundách odísť zo stavu „Pravaruka“. Pri výstupe zo stavu sa premena „a“ prepíše na 0 a vypneme motor2.



Obr. 60 Stavový diagram stavu Prava_ruka

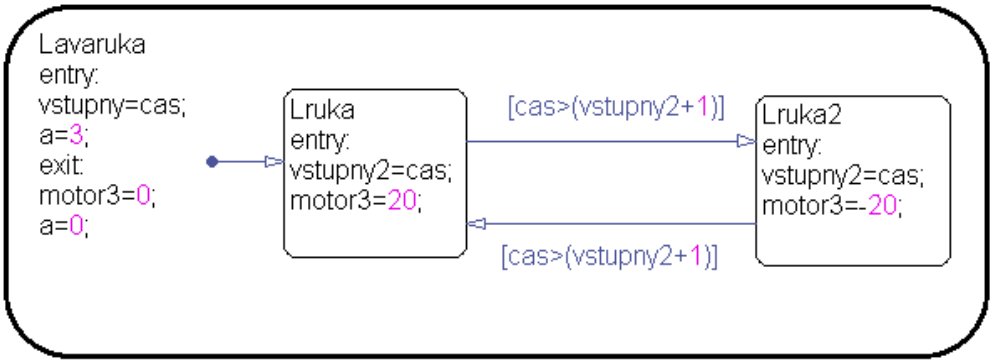
Ako prvý stav sa aktivuje „Pruka“. Vykoná sa „entry“ príkaz, čím zapneme motor2, ktorý ovláda pravú ruku. Do premennej „vstupný2“ zapíšeme aktuálny čas. (Treba si uvedomiť, že premenná „vstupný“ slúži na prechod medzi zostavami a „vstupný2“ na prestup medzi stavmi v nich).

Po uplynutí jednej sekundy prejdeme do stavu „Pruka2“. Opäť si poznačíme aktuálny čas do premennej „vstupný2“ a motor2 prepneme tak, aby zmenil smer pohybu. V stave zotrváme iba sekundu a vrátime sa naspäť do stavu „Pruka“.

Takýmto prepínaním medzi stavmi dosiahneme, aby sa pravá ruka pohybovala približne o 90 stupňov. (Rovnaký výsledok by sme mohli zariadiť ovládaním uhol motora2).

5.3.2.1.2 Stav - Lavaruka

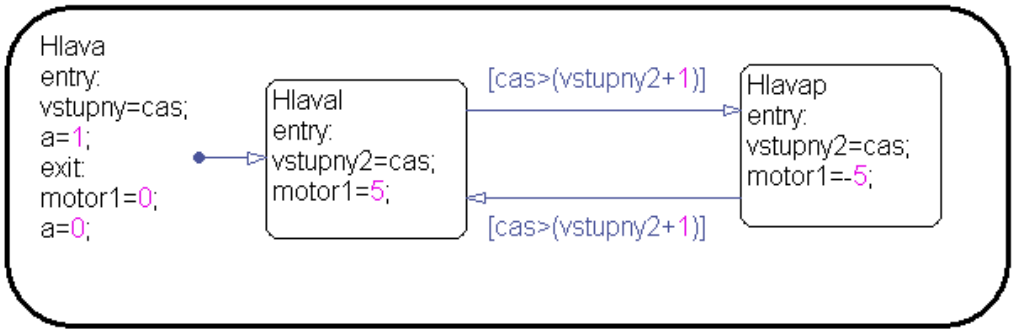
Stav „Lavaruka“ analogicky zodpovedá stavu „Pravaruka“. Do premennej „a“ zapíšeme 3 a pohyb ľavej ruky zabezpečujeme ovládaním motora3.



Obr. 61 Stavový diagram stavu Lava_ruka

5.3.2.1.3 Stav - Hlava

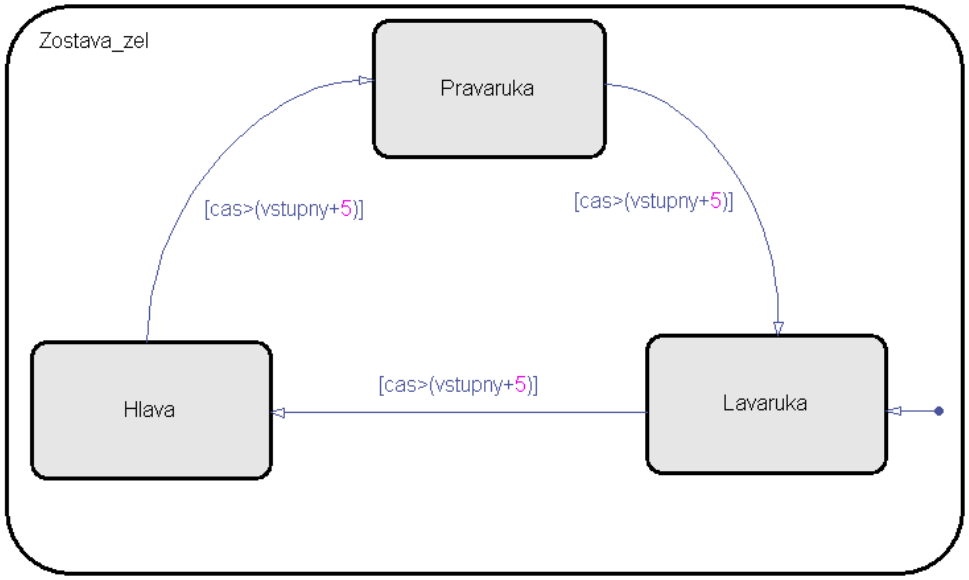
Podobne ovládame aj stav „Hlava“. Motorom1 ovládame otáčanie hlavy a premenná „a“ nadobúda hodnotu 1.



Obr. 62 Stavový diagram stavu Hlava

5.3.2.2 Stav - Zostava_zel

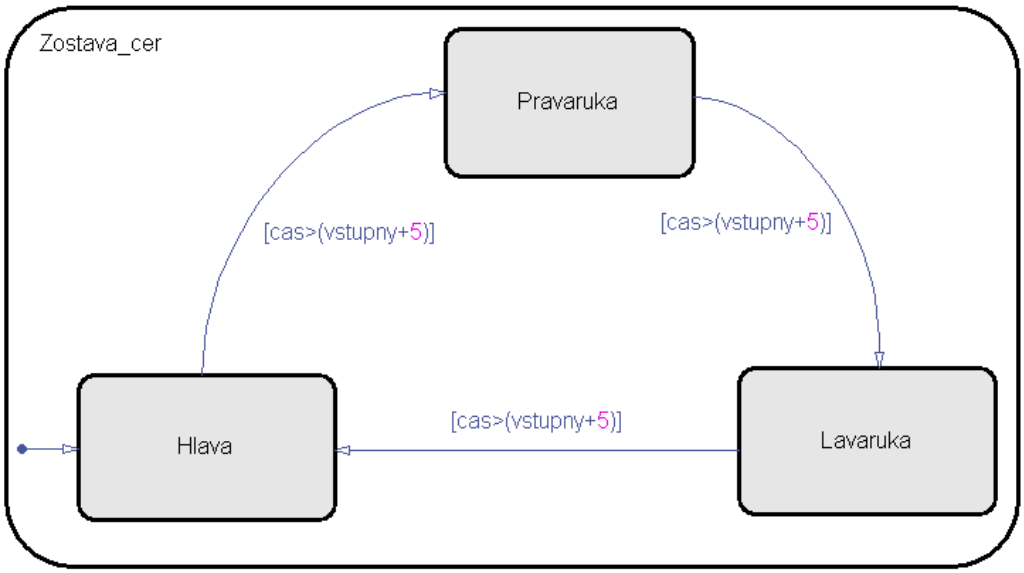
Zostave pre zelenú farbu je presná kópia stavu „Zostava_ora“. Rozdiel je iba v štartujúcim stavom, ktorý je „Lavaruka“.



Obr. 63 Stavový diagram stavu Zelena_zostava

5.3.2.3 Stav - Zostava_cer

Rovnakým duplikátom je aj stav „Zostava_cer“. Stav sa aktivuje, keď svetelný senzor rozpozná červenú farbu na prvom semafory. V tomto stáva sa začne ako prvá hýbať hlava.

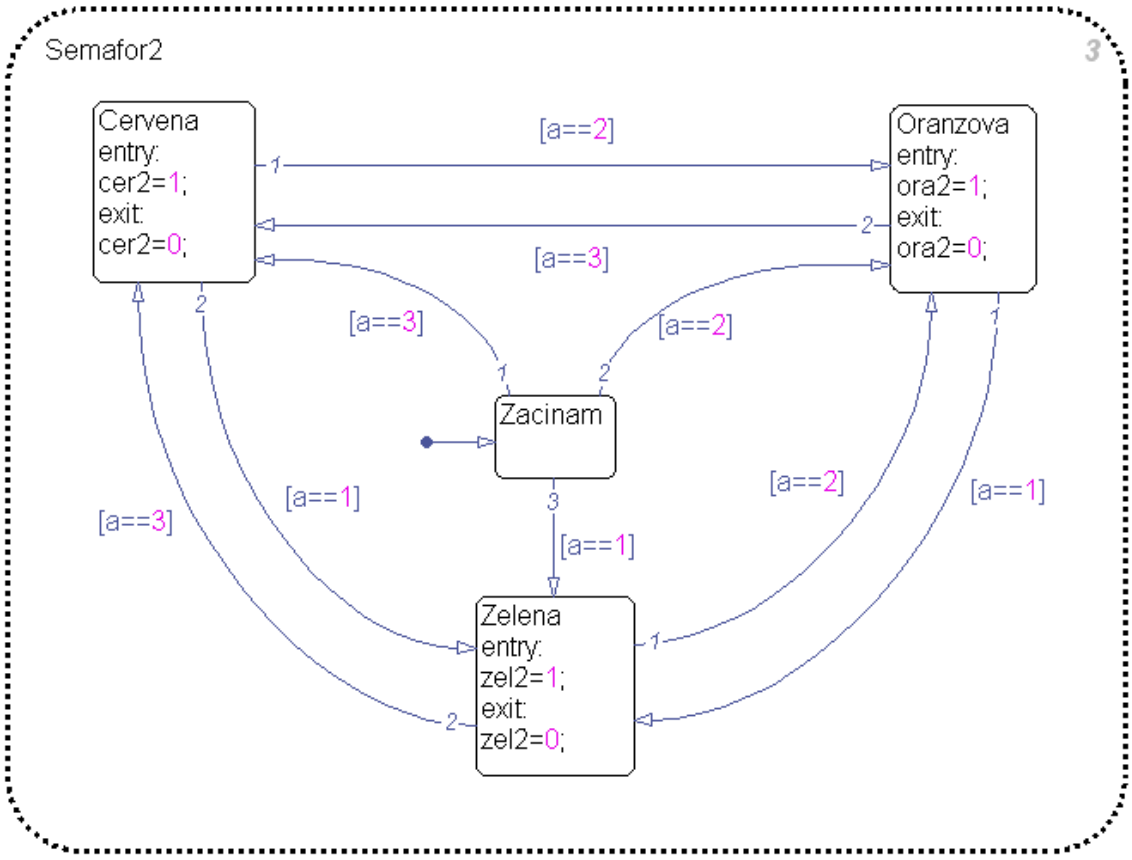


Obr. 64 Stavový diagram stavu Cervena_zostava

5.3.3 Stav - Semafor2

Semafor2 sa už neriadi podľa náhodného čísla zo Simulinku, ale od hodnoty v premennej „a“. Ta sa mení podľa toho v akom stave sa nachádza prvý robot. (Ide o alternatívne riešenie, tento problém sme mohli vyriešiť aj napríklad pomocou príkazu „in( )“).

Do stavu „Cervena“ sa dostaneme ak robot1 bude pohybovať ľavou rukou (a=3). Rozsvieti sa na druhom semafore červené svetlo (cer2=1). Pri prestupe do ďalšieho stavu ju vypneme (cer2=0). Ak robot1 bude hýbať pravou rukou, v premennej „a“ bude hodnota 2 a semafor2 zasvieti oranžové svetlo (ora2=1). Príkazom „exit“ ju vypneme (ora2=0). Stav „Zelena“ rozsvieti zelené svetlo (zel2=1). Bude aktívny, keď prvý robot bude točiť hlavou. Rovnako aj tu pri prestupe do ďalšieho stavu zelené svetlo za sebou vypneme (zel2=0).



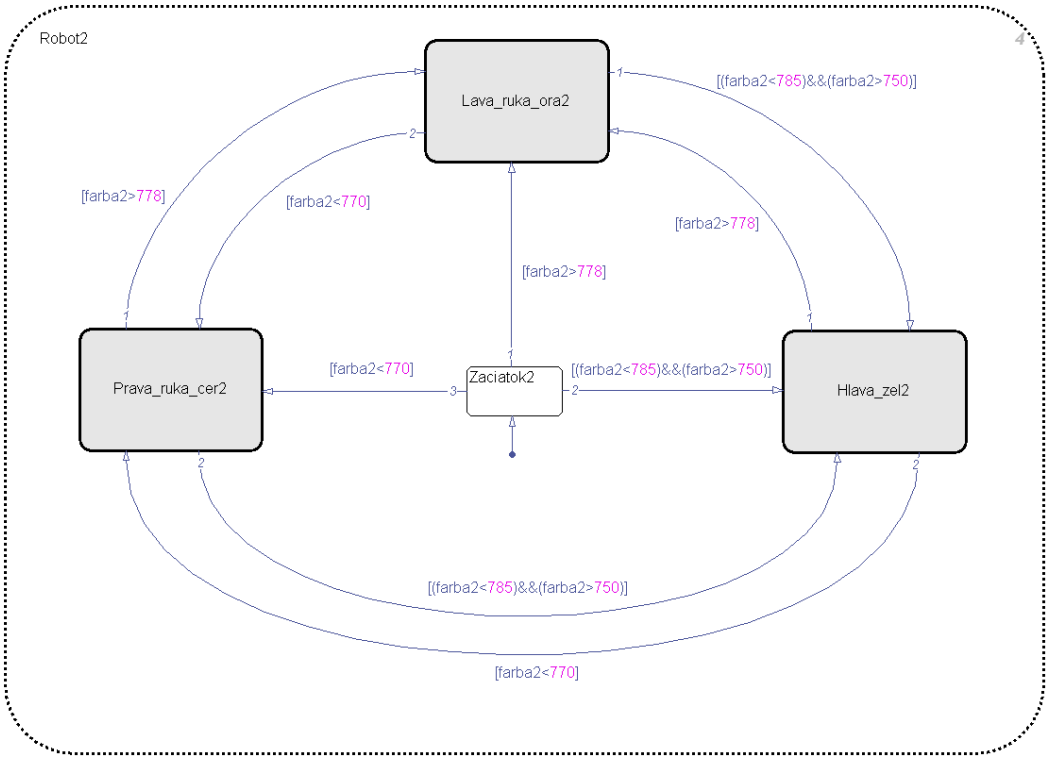
Obr. 65 Stavový diagram stavu Semafor2

5.3.4 Stav - Robot2

Robot2 príjma vo svetelnej podobe informácie od semafora2. Jednotlivé farby rozozná pomocou svetelného senzora2 a zapíše do premennej „farba2“.

Z výsledkov merania svetelného senzora2 sme ku každému stavu (farbe) priradili interval:

- Lava_ruka_ora2 – (778;∞)
- Hlava_zel2 – (750;785)
- Prava_ruka_cer2 – (0;770)

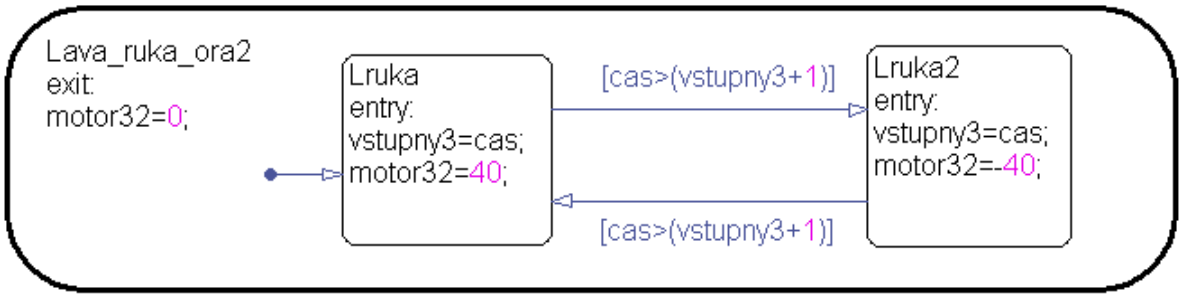


Obr. 66 Stavový diagram stavu Robot2

5.3.4.1 Stav - Lava_ruka_ora2

Do stavu vstúpime, keď svetelný senzor2 zaregistruje oranžovú farbu („farba2“ bude mať hodnotu v intervale (785;∞)). Aktivuje sa stav „Lruka“, v ktorom iniciujeme pohyb ľavej ruky smerom nadol (motor32=40). Po uplynutí jednej sekundy prejdeme do stavu „Lruka2“, v ktorom zmeníme smer ľavej ruky (motor32=-40). Takýmto kolobehom ovládame pohyb

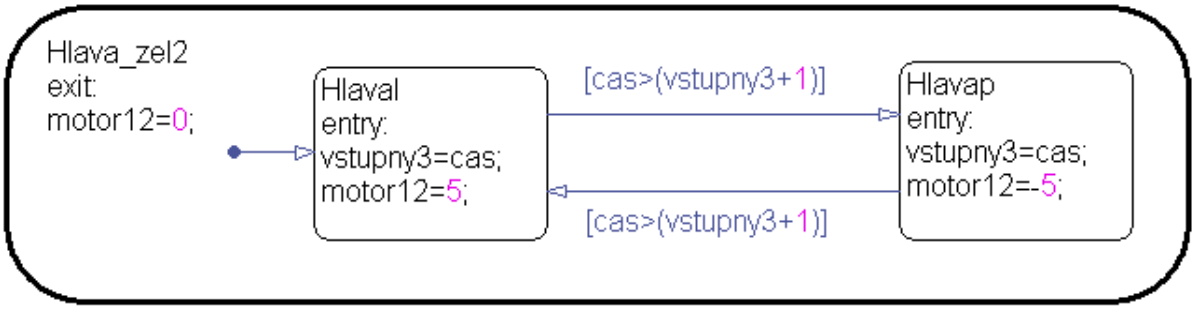
ruky približne o 90 stupňov. Pri výstupe zo stavu „Lava_ruka_ora2“ motor32 vypneme (motor32=0).



Obr. 67 Stavový diagram stavu Lava_ruka_ora2

5.3.4.2 Stav - Hlava_zel2

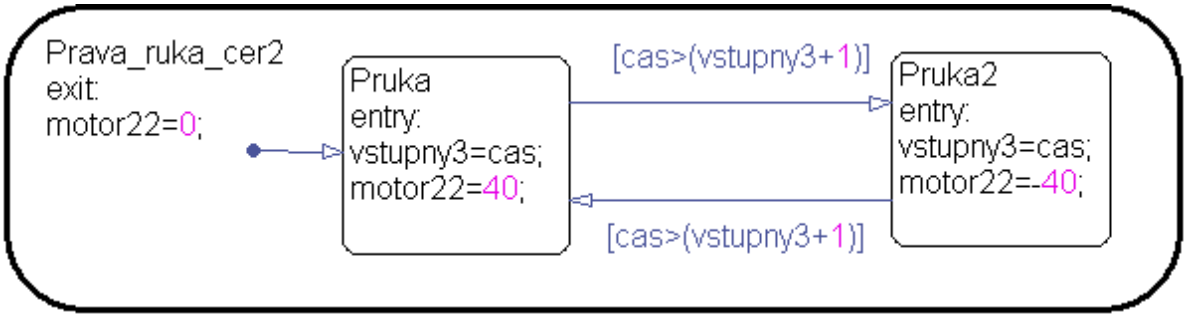
Analogicky ako v stave „Lava_ruka_ora2“ ovládame aj tu pohyb hlavy pomocou motora12.



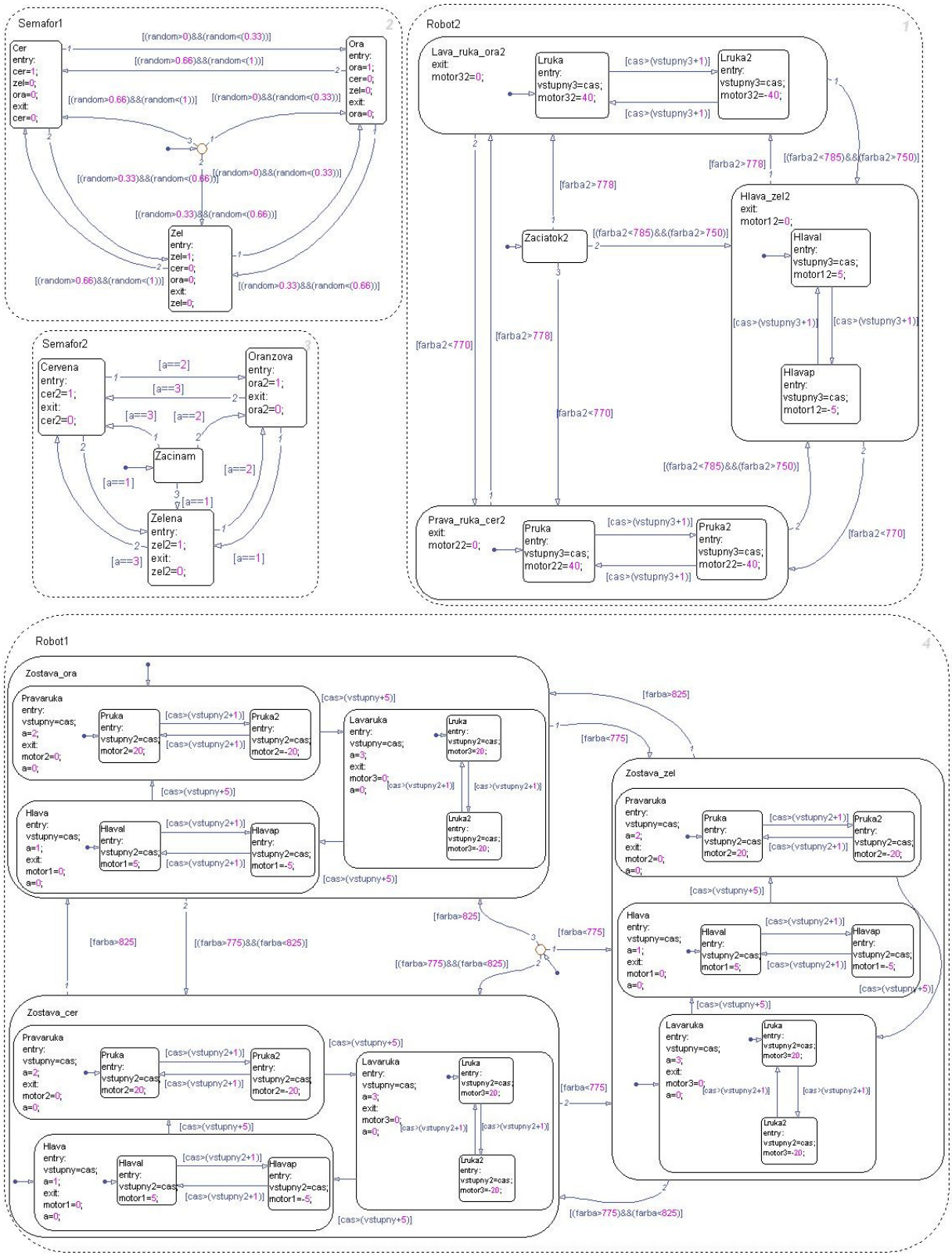
Obr. 68 Stavový diagram stavu Hlava_zel2

5.3.4.3 Stav - Prava_ruka_cer2

Aj v tomto stave používame rovnaký princíp ovládania pravej ruky.



Obr. 69 Stavový diagram stavu Prava_ruka_cer2

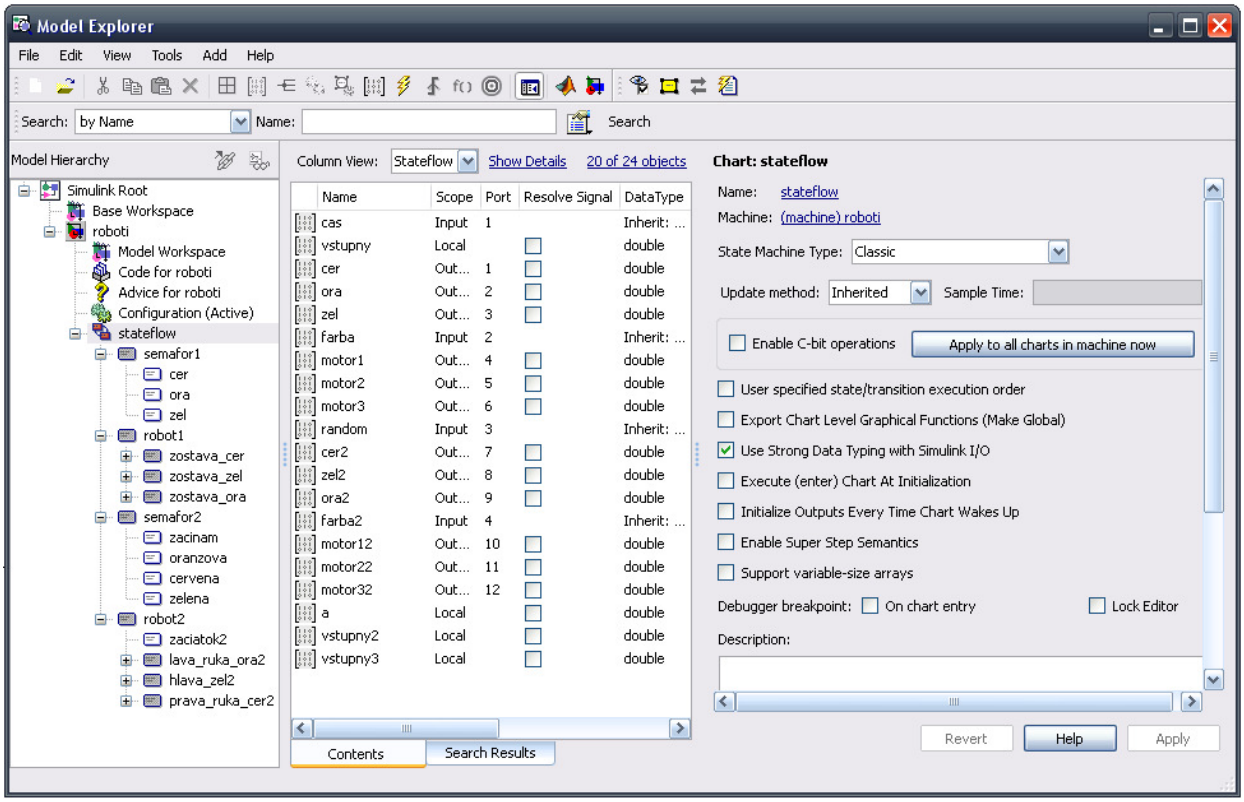


Obr. 70 Stavový diagram celého algoritmu úlohy 3

5.4 Simulink

Do Similinku sme definovali všetky udalosti:

- Vstupy: cas, random, farba, farba2
- Výstupy: motor1, motor2, motor3, motor12, motor22, motor32, cer, ora, zel, cer2, ora2, zel2
- Lokálne premenné: a, vstupny, vstupny2, vstupny3



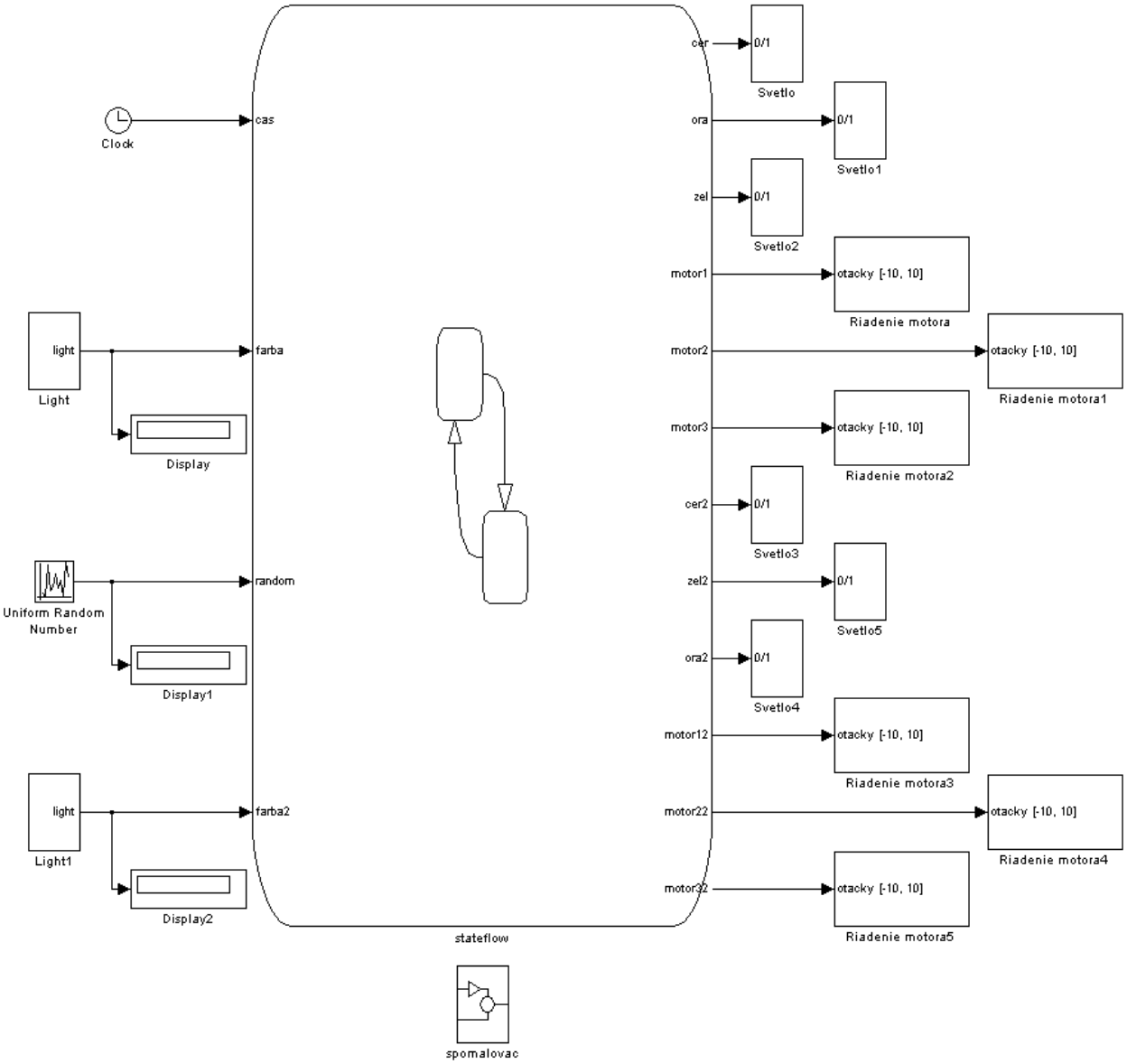
Obr. 71 Model Explorer úlohy 3

Z knižnice sme vybrali bloky, ktoré sme potrebovali:

- Riadenie motora
- Clock
- Light
- Unifirm random number
- Svetlo

- Display
- Spomalovac

Všetky bloky sme spojili k príslušným vstupom (resp. výstupom) a zadali sme správne čísla kociek a portov, ktoré sme si zapísali pri konštrukcii.



Obr. 72 Simulinková schéma úlohy 3

Záver

Hlavným cieľom práce bolo overiť možnosti stavebnice Lego Mindstorms NXT na reálnych procesoch, s ktorými sa môžeme každodenne stretnúť. Model sme riadili pomocou softvéru Stateflow, kde sme pomocou stavového riadenia implementovali program. Prácu sme rozdelili na viacej kapitol.

V prvej kapitole sme predstavili stavebnicu Lego Mindstorms NXT, jeho riadiacu kocku, jednotlivé senzory a akčné členy. Druhá kapitola bola venovaná softvéru Stateflow, ako balíka programu MATLAB. Oboznámili sme sa okrem jeho charakteristiky aj s prácu v ňom. V posledných troch kapitolách sme vysvetlili, ako sme postupovali pri riešení jednotlivých úloh. Na začiatku sme si každú úlohu analyzovali, čím sme získali základné informácie, ktoré sme využili pri konštrukcii modelov a tvorení algoritmu.

V prvej úlohe sme naprogramovali model auta, ktorý na základe vstupného zvukového signálu dokázal automaticky nájsť parkovacie miesto a aj zaparkovať. Program sme prispôsobili tak, aby zaparkované auto sa na vstupný zvykový signál premiestnilo.

Druhá úloha predstavuje nadstavbu prvej úlohy. Zostrojili sme model parkoviska, ktorý okrem parkovacích miest obsahovalo aj ovládací panel a prístupovú rampu. Každé auto fotobunka zaznamenala a vydala príkaz na zastavenie auta. Takto sme zabezpečili, aby auto nevrazilo do rampy. Vodič si na ovládacom paneli môže zistiť či je niektoré parkovacie miesto voľné. Ak sa nachádza voľný parkovací box, tak ovládací panel informuje vodiča, o ktoré miesto ide. Pomocou tlačidiel si vodič môže zvoliť kam má auto zaparkovať.

V poslednej (tretej) úlohe sme overili možnosti prenosu informácii medzi kockami NXT. Zostrojili sme dva modely robotov. Prvý robot na základe náhodného čísla vykonával vopred naprogramované „tanečné zostavy“ a odovzdával informácie druhému robotovi prostredníctvom semafora. Ten rozpoznaním svetla na semafore vedel, akou časťou pohybuje prvý robot a dokázal ho tak presne kopírovať. Pomocou tejto úlohy sme vytvorili a otestovali ako kocky NXT môžu medzi sebou komunikovať.

Literatúra

[1] Mindstorms education – Metodika NXT © 2006 The Lego Group.

[2] **Mathworks, Inc.** Documentation MATLAB. Getting Started. [Online] 23. 11 2009. http://www.mathworks.com/access/helpdesk/help/techdoc/.

[3] **Mathworks, Inc.** Documentation Stateflow. User's Guide. [Online] 3. 12 2009. http://www.mathworks.com/access/helpdesk/help/toolbox/stateflow/.