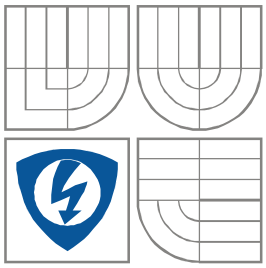
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF CONTROL AND INSTRUMENTATION

METODY ANALÝZY STAVOVÝCH AUTOMATŮ PRO VESTAVNÉ APLIKACE

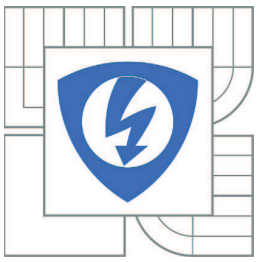
ANALYSIS OF STATE AUTOMATAS FOR EMBEDDED APPLICATIONS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR
Bc. Marek Maťas

VEDOUCÍ PRÁCE
SUPERVISOR
doc. Ing. Pavel Václavek, Ph.D.

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav automatizace a měřicí techniky

Diplomová práce

magisterský navazující studijní obor
Kybernetika, automatizace a měření

Student:	Bc. Marek Maťas	***ID:***	72924
Ročník:	2	***Akademický rok:***	2010/2011

NÁZEV TÉMATU:

Metody analýzy stavových automatů pro vestavné aplikace

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s použitím stavových automatů ve vestavných systémech. Zaměřte se zejména na otázku modelování a posouzení vlastností stavových automatů (minimálnost, dosažitelnost stavu,..). V prostředí Matlab-Simulink vytvořte prostředky pro popis konečných automatů nezávislé na komerčně dostupných toolboxech řešících obdobnou problematiku. Navrhněte vhodnou datovou reprezentaci konečného automatu, nad kterou se následně pokuste implementovat vybraný algoritmus analýzy konečného automatu.

DOPORUČENÁ LITERATURA:

Cassandras, C.G., Lafortune, S.: Introduction to Discrete Event Systems

Termín zadání:	7.2.2011	***Termín odevzdání:***	23.5.2011

Vedoucí práce: doc. Ing. Pavel Václavek, Ph.D.

prof. Ing. Pavel Jura, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

Abstrakt

Táto diplomová práca sa zaoberá analýzou stavových automatov pre vstavané aplikácie. Problematika konečných stavových automatov je rozobraná teoreticky. Dokument ďalej obsahuje návrh prostriedkov pre modelovanie konečných stavových automatov v prostredí Matlab/Simulink. Je navrhnutá dátová reprezentácia konečného automatu. Nad touto dátovou reprezentáciou je aplikovaný algoritmus minimalizácie. Nakoniec je implementovaný algoritmus na generovanie kódu v jazyku C.

Kľúčové slová

konečný stavový automat, vstavaný systém, systémy diskrétnych udalostí, minimalizácia stavového priestoru

Abstract

This master's thesis deals with analysis of state machines for embedded applications. The issue of finite-state machine is described theoretically. The document also contains a proposal for funding for modeling finite state machines in Matlab/Simulink. It is designed data representation of finite automaton. Over this data representation algorithm of minimization is applied. Finally, the algorithm is implemented to generate code in C language.

Keywords

finite state machine, embedded systems, discrete event systems, state space minimization

Bibliografická citácia:

MAŤAS, M. *Metody analýzy stavových automatů pro vestavné aplikace.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2011. 59 s. Vedoucí diplomové práce doc. Ing. Pavel Václavek, Ph.D.

Prehlásenie

„Prehlasujem, že svoju diplomovú prácu na tému "Metody analýzy stavových automatů pro vestavné aplikace" som vypracoval samostatne pod vedením vedúceho diplomovej práce a s použitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor uvedenej diplomovej práce ďalej prehlasujem, že v súvislosti s vytvorením tejto diplomovej práce som neporušil autorské práva tretích osôb, obzvlášť som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a som si plne vedomý následkov porušenia ustanovení § 11 a nasledujúcich autorského zákona č. 121/2000 Zb., vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovení časti druhej, hlavy VI. diel 4 Trestného zákonníka č. 40/2009 Zb.

V Brne dňa: …… ……

podpis autora

Poďakovanie

Ďakujem vedúcemu diplomovej práce doc. Ing. Pavlovi Václavkovi, Ph.D. za účinnú metodickú, pedagogickú a odbornú pomoc a ďalšie cenné rady pri spracovaní mojej diplomovej práce.

V Brne dňa: …… ……

podpis autora

OBSAH

ZOZNAM OBRÁZKOV

ZOZNAM TABULIEK

ÚVOD

Táto práca sa venuje modelovaniu dynamického chovania stavových automatov. Kladie si za cieľ vytvoriť programovú podporu pre modelovanie konečných stavových automatov v prostredí Matlab/Simulink a to bez ohľadu na komerčne dostupné riešenia. Následne vytvoriť vhodnú dátovú reprezentáciu, ktorá bude takto namodelovaný stavový automat reprezentovať a aplikovať nad ňou algoritmus minimalizácie. V poslednom kroku bude naprogramovaný algoritmus generujúci zdrojový kód v jazyku C. Tento kód bude reprezentovať namodelovaný stavový automat.

Úvodné dve kapitoly tejto práce predstavujú teoretický rozbor problematiky. Prvá sa venuje vstavaným systémom. Objasňuje čo si pod týmto pojmom predstavujeme, aké sú ich vlastnosti a oblasti využitia. Druhá kapitola sa zaoberá problematikou stavových automatov. Vysvetlený je tu pojem systémy diskrétnych udalostí ako aj to, ktorú oblasť systémov predstavujú práve tieto systémy. Ďalej sú tu objasnené pojmy ako jazyky a automaty, ich vzájomná súvislosť a operácie na nich definované.

V tretej kapitole nasleduje vlastný návrh riešenia. Podrobne sú popísané všetky komponenty pre modelovanie konečných automatov. Sú spísané vlastnosti každého bloku a vysvetlené významy jednotlivých parametrov. Nechýba ani podrobný návod, ako vytvoriť funkčný model automatu a ako takto vytvorený model používať.

Vo štvrtej kapitole je návrh dátovej reprezentácie. Je vysvetlený význam a práca s touto dátovou reprezentáciou.

Piata kapitola je venovaná ukážke toho, ako pomocou navrhnutého programového riešenia určiť či je automat minimálny alebo nie. Princíp algoritmu minimalizácie je vysvetlený vo všeobecnosti. Následne sú spočítané ekvivalentné stavy neminimálneho automatu ručne a potom pomocou implementovaného algoritmu.

Posledná kapitola sa venuje generovaniu kódu v jazyku C pre konečný stavový automat. Na reálnom príklade aplikácie pre riadenie striedavého indukčného motora je demonštrovaná funkčnosť celého riešenia. Stavový diagram hlavného automatu spolu s vnorenými automatmi je najskôr rozoberaný teoreticky, následne je tento diagram implementovaný v *Simulinku* spolu s ukážkou simulácie. Nakoniec je vysvetlený postup vygenerovania kódu. Pre overenie funkčnosti vygenerovaného kódu je vytvorená konzolová aplikácia, pomocou ktorej môže užívateľ simulovať príchody jednotlivých udalostí a pozorovať aktuálny stav.

1 VSTAVANÉ SYSTÉMY

V úvodnej kapitole si objasníme pojem vstavaný systém (Embedded system - ES).

1.1 Charakteristika

Vstavaný systém alebo tiež zabudovaný systém je vo svojej podstate systém, v ktorom je riadiaci počítač úplne zabudovaný do zariadenia, ktoré ovláda. Na rozdiel od osobných počítačov plní vstavaný systém obyčajne len jednu hlavnú úlohu. Podľa [2] definujeme vstavané systémy ako kombináciu hardwaru a softwaru, ktorých zmyslom je riadiť externý proces, zariadenie alebo systém.

Hardware predstavuje doska s procesorom a ostatná elektronika. Až 98%[2] všetkých vyrobených procesorov sa použije vo vstavaných systémoch, zvyšok v osobných počítačoch. Medzi hlavných výrobcov procesorov pre vstavané systémy patria Atmel, Freescale, Texas Instruments, Microchip atď. Okrem takpovediac klasických ES s mikroprocesorom existujú aj ďalšie platformy. *Embedded PC* sú založené na procesoroch primárne určených do osobných počítačov a využívajú ich prostriedky. Programovateľné hradlové polia (*FPGA – Field Programmable Gate Array*) zasa ponúkajú možnosť paralelného spracovania viacerých operácii, ktoré tak na rozdiel od bežných procesorov nemusia súperiť a zdroje. Hradlové polia preto ponúkajú väčšiu rýchlosť a flexibilitu. Medzi popredných výrobcov v tejto oblasti patria firmy Xilinx a Altera. Návrh celého vstavaného systému vrátane elektroniky na jednom čipe ponúkajú obvody *ASIC (Application Specific Integrated Circuit)*,teda integrované obvody vyrobené na mieru konkrétnej aplikácie.

Pri vývoji vstavaných systémov je nutné prihliadať na špecifiká danej aplikácie a podľa toho musia spĺňať isté základné požiadavky. Sú to napríklad požiadavky na prácu v reálnom čase (RTOS), požiadavky na bezpečnosť zariadenia alebo požiadavky na samostatnú prácu, bez vonkajšieho zásahu. Vstavané systémy musia často pracovať dlhé roky bez akejkoľvek údržby.

1.2 Oblasti použitia

Vstavané systémy sú súčasťou celej rady výrobkov a zariadení obklopujúcich náš každodenný život. Môžeme ich nájsť od najjednoduchších hračiek ako sú hracie narodeninové kartičky, až po zložité zariadenia ako napríklad priemyselné roboty, či vesmírne raketoplány. Medzi hlavné oblasti využitia vstavaných systémov patria:

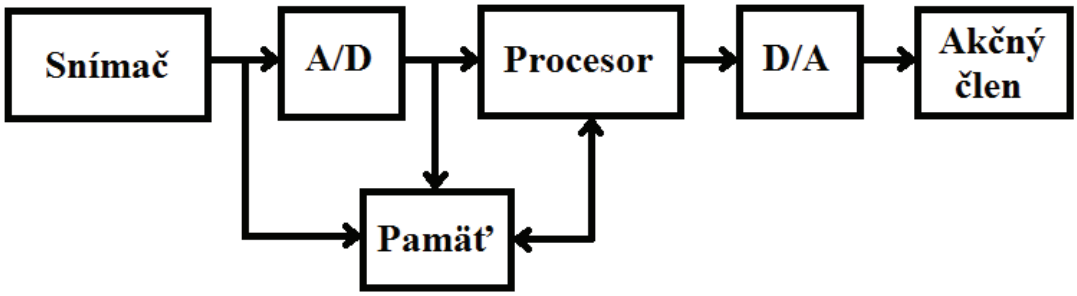
- Spotrebná elektronika
- Letecká elektronika
- Automobilová elektronika
- Lekárske prístroje

- Automatizácia budov
- Priemyselná automatizácia

1.3 Štruktúra

Na obr. 2.1 možno vidieť základnú štruktúru vstavaných systémov. Medzi základné komponenty patria:

- Procesor
- Snímače (teploty, vlhkosti)
- Prevodníky (D/A,A/D)
- Pamäť (interná, externá)
- Akčné členy (motor, ventil)
- Komunikačné periférie



Obr. 1.1 Štruktúra vstavaných systémov

Väčšina aplikácii vo vstavaných systémoch pracuje ako konečný stavový automat. V hlavnej funkcii sa na základe vstupov vyhodnocuje stav aplikácie. Následne sa vykonajú príslušne akcie definované pre daný stav, prípadne sa povolí prechod do iného stavu.

2 STAVOVÉ AUTOMATY

Táto kapitola sa venuje stavovým automatom a ich analýze. Na úvod je vysvetlené čo rozumieme pod pojmom systémy diskrétnych udalostí, rozdelenie a ciele ktoré sa pri ich riešení snažíme dosiahnuť. Ďalej sú objasnené pojmy ako jazyky a automaty, ich vzájomná súvislosť a operácie, ktoré sú definované na jazykoch a na automatoch. Informácie v tejto kapitole sú prevažne čerpané z [1].

2.1 Systémy diskrétnych udalostí

Jednou z možných definícii [4] ako definovať systém je, že je to množina stavov, spolu s množinou prechodov medzi týmito stavmi. Iná definícia, v kybernetike využívaná častejšie, vraví, že systém je množina prvkov, spolu s množinou väzieb medzi nimi navzájom a aj s okolím. Tieto prvky sú tvorené rôznymi fyzikálnymi entitami. Dynamiky systému určuje správanie sa jednotlivých prvkov a systému ako celku. Systémy teda poznáme statické a dynamické. Statické systémy sú z hľadiska analýzy pre nás nedôležité. Dynamické systémy možno ďalej rozdeliť z hľadiska:

- Linearity : lineárne, nelineárne
- Zmeny v čase: t-invariantné (v čase nemenné), t-variantné (meniace sa s časom)
- Času: so spojitým časom, s diskrétnym časom

Z hľadiska stavového priestoru sa systémy rozdeľujú na:

- **Spojité**
- **Diskrétne**

Stavový priestor je v prípade systémov so spojitým stavovým priestorom, ako názov napovedá, spojitý. Hlavným spôsobom popisu takýchto systémov je diferenciálna alebo diferenčná rovnica. Stavový priestor v prípade diskrétnych systémov je diskrétna veličina a mení sa diskrétne, teda skokom. Zvláštnym typom systémov sú systémy hybridné, ktoré sú kombináciou predošlých dvoch typov.

Systém diskrétnych udalostí (Discrete event systems – DES) je taký systém, ktorého stavový priestor je diskrétna množina a vývoj jeho stavu v čase je daný výhradne príchodmi asynchrónnych udalostí. Množina týchto udalostí E je diskrétna. Príchod udalosti spôsobí prechod medzi stavmi systému. Pod pojmom udalosť si možno predstaviť napríklad príchod zákazníka alebo dosiahnutie požadovanej výšky hladiny v nádrži. Udalosť však môže byť aj „nulová“ alebo „prázdna“.

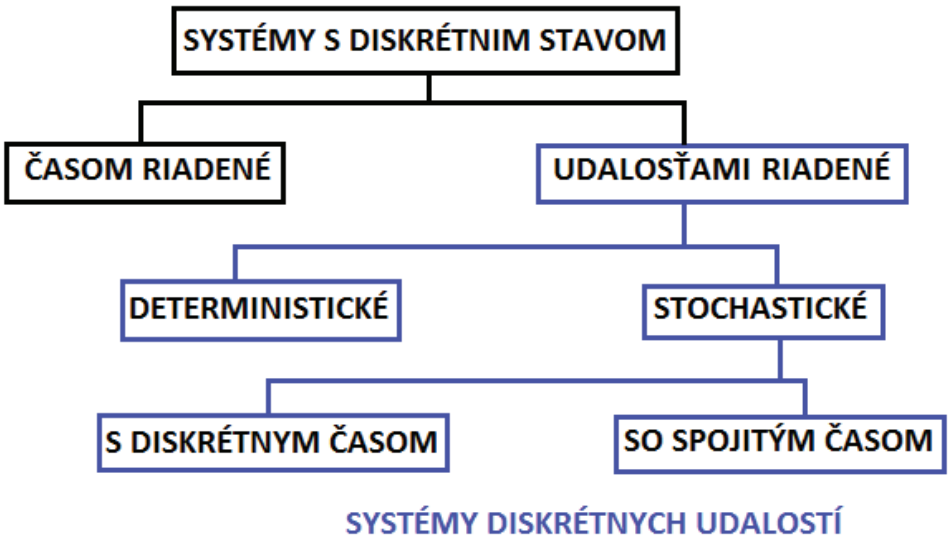
2.1.1 Rozdelenie DES

Pri ďalšej analýze stavových automatov sa zameriame na systémy s diskrétnym stavom, ktoré môžeme ďalej rozdeliť na:

- Časom riadené systémy
- Udalosťami riadené systémy

V prípade časom riadených systémov je pri každom tiku hodín, vybraná jedna udalosť $e \in E$, ktorá môže, ale nemusí vyvolať zmenu systému („nulová“ udalosť).

Ak sa jedná o systém riadený udalosťami, tak platí, že pre každú udalosť $e \in E$ je definovaný proces, ktorý určuje príchody udalosti e. Zmena stavu v tomto prípade je daná kombináciou týchto asynchrónnych súbežne bežiacich procesov.



Obr. 2.1 Klasifikácia systémov diskrétnych udalostí (DES)

Na obr. 2.1 je modrou farbou vyznačená oblasť systémov diskrétnych udalostí. Pre deterministický systém platí, že jeho správanie je jednoznačne dané stavom a postupnosťou udalostí. Pri stochastickom systéme hrá určitú rolu náhoda a jeho stav preto nie je jednoznačne určený.

2.1.2 Ciele pri riešení DES

Dôvodov, prečo sa snažíme systémy diskrétnych udalostí bližšie analyzovať a určiť ich vlastnosti je niekoľko. Ciele, ktoré pri tom sledujeme sa dajú zhrnúť do nasledujúcich bodov:

1. **Modelovanie a analýza**.
   Je to prvý krok k tomu pochopiť, ako reálny systém vlastne pracuje. Snažíme sa vytvoriť model fyzikálneho systému, tak aby sme mohli sledovať jeho správanie pri rôznych podmienkach, odlišných hodnotách parametrov alebo vstupných funkciách.
2. **Návrh a syntéza**.

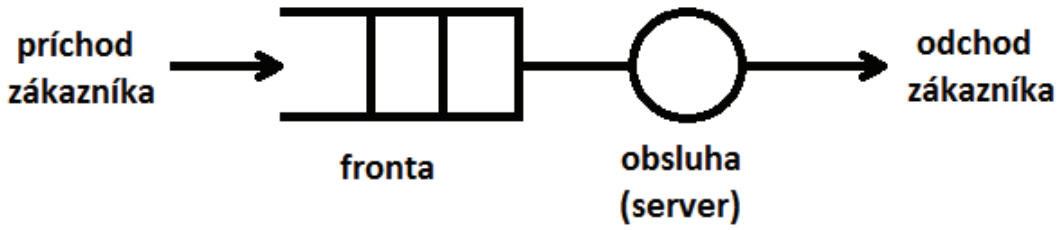
Snažíme sa odpovedať si na otázku: „Ako postaviť systém, ktorý sa bude správať podľa našich požiadaviek?“.

3. **Riadenie**.
   Pokúšame sa vybrať vstupnú funkciu, ktorá zaistí správanie systému podľa našich požiadaviek a pre rôzne operačné podmienky. Z tohto dôvodu potrebujeme model systému, na ktorom by bolo možné testovať a hodnotiť rôzne vstupné funkcie.
4. **Hodnotenie výkonu**.
   Po tom, ako sme systém namodelovali a našli vhodné riadenie sa naskytuje otázka: „Aký výkonný je systém v skutočnosti?“. Merania výkonu systému sú často subjektívne alebo závislé na konkrétnej aplikácii. Je možné, že niekoľko rôznych typov riadenia dokáže splniť naše základné požiadavky.
5. **Optimalizácia**.
   Vzhľadom k tomu že systém je navrhnutý a riadený aby dosiahol určité požadované chovanie, naskytuje sa otázka: „Ako riadiť systém tak, aby dosiahol najlepšie možné chovanie?“. V tomto prípade je nevyhnutné definovať kritérium optimalizácie, podľa ktorého sa bude určovať.

2.1.3 Príklad DES

Vo väčšine nami navrhnutých a vytvorených systémov platí, že pokiaľ chceme v danom okamžiku využívať zdroje systému, ktoré sú užívané niekým iným, musíme počkať kým sa tieto zdroje neuvoľnia. Príkladom takéhoto fungovania je napríklad aj vykonávanie úloh v počítači. V tomto prípade je zdrojom CPU. Ak nastane situácia, že zdroj (CPU) nie je voľný tak úloha, ktorá oň žiada je presunutá do fronty (buffer). Ako náhle sa zdroj uvoľní, prístup k nemu dostane tá úloha, ktorá je vo fronte ďalšia v poradí. Poradie vo fronte určuje čas príchodu úlohy a poradie v akom sa úloha bude vykonávať zasa určuje spôsob implementácie fronty. Základné elementy tvoriace systém tohto typu sú nasledovné:

- **Entity**. V našom príklade ich predstavujú úlohy. Často sú označované ako zákazníci. Entity čakajú na uvoľnenie zdroja.
- **Zdroje**. Poskytujú určité služby entitám (zákazníkom), preto bývajú označované ako obsluha (server). Bývajú obmedzené, a preto na ne entity čakajú.
- **Fronty**. Predstavujú priestor, v ktorom entity čakajú. Správanie sa fronty možno popísať rôznymi pravidlami (FIFO, LIFO,...).



Obr. 2.2 Príklad jednoduchého DES

Na obrázku 2.2 je schematicky zakreslená funkcia jednoduchého príkladu systému diskrétnych udalostí. V reálnom svete si ho môžeme predstaviť napríklad ako situáciu pri bankomate. Frontu predstavuje rad ľudí, ktorí si chcú vybrať peniaze z bankomatu a obsluha je samotný bankomat. Množina udalostí je dvojprvková $E = \{\, príchod\ zákazníka, odchod\ zákaznika \,\}$. Stavový priestor je množina prirodzených čísel $X = \{0,1,2,3, \ldots\}$. Pod stavom systému v čase t, si môžeme predstaviť dĺžku fronty (počet ľudí) v čase t, pričom niekedy je započítavaný aj práve obsluhovaný zákazník (záleží na použitej konvencii).

2.2 Jazyky

Na modelovanie správania sa systémov diskrétnych udalostí (DES), nemožno tak ako v prípade dynamických systémov so spojitým stavom použiť diferenciálne alebo diferenčné rovnice. Preto v prvom rade musíme nájsť adekvátny spôsob modelovania DES, ktorý jednak popíše správanie sa týchto systémov, ale súčasne nám poskytne priestor pre analytické techniky zhodnotenia kvality systémov. Práve pre tieto účely slúžia jazyky, ktoré tak predstavujú základnú metódu modelovania systémov diskrétnych udalostí.

2.2.1 Jazyková reprezentácia DES

K tomu, aby sme pomocou jazykov boli schopný popísať systémy diskrétnych udalostí zavádzame nasledujúce vlastnosti, ktorých platnosť sa v ďalšom texte automaticky predpokladá:

- Množina udalostí E je tvorená pomocou jednotlivých písmen abecedy.
- Množina udalostí E je konečná množina.
- Postupnosť prichádzajúcich udalostí tvorí slovo(reťazec) s.
- Prázdny reťazec ε je taký, ktorý neobsahuje žiadne slovo.
- Dĺžkou slova (reťazca) $|s|$ rozumieme počet udalostí v slove (reťazci), pričom platí $|\varepsilon| = 0$.

- Množina E^* obsahuje všetky reťazce konečnej dĺžky zostavených z prvkov množiny E, vrátane prázdneho reťazca ε.

Potom **jazyk** definovaný na množine E je množina slov konečnej dĺžky zostavených z prvkov množiny E.

Jazyk modelujúci DES je analógiou jazyka ľudského tak ako ho poznáme. Aj tu sú slová zložené z písmen abecedy, pričom niektoré kombinácie (postupnosti) písmen nám dávajú význam (slová), ktorý si vieme predstaviť, analogicky v prípade systémov niektoré postupnosti udalostí majú pre daný systém význam, napríklad v podobe výstupu zo systému.

2.2.2 Operácie s jazykmi

Jazyk vlastne predstavuje množinu slov, a preto operácie, ktoré môžeme aplikovať na jazyky sú dvojakého druhu.

1. Štandardné množinové operácie
    - Zjednotenie
    - Prienik
    - Rozdiel
    - Doplnok vzhľadom k E^*
2. Špecifické operácie
    - Zreťazenie (concatenation)
    - Prefix-uzáver (prefix-closure)
    - Kleenov-uzáver (Kleen-closure)
    - Post-jazyk (post-language)
    - Projekcia (natural projection)

Presné definície jednotlivých operácii spolu s príkladmi možno nájsť v [1].

2.3 Automaty

Na reprezentáciu jazyka slúžia automaty. Automat je zariadenie schopné na základe jasne definovaných pravidiel reprezentovať jazyk. Hovoríme, že automat generuje jazyk. V podstate si môžeme otázky na ktoré sa snažíme odpovedať položiť nasledovne: „Sme schopný postaviť systém, ktorý bude generovať daný jazyk?“ alebo „Aký jazyk daný systém generuje?“.

2.3.1 Definícia

V ďalšom texte upriamime pozornosť na deterministické automaty, ktorých správanie je vopred dané, a pri opakovanej postupnosti tých istých udalostí sa zachovajú vždy

rovnako. Toto tvrdenie ale neplatí pre automaty stochastické, ktoré pracujú s neurčitosťou (pravdepodobnosťou).

Deterministický automat označený G je šestica

$$G = (X, E, f, \Gamma, x_0, X_m) \quad (1)$$

kde:

X je množina stavov.

E je konečná množina udalostí definovaných v G.

$f : X \times E \rightarrow X$ je funkcia prechodu, označenie $f(x, e) = y$ znamená, že ak je automat v stave x a nastane udalosť e , potom prejde do stavu y.

$\Gamma : X \rightarrow 2^E$ je funkcia aktívnych udalostí. Označenie $\Gamma(x)$ je množina všetkých udalostí e, pre ktoré je funkcia $f(x, e)$ definovaná a nazýva sa množina aktívnych udalostí automatu G v stave x.

x_0 je počiatočný stav

$X_m \subseteq X$ je množina označených stavov

Nedeterministický automat označený G je šestica

$$G = (X, E \cup \{\varepsilon\}, f_{nd}, \Gamma, x_0, X_m) \quad (2)$$

kde:

X je množina stavov.

E je konečná množina udalostí definovaných v G.

$f_{nd} : X \times E \cup \{\varepsilon\} \rightarrow 2^X$ je funkcia prechodu $f(x, e) \subseteq X$

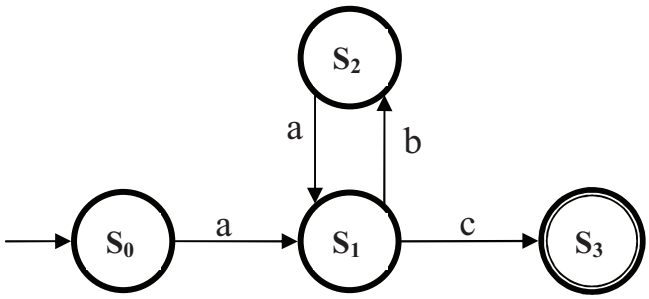
$\Gamma : X \rightarrow 2^E$ je funkcia aktívnych udalostí.

$x_0 \subseteq X$ je počiatočný stav

$X_m \subseteq X$ je množina označených stavov

2.3.2 Modelovanie stavových automatov

Existujú dva základné popisy konečných stavových automatov, popis pomocou **stavového diagramu** alebo **tabuľky prechodov**. Tabuľka prechodov obsahuje informácie o jednotlivých prechodoch v prípade, ak je automat v danom stave a na vstupe sa objaví udalosť. S tabuľkou prechodov sa možno najčastejšie stretnúť pri návrhu logických obvodov. Stavový diagram je grafickou realizáciou automatov a predstavuje najnázornejší popis. Jedná sa v podstate o orientovaný graf, znázorňujúci jednotlivé stavy pospájané hranami. Tieto hrany reprezentujú funkciu prechodu a znázorňujú do akých stavov a za akých podmienok sa môže konečný automat z počiatočného stavu dostať. Príklad automatu popísaného stavovým diagramom je na Obr.2.3.



Obr. 2.3 Príklad jednoduchého automatu

Pre takto popísaný automat platí:
Množina stavov je štvorprvková:
$X = \{S_0, S_1, S_2, S_3\}$
Množina udalostí je trojprvková :
$E = \{a, b, c\}$
Príklad funkcie prechodu pre stav S_1 a udalosť b
$f(S_1, b) = S_2$
Množina aktívnych udalostí v stave S_1:
$\Gamma(S_1) = \{b, c\}$
Počiatočný stav:
$x_0 = S_0$
Množina označených stavov:
$X_m = \{S_3\}$
Medzi slová, ktoré automat generuje patria napríklad: $ac, abac, ababac$ ...

2.3.3 Ekvivalencia a blokovanie automatov

Najskôr si ukážeme ako súvisí jazyk s automatom a definíciu generovaného a označeného jazyka, aby sme mohli zaviesť definície pre ekvivalenciu a blokovanie automatov.
Máme automat G definovaný šesticou:

$$G = (X, E, f, \Gamma, x_0, X_m) \tag{3}$$

Jazyk generovaný automatom G je množina:

$$\mathcal{L}(G) := \{s \in E^* | \, f(x_0, s) je \; definovaná\} \tag{4}$$

Jazyk označený automatom G je množina:

$$\mathcal{L}_m(G) := \{s \in \mathcal{L}(G) | \, f(x_0, s) \in X_m\} \tag{5}$$

Jazyková ekvivalencia automatov:
Dva automaty G_1a G_2 sú jazykovo ekvivalentné, pokiaľ platí

$$\mathcal{L}(G_1) = \mathcal{L}(G_2) \tag{6}$$

$$\mathcal{L}_m(G_1) = \mathcal{L}_m(G_2) \tag{7}$$

Znamená to, že automaty generujú rovnaký jazyk, a súčasne jazyk označený automatom G_1 je rovnaký ako automatom G_2.

Blokovanie automatu:

Automat sa môže zablokovať dvoma rôznymi spôsobmi:

1. *Deadlock*. Pri deadlocku automat dosiahne stav x, ktorý nie je označený a súčasne nie sú v tomto stave definované žiadne prechody, množina aktívnych udalostí je prázdna. Matematicky zapísané:
   $\Gamma(x) = \emptyset \wedge x \notin X_m$
2. *Livelock*. Automat sa navonok tvári že je živý, pretože môže prechádzať medzi stavmi, nemôže však opustiť dosiahnutú množinu stavov.

Súhrnne môžeme definovať:

Automat G je **blokujúci**, ak platí:

$$\overline{\mathcal{L}_m(G)} \subset \mathcal{L}(G) \tag{8}$$

Automat G je **neblokujúci**, ak platí

$$\overline{\mathcal{L}_m(G)} = \mathcal{L}(G) \tag{9}$$

Kde $\overline{\mathcal{L}_m(G)}$ je operácia prefix-uzáver označeného jazyka.

2.3.4 Operácie s automatmi

K tomu, aby sme dokázali analyzovať systémy diskrétnych udalostí modelované pomocou automatov zavádzame sadu operácii nad automatom, ktoré modifikujú graf prechodov stavu. Taktiež potrebujeme operácie, ktoré nám dovoľujú kombinovať alebo skladať spolu dva alebo viac automatov. Potom môžeme modely rozsiahlych systémov vytvoriť na základe modelov jednotlivých komponentov systému.

- **Unárne operácie**
  - Dosiahnuteľná časť (Accessible part)
  - Spätne dosiahnuteľná časť (Coaccessible part)
  - Orezanie
  - Projekcia, inverzná projekcia
  - Doplnok
- **Operácie skladania automatov**
  - Súčin
  - Paralelná kombinácia
- **Úprava stavového priestoru**
- **Konštrukcia pozorovateľa stavu**
- **Ekvivalencia automatov**

Presné definície a príklady všetkých operácii je možné nájsť v [1] na strane 76. Algoritmus pre minimalizáciu stavového priestoru bude pre lepšiu názornosť vysvetlený až v kapitole 5, kde bude naň spočítaný príklad. V tejto časti si uvedieme definíciu pre

dosiahnuteľnú časť, spätne dosiahnuteľnú časť a orezanie automatu ktoré budeme potrebovať práve pri minimalizácii.

Dosiahnuteľnú časť predstavuje tá časť automatu G, ktorá je z počiatočného stavu x_0 dosiahnuteľná niektorým reťazcom z jazyka generovaného automatom $\mathcal{L}(G)$. Všetky stavy spolu s príslušnými prechodmi, ktoré nie sú dosiahnuteľné odstránime a to tak, že vykonáme operáciu

$$Ac(G) = \left(X_{ac}, E, f_{ac}, \Gamma, x_0, X_{ac_m}\right) \tag{10}$$

kde:

$X_{ac} = \{x \in X | (\exists s \in E^*)[f(x_0, s) = x]\}$

$X_{ac_m} = X_m \cap X_{ac}$

$f_{ac} = f|X_{ac} \times E \to X_{ac}$

Táto operácia nemá žiaden vplyv na jazyk generovaný ani označený automatom. Jej výsledkom je dosiahnuteľný automat, preto automat G je dosiahnuteľný ak platí:

$$G = Ac(G) \tag{11}$$

Spätne dosiahnuteľnú časť automatu predstavujú tie stavy, od ktorých existuje v grafe prechodov trasa po niektorý z označených stavov. Operácia $CoAc(G)$ je definovaná tak, aby odstránila z automatu všetky stavy, ktoré nie sú spätne dosiahnuteľné.

$$CoAc(G) = \left(X_{coac}, E, f_{coac}, \Gamma, x_{0_{coac}}, X_m\right) \tag{12}$$

kde:

$X_{coac} = \{x \in X | (\exists s \in E^*)[f(x_0, s) \in x]\}$

$x_{0_{coac}} = x_0$ ak $x_{0_{coac}} \in X_{coac}$ inak nedefinovaný

$f_{coac} = f|X_{ac} \times E \to X_{coac}$

Opäť platí, že automat je spätne dosiahnuteľný pokiaľ:

$$G = CoAc(G) \tag{13}$$

Operácia **orezanie automatu** $Trim(G)$ je kombináciou dosiahnuteľnej a spätne dosiahnuteľnej časti automatu. Platí:

$$Trim(G) = CoAc\big(Ac(G)\big) = Ac\big(CoAc(G)\big)$$

3 STAVOVÝ AUTOMAT V MATLABE

Konečný stavový automat, predstavuje stavový automat s konečným počtom stavov. V nasledujúcom texte sa bude pod pojmom stavový automat chápať vždy automat s konečným počtom stavov a nebude sa táto vlastnosť ďalej zdôrazňovať.

Na trhu existuje dostupné riešenie na modelovanie stavových automatov. Je ním toolbox *Stateflow* od spoločnosti MathWorks. Tento program je plne integrovaný do výpočtového systému Matlab/Simulink. Primárne je *Stateflow* určený na modelovanie a následné simulovanie systémov diskrétnych udalostí, teda systémov reagujúcich na určité udalosti, prípadne sekvenciu udalostí.

Úlohou tejto práce bolo nájsť spôsob ako umožniť modelovanie konečných stavových automatov bez využitia toolboxu *Stateflow*, ale s použitím základných programových prostriedkov ktoré ponúka *Matlab* a *Simulink*.

Pre modelovanie stavových automatov v *Simulinku* bol vytvorený programový prostriedok pozostávajúci z piatich samostatných blokov:

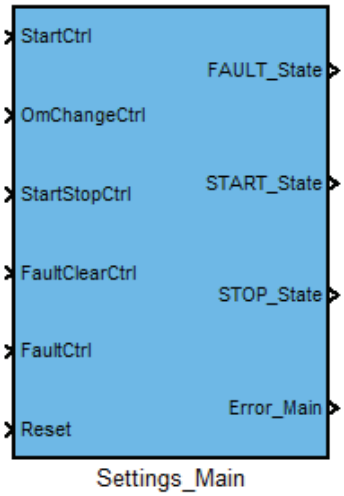
- BLOK NASTAVENIA AUTOMATU
- BLOK STAVU
- BLOK CHYBOVÉHO STAVU
- BLOK VNORENÉHO AUTOMATU
- BLOK FUNKCIE

Aby výsledná schéma bola čo najprehľadnejšia všetky bloky sú farebne rozlíšené. Pre nastavenie celého automatu, definovanie sady udalostí a ako vstup pre tieto udalostí slúži **blok nastavenia automatu** (modrý blok). Tento blok na výstupoch dáva informáciu o aktuálnom stave automatu. Ďalším blokom je **blok stavu** (zelený blok) predstavujúci jeden konkrétny stav automatu. Užívateľ má možnosť nadefinovať akcie, ktoré sa v tomto bloku vykonajú. Bloky stavov možno ľubovoľne kombinovať a vytvoriť tak základnú štruktúru automatu. V prípade ak je to treba, možno použiť aj **blok vnoreného automatu** (oranžový blok), ktorý môže mať ľubovoľný počet vstupov a výstupov. Užívateľ v ňom môže nadefinovať nový automat so samostatným blokom nastavenia. Ak je potrebné nadefinovať aj akcie prechodu medzi stavmi, je možné použiť **blok funkcie** (šedý blok), ktorý sa vkladá priamo na prechod a vykoná sa vždy keď k tomuto prechodu medzi stavmi dôjde. Súčasťou automatu v *Simulinku* je aj **blok chybového stavu** (červený blok). Slúži pre detekciu chyby. Chybou sa rozumie príchod udalosti v mieste, kde nie je očakávaná, výsledkom čoho je práve prechod do chybového stavu. V jednej simulačnej schéme môže byť ľubovoľný počet na sebe nezávislých stavových automatov. Všetky automaty však musia mať definovaný unikátny *Tag* (označenie).

Funkcie a vlastnosti jednotlivých blokov a význam každého parametru budú vysvetlené v ďalšom texte.

3.1 Blok nastavenia automatu

Tento blok slúži k základnému nastaveniu celého automatu. Zároveň doň vstupujú prichádzajúce udalosti a vystupujú z neho informácie o aktuálnom stave automatu. Možno priamo z neho vyčítať aké udalosti sú definované v automate, kam sa majú pripojiť a tiež v akých stavoch sa automat môže nachádzať. Informácia o aktuálnom stave môže byť zobrazená. Príklad bloku nastavenia automatu je na obr.3.1.



Obr. 3.1 Blok nastavania automatu.

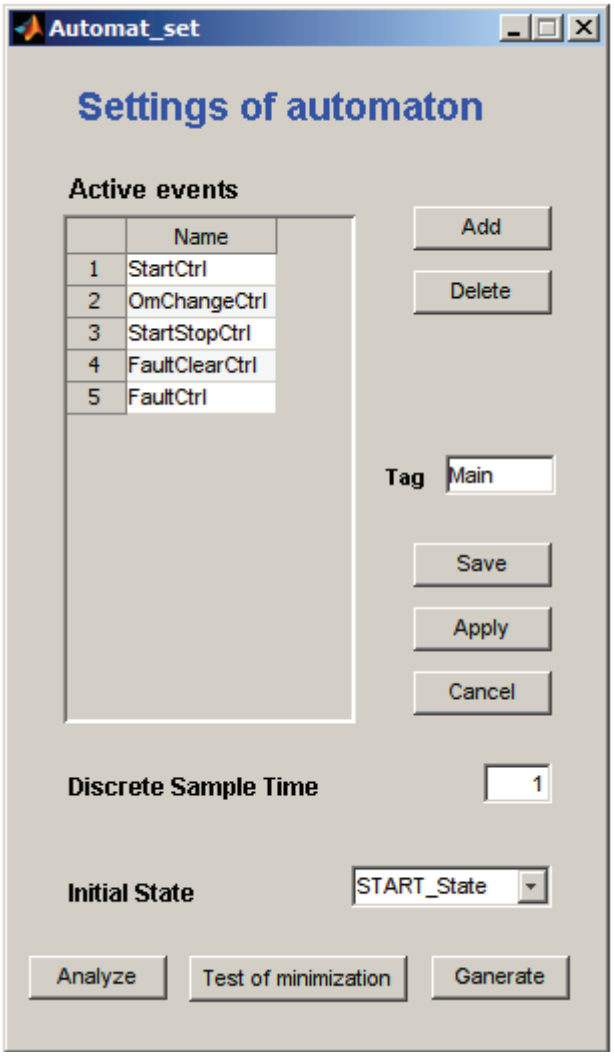
Užívateľské rozhranie spolu s vnútornou štruktúrou je navrhnuté tak, aby tento blok spĺňal nasledujúce **požiadavky**:

- Definuje množinu aktívnych udalostí automatu.
- Definuje *Tag* automatu.
- Definuje vzorkovaciu periódu automatu.
- Definuje počiatočný stav.
- Vytvára univerzálnu dátovú reprezentáciu automatu.
- Dokáže určiť či je automat minimálny, a ak nie, tak určí ekvivalentné stavy.
- Dokáže generovať kód v jazyku C, ktorý reprezentuje správanie sa namodelovaného automatu v *Simulinku*.

Popri vstupoch pre každú udalosť si blok automaticky generuje aj vstup pre signál *Reset*, tento názov je preto rezervovaný a nemôže sa tak volať žiadna iná udalosť. Viac o význame tohto signálu bude v kapitole 3.3.

3.1.1 Užívateľské rozhranie

Na obr.3.2 je príklad vyplneného bloku nastavenia automatu s definovanou sadou udalostí, identifikátorom (Tag) *Main*, vzorkovacou periódou jedna sekunda a s počiatočným stavom *START_State*.



Obr. 3.2 Užívateľské rozhranie pre nastavenie automatu

Funkcie a význam jednotlivých prvkov nastavenia sú nasledovné:

Tlačidlá:

- **Add:** Pridá prázdne pole na koniec tabuľky, v ktorej sú definované udalosti.
- **Delete:** Vymaže posledné pole v tabuľke udalostí.
- **Save:** Uloží všetky nastavenia do masky bloku, vytvorí vnútornú štruktúru a zavrie okno.
- **Apply:** Uloží všetky nastavenia do masky bloku, vytvorí vnútornú štruktúru, ale okno ponechá otvorené.
- **Cancel:** Neuloží žiadne nastavenia, ponechá predchádzajúce a zavrie okno.
- **Analyze:** Analyzuje automat a vytvorí dátovú reprezentáciu v podobe textového súboru s názvom „*Automat_Tag.txt*“, kde namiesto „*Tag*“ je definované označenie automatu. Tento textový súbor obsahuje základné informácie o automate. Viac o dátovej reprezentácii sa nachádza v kapitole 4.

- **Test of minimization:** Definovaný algoritmus minimalizácie určí, či je automat minimálny, ak nie je tak určí ekvivalentné stavy. Výsledok vypíše do príkazového okna v prostredí *Matlab*. Viac o algoritme ako aj príklad neminimálneho automatu je v kapitole 5.
- **Generate:** Vygeneruje kód v jazyku C, ktorý je možno priamo použiť bez nutnosti ďalších úprav. Kód je odladený v programe *Microsoft Visual Studio 2008*.

Tabuľka:

- **Active events:** V tejto tabuľke sa definujú aktívne udalosti automatu, je možnosť vymazávať udalosti a vytvárať nové. Užívateľ má tiež možnosť premenovať už existujúce a použité udalosti a tie sa po uložení nových nastavení automaticky premenujú vo všetkých stavoch, v ktorých sú použité. Zároveň je však programovo ošetrené, aby nebolo možné vymazať už použité udalosti.

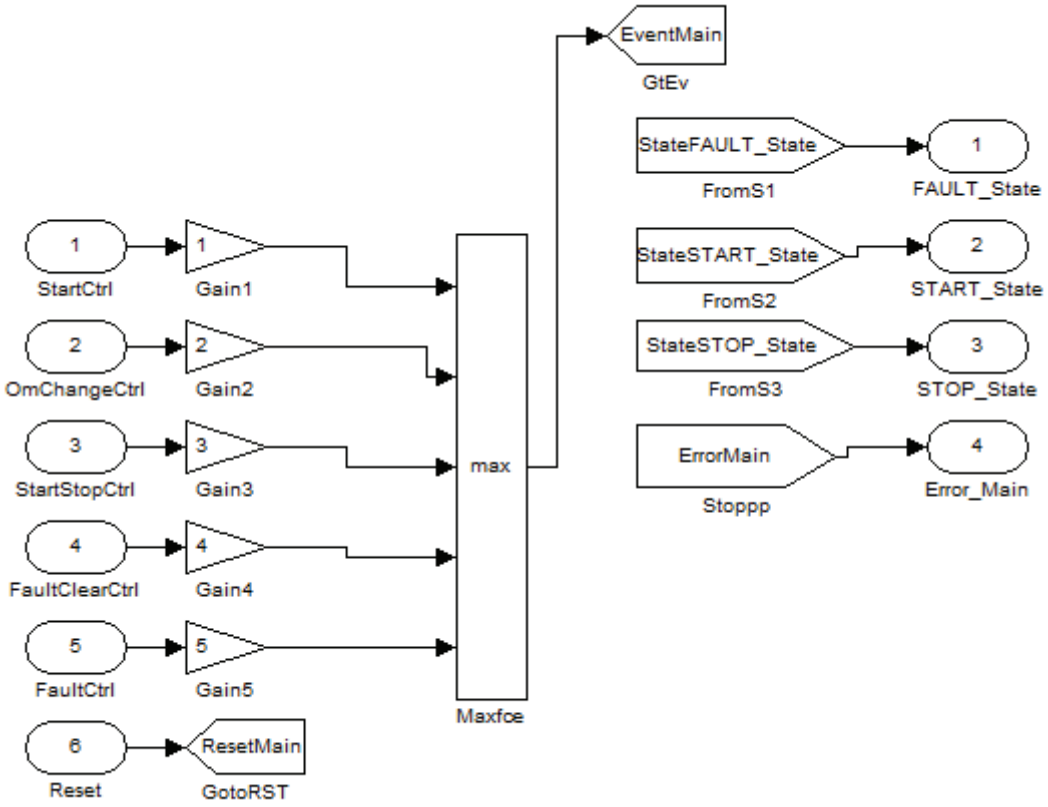
Textové polia:

- **Tag:** Nastavenie *Tagu* pre automat. Tento názov musí byť unikátny, čo je aj programovo zabezpečené. Slúži ako identifikátor daného automatu. Od jeho názvu sú odvodené aj názvy signálov pre komunikáciu medzi blokmi tak, aby nedochádzalo ku kolíziám.
- **Discrete Sample Time:** Nastavenie času vzorkovania, v ktorom všetky stavy daného automatu pracujú a vyhodnocujú signál. Je dôležité, aby prichádzajúce udalosti boli vzorkované rovnako. Ak udalosť trvá dlhšie ako je vzorkovacia perióda automatu, potom je automatom vyhodnotená ako viacnásobný opakovaný príchod jednej a tej istej udalosti po sebe.
- **Initial State:** Nastavenie počiatočného stavu automatu. Automat z tohto stavu vychádza na začiatku. Do počiatočného stavu sa automat dostane aj pri príchode signálu *Reset*.

V prípade ak sa zmení názov niektorého stavu alebo ak je vytvorený nový stav automatu, prípadne je niektorý stav vymazaný, tak je potrebné znova otvoriť a uložiť blok nastavení. Dôvodom je, že vnútorná štruktúra sa vytvára na základe aktuálnych stavov patriacich k automatu, ktoré si tento blok pri ukladaní automaticky vyhľadá. Ak je však vnútorná štruktúra bloku už vytvorená, následne sa zmení názov alebo počet stavov a spustí sa simulácia, môže dôjsť ku chybovému hláseniu.

Užívateľské rozhranie (GUI) spolu so všetkými funkciami je nadefinované v súboroch *Automat_set.fig* a *Automat_set.m*, ktoré sú súčasťou priloženého CD.

3.1.2 Vnútorná štruktúra

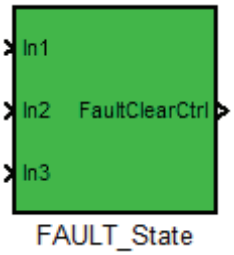


Obr. 3.3 Vnútorná štruktúra bloku nastavenia

Na obr.3.3 je vnútorná štruktúra bloku nastavenia. Logika celého automatu funguje tak, že každá udalosť predstavuje číslo od 1 do *n* (*n* je počet udalostí), *Reset* predstavuje samostatný signál. S týmito hodnotami ďalej pracujú všetky stavy. Funkcia *max* realizuje to, aby sa do stavov v jednom okamžiku poslala len jedna udalosť, a nenastala neočakávaná situácia.

3.2 Blok stavu

Ide o samostatný blok, ktorý predstavuje jeden stav. V jednom automate ich môže byť ľubovoľný počet, musia mať však unikátne názvy. Pri pohľade na blok sa dá priamo určiť na aké udalosti stav reaguje a aký prechod pri nich nastane.



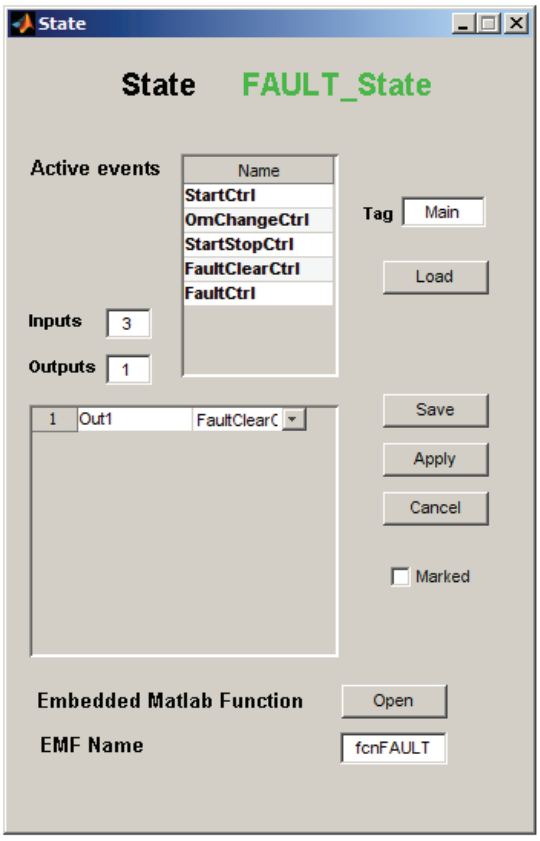
Obr. 3.4 Blok stavu

Požiadavky kladené na blok stavu a jeho užívateľské rozhranie môžeme zhrnúť nasledovne:

- Definuje počet vstupov stavu. Všetky vstupy sú si rovnocenne a ktorýmkoľvek z nich sa možno do stavu dostať.
- Definuje počet výstupov stavu. Každý výstup reaguje na inú udalosť.
- Definuje pre každý výstup konkrétnu udalosť, pri príchode ktorej nastane prechod na danom výstupe.
- Užívateľ má možnosť definovať akcie (funkcie), ktoré sa vykonajú ak sa automat do stavu dostane.
- Pre potreby algoritmu minimalizácie je možnosť určiť či je stav označený, alebo nie.

3.2.1 Užívateľské rozhranie

Príklad užívateľského rozhrania pre stav *FAULT_State* je na obr.3.5. Tento stav je naviazaný na automat s označením (Tag) *Main.* Má tri vstupy, jeden výstup pričom nie je označený a funkcia, ktorá sa pri príchode do tohto stavu vykoná má označenie *fcnFAULT.*



Obr. 3.5 Užívateľské rozhranie pre blok stavu *FAULT_State*

Funkcie a význam jednotlivých prvkov nastavenia sú nasledovné:
Tlačidlá:

- **Load**: Načítanie definovaných udalostí automatu do tabuľky **Active event**. V prípade ak by sa stav naviazal na iný automat zmenou *Tagu*, je možné týmto tlačidlom načítať do tabuľky aktívnych udalostí nové udalosti definované pre nový automat.
- **Save**: Uloží všetky nastavenia do masky bloku, vytvorí vnútornú štruktúru a zavrie okno.
- **Apply**: Uloží všetky nastavenia do masky bloku, vytvorí vnútornú štruktúru ale okno ponechá otvorené.
- **Cancel**: Neuloží žiadne nastavenia, ponechá predchádzajúce a zavrie okno.
- **Open**: Otvorí *Embedded Matlab Function* pre tento stav, do ktorej má možnosť užívateľ dopísať kód programu, ktorý sa vykoná keď sa automat do tohto stavu dostane.

Tabuľky:

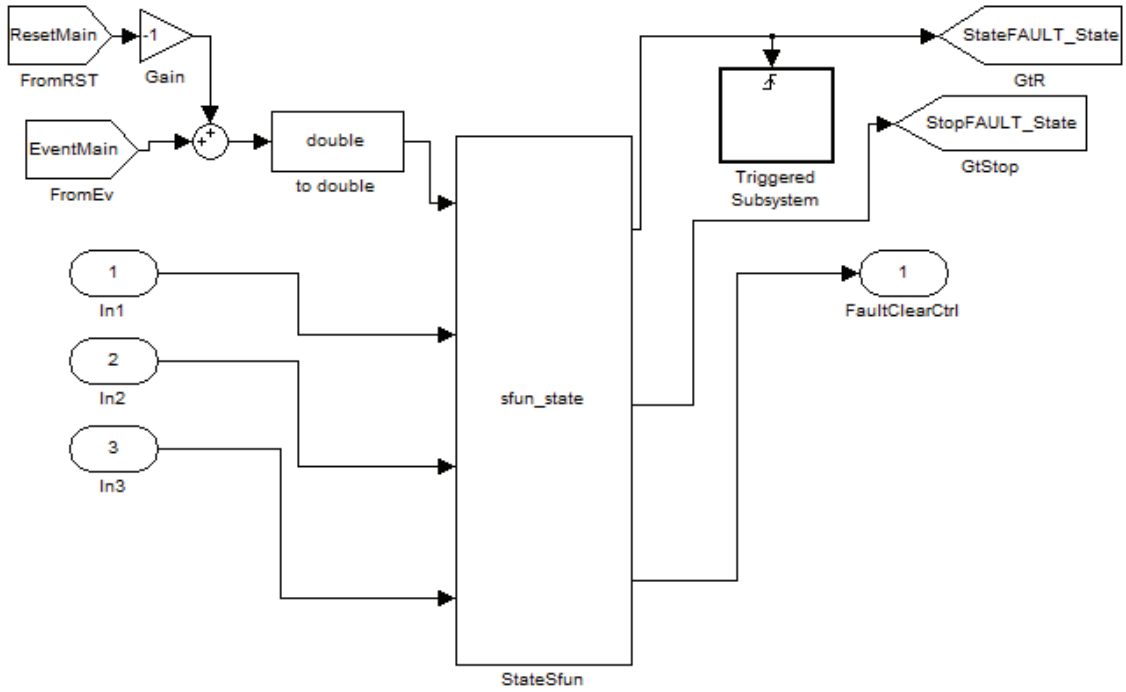
- **Active events:** Tabuľka, ktorá obsahuje všetky udalosti definované pre konkrétny automat. V prípade ak je z nejakého dôvodu prázdna, stačí kliknúť na *Load*. Ak sa ani potom nenaplní udalosťami, tak chyba bude najpravdepodobnejšie v zle zadanom *Tagu*.
- **Tabuľka výstupov:** V tejto tabuľke sa automaticky vytvorí toľko riadkov, koľko je definovaných výstupov stavu. Pre každý výstup si užívateľ môže vybrať udalosť. Programovo je ošetrené, aby nebolo možné jednu udalosť definovať pre viac výstupov.

Textové polia:

- **Tag:** Určuje ku ktorému automatu je stav naviazaný.
- **Inputs:** Nastavenie počtu vstupov stavu. Tento počet nie je ničím obmedzený.
- **Outputs:** Nastavenie počtu výstupov. Maximálny počet je obmedzený počtom aktívnych udalostí, pretože každý výstup musí reagovať na rôznu udalosť.
- **EMF Name:** Predstavuje názov pre ***E**mbedded **M**atlab **F**unction.* Pre správnosť simulácie automatu nemá tento názov žiaden vplyv. Slúži pri následnom spracovaní schémy automatu a pri algoritme minimalizácie.

Užívateľské rozhranie (GUI) spolu so všetkými funkciami je nadefinované v súboroch: *State.fig a State.m*, ktoré sú súčasťou priloženého CD.

3.2.2 Vnútorná štruktúra

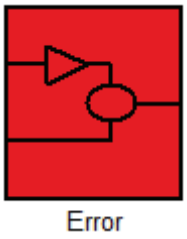


Obr. 3.6 Štruktúra stavu *FAULT_State*

Na obr. 3.6 je zobrazená štruktúra bloku stavu. Základom je *S-funkcia*, ktorá je bližšie popísaná v kapitole 3.4. Do tejto *S-funkcie* vstupujú prichádzajúce udalosti a vstupy stavu. Výstupom je informácia o tom, či je tento stav aktívny alebo nie. V prípade ak príde neočakávaná udalosť, tak vyvolá prechod do chybového stavu. Ak príde udalosť očakávaná tak vyvolá prechod na výstupe. V bloku *Triggered Subsystem* je schovaná *Embedded Matlab Function.* Táto funkcia sa vykoná len pri vzostupnej hrane signálu a zabezpečí tak jednorazové spustenie tejto funkcie.

3.3 Blok chybového stavu

Blok chybového stavu slúži k detekcii chyby v automate. Pod pojmom chyba sa rozumie príchod udalosti v čase a mieste, v ktorom táto udalosť nie je očakávaná. V takejto situácii automat samostatne prejde do chybového stavu (červené označenie) a prestáva reagovať na prichádzajúce udalosti. Do počiatočného stavu sa automat dostane príchodom signálu *Reset*.



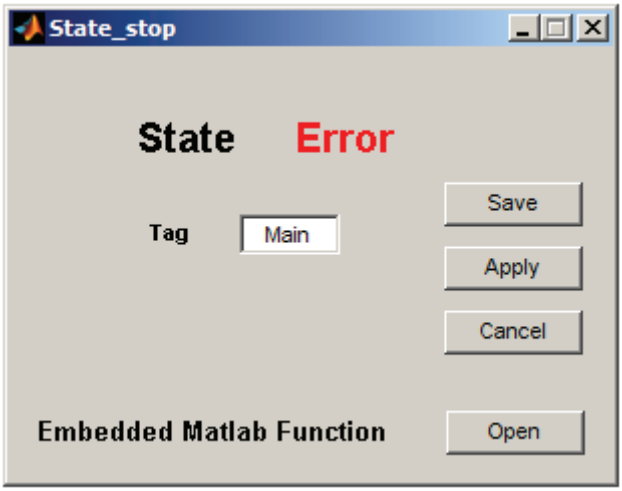
Obr. 3.7 Blok chybového stavu

Požiadavky na blok chybového stavu sú:

- Automat musí detekovať neočakávanú postupnosť udalostí.
- Užívateľ má možnosť definovať akcie, ktoré sa vykonajú ak takáto situácia nastane.
- Automat musí byť schopný sa dostať späť do počiatočného stavu.

3.3.1 Užívateľské rozhranie

Rozhranie pre tento blok je strohé. Pred spustením simulácie, po tom ako je celý automat poskladaný z jednotlivých blokov, je potrebné tak ako aj v prípade bloku nastavenia aj chybový blok otvoriť a znova uložiť. Dôvod je vytváranie vnútornej štruktúry, ktorá sa tvorí na základe aktuálneho počtu stavov priradených k automatu a ich mien.



Obr. 3.8 Užívateľské rozhranie pre chybový stav *Error*

Tlačidlá:

- **Save:** Uloží všetky nastavenia do masky bloku, vytvorí vnútornú štruktúru a zavrie okno.
- **Apply:** Uloží všetky nastavenia do masky bloku, vytvorí vnútornú štruktúru, ale okno ponechá otvorené.
- **Cancel:** Neuloží žiadne nastavenia, ponechá predchádzajúce a zavrie okno.

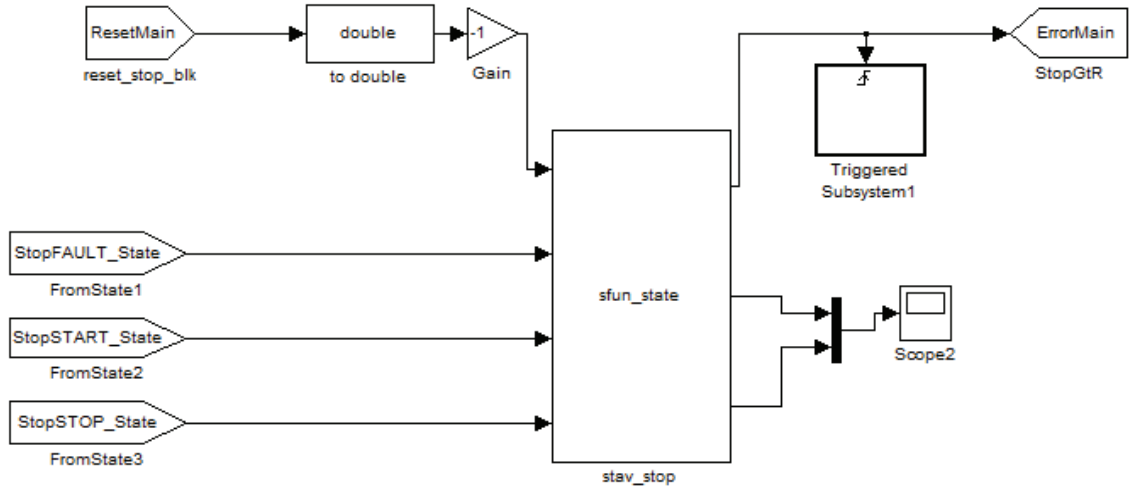
- **Open:** Otvorí *Embedded Matlab Function* pre chybový stav, do ktorej má možnosť užívateľ dopísať kód programu, ktorý sa vykoná keď sa automat dostane do tohto stavu.

Textové pole:

- **Tag:** Určuje ku ktorému automatu je chybový stav naviazaný.

Užívateľské rozhranie (GUI) spolu so všetkými funkciami je nadefinované v súboroch: *State_stop.fig a State_stop.m*, ktoré sú súčasťou priloženého CD.

3.3.2 Vnútorná štruktúra

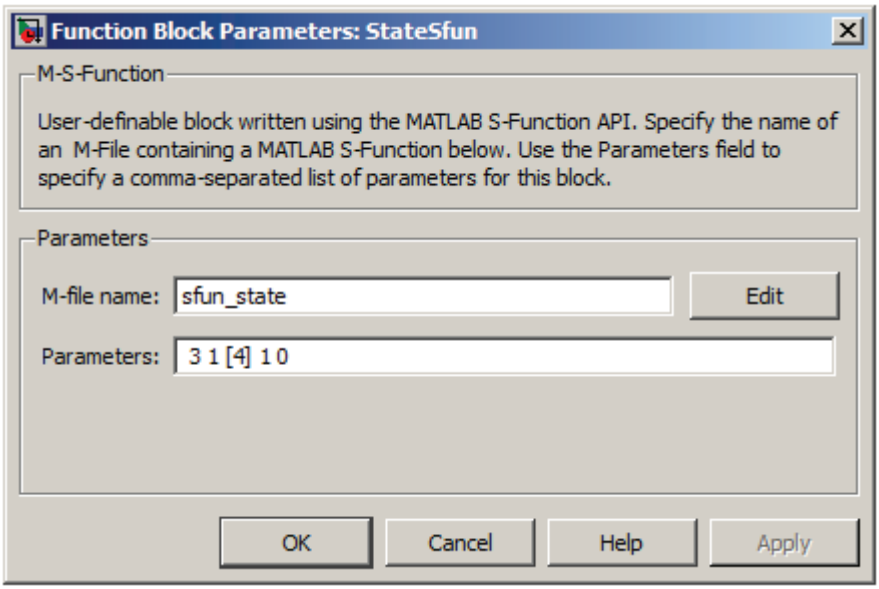


Obr. 3.9 Štruktúra chybového stavu *Error*

Vnútorná štruktúra je veľmi podobná s blokom stavu. Základom je S-funkcia do ktorej vstupuje signál *Reset* a informácie zo všetkých stavov o tom, či nenastala neočakávaná udalosť.

3.4 S-funkcia realizujúca stav

Logika realizujúca funkcie pre každý stav je naprogramovaná v S-funkcii s názvom *sfun_state.m*. Je typu Level-2 M-file S-function. Na obr. 3.10 je príklad dialógového okna pre stav FAULT_State.



Obr. 3.10 Dialogové okno S-funkcie.

3.4.1 Parametre S-funkcie

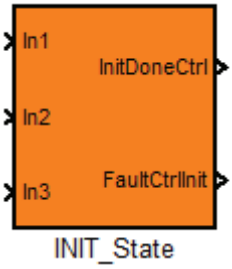
S-funkcia ma päť dialógových parametrov zapísaných v tomto poradí {**In Out [Events] SampleTime Init**}. Ich význam je nasledovný:

- **In:** Počet vstupov stavu.
- **Out:** Počet výstupov stavu.
- **Events:** Aktívne udalosti stavu v poradí, v akom sú naviazané na výstupy. V kapitole 3.1.2 je vysvetlená logika predávania udalostí medzi blokmi, ktoré tak majú význam celého kladného čísla (-1 je pre udalosť *Reset*, 0 sa nepoužíva). Poradie čísel (udalostí) pri parametre Events udáva zároveň poradie výstupov na ktorých budú udalosti naviazané.
- **SampleTime:** Vzorkovacia perióda pre S-funkciu.
- **Init:** Inicializácia stavu pre počiatočný stav je 1, pre ostatné 0.

Všetky parametre S-funkcie sa pri simulácii nastavia automaticky, a nie je potrebné ich nastavovať ručne, alebo prepisovať.

3.5 Blok vnoreného automatu

Tento blok plní funkciu vnoreného automatu. Navonok patrí k nadradenému automatu a pri vytváraní dátovej reprezentácie sa s ním počíta ako so stavom automatu. Vo vnútri však beží samostatný automat definovaný svojim *Tagom.* V prípade prechodu do tohto stavu sa predá riadenie vnorenému automat, ktorý následne môže predať nadradenému cez svoje výstupy. Na obr. 3.11 je príklad bloku, ktorý predstavuje vnorený automat s troma vstupmi a dvoma výstupmi. Ich počet je ľubovoľný a ich vytváranie je na užívateľovi.



Obr. 3.11 Blok vnoreného automatu.

3.5.1 Užívateľské rozhranie

Obr. 3.12 Užívateľské rozhranie pre vnorený automat *INIT_State*.

Význam a funkcie tlačidiel a textových polí užívateľského rozhrania sú nasledovné:
Tlačidlá:

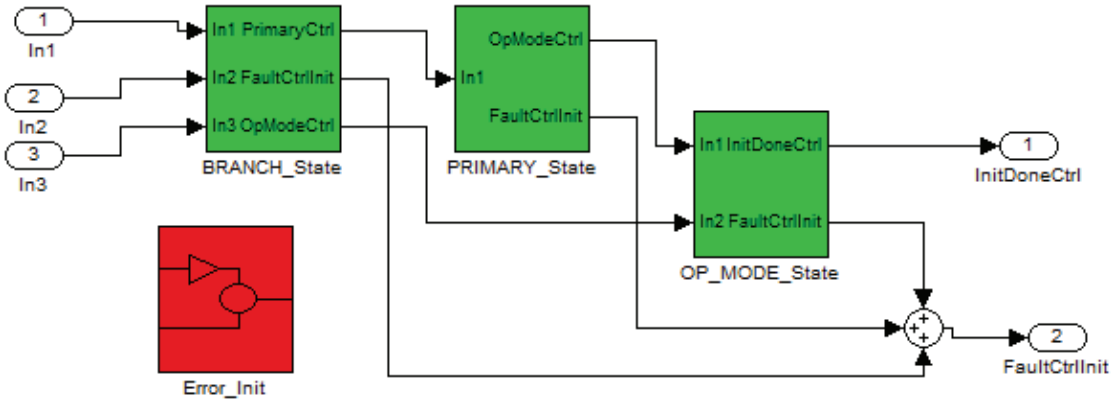
- **Save:** Uloží všetky nastavenia do masky bloku a zavrie okno.
- **Apply:** Uloží všetky nastavenia do masky bloku, ale okno ponechá otvorené.
- **Cancel:** Neuloží žiadne nastavenia, ponechá predchádzajúce a zavrie okno.
- **Open:** Otvorí sa modelovacia schéma do ktorej je možné vytvoriť vnorený automat. Zároveň sa uložia nastavenia do masky bloku a okno sa zavrie.

Textové pole:

- **Tag:** Určuje ku ktorému automatu je vnorený automat naviazaný.
- **Label:** Označenie vnoreného automatu. Pomocou tohto názvu sa dajú rozlíšiť dva rôzne vnorené automaty naviazané na jeden *Tag* a používajúce tú istú sadu udalostí.

Príklad vytvoreného vnoreného automatu je na obr. 3.13. Tento vnorený automat nemá počiatočný stav a dá sa do neho dostať len z vonku. Vnorený automat predstavuje plnohodnotný automat z hľadiska toho čo všetko musí obsahovať a toho ako sa chová.

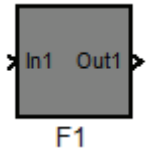
Dôležité **upozornenie** pri vytváraní štruktúry vnoreného automatu sa týka toho, ako naviazať výstupy zo stavov na výstupy z bloku (*Outport*). Je potrebné zachovať istú konvenciu. Ak sú výstupy z bloku (*Outport*) zoradené z hora dole podľa toho ako nasledujú (Out1,Out2 ...), tak potom ak sa výstup zo stavu napája na Out2, tak tento výstup musí byť ako druhý v poradí z daného bloku. **Poradie** výstupu zo stavu a **číslo výstupu** (*Outport*) z vnoreného automatu musí byť rovnaké. Toto obmedzenie nemá vôbec žiaden vplyv na funkciu a na simuláciu automatu. Aby však boli pri vytváraní kódu v jazyku C všetky udalosti a stavy vnoreného automatu naviazané správne na stavy nadradeného automatu je vhodné brať ohľad na toto pravidlo.



Obr. 3.13 Schéma vnoreného automatu.

Užívateľské rozhranie (GUI) spolu so všetkými funkciami je nadefinované v súboroch: *State_nested.fig a State_nested.m*, ktoré sú súčasťou priloženého CD.

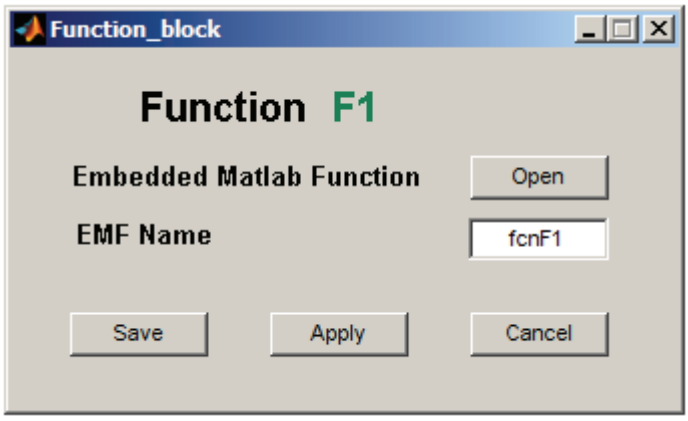
3.6 Blok funkcie



Obr. 3.14 Blok funkcie *F1*.

Blok funkcie má jeden vstup a jeden výstup. Vkladá sa na prechod medzi dvomi stavmi. Dajú sa v tomto bloku definovať akcie prechodu, ktoré sa vykonajú vždy, keď nastane prechod.

3.6.1 Užívateľské rozhranie



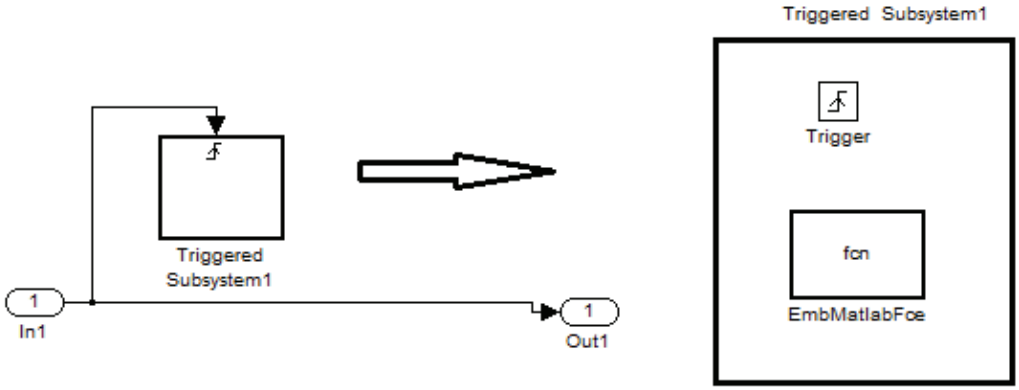
Obr. 3.15 Užívateľské rozhranie pre blok funkcie *F1*.

Význam a funkcie tlačidiel a textových polí užívateľského rozhrania sú nasledovné:
Tlačidlá:

- **Save:** Uloží všetky nastavenia do masky bloku a zavrie okno.
- **Apply:** Uloží všetky nastavenia do masky bloku, ale okno ponechá otvorené.
- **Cancel:** Neuloží žiadne nastavenia, ponechá predchádzajúce a zavrie okno.
- **Open:** Otvorí *Embedded Matlab Function* pre blok funkcie, do ktorej má možnosť užívateľ dopísať kód programu, ktorý sa vykoná keď sa automat do tohto stavu dostane.

Textové pole:

- **EMF Name:** Predstavuje názov pre ***Embedded Matlab Function.*** Pre správnosť simulácie automatu nemá tento názov žiaden vplyv. Slúži pri následnom spracovaní schémy automatu a pri algoritme minimalizácie.



Obr. 3.16 Štruktúra bloku funkcie.

Užívateľské rozhranie (GUI) spolu so všetkými funkciami je nadefinované v súboroch: *Function_block.fig* a *Function_block.m*, ktoré sú súčasťou priloženého CD.

3.7 Postup pri vytváraní automatu v Simulinku

Aby model automatu fungoval správne, je vhodné dodržať niekoľko základných pravidiel pri jeho vytváraní a dodržovať nasledujúci postup:

1. Vyplníme blok nastavenia. Zvolíme unikátny *Tag* pre tento automat, určíme vzorkovaciu periódu s ktorou bude automat pracovať a nadefinujeme množinu udalostí.
2. Vytvoríme si v schéme toľko stavov a vnorených automatov, koľko ich potrebujeme. Nesmieme zabudnúť ani na chybový stav.
3. Každý jeden stav v prvom rade naviažeme na blok nastavenia a načítame doň udalosti. Potom určíme počet vstupov, výstupov a naviažeme aktívne udalosti stavu na jeho výstupy. Určíme či stav je označený alebo nie. Nadefinujeme akcie, ktoré sa v stave musia vykonať. Postup opakujeme pre všetky stavy automatu.
4. Pospájame stavy medzi sebou ako potrebujeme.
5. Nakoniec naviažeme k automatu aj chybový stav, prípadne nadefinujeme akcie stavu a všetko uložíme.
6. Ak používame v schéme aj vnorené automaty, tak obdobným spôsobom nadefinujeme ich vnútornú štruktúru. Vnorený automat ma vlastný blok nastavení s vlastnou sadou udalostí a vlastný chybový stav. Na jeden blok nastavení môže byť naviazaných viacero vnorených automatov.
7. Všade tam, kde sa na prechode medzi stavmi musí vykonať nejaká akcia, vložíme blok funkcie a nadefinujeme akcie prechodu.
8. Nakoniec je potrebné definovať v bloku nastavení počiatočný stav automatu a všetko treba uložiť

Pre modelovanie automatu platia nasledujúce obmedzenia:

- Ak sa zmení názov niektorého stavu, alebo sa zmení počet stavov pridaním alebo vymazaním nejakého, je potrebné znova otvoriť a uložiť blok udalostí a blok chybového stavu. Je to preto, aby sa správne nakonfigurovali vnútorné štruktúry blokov.
- Jednotlivé udalosti vstupujúce do bloku udalostí treba upraviť tak, aby ich vzorkovacia perióda bola zhodná s tou, ktorú používa automat. Zároveň ich amplitúda musí byť rovná jednej.
- Zmena názvu niektorej udalosti v bloku udalostí sa automaticky prejaví aj vo všetkých stavoch a nie je možné vymazať udalosť ktorá je použitá v niektorom stave. Automaticky sa prejaví na všetkých stavoch aj zmena počiatočného stavu, či veľkosť vzorkovacej periódy.
- Nie je možné zmeniť *Tag* automatu v bloku nastavenia, ak sa na tento *Tag* odkazuje aspoň jeden stav.

4 DÁTOVÁ REPREZENTÁCIA

Ďalším úkolom bolo vytvoriť vhodnú dátovú reprezentáciu pre konečný stavový automat namodelovaný v *Simulinku* pomocou navrhnutých komponentov v kapitole 3. S touto dátovou reprezentáciou by malo byť možné ďalej pracovať a mala by zachytiť všetky vlastnosti a parametre namodelovaného automatu. V tejto kapitole bude vysvetlené čo sa pod pojmom dátová reprezentácia vlastne rozumie a ako sa pomocou nej dá spätne poskladať schéma automatu.

Najvhodnejší spôsob ako uchovať informácie o automate sa ukázal v podobe textového súboru s pevnou štruktúrou textu, v ktorej by sa dali jednoducho nájsť všetky vlastnosti automatu. Pri vytváraní tejto štruktúry bolo dôležité si určiť parametre automatu, ktoré sú podstatné a treba ich uchovať a ktoré nie. Medzi podstatné údaje samozrejme patria informácie o všetkých stavoch, aktívnych udalostiach a ich prechodoch.

Výsledná štruktúra je písaná čo možno najviac zrozumiteľne a prehľadne. Všetky informácie sú uložené v takej podobe, že najskôr je riadok, ktorý informuje o tom, čo bude za ním nasledovať. Táto vlastnosť sa dá s výhodou využiť. Stačí nájsť príslušný reťazec textu, napríklad `TAG:`,zistiť na ktorom riadku sa nachádza a na nasledujúcom už bude priamo *Tag* automatu, napríklad: `Main`. Textový súbor je rozdelený do troch častí. V prvej sú základné nastavenia automatu (SETTINGS-OF-AUTOMATON), v druhej sú informácie o stavoch a ich prechodoch (STATES) a v tretej sú informácie o vnorených automatoch (NESTED-AUTOMATA). Význam jednotlivých riadkov vo vygenerovanom textovom súbore je:

SETTINGS-OF-AUTOMATON - Základné nastavenie automatu

`TAG:`	*Tag* automatu.
`Number_of_states:`	Počet základných stavov automatu. Nepočíta sa chybový stav.
`Regular_states:`	Informácie o každom stave v podobe: NÁZOV FUNKCIA OZNAČENIE („on“ ak je označený, inak „off“)
`Stop_state:`	Názov chybového stavu.
`Initial_state:`	Počiatočný stav automatu.
`Set_of_events:`	Sada definovaných udalostí automatu. Nie je tu vypísaná udalosť *Reset*.

STATES – Informácie o stavoch a ich prechodoch

Nasledujúca časť sa opakuje pre každý stav:	
`State:`	Informácie o stave v podobe: NÁZOV FUNKCIA OZNAČENIE („on“ ak je označený, inak „off“)
`Active_events_set:`	Aktívne udalosti stavu.
`Transition_function:`	Počet prechodových funkcií. Prechodové funkcie stavu.

Tvar prechodovej funkcie má tiež svoju štruktúru a skladá sa z niekoľkých reťazcov:

- Prvý reťazec je počiatočný stav prechodu.
- Nasleduje udalosť prechodu.
- Na prechode sa môžu nachádzať tri typy blokov a pre každý je iné kľúčové slovo :
  `F==` : je označenie pre blok funkcie.
  `N==` : je označenie pre blok vnoreného automatu
  `O==` : je označenie pre outport (väčšinou pre vnorené automaty)
- Za kľúčovým slovom nasleduje názov bloku. Ak je na prechode blok vnoreného automatu a má viac ako jeden výstup, tak je za jeho meno pridané číslo výstupu ktorým sa pokračuje v prechode (napríklad `-Out1`).
- Za názvom je identifikátor bloku, pre blok funkcie je tam *EMF Name* a pre vnorený automat *Label*.
- Posledný reťazec označuje cieľový stav prechodu. V prípade ak sa na prechode objaví outport, napíše sa namiesto cieľového stavu `_none`.

Počet reťazcov v prechodovej funkcii je vždy 3*(n+1), kde n je počet blokov na prechode a môže vyzerať napríklad takto:

```
START_State StartCtrl F==  F1 fcnF1 N== INIT_State-Out1
Initialization STOP_State
```

Tento zápis znamená, že zo stavu `START_State` sa pri príchode udalosti `StartCtrl` dostane automat do stavu `STOP_State`. Na prechode sa pritom nachádza blok funkcie `F1` (*EMF Name* je `fcnF1`) a vnorený automat, z ktorého sa pokračuje v prechode cez jeho prvý výstup `INIT_State-Out1` (*Labe* je `Initialization`).

NESTED-AUTOMATA – informácie o vnorených automatoch

`Number_of_nested_automata:`	Počet vnorených automatov.
Nasledujúca časť sa opakuje pre každý vnorený automat:	
`Nested_automaton:`	Informácie o vnorenom automate v tvare: NÁZOV LABEL POČET VÝSTUPOV TAG VNORENÉHO AUTOMATU
`Transition_function:`	Počet prechodových funkcií. Prechodové funkcie v tvare počiatočný stav, konečný stav

5 MINIMALIZÁCIA AUTOMATU

Táto kapitola sa venuje problematike minimalizácie namodelovaného konečného automatu. Najskôr je vysvetlený algoritmus minimalizácie vo všeobecnosti, následne je ukážka na konkrétnom príklade. Na úvod by bolo vhodné zdôrazniť, že aplikovaný algoritmus nezlúči ekvivalentné stavy automatu. Dokáže však určiť či je daný automat minimálny a ak nie je, tak vypíše dvojice ekvivalentných stavov. Úprava automatu do minimálneho tvaru je ponechaná na užívateľovi.

5.1 Algoritmus minimalizácie

Algoritmus minimalizácie možno popísať v niekoľkých krokoch:

1. Upravíme automat tak, aby generoval jazyk E^* pričom označený jazyk musí zostať nezmenený. Toto docielime pridaním neoznačeného stavu a doplnením prechodov pre každý stav a každú jeho neaktívnu udalosť do novo vytvoreného stavu.
2. Označíme všetky dvojice stavov (x, y), kde $x \in X_m \wedge y \notin X_m$. To znamená, že jeden je označený a druhý nie.
3. Pre každú dvojicu ktorá ešte nie je označená vykonáme:
    a. Ak stav $\big(f(x,e), f(y,e)\big)$ je označený pre niektorú udalosť $e \in E$, tak označíme aj dvojicu (x, y) a všetky dvojice (w, z), ktoré sú pripísané na zozname dvojice (x, y).
    b. Ak stav $\big(f(x,e), f(y,e)\big)$ nie je označený pre žiadnu udalosť $e \in E$, tak pokiaľ $f(x,e) \neq f(y,e)$ pridáme dvojicu (x, y) do zoznamu priradeného dvojici $\big(f(x,e), f(y,e)\big)$.
4. Dvojice stavov, ktoré po dokončení algoritmu ostanú neoznačené predstavujú ekvivalentné stavy a môžu byť zlúčené, pričom sa jazyk generovaný automatom nezmení.

5.2 Matlab Pointer Library

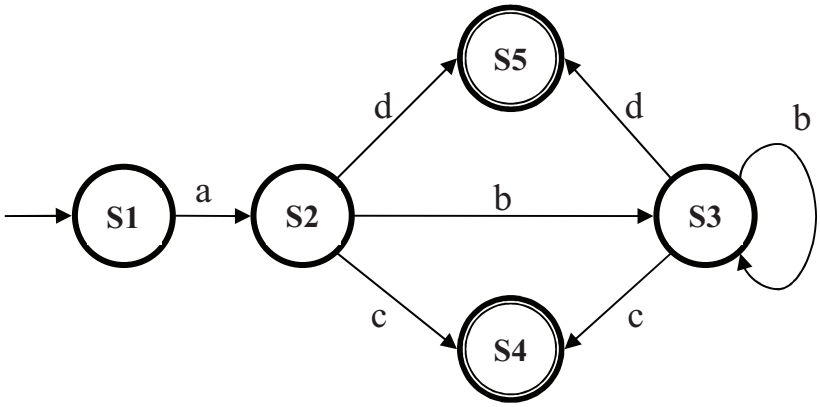
Matlab Pointer Library predstavuje knižnicu, ktorá rozširuje prostredie *Matlab* o možnosti používania ukazovateľov (pointers). Knižnica je voľne šíriteľná. Jedná sa o staršiu verziu z roku 2006 a je dostupná z [4]. Mne túto knižnicu odporučil vedúci diplomovej práce, ktorý už mal s ňou skúsenosti. Nachádza sa aj na priloženom CD k tejto práci.

Funkcie tejto knižnice využívam pri určovaní minimálnosti automatu, preto je potrebné implementovať túto knižnicu do prostredia *Matlab*, aby program fungoval správne. Inštalácia je jednoduchá a rýchla. Zvolil som túto možnosť, pretože použitie

ukazovateľov sa ukázalo ako najefektívnejší spôsob uchovania informácii o prechodoch.

5.3 Príklad neminimálneho automatu

Najskôr vykonáme minimalizáciu automatu na obr.5.1 podľa algoritmu popísaného v kapitole 5.1. Následne namodelujeme automat v Simulinku a použijeme implementovaný algoritmus k tomu, aby určil ekvivalentné stavy. Automat, ktorý budeme minimalizovať je na obr. 5.1.



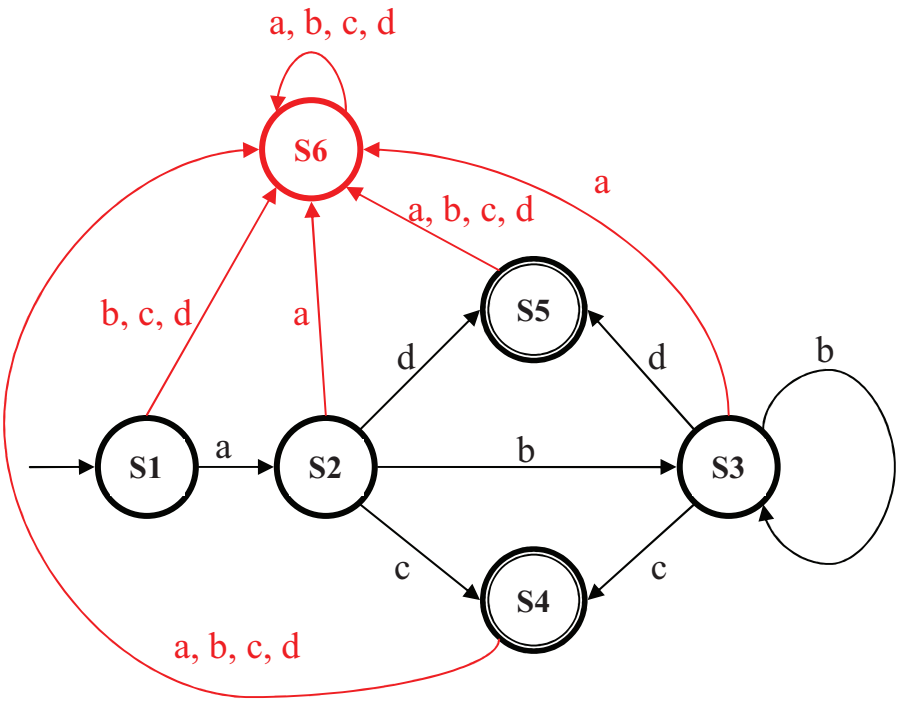
Obr. 5.1 Príklad neminimálneho automatu

Pre tento automat platí:
Množina stavov:
$X = \{S1, S2, S3, S4, S5\}$
Množina označených stavov:
$X_m = \{S4, S5\}$
Množina udalostí je :
$E = \{a, b, c, d\}$
Príklad funkcie prechodu pre stav $S1$ a udalosť A:
$f(S1, a) = S2$
Množina aktívnych udalostí v stave S_1:
$\Gamma(S_1) = \{a\}$
Počiatočný stav:
$x_0 = S1$

5.3.1 Minimalizácia automatu výpočtom

Postupovať pri minimalizácii budeme presne podľa algoritmu popísaného v kapitole 5.1.

Krok 1: Doplníme automat tak, aby generoval jazyk E^*. Výsledná schéma je na obr.5.2.



Obr. 5.2 Upravená schéma automatu.

Krok 2: Do pripravenej tabuľky dvojíc stavov označíme dvojice kde jeden stav je označený a druhý nie je.

	S1	S2	S3	S4	S5
S6				*	*
S5	*	*	*		
S4	*	*	*		
S3					
S2					

Tab. 5.1 Dvojice stavov po druhom kroku minimalizácie.

Krok 3: Postupne prechádzame všetky zatiaľ neoznačené dvojice stavov.

Pre dvojicu $(S1, S2)$ určíme:

$\big(\boldsymbol{f(S1, a)}, \boldsymbol{f(S2, a)}\big) = \boldsymbol{(S2, S6)}$, dvojica $(S2, S6)$ nie je zatiaľ označená, preto nemôžeme v tejto chvíli označiť ani dvojicu $(S1, S2)$, pridáme ju ale na zoznam dvojice $(S2, S6)$. Keď sa nám podarí dokázať že sa nejedná o ekvivalentnú dvojicu označíme v tabuľke aj všetky dvojice na jej zozname.

$\big(\boldsymbol{f(S1, b)}, \boldsymbol{f(S2, b)}\big) = \boldsymbol{(S6, S3)}$, opäť tá istá situácia ako v predošlom prípade. Pridáme teda dvojicu $(S1, S2)$ aj na zoznam dvojice $(S6, S3)$.

$\big(\boldsymbol{f(S1, c)}, \boldsymbol{f(S2, c)}\big) = \boldsymbol{(S6, S4)}$, vidíme v tabuľke že dvojica $(S6, S4)$ je už označená, preto môžeme označiť aj dvojicu $(S1, S2)$ a už sa ňou viac zaoberať nemusíme.

	S1	S2	S3	S4	S5
S6				*	*
S5	*	*	*		
S4	*	*	*		
S3					
S2	*				

Tab. 5.2 Tabuľka po preskúmaní dvojice (S1,S2).

Ako ďalšiu dvojicu preskúmame $(S1, S3)$:

$\big(\boldsymbol{f}(\boldsymbol{S1}, \boldsymbol{a}), \boldsymbol{f}(\boldsymbol{S3}, \boldsymbol{a})\big) = (\boldsymbol{S2}, \boldsymbol{S6})$, táto dvojica zatiaľ označená nie je, preto na jej zoznam pridáme dvojicu $(S1, S3)$.

$\big(\boldsymbol{f}(\boldsymbol{S1}, \boldsymbol{b}), \boldsymbol{f}(\boldsymbol{S3}, \boldsymbol{b})\big) = (\boldsymbol{S6}, \boldsymbol{S3})$, dvojicu $(S1, S3)$ zapíšeme aj na zoznam dvojice $(S6, S3)$, ktorá tiež ešte nie je označená.

$\big(\boldsymbol{f}(\boldsymbol{S1}, \boldsymbol{c}), \boldsymbol{f}(\boldsymbol{S3}, \boldsymbol{c})\big) = (\boldsymbol{S6}, \boldsymbol{S4})$, v tabuľke je dvojica $(S6, S4)$ označená, preto rovno označíme dvojicu $(S1, S3)$ a nemusíme sa ňou ďalej zaoberať

	S1	S2	S3	S4	S5
S6				*	*
S5	*	*	*		
S4	*	*	*		
S3	*				
S2	*				

Tab. 5.3 Tabuľka po preskúmaní dvojice (S1,S3).

Ďalšou neoznačenou dvojicou je $(S1, S6)$:

$\big(\boldsymbol{f}(\boldsymbol{S1}, \boldsymbol{a}), \boldsymbol{f}(\boldsymbol{S6}, \boldsymbol{a})\big) = (\boldsymbol{S2}, \boldsymbol{S6})$, na zoznam dvojice $(S2, S6)$ pridáme dvojicu $(S1, S6)$

$\big(\boldsymbol{f}(\boldsymbol{S1}, \boldsymbol{b}), \boldsymbol{f}(\boldsymbol{S6}, \boldsymbol{b})\big) = \big(\boldsymbol{f}(\boldsymbol{S1}, \boldsymbol{c}), \boldsymbol{f}(\boldsymbol{S6}, \boldsymbol{c})\big) = \big(\boldsymbol{f}(\boldsymbol{S1}, \boldsymbol{d}), \boldsymbol{f}(\boldsymbol{S6}, \boldsymbol{d})\big) = (\boldsymbol{S6}, \boldsymbol{S6})$, táto informácia nemá pre nás žiadnu hodnotu.

Pre dvojicu $(S2, S3)$:

$\big(\boldsymbol{f}(\boldsymbol{S2}, \boldsymbol{a}), \boldsymbol{f}(\boldsymbol{S3}, \boldsymbol{a})\big) = (\boldsymbol{S6}, \boldsymbol{S6})$,

$\big(\boldsymbol{f}(\boldsymbol{S2}, \boldsymbol{b}), \boldsymbol{f}(\boldsymbol{S3}, \boldsymbol{b})\big) = (\boldsymbol{S3}, \boldsymbol{S3})$,

$\big(\boldsymbol{f}(\boldsymbol{S2}, \boldsymbol{c}), \boldsymbol{f}(\boldsymbol{S3}, \boldsymbol{c})\big) = (\boldsymbol{S4}, \boldsymbol{S4})$,

$\big(\boldsymbol{f}(\boldsymbol{S2}, \boldsymbol{d}), \boldsymbol{f}(\boldsymbol{S3}, \boldsymbol{d})\big) = (\boldsymbol{S5}, \boldsymbol{S5})$, vo všetkých prípadoch z oboch stavov po príchode niektorej udalosti nastal prechod do rovnakého stavu. Dvojicu $(S2, S3)$ preto nemôžeme ani označiť a ani pridať na zoznam niektorej inej dvojice.

Pre dvojicu $(S2, S6)$:

$\big(\boldsymbol{f}(\boldsymbol{S2}, \boldsymbol{a}), \boldsymbol{f}(\boldsymbol{S6}, \boldsymbol{a})\big) = (\boldsymbol{S6}, \boldsymbol{S6})$, nemôžeme nič vyvodiť.

$\big(\boldsymbol{f}(\boldsymbol{S2}, \boldsymbol{b}), \boldsymbol{f}(\boldsymbol{S6}, \boldsymbol{b})\big) = (\boldsymbol{S3}, \boldsymbol{S6})$, pre túto dvojicu sa nám už podarilo preukázať, že sa nejedná o ekvivalentné stavy, preto aj dvojicu $(S2, S6)$ môžeme v tabuľke označiť.

Zároveň môžeme označiť aj všetky dvojice, ktoré sú na jej zozname a teda dvojicu $(S1, S6)$.

	S1	S2	S3	S4	S5
S6	*	*		*	*
S5	*	*	*		
S4	*	*	*		
S3	*				
S2	*				

Tab. 5.4 Tabuľka po preskúmaní dvojice (S2,S6).

Pre dvojicu $(S3, S6)$:

$\big(\boldsymbol{f}(\boldsymbol{S3}, \boldsymbol{a}), \boldsymbol{f}(\boldsymbol{S6}, \boldsymbol{a})\big) = (\boldsymbol{S6}, \boldsymbol{S6})$, nemôžeme nič vyvodiť.

$\big(\boldsymbol{f}(\boldsymbol{S3}, \boldsymbol{b}), \boldsymbol{f}(\boldsymbol{S6}, \boldsymbol{b})\big) = (\boldsymbol{S3}, \boldsymbol{S6})$, nemôžeme nič vyvodiť

$\big(\boldsymbol{f}(\boldsymbol{S3}, \boldsymbol{c}), \boldsymbol{f}(\boldsymbol{S6}, \boldsymbol{c})\big) = (\boldsymbol{S4}, \boldsymbol{S6})$, je označená v tabuľke, preto ani dvojica $(S3, S6)$ nie je ekvivalentná.

Posledná dvojica, ktorú musíme preskúmať je $(S4, S5)$:

$\big(\boldsymbol{f}(\boldsymbol{S4}, \boldsymbol{a}), \boldsymbol{f}(\boldsymbol{S5}, \boldsymbol{a})\big) = (\boldsymbol{S6}, \boldsymbol{S6})$,

$\big(\boldsymbol{f}(\boldsymbol{S4}, \boldsymbol{b}), \boldsymbol{f}(\boldsymbol{S5}, \boldsymbol{b})\big) = (\boldsymbol{S6}, \boldsymbol{S6})$,

$\big(\boldsymbol{f}(\boldsymbol{S4}, \boldsymbol{c}), \boldsymbol{f}(\boldsymbol{S5}, \boldsymbol{c})\big) = (\boldsymbol{S6}, \boldsymbol{S6})$,

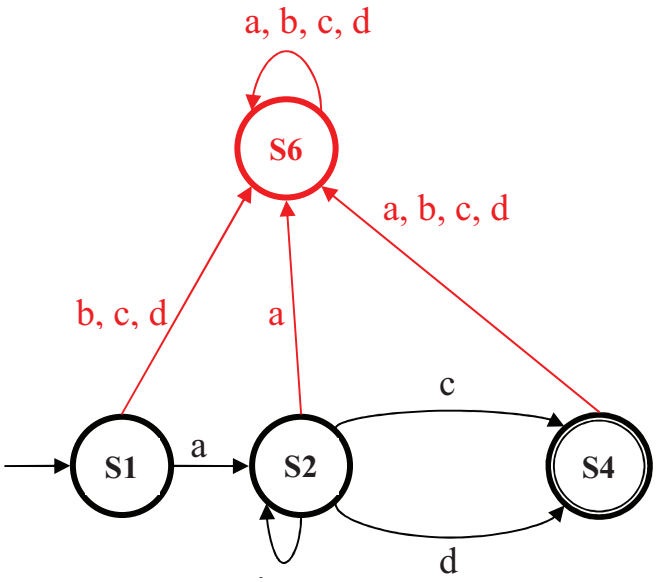
$\big(\boldsymbol{f}(\boldsymbol{S4}, \boldsymbol{d}), \boldsymbol{f}(\boldsymbol{S5}, \boldsymbol{d})\big) = (\boldsymbol{S6}, \boldsymbol{S6})$, dvojicu $(S4, S5)$ nemôžeme označiť.

Krok 4: Konečná tabuľka dvojíc stavov je:

	S1	S2	S3	S4	S5
S6	*	*	*	*	*
S5	*	*	*		
S4	*	*	*		
S3	*				
S2	*				

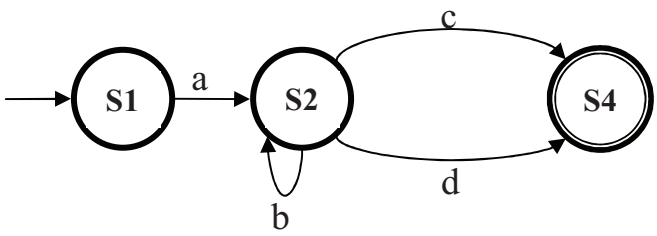
Tab. 5.5 Konečná tabuľka dvojíc.

V tab.5.5 zostali neoznačené dvojice ekvivalentných stavov $(S2, S3)$ a $(S4, S5)$. Tieto stavy môžeme zlúčiť a jazyk generovaný automatom sa nezmení.



Obr. 5.3 Po zlúčení ekvivalentných stavov.

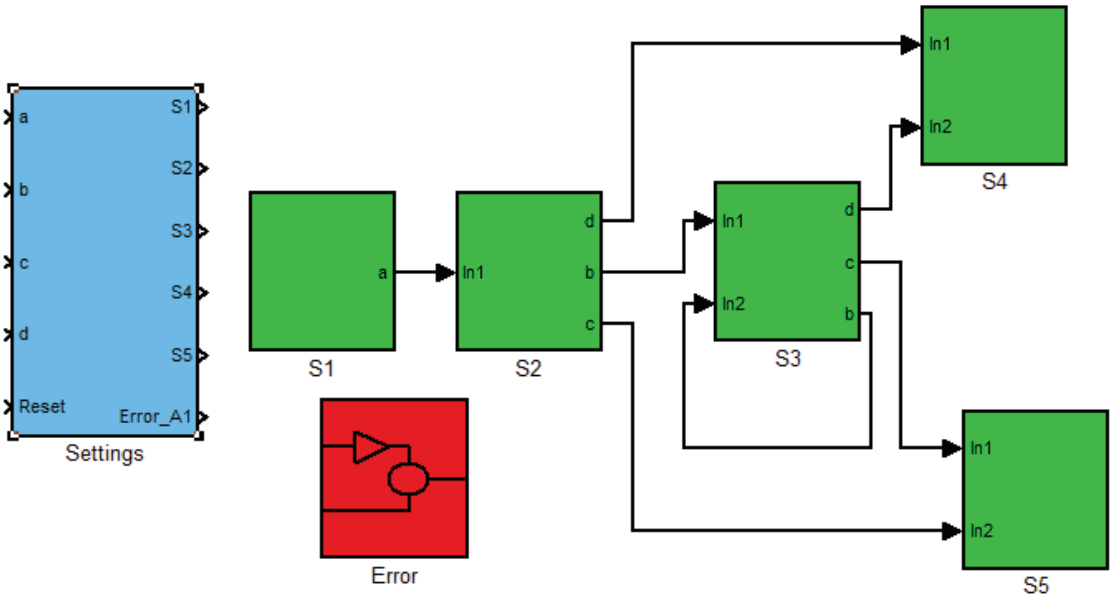
Nakoniec odstránime nedosiahnuteľnú časť a spätne nedosiahnuteľnú časť. Automat nedosiahnuteľnú časť neobsahuje a spätne nedosiahnuteľnou časťou je stav *S*6. Výsledná schéma automatu po minimalizácii je na obr. 5.4.



Obr. 5.4 Schéma automatu po minimalizácii

5.3.2 Test minimálnosti v Simulinku

Teraz automat z obr. 5.1 namodelujeme v *Simulinku* a ukážeme si ako funguje test minimálnosti. Schéma namodelovaného automatu je na obr. 5.4. Názvy funkcií jednotlivých stavov sú rovnaké, pretože majú vplyv na to, či sa dva stavy pokladajú za ekvivalentné, alebo nie. Stav *S*4 a *S*5 sú označené a majú zaškrtnuté *Marked*.



Obr. 5.5 Model automatu v Simulinku

Po stlačení tlačidla **Test of minimization** sa spustí algoritmus, ktorý určí či je schéma automatu minimálna, a ak nie je, tak určí ekvivalentné stavy. Výsledok sa objaví v príkazovom okne v *Matlabe* a pre takto namodelovaný automat sa vypíše:

```
– – – – – – – – – – – – – – – – – – – – – – – – –
– – – – – – AUTOMAT– – – – – – – – – – – – –
– – – – – – – – – – – – – – – – – – – – – – – – –
Automat_A1.txt
**************************************
– – – – – – – – – – – – – – – – – – – – – – – – –
Automaton is not minimal!!!
**************************************
Equivalent states are:
S2
S3
– – – – – – – – – – – – – – – – – – – – – – – – –
S4
S5
– – – – – – – – – – – – – – – – – – – – – – – – –
**************************************
```

Podľa tohto výpisu nie je automat minimálny a ekvivalentné dvojice stavov sú $(S2, S3)$ a $(S4, S5)$. Porovnaním tohto výsledku s tým, čo nám vyšiel v kapitole 5.3.1, kde sme ho počítali ručne zistíme, že test minimalizácie funguje správne.

Test minimalizácie si vie poradiť aj so situáciou ak sa na prechode objaví blok funkcie alebo blok vnoreného automatu. Potom sú prechody charakterizované nielen udalosťou ktorá ich vyvolala, ale aj funkciou ktorá sa pri nich vykoná, prípadne

vnoreným automatom cez ktorý sa prechádza. Pre dva ekvivalentné stavy v tomto ponímaní platí, že nielen že sú ekvivalentné z hľadiska ich prechodov, ale aj z hľadiska ich funkcie (akcie ktorá sa v nich vykonáva).

Ak by sme napríklad zmenili názov funkcie v stave $S4$ a bol by iný ako v stave $S5$ už by neboli ekvivalentné a výpis by vyzeral nasledovne:

```
----------AUTOMAT----------
Automat_A1.txt
**********
----------
Automaton is not minimal!!!
**********
Equivalent states are:
S2
S3
----------
**********
```

6 GENEROVANIE KÓDU

Kapitola sa zaoberá tým, ako vygenerovať kód v jazyku C reprezentujúci správanie sa konečného automatu namodelovaného v *Simulinku*. Na úvod bude vysvetlený algoritmus generovania kódu.

Následne bude tento algoritmus demonštrovaný na príklade. Budeme modelovať reálny stavový automat aplikácie na riadenie asynchrónneho striedavého motora. Vysvetlíme si ako vyzerá stavový diagram aplikácie spolu s funkciou každého stavu. Ukážeme si ako tento stavový diagram namodelovať spolu s výsledkami simulácie.

Nakoniec bude uvedený postup ako vygenerovať kód v jazyku C pre namodelovaný automat. Funkčnosť kódu si overíme v jednoduchej konzolovej aplikácii vytvorenej v programe *Microsoft Visual Studio 2008*. Pomocou tejto aplikácie môže užívateľ simulovať príchod jednotlivých udalostí a uvidí aktuálny stav v ktorom sa nachádza automat.

6.1 Algoritmus generovania

Generovanie zdrojového kódu prebieha v niekoľkých krokoch:

1. Vytvorenie dátovej reprezentácie hlavného automatu (v prípade ak ešte nie je vytvorená).
2. Analýza dátovej reprezentácie hlavného automatu a vytvorenie hlavičkového súboru s názvom *Automat_Tag.h*.
3. Ak sa v hlavnom automate nachádzajú vnorené automaty, tak sa analyzuje dátová reprezentácia každého z nich a vygeneruje sa vlastný hlavičkový súbor s názvom *Name_Tag.h* (*Name* predstavuje názov bloku vnoreného automatu).
4. Nakoniec sa vygeneruje zdrojový kód celého automatu aj s vnorenými automatmi s názvom *Automat_Tag.c*.

Pre **analýzu** dátovej reprezentácie je vytvorená funkcia s prototypom: *[Event,State,Nested,stopSt,initSt] = fAnalyzeTxt(Tag)*, kde *Event* sú definované udalosti automatu, *State* sú pointre na stavy, *Nested* sú štruktúry vnorených automatov, *stopSt* je chybový stav, *initSt* je počiatočný stav. V premenných *State* a *Nested* sú schované aj vlastnosti prechodových funkcií.

Funkcia **generujúca** hlavičkový súbor má prototyp: *fCreateH(Tag,isN,Nname)*, kde *Tag* je označenie automatu, *isN* je 0 pre hlavný automat, 1 pre vnorené, *Nname* je meno bloku vnoreného automatu, v prípade hlavného automatu na tomto parametre nezáleží.

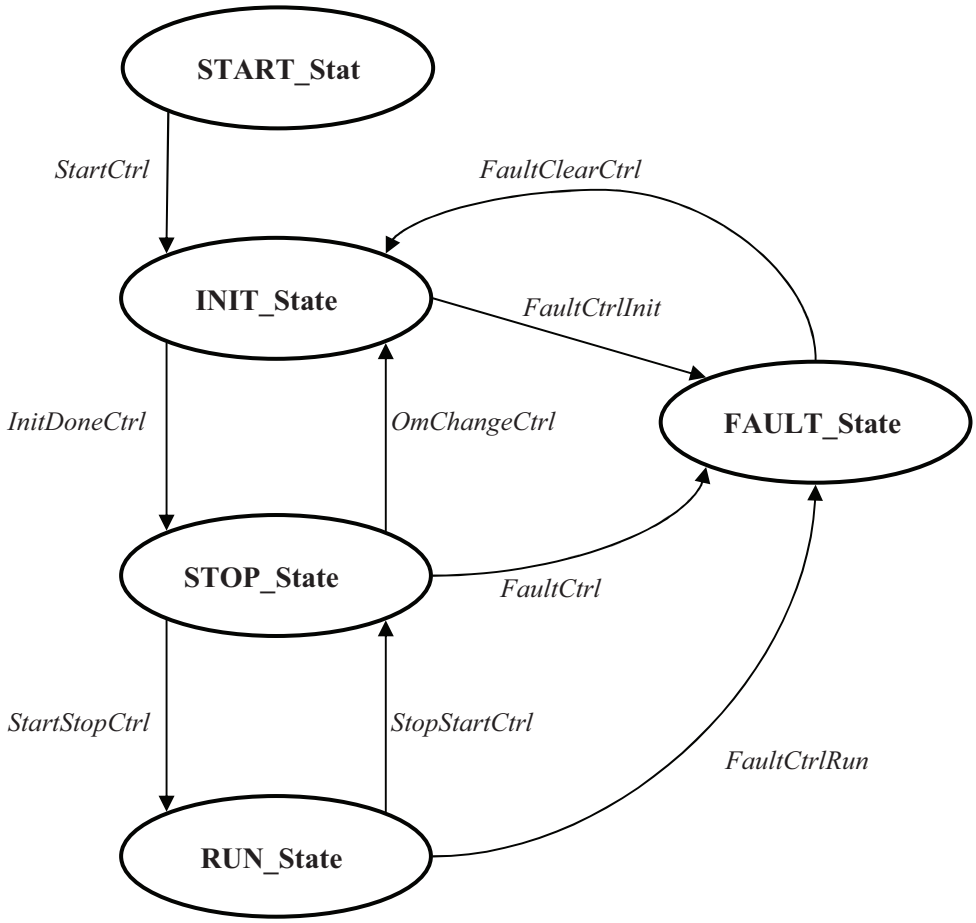
Funkcia **generujúca** zdrojový súbor má prototyp: *fCreateC(Tag)*.

Výgenerovaný kód je priamo použiteľný a odladený v programe *Microsoft Visual Studio 2008*.

6.2 Príklad konečného automatu

Stavový diagram je prevzatý z [3] s menšími úpravami. Originál pozostáva zo štyroch stavov: INIT_State, STOP_State, RUN_State, a FAULT_State. Kvôli simulácii v *Simulinku* je pridaný aj piaty stav START_State a udalosť *StartCtrl.* Tento stav slúži ako počiatočný stav celej simulácie. Pri reštarte sa automat automaticky dostane do tohto stavu. Ďalej udalosť *StartStopCtrl* znamená prechod zo stavu STOP_State do RUN_State a *StopStartCtrl* opačný prechod. V pôvodnom diagrame bola iba jedna udalosť a hodnota *StartStopCtrl* = 1 znamenala prechod zo STOP_State do RUN_State a hodnota *StartStopCtrl* = 0 opačný prechod.

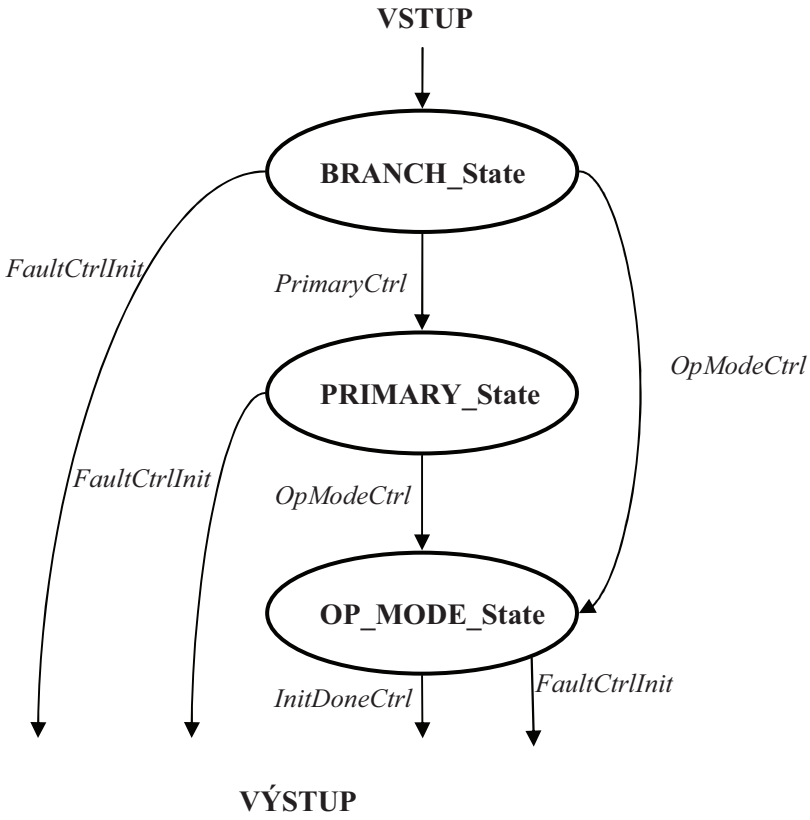
INIT_State a RUN_State predstavujú vnorené automaty. Každý z nich má svoje vlastné stavy a svoju sadu udalostí na ktoré reaguje.



Obr. 6.1 Stavový diagram aplikácie

6.2.1 Vnorený automat INIT_State

Po resete aplikácie sa automat dostane do vnoreného automatu INIT_State, kde prebieha inicializácia. Obsahuje tri podstavy: BRANCH_State, PRIMARY_State, OP_MODE_State. BRANCH_State rozhoduje o tom, či sa bude vykonávať primárna inicializácia, ak áno tak predá riadenie stavu PRIMARY_State, ak nie tak prejde automat priamo do OP_MODE_State. Do tohto stavu sa dostane automat aj vtedy keď je primárna inicializácia hotová. OP_MODE_State predstavuje operačný mód. Ten môže byť nastavený ako manuálny alebo automatický (počítačom riadený).

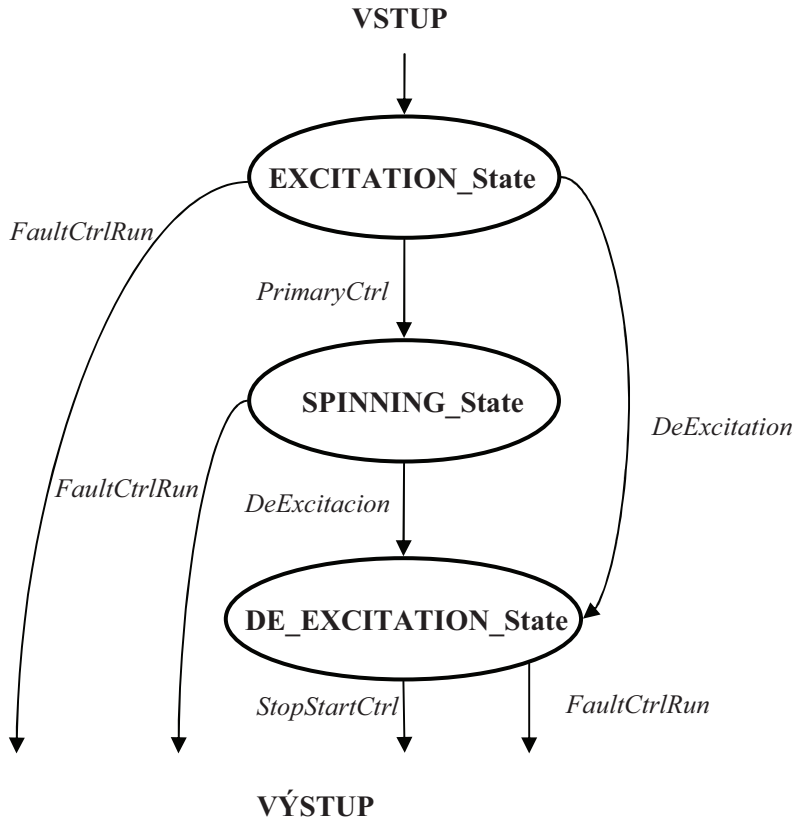


Obr. 6.2 Vnorený automat INIT_State

6.2.2 Vnorený automat RUN_State

Aplikácia sa môže dostať zo stavu STOP_State do vnoreného automatu RUN_State, ktorý predstavuje beh motora. V stave EXCITATION_State nastáva excitácia motora. V prípade ak príde príkaz k zastaveniu excitácie nastane prechod automatu priamo do stavu DE_EXCITATION_State. Keď dosiahne nominálnu veľkosť prúdu v rotore prejde automat do stavu SPINNING_State. V tomto stave je motor rozbehnutý a môže zrýchľovať alebo spomaľovať podľa požiadaviek. Ak príde príkaz pre zastavenie motora, prejde automat do stavu DE_EXCITATION_State. V tomto stave prebieha

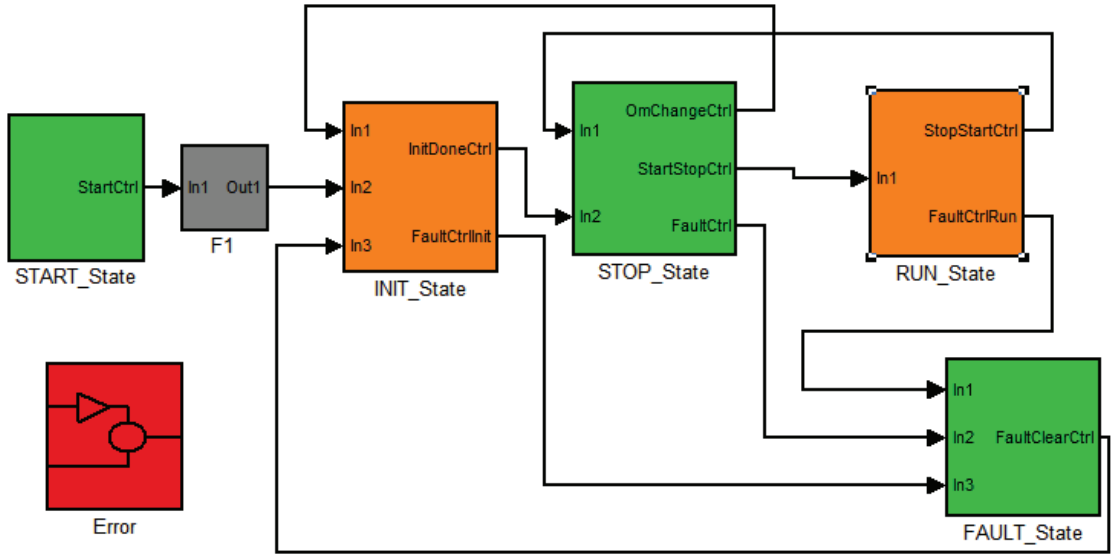
deexcitácia motora. Po nej sa predá riadenie nadradenému automatu do STOP_State.



Obr. 6.3 Vnorený automat RUN_State

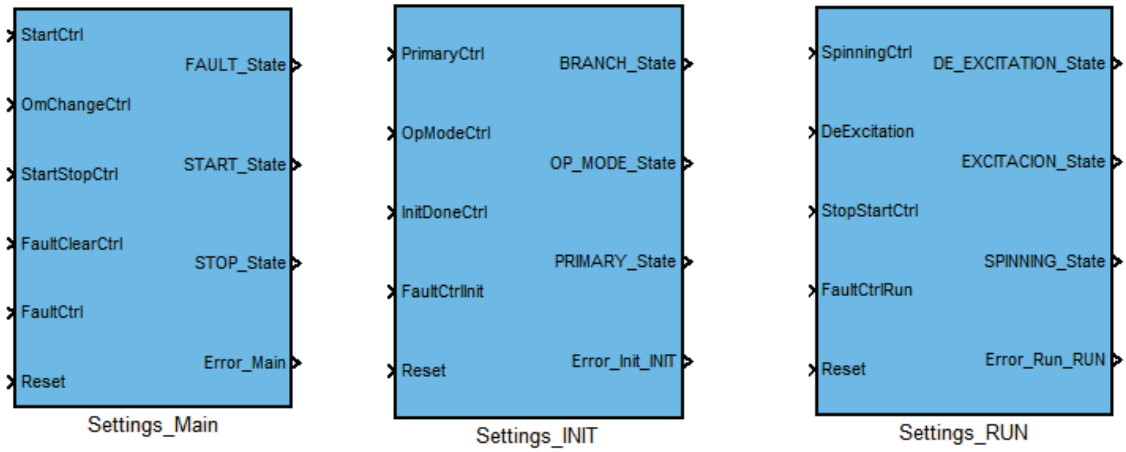
6.3 Implementácia v Simulinku

Stavový diagram aplikácie možno namodelovať v *Simulinku* pomocou schémy, ktorá je na obr. 6.4. Funkcia *F1* predstavuje akcie, ktoré sa vykonávajú pri prechode z počiatočného stavu simulácie. Potom už stavový automat pracuje až pokiaľ nenastane nejaká chyba (neočakávaná udalosť), vtedy sa dostane do chybového stavu *Error*. Do počiatočného stavu sa opäť dostane príchodom udalosti *Reset*. Ak nastane neočakávaná udalosť v niektorom vnorenom automate, prejde tento vnorený automat do svojho chybového stavu. Z neho sa opäť dostane príchodom signálu *Reset* definovaného v jeho sade udalostí.



Obr. 6.4 Model automatu v Simulinku

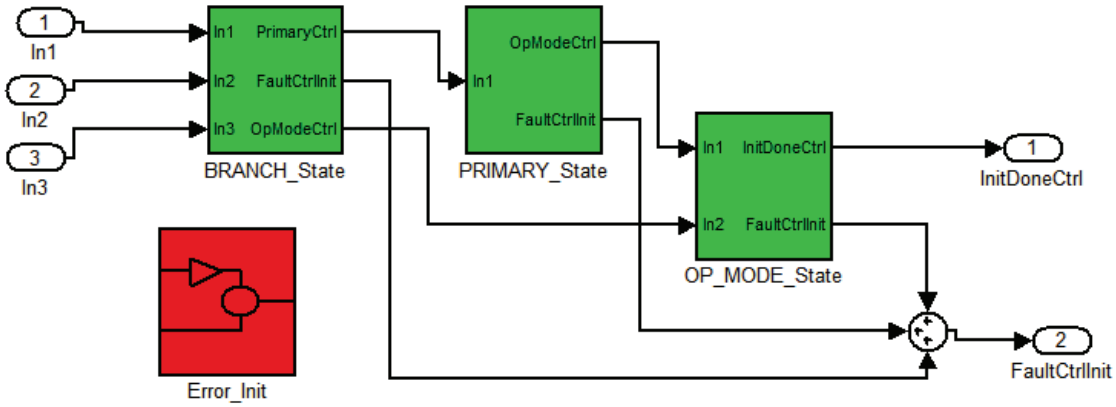
Pretože schéma obsahuje dva vnorené automaty, musíme pre každý definovať vlastné označenie (*Tag*) a vlastnú sadu udalostí. Celá schéma preto pozostáva z troch nezávislých automatov. Hlavný automat má označenie *Main*, inicializačný automat *INIT*, a automat riadenia motora *RUN*. Bloky nastavenia pre každý automat sú na obr. 6.5.



Obr. 6.5 Blok nastavenia pre každý automat

6.3.1 Vnorený automat INIT_State

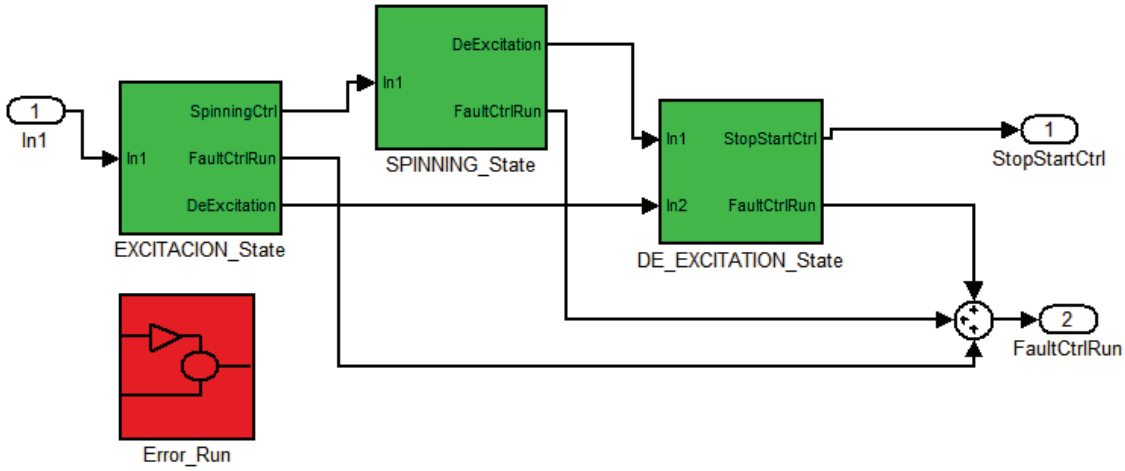
Implementácia inicializačného automatu v *Simulinku* je na obr. 6.6. Do bloku INIT_State vedú tri vstupy. Všetky vedú do stavu BRANCH_State. Z každého stavu vedie výstup v prípade poruchy do nadradeného automatu. V prípade úspešného dokončenia inicializácie sa predá riadenie hlavnému automatu cez výstup *InitDoneCtrl*.



Obr. 6.6 Schéma vnoreného automatu INIT_State.

6.3.2 Vnorený automat RUN_State

Schéma pre automat riadenia motora je na obr. 6.7. Do neho vedie jeden vstup. V prípade poruchy opäť vedie z každého stavu výstup do nadradeného automatu.



Obr. 6.7 Schéma vnoreného automatu RUN_State.

6.3.3 Výsledky simulácie v Simulinku

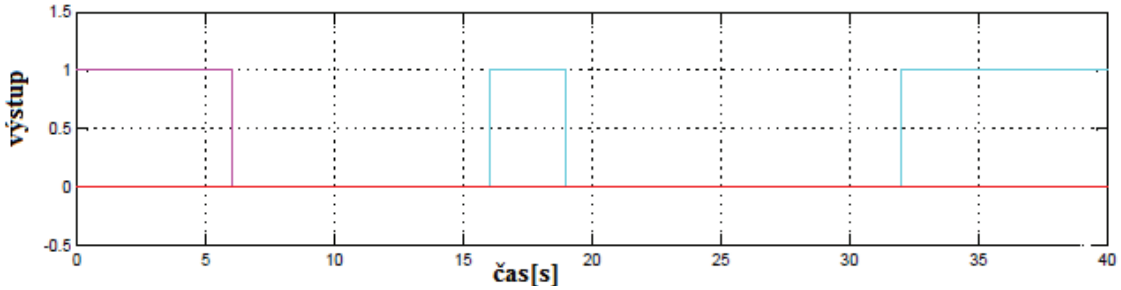
Vzorkovacia perióda hlavného aj vnorených automatov je jedna sekunda. Udalosti prichádzajú vo vopred nastavených časoch. Ich príchod je nasledovný:

Čas [s]	Udalosť	Čas [s]	Udalosť
5	*StartCtrl*	18	*StartStopCtrl*
9	*PrimaryCtrl*	21	*SpinningCtrl*
12	*OpModeCtrl*	28	*DeExcitation*
15	*InitDoneCtrl*	35	*StopStartCtrl*

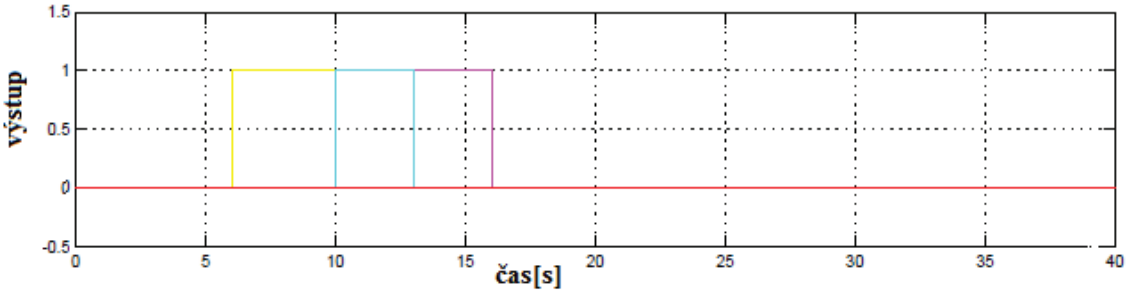
Tab. 6.1 Časy príchodov udalostí pre simuláciu.

Postupnosť stavov pre hlavný automat je vidieť na grafe 6.1, pre inicializačný automat na grafe 6.2 a pre automat riadenia motora na grafe 6.3.

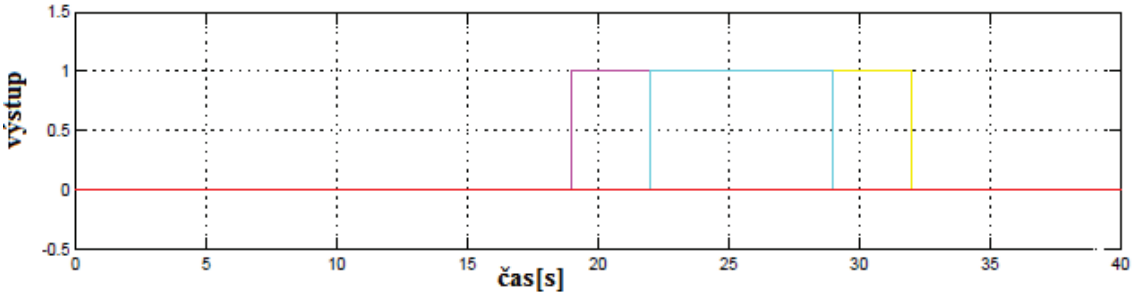
Graf 6.1 Priebeh stavov pre hlavný automat *Main* (fialová – START_State, modrá – STOP_State)



Graf 6.2 Priebeh stavov pre inicializačný automat *INIT* (žltá – BRANCH_State, modrá – PRIMERY_State, fialová – OP_MODE_State)



Graf 6.3 Priebeh stavov pre automat riadenia motora *RUN* (fialová – EXCITATION_State, modrá – SPINNING_State, žltá – DE_EXCITATION_State)



V čase 6 sekúnd predal hlavný automat riadenie vnorenému automatu INIT_State. Inicializácia prešla v poriadku cez všetky tri stavy a v čase 16 sekúnd sa hlavný automat dostal do stavu STOP_State. V čase 19 sekúnd sa dostal automat riadenia motoru do stavu EXCITATION_State. Potom prešiel do stavu SPINNING_State (22s) a nakoniec do DE_EXCITATION_State. Po deexcitácii sa automat vrátil späť do STOP_State.

6.3.4 Postup pri generovaní kódu

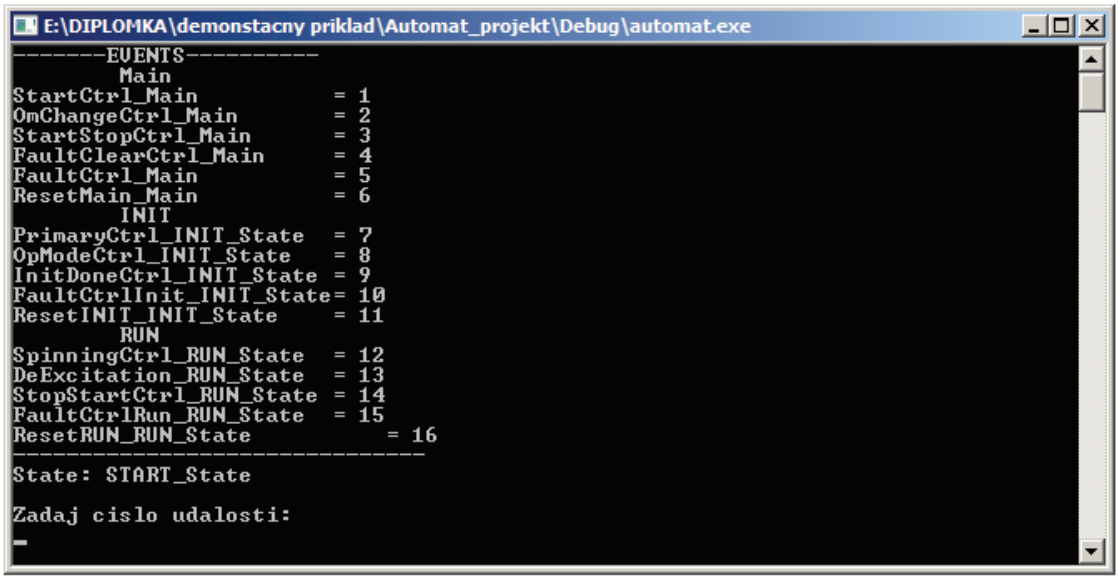
Tento postup je veľmi jednoduchý a všetko sa generuje automaticky. Je však potrebné dodržať istý postup:

1. Vytvoriť schému automatu poskladaním z jednotlivých blokov.
2. Všetky vnorené automaty, ktoré sú súčasťou hlavného automatu musia byť analyzované. Docielime to stlačením tlačidla **Analyze** v bloku nastavenia pre každý automat (modrý blok). Následne sa vytvorí pre každý automat textový súbor nesúci informácie o automate (dátová reprezentácia).
3. Následne stačí stlačiť tlačidlo **Generate** v bloku nastavení pre hlavný automat. Nie je potrebné generovať kód pre všetky vnorené automaty. Program pracuje tak, že v prvom rade sa vygeneruje hlavičkový súbor pre hlavný automat v tvare *Automat_Tag.h* a pre všetky vnorené automaty v tvare *Name_Tag.h*. V hlavičkovom súbore sú definované udalosti a stavy pre konkrétny automat v podobe výčtových typov (enumerátov). Následne sa vygeneruje zdrojový kód v jazyku C v tvare *Automat_Tag.c*.

Pre automat podľa schémy na obr. 6.4 sa vygenerujú nasledovné súbory:
Automat_Main.c – zdrojový kód automatu spolu s vnorenými stavmi automatu.
Automat_Main.h – hlavičkový súbor hlavného automatu.
INIT_State_INIT.h – hlavičkový súbor vnoreného automatu INIT_State.
RUN_State_RUN.h – hlavičkový súbor vnoreného automatu RUN_State.

6.3.5 Testovanie kódu

Vygenerovaný kód bol odladený a otestovaný v programe *Microsoft Visual Studio 2008*. Bola vytvorená konzolová aplikácia, kde si užívateľ stlačením príslušného čísla môže simulovať príchod jednotlivých udalostí a sledovať ako sa vyvíja aktuálny stav automatu. Okno konzolovej aplikácie možno vidieť na obr. 6.8.



Obr. 6.8 Okno konzolovej aplikácie

ZÁVER

V tejto práci bol vytvorený a predstavený programový prostriedok pre modelovanie konečných stavových automatov v prostredí *Matlab/Simulink* v podobe piatich základných blokov.

Pre analýzu stavového automatu bola vytvorená štruktúra dátovej reprezentácie konečného automatu, ktorá má podobu textového súboru. Tento súbor možno načítať v ktorejkoľvek aplikácii a na základe pravidiel popísaných v tejto práci z nej možno dostať pôvodnú schému stavového automatu s kompletnými prechodovými funkciami.

V ďalšom kroku bol nad dátovou reprezentáciou aplikovaný algoritmus minimalizácie. Jeho výstupom je informácia o tom, či je namodelovaný konečný automat minimálny. V prípade ak je neminimálny, tak sú identifikované dvojice ekvivalentných stavov. Prípadné zlúčenie týchto dvojíc je ponechané na užívateľa.

V prípade ak je konečný automat namodelovaný správne, tzn. tak ako je popísané v tejto práci, je implementovaná aj možnosť vygenerovať si zdrojový súbor v jazyku C pre tento automat. Vygenerovaný zdrojový súbor priamo zlučuje nadradený automat s prípadnými vnorenými automatmi. Aby bolo možné zhodnotiť funkčnosť vygenerovaného kódu, bola vytvorená konzolová aplikácia pomocou ktorej možno simulovať príchod udalostí a pozorovať aktuálny stav.

Tesne pred odovzdaním tejto práce sa v najnovšej verzii programu *Matlab* objavili neočakávané, a v istom zmysle aj nepochopiteľné zmeny v názvoch niektorých základných blokoch. Je preto potrebné pozmeniť niektoré názvy v programe, aby navrhnuté riešenie fungovalo bez problémov aj v najnovšej verzii programu.

Literatúra

[1] CASSANDRAS, Ch.G., LAFORTUNE, S. *Introduction to Discrete Event Systems, 2nd ed.* Springer, 2008. 771 s. ISBN 978-0-387-33332-8.

[2] SROVNAL, Vilém. Vestavné systémy: charakteristika, vývoj, použití. *AUTOMA*. 2007, 10, s. 4-8. Dostupný také z WWW: <http://www.odbornecasopisy.cz/download/automa/2007/au100704.pdf>.

[3] 3-Phase AC Induction Motor Vector Control Using a 56F80x, 56F8100 or 56F8300 Device. *Freescale Semiconductor, Application Note* [online]. 2005, 2, [cit. 2011-05-16]. Dostupný z WWW: <http://tungvp.files.wordpress.com/2010/03/3-phase-ac-induction-motor-vector-control-using-dsp56f80x.pdf>.

[4] VAVŘÍN, Pert. *Teorie dynamických systémů*. Brno : Rektorát Vysokého učení technického v Brně, 1989. Základní pojmy a definice z teorie systémů, s. 177.

[5] *Code.google.com* [online]. 2007 [cit. 2011-05-16]. Project Hosting on Google Code. Dostupné z WWW: <http://code.google.com/p/pointer/>.

[6] *MATLAB Central* [online]. 2011 [cit. 2011-05-17]. Simulink user community. Dostupné z WWW: <http://www.mathworks.com/matlabcentral/>.

[7] *C Programming and C++ Programming* [online]. 2009 [cit. 2011-05-17]. C programming.com. Dostupné z WWW: <http://www.cprogramming.com/>.

[8] *Tutorial - Controlling The Real World With Computers* [online]. 2008 [cit. 2011-05-17]. Control And Embedded Systems. Dostupné z WWW: <http://www.learn-c.com/>.

Zoznam skratiek

Skratka	Popis
RTOS	*Real Time Operating System.* Operačný systém reálneho času
DES	*Discrete Event Systems*. Systémy diskrétnych udalostí.
FPGA	*Field Programmable Gate Array*. Programovateľné hradlové pole.
ASIC	*Application Specific Integrated Circuit*. Zákaznícky obvod vyrobený na mieru aplikácie.
LIFO	*Last In, First Out*.
FIFO	*First In, Last Out*.
CPU	*Central Processing Unit.* Centrálna procesorová jednotka

Zoznam príloh

CD-ROM Obsahuje:

- *DP_Matas_Marek_2011.pdf*.
- Simulačné schéma neminimálneho automatu (kapitola 5.3).
- Simulačné schéma stavového automatu aplikácie (kapitola 6.3).
- Konzolovú aplikáciu (kapitola 6.3.1).
- *pointers-01.00.05.rar* – Matlab Pointer Library.
- Všetky navrhnuté komponenty v Matlabe (kapitola 3).