POUŽITIE KONEČNÉHO AUTOMATU PRI PROGRAMOVANÍ PLC
USING OF FINITE AUTOMATON AT PROGRAMMING PLC

Karol Rástočný, Juraj Ždánsky
Katedra riadiacich a informačných systémov, Elektrotechnická fakulta, Žilinská univerzita, Veľký diel, 010 26 Žilina

Abstrakt Článok sa zaoberá systematickými postupmi pri programovaní programovateľných logických automatov (PLC), ktoré vychádzajú z algebraického opisu správania sekvenčného obvodu spôsobom konečného automatu. Takýto prístup zefektívňuje prácu programátora a umožňuje použiť formalizmy v celom procese vývoja systému, t. j. od procesu analýzy požiadaviek až po proces verifikácie a validácie vytvoreného programu. V článku sa uvažuje s použitím priečkovej logiky pri programovaní PLC.

Summary The paper is concerning with systematic advances at programming programmable logic controllers (PLC), which comes out from algebraic description of behaviour of sequential circuit, in the way of finite automaton. This kind of access is streamlining the work of a programmer and enabling to use formalisms in the of whole process of system development, that is from process of analysing demands to process of verification and validation created program. The paper considers about using of ladder diagram at programming PLC.

1. ÚVOD

Automatické riadenie sa uplatňuje v najrozličnejších sférach života, počnúc jednoduchými procesmi a končiac zložitými bezpečnostne kritickými procesmi. Jedným z vhodných prostriedkov na realizáciu automatického riadenia sú aj programovateľné logické automaty (PLC), čomu zodpovedá aj ich obvodové aj technologické riešenie. Kvalita riadenia závisí od kvality riadiaceho programu, preto treba pri jeho tvorbe použiť vhodné metódy a postupy, ktoré umožnia vytvoriť program s požadovanými spoľahlivostnými a bezpečnostnými vlastnosťami 0.

Pre bezpečnostne kritické systémy je dôležité, aby na báze formálnych prípadne poloformálnych metód bol v procese analýzy požiadaviek vytvorený prehľadný a zrozumiteľný model, ktorý umožní odstrániť prípadné nejasnosti alebo protirečenia v neformálnej špecifikácii a umožní preskúšať komplexnosť a bezchybnosť špecifikácie. Použitie vhodného formalizmu na opis správania riadiaceho systému (vytvorenie modelu) výrazne zefektívni prácu programátora a minimalizuje systematické chyby v programe.

Ak riadiaci systém možno považovať za sekvenčný obvod, dá sa jeho činnosť opísať (modelovať) algebraickým systémom – konečným automatom. Použitie tohto modelu na tvorbu programu si vyžaduje stanoviť vhodnú metodiku v závislosti od spôsobu zápisu konečného automatu a od spôsobu činnosti PLC.

2. KONEČNÝ AUTOMAT

Konečný automat $\boldsymbol{M}$ je usporiadaná pätica

$$\boldsymbol{M} = (A, S, U, p, v), \tag{1}$$

kde $\boldsymbol{A}$ je množina vstupných symbolov (vstupy), $\boldsymbol{S}$ je množina stavov a $\boldsymbol{U}$ je množina výstupných symbolov (výstupy); p a v sú zobrazenia typu:

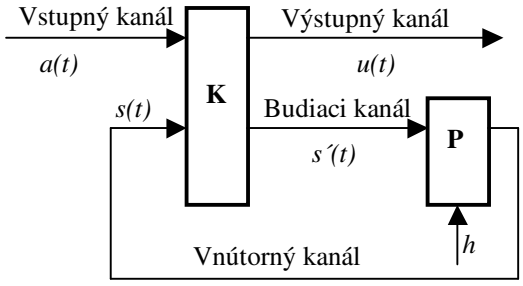
$$p : \boldsymbol{S} \times \boldsymbol{A} \rightarrow \boldsymbol{S} \tag{2}$$
$$v : \boldsymbol{S} \times \boldsymbol{A} \rightarrow \boldsymbol{U} \text{ (príp. } v : \boldsymbol{S} \rightarrow \boldsymbol{U} \text{)},$$

pričom zobrazenie p sa nazýva prechodová funkcia a zobrazenie v výstupná funkcia [1].

Konečný automat, ktorého výstupná funkcia v má obor $\boldsymbol{S} \times \boldsymbol{A}$, t.j. priraďuje určitý výstupný symbol každej dvojici (stav, vstup), sa nazýva Mealyho automat. Ak zobrazenie v má obor $\boldsymbol{S}$, t.j. priraďuje výstupný symbol každému stavu, ide o Moorov automat. Obidva tieto typy automatov môžu byť realizované ako synchrónny sekvenčný obvod (SSO) alebo asynchrónny sekvenčný obvod (ASO).

Inými spôsobmi stavového zápisu konečného automatu sú zápisy pomocou prechodovej tabuľky, inverznej prechodovej tabuľky a prechodového grafu (stavového diagramu).

Základný štruktúrny model synchrónneho sekvevenčného obvodu je na obr. 1. Systém sa skladá z dvoch podsystémov K a P, pričom K je kombinačná časť a P je pamäťová časť sekvenčného obvodu.



Obr. 1. Základný model synchrónneho sekvenčného obvodu
Fig. 1. Base model of the synchronous sequential circuit

Pamäťová časť vytvára jednoduché oneskorenie o jeden takt, t.j.

$$s_{(t+1)} = s'_{(t)}, \tag{3}$$

kde t je diskrétny čas definovaný synchronizačnou premennou h. Synchronizačná premenná h definuje okamihy, v ktorých môžu nastávať prechody medzi stavmi. Potom správanie obvodu zobrazeného na obr. 1 v čase t možno opísať vzťahmi

$$s_{(t+1)} = s'_{(t)} = p_{(s(t),a(t))}, \quad (4)$$
$$u_{(t)} = v_{(s(t),a(t))},$$

kde symboly $a_{(t)}$ a $u_{(t)}$ označujú konkrétne vstupné a výstupné vektory systému v čase t; symboly $s_{(t)}$ a $s_{(t+1)}$ označujú konkrétne stavy, ktoré sa vyskytujú vo vnútornom kanále v čase t a v čase $t+1$; $s'_{(t)}$ označuje vektor stavových premenných, ktorý zodpovedá konkrétnemu stavu v čase $t+1$ (v podstate ide o nasledujúci stav).

Kombinačná časť realizuje zobrazenia p a v, ktoré každej dvojici $s_{(t)}$, $a_{(t)}$ priradia $s'_{(t)}$ v budiacom kanále a $u_{(t)}$ vo výstupnom kanále.

Asynchrónny sekvevenčný obvod sa od SSO odlišuje tým, že nie je riadený synchronizačným vstupom (vstup pre synchronizačnú premennú h). Prechody medzi stavmi sú pri týchto obvodoch inicializované zmenami vstupných a stavových premenných obvodu.

3. PROGRAMOVANIE PLC

Rozšíreným spôsobom programovania PLC je tzv. priečková logika (*Ladder Diagram*) – priečkový program. Ide o grafickú metódu programovania založenú na technikách používaných pre reléové obvody. Priečky takéhoto programu sú zoradené pod sebou, pričom vykonávanie programu prebieha zľava doprava po jednotlivých inštrukciách a zhora dole po jednotlivých priečkach. Inštrukcie sa umiestňujú do priečky, pričom v jej ľavej časti sa umiestňujú inštrukcie (vstupné inštrukcie) ovplyvňujúce činnosť inštrukcií umiestnených v pravej časti priečky (výstupné inštrukcie). Jednotlivé časti priečky programu môžu mať paralelné vetvy, v ktorých sú tiež umiestnené inštrukcie. Tak možno vo vstupnej časti priečky zadať rozvetvenú podmienku, ktorá z logického hľadiska predstavuje použitie operátorov OR a AND. Vo výstupnej časti priečky možno paralelným vetvením dosiahnuť nastavenie niekoľkých výstupov súčasne.

Ďalšie spôsoby programovania PLC sú uvedené v norme 0.

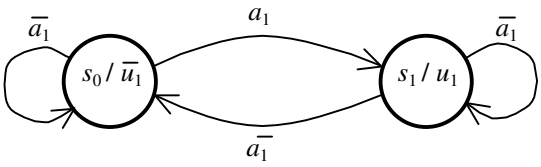
Väčšina PLC pracuje pod jednoduchým operačným systémom, ktorý zaisťuje výkon operačného cyklu (*scan*) podľa obr. 2. Počas tohto opakujúceho sa cyklu sa okrem užívateľského programu vykonávajú aj činnosti, ktoré sú firemne definované.

Čas vykonávania programu je spravidla kontrolovaný kontrolným časovačom (*Watch Dog Timer*). Kontrolný časovač (v podstate ide o čítač) sa spustí na začiatku periodicky sa opakujúcej činnosti procesora, ktorá je sledovaná a odmeriava vopred nastavený časový interval, v rámci ktorého vyžaduje pravidelnú obsluhu od procesora (nastavenie počiatočnej hodnoty). Ak uplynie nastavený čas a procesor nevykoná požadovanú obsluhu kontrolného časovača, tak kontrolný časovač reaguje na vzniklú situáciu vopred definovaným spôsobom (zastavenie činnosti, nastavenie výstupov PLC do stavu logická nula, poruchové hlásanie, ...). Dôvodom prečo procesor nevykonal obsluhu kontrolného časovača môže byť napríklad chyba softvéru, chyba v dôsledku poruchy hardvéru alebo vplyvu operačného prostredia.

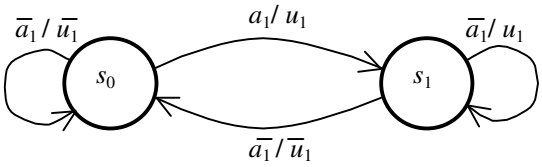
Obr. 2. Operačný cyklus PLC
Fig. 2. Operation cycle of PLC

4. TVORBA PROGRAMU ZO STAVOVÉHO DIAGRAMU

Konečný automat možno opísať aj orientovaným grafom, ktorý sa nazýva stavový diagram alebo prechodový graf. Vrcholy grafu reprezentujú stavy automatu. Orientované hrany zodpovedajú prechodom medzi stavmi a sú ohodnotené vstupnými symbolmi $a_i \in A$, ktoré aktivujú prechod automatu z jedného stavu do druhého stavu. Ak každému stavu automatu je priradený výstupný symbol $u_i \in U$, potom ide o Moorov automat (obr. 3). Ak výstupný symbol je priradený prechodu, potom ide o Mealyho automat (obr. 4). automatu.



Obr. 3. Stavový diagram Moorovho automatu
Fig. 3. State diagram of Moore automaton



Obr. 4. Stavový diagram Mealyho automatu
Fig. 4. State diagram of Mealy automaton

V oboch prípadoch však treba definovať (prípadne aj vhodným spôsobom v grafe označiť) začiatočný stav. Systém možno do začiatočného stavu nastaviť buď vhodnou voľbou kódového slova pre začiatočný stav, alebo vhodným riešením v programe.

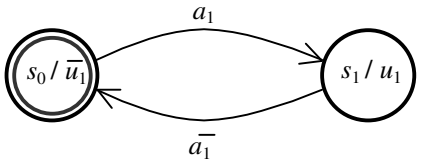
Snaha o efektívnosť výpočtu vedie k nutnosti zaoberať sa problematikou minimalizácie počtu stavov a prechodov automatu.

Pri minimalizácii počtu stavov sa vychádza z tvrdenia, že dva stavy s a $s´$ sú zlučiteľné vtedy, ak pre všetky vstupné symboly $a_i \in A$ platí, že

$$v_{(s,a_i)} = v_{(s',a_i)} \ , \qquad (5)$$

Existuje viacero metód minimalizácie počtu stavov, ktoré sú algoritmizovateľné a možno ich použiť v procese syntézy automatu 0, 0.

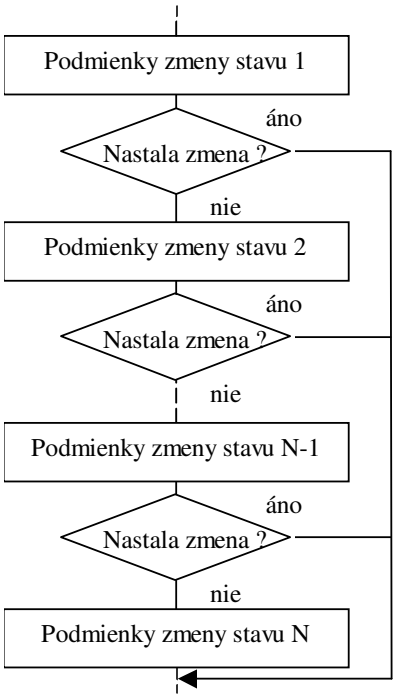
Ďalšie zefektívnenie výpočtu možno dosiahnuť znížením počtu prechodov. Napríklad tak, že zo stavového diagramu sa vylúčia všetky prechody, ktoré nespôsobujú zmenu stavu automatu. Na obr. 5 je znázornený stavový diagram, ktorý vznikne po úprave diagramu na obr. 3. Základný stav je označený dvojitým kruhom.



Obr. 5. Upravený stavový diagram automatu
Fig. 5. Modified state diagram of automaton

Ak počas jedného operačného cyklu je dovolená len jedna zmena stavu (napríklad z dôvodu vylúčenia hazardov a zaistenia stability obvodu), tak v programe musia byť použité „pomocné“ stavové premenné, ktoré zaznamenávajú požiadavku na zmenu stavu počas operačného cyklu. Skutočná zmena stavu sa uskutoční spravidla až na konci programu (nie je to podmienkou správnej činnosti). Efektívnosť výkonu programu možno zvýšiť vhodným poradím priečok a vetvení v priečkach. Pomocné stavové premenné nie sú potrebné, ak po zmene stavu sa ukončí výkon programu v tomto programovom cykle (obr. 6). Takéto riešenie vedie k výraznému skráteniu času výkonu programu. V tomto prípade ide o určitú analógiu so SSO a časový interval prechodov medzi stavmi je daný časom operačného cyklu PLC. Kódovanie stavov (priradenie stavových premenných jednotlivým stavom) systému si v tomto prípade nevyžaduje špeciálny prístup. Na priradení kódových slov jednotlivým stavom v podstate nezáleží. Možno použiť binárny kód, ktorého dĺžka L závisí od počtu stavov automatu (N).

$$N \le 2^L \ . \qquad (6)$$



Obr. 6. Skrátenie behu programu
Fig. 6. Shortening of program running

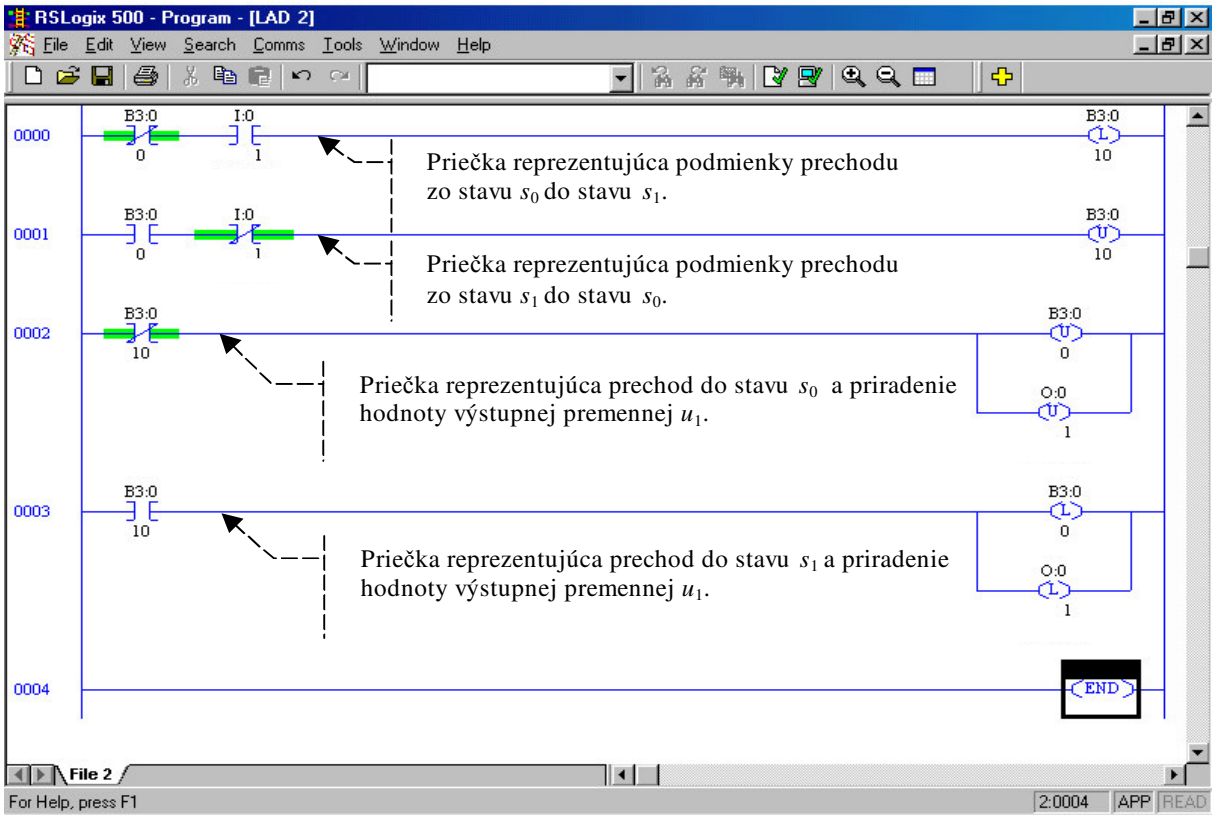
Na obr. 7 a obr. 8 sú zobrazené programy zodpovedajúci stavovému diagramu podľa obr. 5, ktoré sú vytvorené v programovom prostredí RSLogix 500 (firma Rockwell Automation). Program na obr. 7 využíva pomocné stavové premenné a program na obr. 8 využíva inštrukciu TND ukončenia behu užívateľského programu počas tohto operačného cyklu. Priradenie prvkov pamäte jednotlivým premenným je uvedené v tab.1.

Tab. 1. Kódovanie premenných
Tab. 1. Variable coding

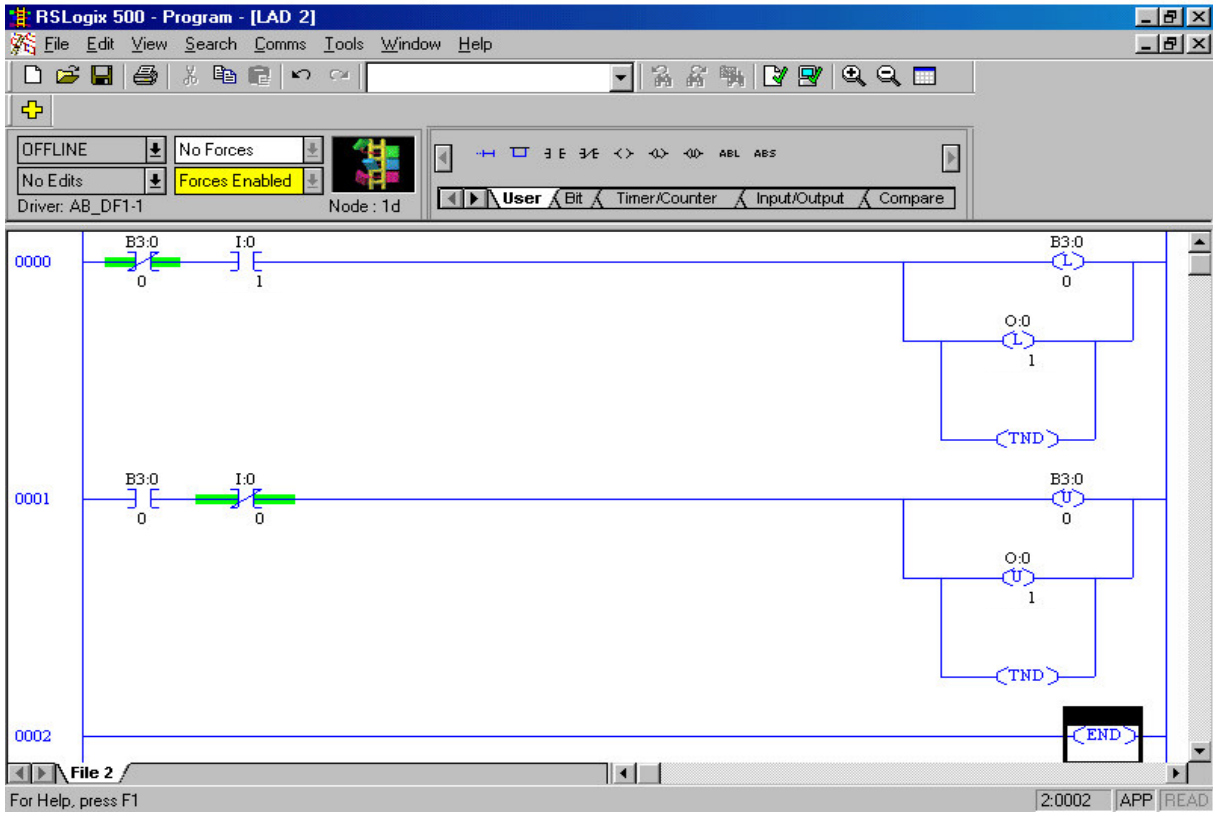
	Stavová premenná	Pomocná stav. premenná
s_0	B3:0/0 = log.0	B3:0/10 = log.0
s_1	B3:0/0 = log.1	B3:0/10 = log.1
	Vstupná premenná	
a_1	I:0/1	
	Výstupná premenná	
u_1	O:0/1	

V programe sú použité nasledujúce operácie:

- XIC ⊣ ⊢ inštrukcia je priechodná, ak adresný bit má hodnotu log. 1;
- XIO ⊣/⊢ inštrukcia je priechodná, ak adresný bit má hodnotu log. 0;
- OTL ⊣(L)⊢ ak je priečka priechodná, tak nastavuje adresný bit na hodnotu log.1; hodnotu tohto bitu možno zmeniť inou vhodnou inštrukciou (napr. OTU);
- OTU ⊣(U)⊢ ak je priečka priechodná, tak nastavuje adresný bit na hodnotu log.0; hodnotu tohto bitu možno zmeniť inou vhodnou inštrukciou (napr. OTL).



Obr. 7. Program s pomocnými stavovými premennými
Fig. 7. Program with auxiliary state variable



Obr. 8. Program s inštrukciou skoku
Fig. 8. Program with instruction of jump

Ak počas jedného operačného cyklu je tolerovaná viacnásobná zmena stavu, tak potom ide o analógiu s ASO. V tomto prípade nie sú potrebné pomocné stavové premenné, ale výkon programu môže byť ovplyvnený poradím priečok v programe a vhodným kódovaním stavov systému. Takéto riešenie je pre kritické aplikácie nevhodné, pretože dôkaz funkčnej bezpečnosti je pri zložitejších systémoch značne problematický 0.

5. TVORBA PROGRAMU Z PRECHODOVEJ TABUĽKY

Inou možnosťou stavového zápisu konečného automatu je zápis pomocou prechodovej tabuľky alebo inverznej prechodovej tabuľky .

Prechodová tabuľka a inverzná prechodová tabuľka sú podobné stavové zápisy správania sekvenčného obvodu. Obidva tieto zápisy možno najjednoduchšie dostať prepisom zo stavového diagramu.

Na mieste je otázka, prečo vlastne prepisovať stavový diagram na iný spôsob zápisu konečného automatu, ak možno program realizovať priamo zo stavového diagramu. Jedným z dôvodov môže byť aj sprehľadnenie procesu minimalizácie počtu stavov automatu. Tvorba programu z prechodovej tabuľky (inverznej prechodovej tabuľky) nadobúda význam aj vtedy, keď prechodová tabuľka (inverzná prechodová tabuľka) je výsledkom syntézy konečného automatu použitím formálnych prostriedkov, napríklad regulárnych výrazov. Proces tvorby prechodovej tabuľky z regulárnych výrazov je algoritmizovateľný, pričom možno vygenerovať tabuľku, ktorá reprezentuje úplný konečný automat s minimálnym počtom stavov. Ak z neformálnej špecifikácie systému vyplýva, že niektoré kombinácie vstupných premenných pre niektoré stavy nie sú definované, potom tieto nedefinované prechody možno vhodne využiť na identifikáciu neželaných kombinácií vstupných premenných pri danom stave automatu alebo na detekciu poruchy (takáto kombinácia je znakom poruchy) a na prevod systému do vopred definovaného bezpečného stavu pri poruche.

Postup pri tvorbe programu z prechodovej tabuľky (inverznej prechodovej tabuľky) je zhodný s postupom použitým pri tvorbe programu zo stavového diagramu.

Proces syntézy sekvenčného obvodu nemusí skončiť vygenerovaním prechodovej tabuľky, ale môže pokračovať až do momentu zostavovania budiacich a výstupných funkcií. Spôsob tvorby programu z týchto funkcií sa zdá byť veľmi výhodný, pretože vedie k najkratšiemu programu v porovnaní s programami vytvorenými podľa metodík uvádzaných v tomto príspevku. Aj napriek tomuto faktu, nie je takého riešenie pre bezpečnostne kritické systémy vhodné, pretože dochádza k strate prehľadnosti a zrozumiteľnosti modelu. Tým sa zvyšuje pravdepodobnosť omylu a zhoršujú sa podmienky pre verifikáciu a validáciu systému.

6. ZÁVER

Jednoznačne nemožno povedať, ktorá z uvedených metodík tvorby programu na základe zápisu konečného automatu je najvýhodnejšia. Každá z týchto metodík má svoje výhody aj nevýhody. Voľba metódy bude závisieť, okrem iného, aj od vlastností aplikačného hardvéru. Pri tvorbe programu pre bezpečnostne kritické systémy sa treba vyhnúť takým postupom, ktoré síce minimalizujú kód (programové triky, zhusťovanie kódu, spätné skoky, ...), ale môžu byt zdrojom systematických chýb v programe.

Štruktúra softvéru bezpečnostne kritických systémov sa musí vyznačovať jednoduchosťou a prehľadnosťou, aby sa v procese verifikácie a validácie dala overiť jeho správna funkcia. Preto v prípade zložitých riadiacich systémov treba zvážiť dekompozíciu systému na moduly. Dôležité je, aby bola vhodne zvolená úroveň dekompozície systému. Všeobecne platí, že čím sú moduly menšie, tým sú jednoduchšie, ale na druhej strane je zložitejšie riadenie ich vzájomnej koordinácie. Moduly treba vytvoriť tak, aby modul bol jasný a zrozumiteľný, aby zodpovedal špecifickej funkcii, aby mal čo najmenší počet vstupov a výstupov, aby rozhrania medzi modulmi boli presne definované a výmena informácií medzi modulmi bola čo najmenšia.

Článok bol spracovaný za podpory grantovej úlohy VEGA 1/1044/04: Teoretický aparát pre implementáciu e-safety do integrovaných dopravných systémov.

LITERATÚRA

[1] Brikhoff, G; Bartee, T.: Aplikovaná algebra. Vydavateľstvo Alfa, Bratislava, 1970.

[2] Frištacký, N., Kolesár, M., Kolenička, J., Hlavatý, J. Logické systémy. Vydavateľstvo Alfa, Bratislava, 1986.

[3] Ždánsky, J.: Využitie teórie konečných automatov pri programovaní PLC. Diplomová práca. 2003.

[4] STN EN 61508: Funkčná bezpečnosť elektrických/elektronických/programovateľných elektronických bezpečnostných systémov. 2002.

[5] STN IEC 61131-3: Programovateľné regulátory. Časť 3: Programovacie jazyky. 2001.