Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií
FIIT-13428-56206

Bc. František Kudlačák

Syntéza asynchrónnych sekvenčných obvodov

Diplomová práca

Študijný program: Počítačové a komunikačné systémy a siete
Študijný odbor: 9.2.4 Počítačové inžinierstvo
Miesto vypracovania: Ústav počítačových systémov a sietí, FIIT STU Bratislava
Vedúci práce: doc. RNDr. Elena Gramatová, PhD.

Máj 2013

Anotácia

Slovenská technická univerzita v Bratislave
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ

Syntéza asynchrónnych sekvenčných obvodov

Študijný program: Počítačové a komunikačné systémy a siete
Autor: Bc. František Kudlačák
Vedúci bakalárskej práce: doc. RNDr. Elena Gramatová, PhD.

Máj 2013

Diplomová práca sa zaoberá syntézou asynchrónnych sekvenčných obvodov, umožňujúcou vytvoriť asynchrónnu reprezentáciu automatu pomocou logických obvodov. Navrhnutý programový systém načíta opis automatu vo formáte VHDL, ktorý je výstupom návrhového prostriedku HDL Designer a vytvorí optimálnu reprezentáciu automatu vzhľadom na rýchlosť, veľkosť a spotrebu. Výstupná reprezentácia automatu je vo formáte VHDL, a je možné ju simulovať v simulačnom prostriedku ModelSim. Programový systém zahŕňa aj implementáciu na vysokovýkonnom výpočtovom systéme s využitím komunikačnej knižnice MPI. Súčasťou práce je aj formálny opis implementovanej metódy návrhu zakódovania stavov v automate.

Na otestovanie funkčnosti programového systému bol použitý testovací prístup čiernej skrinky a výstupy boli odsimulované v simulačnom prostriedku ModelSim. Efektívnosť programu na vysokovýkonnom výpočtovom systéme bola otestovaná vzhľadom na veľkosť vstupného automatu a počet uzlov, podieľajúcich sa na výpočte.

Annotation

Slovak University of Technology Bratislava
FACULTY OF INFORMATICS AND INFORMATION TECHNOLOGIES

Synthesis of asynchronous sequential circuits

Degree Course: Computer and Communication Systems and Networks
Author: Bc. František Kudlačák
Supervisor: doc. RNDr. Elena Gramatová PhD.

May 2013

The diploma thesis deals with design of asynchronous sequential circuits, which allows to design asynchronous representation of an automat. This representation is composed from digital logic. Designed system loads the automat description in VHDL format, which is output format of the computer-aided design system HDL Designer. Output representation of the automat is in VHDL syntax, and it is possible to simulate output in the simulation environment ModelSim. Implementation of designed system includes implementation for the parallel super computing. Communication in the parallel system is provided by the communication library MPI. The thesis also contains formal description of the implemented method for design of state coding.

The black box testing approach was used during the function tests, and outputs were simulated in the simulation environment ModelSim. Effectiveness of parallel implementation for super computing, was tested with different sizes of input automat, and with different numbers of computing nodes in the super computing system.

Poďakovanie

Za odborné vedenie a konzultácie pri vytváraní diplomovej práce ďakujem vedúcej práce doc. RNDr. Elene Gramatovej, PhD.. Za konzultácie ďakujem Ing. Michalovi Kudlačákovi a za odbornú pomoc pri implementácii na vysokovýkonnom výpočtovom systéme ďakujem Mgr. Lukášovi Demovičovi, PhD. z Výpočtového strediska SAV.

Prehlásenie

Prehlasujem, že som diplomovú prácu vypracoval samostatne, pričom bola použitá uvedená literatúra.

.....

Bc. František Kudlačák

Obsah

Zoznam príloh

Zoznam obrázkov

Zoznam tabuliek

1 Úvod

V súčasnosti je väčšina digitálnych sekvenčných obvodov navrhovaná synchrónne. Všetky časti navrhnutého systému zdieľajú jednu spoločnú synchronizačnú premennú – hodinový signál. Naproti tomu asynchrónne sekvenčné obvody pri komunikácii medzi modulmi využívajú tzv. „*handshaking*“[1]. Ide o kontrolu pomocou synchronizačných impulzov medzi jednotlivými časťami systému, kde sú dané synchronizačné signály závislé na stave samotnej časti, respektíve na hodnote vstupných signálov. Asynchrónne sekvenčné obvody pracujú rýchlejšie a majú nižšiu spotrebu pri rovnakej funkčnosti ako synchrónne sekvenčné obvody.

Diplomová práca sa zaoberá metódami návrhu asynchrónnych sekvenčných obvodov, výberom a opisom vhodnej metódy a následne návrhom programového systému, pozostávajúceho z obslužného programu, jednoprocesorového programu a programu pre vysokovýkonný výpočtový systém. Implementácia realizuje návrh vnútorného kódu pamäťovej časti, ako aj návrh a reprezentáciu kombinačných častí obvodu. Riešenie sa zameriava na návrh optimálneho digitálneho obvodu vzhľadom na rýchlosť a veľkosť pamäťovej časti, ako aj na automatizáciu návrhu obvodu.

Práca sa skladá z ôsmich častí. V prvých dvoch kapitolách sú zadefinované základné pojmy a analýza metód návrhu asynchrónnych sekvenčných obvodov. Špecifikácia požiadaviek na navrhovaný programový systém je uvedená v tretej kapitole. Štvrtá kapitola obsahuje navrhnutý formálny opis použitej metódy návrhu pamäťovej časti. Návrh programového systému skladajúci sa z troch častí je opísaný v piatej kapitole. V nasledujúcej kapitole je uvedená implementácia navrhnutej metódy, ktorá používateľovi ponúka prostriedok pre návrh vnútorného kódu a pre návrh budiacich funkcií asynchrónneho sekvenčného automatu s vyhodnotením optimálnosti podľa rôznych parametrov (rýchlosť, plocha, spotreba). Pri implementácii je potrebný vysoký výpočtový výkon, a preto je implementácia realizovaná aj v prostredí vysokovýkonného výpočtového systému. Dosiahnuté výsledky sú porovnané s výsledkami realizácie na jednojadrovom výpočtovom systéme v kapitole sedem. V poslednej kapitole je vyhodnotenie dosiahnutých výsledkov.

2 Analýza návrhu asynchrónnych sekvenčných obvodov

Analýza opisuje oblasť návrhu asynchrónnych sekvenčných obvodov, definuje základné pojmy, uvádza existujúce riešenia a existujúce metódy návrhu asynchrónnych sekvenčných obvodov.

2.1 Základné pojmy

V kapitole základných pojmov sú uvedené základné definície pre pochopenie fungovania asynchrónnych sekvenčných obvodov.

2.1.1 Charakteristika fundamentálnych automatov

Definícia 2.1
Konečný stavový automat **A** je usporiadaná pätica [3]:
A = (X, S, Y, δ, λ),

- X - konečná neprázdna množina vstupov do automatu **A**.
- S - konečná neprázdna množina stavov S_1, S_2, ..., S_R automatu **A**.
- Y - konečná neprázdna množina výstupov z automatu **A.**
- δ - prechodová (vnútorná) funkcia automatu **A,** pre ktorú platí: δ: S x X → S
- λ - výstupná funkcia automatu **A,** pre ktorú platí: S x X → Y (automat typu Mealy), λ': S → Y (automat Moore).

Automat typu Moore určuje výstupné hodnoty na základe aktuálneho stavu, v ktorom sa automat nachádza. Automat typu Mealy určuje výstupné hodnoty na základe aktuálneho stavu, v ktorom sa automat nachádza a na základe vstupu, idúceho do automatu.
Medzi automatom typu Moore a automatom typu Mealy je možné urobiť konverziu [6].

Konečný automat, ktorý neobsahuje žiadnu slučku, je formulovaný pomocou fundamentálneho automatu (Finite State Machine – FSM).

Definícia 2.2 [3]
Automat A je fundamentálny automat, ak prechody daného automatu spĺňajú podmienku: Každé vstupné slovo X, aplikované v ktoromkoľvek stave S, má za následok:

a. Priamy prechod do vnútorného stavu S, ktorý je vzhľadom k X stabilný.
b. Sériu prechodov cez nestabilné stavy, ktorá končí vo vnútornom stave S, stabilnom vzhľadom k X.

Definícia 2.3 [3]
Fundamentálny automat je automatom n- tého rádu, ak najdlhšia séria prechodov medzi stavmi je dlhá n prechodov.

Ak sa vo fundamentálnom automate nachádzajú len priame prechody, čiže prechody s dĺžkou jedna, tak sa automat nazýva fundamentálnym automatom prvého rádu.

Automaty vyšších rádov majú vyššie oneskorenie, ktoré je možné vypočítať podľa vzťahu (2.1).

$$f_{\max} = \frac{1}{n.(T_{mokc} + T_{mopc})}, \text{ kde} \tag{2.1}$$

$f_{\max}$ - oneskorenie pri prechode medzi dvoma stabilnými stavmi,
n - rád fundamentálneho automatu,
T_{mokc} - maximálne oneskorenie kombinačnej časti,
T_{mopc} - maximálne oneskorenie pamäťovej časti.

2.1.2 Charakteristika asynchrónnych sekvenčných systémov

Asynchrónne sekvenčné systémy pri vykonávaní funkcie nie sú synchronizované špeciálnou synchronizačnou premennou. V praktickom návrhu majú jednotlivé časti obvodu určité oneskorenia, dokonca sa do obvodu pridávajú oneskorovacie členy pre zabezpečenie očakávaného správania sa systému, čo má za následok spomalenie výpočtovej rýchlosti logického systému [5].

Asynchrónne sekvenčné systémy majú oproti synchrónnym sekvenčným systémom viaceré výhody:

a. Asynchrónne systémy sú všeobecnejšie ako synchrónne. To znamená, že synchrónne systémy možno reprezentovať asynchrónnymi systémami prostredníctvom zavedenia synchronizačnej premennej do vstupu asynchrónneho automatu. Synchrónne sekvenčné systémy sú podmnožinou asynchrónnych sekvenčných systémov. Asynchrónnymi sekvenčnými systémami je možné vykonávať rovnaké funkcie ako pri synchrónnych sekvenčných systémoch, čo však naopak neplatí [5].

b. Asynchrónne sekvenčné systémy môžu dosahovať vyššie rýchlosti pri vykonávaní svojej funkcie, a to z dôvodu neobmedzenia ich rýchlosti hodinovým signálom. Práca obvodu je pri synchrónnych sekvenčných obvodoch možná len pri príchode nábežnej hrany hodinového signálu a počas jeho stabilnej hodnoty je synchrónny obvod necitlivý na zmenu vstupov. Asynchrónne obvody reagujú hneď na zmenu vstupných premenných a ich citlivosť je stála. Z tohto dôvodu nie je potrebné dodržiavať šírku vstupných impulzov.
   Najvyššia rýchlosť práce obvodu je pri synchrónnych sekvenčných obvodoch obmedzená šírkou hodinového signálu. Asynchrónne obvody majú rýchlosť takú, akú im umožňuje fyzická realizácia obvodov.

c. Asynchrónne sekvenčné systémy môžu vykonávať funkciu aj tam, kde je privedenie hodinového signálu z pohľadu realizácie náročné alebo drahé. Z tohto dôvodu môžu byť malé asynchrónne obvody lacnejšie ako malé synchrónne obvody.

2.2 Návrh a reprezentácia vnútorného kódu

Táto časť sa venuje spôsobom opisu automatu a ich vnútorného kódu. Následne sú opísané základné možnosti návrhu vnútorného kódu.

2.2.1 Zápisy automatov

Reprezentácia automatov má za úlohu vizualizovať fungovanie automatu a uľahčiť automatizovaný návrh.

a. Prechodová tabuľka

Prechodová tabuľka je zobrazenie automatu, reprezentujúce stavy automatu, vstupy automatu a výstupy automatu. V tabuľke sú uvedené prechodové funkcie a výstupné funkcie pomocou vymenovania možností, čo sa pri určitých vstupoch pre aktuálny stav udeje.

Automat typu Mealy (tabuľka 2.1) je reprezentovaný tabuľkou opisujúcou stavy, vstupné premenné a výstupné premenné. Výstupná funkcia je závislá nielen od aktuálneho stavu, ale aj od vstupných premenných.

Tabuľka 2.1 Automat typu Mealy

	X		**Y**	
	0	**1**	**X=0**	**X=1**
A	A	B	0	0
B	A	C	0	0
C	A	D	0	0
D	E	D	0	0
E	F	B	1	0
F	A	B	0	0

Tabuľka automatu typu Moore (tabuľka 2.2) reprezentuje stavy automatu, vstupné premenné a výstupné premenné. Výstupná funkcia je závislá len na aktuálnom stave, v ktorom sa automat nachádza.

Tabuľka 2.2 Automat typu Moore

	X=0	X=1	Y
A	A	B	0
B	A	C	0
C	A	D	0
D	E	D	0
E	F	B	0
F	A	B	1

Reprezentácia pomocou tabuľky je vhodná pre automatizovaný návrh, lebo sa jednoducho implementuje.

b. Prechodový graf

Prechodový graf je grafické znázornenie prechodového automatu [5].

2.2.2 Vnútorný kód a jeho reprezentácia

Vnútorný kód v dvojkovej sústave je možné reprezentovať pomocou kódovacej tabuľky, v ktorej sa nachádzajú kódy pre každý stav použitý v automate. Z kódovacej tabuľky a opisu automatu je možné vytvoriť reprezentáciu automatu pomocou logických obvodov. Sú to hlavné nástroje pri návrhu pamäťovej časti asynchrónnych sekvenčných obvodov a kombinačnej logiky, ktorá je k nej pridružená.

2.2.3 Požiadavky na vnútorný kód

Navrhnutý vnútorný kód musí spĺňať požiadavky v prvom rade na spoľahlivosť, v druhom rade na rýchlosť. Typy prechodov medzi dvoma stavmi, ktoré majú svoj vnútorný kód, sú nasledovné:

a. Bezsúbehový prechod - je priamy prechod, stavové premenné pôvodného stavu a cieľového stavu sa menia práve v jednom bite, čiže ich Hamingova vzdialenosť sa rovná jednej [3].

b. Súbehový prechod - nastáva pri prechode z pôvodného do cieľového stavu, keď sa mení viac ako jedna stavová premenná. Hamingova vzdialenosť kódu pôvodného stavu a kódu cieľového stavu je väčšia ako jedna. Súbehový prechod môže byť kritický a nekritický. Nekritický súbehový prechod pre každé poradie zmeny vnútorného kódu vedie do rovnakého stavu, teda prechod je pre každú možnosť úspešný. Pri kritickom súbehu existuje možnosť zmeny stavových premenných tak, aby prechod nebol úspešný.
V tabuľke 2.3 je určená reprezentácia stavov. Nachádzajú sa v nej dva súbehy. Prvý 3 → 1 a druhý 2 → 4.

Tabuľka 2.3 Súbehový automat

	X_1, X_2					
S	**00**	**10**	**11**	**01**	Y	z_1,z_2
1	**1**	4	**1**	**1**	0	00
2	1	**2**	**2**	4	0	01
3	**3**	2	1	**3**	0	11
4	3	**4**	1	**4**	1	10

Pre prechod 3 → 1, pre vstup 11 sa menia obidve stavové premenné súčasti, a to z 11 do 00. Zmena stavových premenných musí prísť v sekvencii za sebou. Existujú práve dve možnosti týchto sekvencií:

1. z_2 sa mení z 1 na 0, obvod prechádza do stavu 4. Avšak pri vstupe 11 aj stav 4 prechádza do stavu 2, takže výsledný stav je zhodný s požadovaným. Ale pri prechode vznikla prechodná chyba. Výstupná premenná mala chvíľu hodnotu jedna.

2. z_1 sa mení z 1 na 0 a obvod prechádza do stavu 2, ale stav dva je pri vstupe 11 stabilný. Prechod teda nie je úspešný.

Prechod 3 → 1 je kritický, preto vnútorný kód nie je vhodný pre návrh kombinačnej časti. Je potrebné zvoliť iné zakódovanie stavov.

Dĺžka najdlhšieho nekritického súbehu určuje maximálne oneskorenie v obvode (T_{max}), ktoré je dané vzťahom (2.2).

$$T_{max} = n \,.\, T_{P,} \tag{2.2}$$

kde n je maximálna dĺžka súbehu a T_P je čas, za ktorý sa vykoná jeden prechod.

Ak pri návrhu zakódovania pamäťovej časti sú všetky prechody bezsúbehové alebo nekriticky súbehové, je možné vytvoriť budiace funkcie bez hazardov. Ak sú v kóde kritické súbehy, je potrebné zvoliť iné zakódovanie stavov automatu.

2.2.4 Návrh vnútorného kódu

Návrh a výber vnútorného kódu je algoritmicky náročný proces. Pre jeho výber neexistuje algoritmus, ktorý by poskytol bez otestovania veľa možností a optimálne výsledky, vzhľadom na počet stavových premenných automatu alebo na počet premenných v kombinačnej časti obvodu. Sú známe dva hlavné prístupy vytvárania vnútorného kódu stavov automatu, ktoré sú ďalej opísané.

a. Priama realizácia automatu

Priamou realizáciou zakódovania stavov automatu sa nazýva funkcia f: $S \rightarrow \{0,1\}^k$, ktorá priradí každému stavu príslušné zakódovanie. Zakódovania sú pre každý stav rozdielne. Zároveň platí, že pri vytváraní ekvivalentného automatu sa použije binárny kód pre zakódovanie stavov, ktorý je bezsúbehový, alebo ide o kód bez kritických súbehov. Ak navrhnuté zakódovanie vnútorných stavov automatu spĺňa tieto požiadavky, hovoríme o priamom kóde [3].
Podmienky, ktoré musí spĺňať postup vytvárania vnútorného kódu, sú nasledovné:

Pre konštrukciu priamych kódov musí byť splnené pravidlo 2.1 [3].

Pravidlo 2.1

Ak je prechod $s_i \rightarrow s_j$ s dĺžkou väčšou ako jedna, tak každý stav, do ktorého sa obvod môže pri prechode dostať, musí určiť cieľový stav prechodu, ktorý je zhodný s s_j.

Takýto prechod nazývame nekritickým súbehom. Ak prechod nespĺňa pravidlo 2.1, nazývame ho kritickým súbehom.

b. Nepriama realizácia automatu [3].

Nepriama realizácia automatu je založená na vytvorení nového automatu, ktorého výstupy sú rovnaké ako v pôvodnom automate, ale realizácia vnútorných stavov je odlišná. Kritické súbehy sú odstránené nie aplikovaním pravidla 2.1, ale zmenou prechodov medzi stavmi v automate. Keďže sú menené samotné prechody, rád automatu sa môže meniť. Ak rád automatu rastie, jeho rýchlosť klesá.

2.2.5 Multikódy

Multikódy sú univerzálne kódy pre určitý počet stavov. Zaručujú, že pri určitom počte stavových premenných a vybranom spôsobe zakódovania sa v zakódovaní nevyskytujú kritické súbehy. Multikódy neposkytujú optimálne riešenie vzhľadom na veľkosť obvodu. Niektoré známe typy multikódov sú:

- univerzálny kód Saucier
- Hammingov multikód
- univerzálny multikód

2.3 Návrh a reprezentácia kombinačnej časti

V nasledujúcej časti bude opísaná kombinačná časť asynchrónnych sekvenčných obvodov. Keďže na rozdiel od synchrónnych obvodov sa zmeny vstupov spracúvajú stále, môžu v obvode vzniknúť neželané stavy. Tieto situácie sa nazývajú hazardy.

2.3.1 Hazardy

Každá časť obvodu má určité oneskorenie vplývajúce na celkové správanie obvodu a ovplyvňuje nielen prechody medzi stavmi, ale aj výstupné premenné. Chyby, ktoré môžu nastať v obvode, sú rôzne. Obvod sa nemusí dostať do želaného stavu. Výstupné premenné nadobúdajú hodnotu, ktorú nemajú, alebo výstupná hodnota nie je stabilná. Obvod, ktorý sa nespráva podľa očakávania a nachádza sa v ňom hazard, nazývame hazardným obvodom [3].

Rozlišujú sa tieto štyri typy hazardov:

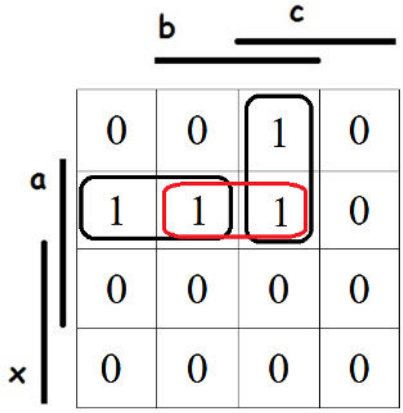
a. Statický hazard [3].

Ak $f(a_i) = f(a_j) = 0$ a pri zmene vstupu $a_i \rightarrow a_j$ sa na výstupe obvodu zmení hodnota aspoň raz na 1, ide o statickú chybu v hodnote 0 respektíve o B-hazard-0.

Ak $f(a_i) = f(a_j) = 1$ a pri zmene vstupu $a_i \rightarrow a_j$ sa na výstupe obvodu zmení hodnota aspoň raz na 0, ide o statickú chybu v hodnote1 respektíve o B-hazard-1.

Statické hazardy sú spôsobené realizáciou pamäťovej časti asynchrónneho sekvenčného obvodu. Pri použití preklápacích obvodov typu JK nie je potrebné ošetrovať statické hazardy, lebo realizácia preklápacích obvodov JK vylučuje výskyt statického hazardu v obvode.

Pri použití preklápacích obvodov SR nie je potrebné riešiť B-hazardy-0, pretože preklápacie obvody ST sú voči B-hazardu-0 imúnne. Je však potrebné vytvoriť budiace funkcie, ktoré sú imúnne voči B-hazardu-1. Bezhazardná minimalizácia pozostáva v pokrytí všetkých možných susedov v Karnaughovej mape. Na obrázku 2.4 je uvedená bezhazardná minimalizácia. Počet logických členov v budiacej funkcii síce stúpol, ale oneskorenia sú pre všetky možné prechody rovnaké. Pri prechode medzi stavmi teda nedôjde k neželanému výstupu.



Obrázok 2.1 Bezhazardná minimalizácia

b. Dynamický hazard [3].

Ak $f(a_i) \neq f(a_j)$ a pri zmene vstupu $a_i \rightarrow a_j$ sa výstup z automatu zmení viac ako raz, tak sa v obvode nachádzajú dynamické hazardy. Tento hazard je spôsobený len pri

dĺžke prechodu väčšej ako dva. Ošetrenie tohto hazardu je založené na vložení oneskorenia do výstupnej logiky.

c. Podstatný hazard.

Podstatný hazard je definovaný v definícii 2.4 [2].

Definícia 2.4

Nech A = (X, S, Y, δ, λ) je fundamentálny automat 1. rádu. Potom v automate A existuje podstatný hazard, ak pri prechode stavov $s_1 \rightarrow s_2$, nastane zmena hodnoty na vstupe a následne platí $\delta(s_1, a_2\ a_1\ a_2) \neq \delta(s_1, a_2) = s_2$.

V tabuľke 2.4 je podstatný hazard ak je stav 1 zakódovaný kódom 00 a stav 2 je zakódovaný kódom 11. V prechode 1 → 2 po príchode 1 je potrebné, aby obvod prešiel do stavu 2. Ale ak príde kombinácia (s_1,101), obvod prejde do neželaného stavu 4.

Tabuľka 2.4 Podstatný hazard

	0	**1**
1	1	2
2	3	2
3	3	4
4	1	4

Návrh odstránenia podstatného hazardu je opísaný v [3].

d. Funkčný hazard

Funkčný hazard nastáva v prípade, ak sa majú súčasne zmeniť hodnoty na dvoch výstupných premenných, čo pri realizácii nie je možné. Funkčný hazard je chyba zadania automatu. Je možné ju zistiť kontrolou všetkých prechodov v automate. Ak pre všetky prechody v automate platí, že výstupné hodnoty pre počiatočný stav prechodu sa menia maximálne v jednej hodnote oproti výstupným hodnotám v cieľovej časti prechodu, tak automat nemá funkčné hazardy.

Hazardy je možné ošetriť pomocou zmeny kódovania stavov, alebo v zmene kombinačnej časti, pričom požadovaná funkčnosť musí byť zachovaná. Rozšírenou možnosťou je aj pridávanie oneskorení na výstup obvodu. Toto riešenie ale má negatívny dopad na rýchlosť obvodu, avšak znižuje nielen zložitosť návrhu, ale aj náklady na výrobu obvodu.

2.3.2 Návrh vnútorného kódu

Návrh vnútorného kódu je realizovaný pomocou algoritmu využívajúceho rozklad stavov [2]. Po vytvorení rozkladov stavov sa zistí pokrytie jednotlivých rozkladov a z množín, ktoré pokrývajú všetky stavy, je vytvorený vnútorný kód.

Návrh priameho kódu pozostáva z vytvorenia dvojblokového rozkladu pre každý možný vstup z automatu **A**, v ktorom je opísaná množina stavov **S.** Dvojblokový rozklad, inak nazývaný dichotómia, je dvojica množín, v ktorých sú stavy z množiny **S.** V jednom bloku sú stavy, ktoré majú pre stavovú premennú logickú hodnotu nula alebo jedna. Stavy v druhej množine dichotómie majú pre rovnakú stavovú premennú opačné logické hodnoty. Podmienky pre priamy kód pomocou rozkladov sú definované takto[2]:

1. Pre všetky dvojice prechodov $s_i \rightarrow s_j$, $s_k \rightarrow s_l$, pričom i, j, k, l $\in$(1, n), kde n je počet stavov a platí $s_j \neq s_l$ pri vstupe **a** existuje aspoň jeden rozklad $\tau \geq (s_i, s_j; s_k, s_l)$.
2. Pre každý prechod $s_i \rightarrow s_j$ a stabilný stav s_k pričom i, j, k $\in$(1, n), pri vstupe **a** existuje aspoň jeden rozklad $\tau \geq (s_i, s_j; s_k)$.
3. Pre každú dvojicu stabilných stavov s_i s_j pričom i, j $\in$(1, n), pre ktoré platí $s_i \neq s_j$ existuje aspoň jeden rozklad $\tau \geq (s_i; s_j)$.

Nájdenie priameho kódu pre automat **A** s prechodovou funkciou **p** prebieha v nasledujúcich častiach:

1. Pre každý možný vstup **a** automatu **A** je vytvorený zoznam všetkých dichotómií.
2. Po zlúčení všetkých zoznamov z časti jedna sa vytvorí minimálna množina dichotómií.

3. Z minimálnej množiny dichotómií je vytvorená množina úplných dichotómií.
4. Je vybratá časť z množiny úplných dichotómií, ktorá pokrýva všetky dichotómie zo zoznamu minimálnych dichotómií.

V tretej časti riešenia sa použije 2.2 o spájaní dichotómií [2]:

Pravidlo 2.2

Nech $\pi_1 = (Q_i; Q_j)$ a $\pi_2 = (Q_k; Q_l)$ sú dichotómie, kde Q je množina stavov a Q_i, Q_j Q_k, $Q_l \in Q$, potom platí:

- $\pi_1 + \pi_2 = \{Q_i \cup Q_j ; Q_k \cup Q_l\}$, ak $(Q_j \cap Q_k = 0) \wedge (Q_i \cap Q_l = 0)$,
- $\pi_1 + \pi_2 = \{Q_i \cup Q_l ; Q_j \cup Q_k\}$, ak $(Q_j \cap Q_l = 0) \wedge (Q_i \cap Q_k = 0)$.

Aplikácia pravidla 2.2 je uvedená na príklade 2.1.

Príklad 2.1. Automat, pre ktorý sa má navrhnúť priamy kód, je zadaný v tabuľke 2.5. [5]

Tabuľka 2.5 Zadanie automatu pre návrh vnútorného kódu

	X_1, X_2				
S	**00**	**10**	**11**	**01**	**Y**
1	**1**	4	**1**	**1**	0
2	1	**2**	5	5	0
3	**3**	2	1	**3**	0
4	3	**4**	**4**	**4**	0
5	**5**	4	**5**	**5**	1

1. Zostavenie zoznamu dichotómií.
   a. Pri vstupe $(X_1, X_2)=(0, 0)$ existujú prechody $2 \rightarrow 1$, $4 \rightarrow 3$, a stabilný stav 5. Dichotómie sa vytvoria kombináciou týchto prvkov. Sú to:
      $\pi_1=\{1,2 ; 3,4\}$,
      $\pi_2=\{1,2 ; 5\}$,
      $\pi_3=\{3,4 ; 5\}$.

   b. Pri vstupe $(X_1, X_2)=(1,0)$ existuje trojica prechodov $1 \rightarrow 4,5 \rightarrow 4$, $3 \rightarrow 2$. Keďže prvá dvojica prechodov má spoločný stav, treba pokryť dichotómie:
      $\pi_4=\{1,4 ; 2,3\}$,
      $\pi_5=\{4,5 ; 2,3\}$.

c. Pri vstupe $(X_1, X_2)=(1,1)$ existujú prechody $3 \rightarrow 1$, $2 \rightarrow 5$ a stabilný stav 4. Dichotómie sa vytvoria kombináciou týchto prvkov. Sú to:
$\pi_6=\{1,3 ; 2,5\}$,
$\pi_7=\{1,3 ; 4\}$,
$\pi_8=\{2,5 ; 4\}$.

d. Pri vstupe $(X_1, X_2)=(0,1)$ existuje jeden prechod $2 \rightarrow 5$ a trojica stabilných stavov 1,3,4. Kombináciou sa vygeneruje nasledujúca množinu prvkov:
$\pi_9=\{1 ; 2,5\} <\pi_6$,
$\pi_{10}=\{1 ; 3\} <\pi_1,\pi_4$,
$\pi_{11}=\{1 ; 4\} <\pi_1,\pi_7$,
$\pi_{12}=\{3 ; 2,5\} <\pi_6$,
$\pi_{13}=\{4 ; 2,5\} =\pi_8$,
$\pi_{14}=\{3 ; 4\} <\pi_4, \pi_5, \pi_7$.

2. Výber minimálneho zoznamu dichotómií. Keďže dichotómie π_9 až π_{14} sú už pokryté, nie je potrebné ich vkladať do minimálneho zoznamu dichotómií. Ostatné dichotómie sú vložené do minimálneho zoznamu dichotómií.

3. Aplikovaním pravidla 2.2 vznikne množina minimálnych rozkladov, kde je každý prvok pokrytý aspoň jedenkrát. Po aplikovaní tohto pravidla vznikne množina $\{\tau_1,\tau_2, \ldots.\tau_8\}$, kde
$\tau_1 = \pi_1 + \pi_2 = \{1,2 ; 3,4,5\} > \pi_1,\pi_2$,
$\tau_2 = \pi_2 + \pi_3 = \{5 ; 1,2,3,4\} > \pi_2,\pi_3$,
$\tau_3 = \pi_3 + \pi_4 = \{2,5 ; 1,3,4\} > \pi_3, \pi_4,\pi_7$,
$\tau_4 = \pi_4 + \pi_5 = \{1,4 ; 2,3,5\} > \pi_4, \pi_5$,
$\tau_5 = \pi_5 + \pi_6 = \{2,3 ; 1,4,5\} > \pi_5, \pi_6$,
$\tau_6 = \pi_6 + \pi_8 = \{1,2,3 ; 4,5\} > \pi_6,\pi_8,\pi_2$,
$\tau_7 = \pi_7 + \pi_8 = \{1,3 ; 2,4,5\} > \pi_7, \pi_8$,
$\tau_8 = \pi_8 + \pi_2 = \{4,5 ; 1,2,3\} > \pi_8, \pi_2, \pi_6$.

4. Množina $\{\tau_1,\tau_2, \ldots.\tau_8\}$ pokrýva všetky dichotómie z minimálneho zoznamu dichotómií, a tak aj všetky dichotómie. Generuje kód ôsmych stavových premenných,

ale je možné vytvoriť minimálny výber τ, kde bude každá dichotómia z minimálneho zoznamu dichotómií pokrytá aspoň jedenkrát. Použité dichotómie sú: τ_3, τ_6, τ_1, τ_4.

5. V piatej časti je vytvorené zakódovanie stavov podľa vytvorenej množiny úplných dichotómií. Pri vytváraní kódu je možné zvoliť, ktorá časť dichotómie bude reprezentovaná v kóde logickou jednotkou, a ktorá časť bude reprezentovaná logickou nulou. Toto rozhodnutie je možné urobiť pre každú úplnú dichotómiu, čo nám dáva 16 rozdielnych vnútorných kódov. Použité dichotómie a kódy stavov sú uvedené v tabuľke 2.6.

   Tabuľka 2.6 Možný vnútorný kód

	τ_3	τ_6	τ_1	τ_4
1	0	1	1	1
2	1	1	1	0
3	0	1	0	0
4	0	0	0	1
5	1	0	0	0

6. V poslednom kroku sa doručí prechodová funkcia. Sú vytvorené záchytné stavy tak, aby bola ošetrená každá možná zmena stavových premenných. Výstupom tejto časti je nový automat, rozšírený o nové záchytné stavy.

2.4 Analýza existujúcich riešení

V problémovej oblasti boli nájdené tri implementované metódy, ktoré sa snažia softvérovo riešiť problematiku návrhu asynchrónnych sekvenčných obvodov. Ani jedna implementácia nerieši optimálnosť riešenia, ani výpočet na vysokovýkonnom výpočtovom systéme. Veľký problém pri rozvíjaní tejto oblasti je v zameraní sa na synchrónne sekvenčné systémy, keďže ich návrh je oveľa jednoduchší, a aj časovo menej náročný. Samotná realizácia je lacnejšia, lebo používa menej logických členov. Avšak asynchrónne sekvenčné obvody pri správnom návrhu ponúkajú tzv. maximálnu rýchlosť, kde oneskorenie je určené len fyzickou realizáciu logický členov. Rýchlosť nie je obmedzená synchronizačnou premennou.

2.4.1 Metóda návrhu s využitím multikódu

Softvérové riešenie, ktoré aspoň z časti rieši danú problematiku, je návrh pomocou multikódov [4]. Riešenie prebieha cez tvorbu multikódu, presnejšie Hammingového multikódu. Celá práca sa snaží riešiť len návrh vnútorného kódu, návrh budiacich funkcií nerieši. Preto ako komplexnejšie riešenie je daná práca nepoužiteľná. Program je prehľadný, javí sa ako fungujúci pri návrhu Hammingového multikódu.

2.4.2 Metóda návrhu pomocou priameho kódu

Metóda rieši návrh vnútorného obvodu pomocou priameho kódu [5]. Použitý algoritmus negeneruje optimálny vnútorný kód, ale generuje tabuľku rozkladov, ktorá sa dá využiť pri tvorbe optimálneho kódu. Navrhnutý programový systém rieši návrh kombinačnej časti len čiastočne a neošetruje všetky možné hazardy. Taktiež nie sú implementované algoritmy na odstránenie všetkých hazardov. Algoritmus pre generovanie vnútorného kódu bude využitý v navrhovanom prístupe, a to v jeho úvodnej fáze.

2.4.3 Metóda návrhu pomocou C - blokov

C - blok je logický obvod s preddefinovanou funkciou. Návrh pomocou C - blokov je vhodný pre FPGA. Systémom, ktorý pracuje s C - blokmi je systém Balsa. Súčasťou systému Balsa je systém Petrify [7], ktorý rieši návrh asynchrónnych sekvenčných obvodov, zadaných pomocou Petriho sietí. Taktiež neposkytuje optimálnu realizáciu asynchrónneho sekvenčného obvodu, ale rieši odstránenie hazardov v kombinačnej časti, ako aj návrh pamäťovej časti. Keďže tento systém využíva algoritmus založený na kombinácii C – blokov, nie je možné ho využiť v navrhovanom prístupe.

2.5 Zhodnotenie metód návrhu

Opísané metódy návrhu asynchrónnych sekvenčných obvodov majú svoje výhody aj nevýhody. Návrh pomocou multikódov je vhodný len pre obvody, ktoré majú malý počet vstupov a malý počet stavov. Pre veľké obvody ponúka návrh pomocou priameho kódu oveľa lepšie riešenia, čiže počet obvodov vo fyzickej realizácii asynchrónneho sekvenčného obvodu je rádovo nižší. Výhodou návrhu pomocou multikódov je rýchlosť obvodu. Keďže maximálna

dĺžka prechodu po zakódovaní je jedna, tak oneskorenie v obvode je minimálne, lebo na prechod do druhého stavu je potrebná zmena len jednej stavovej premennej. V prípade priameho kódu je maximálna dĺžka prechodu dve preklopenia. Zároveň redukcia použitých obvodov je výrazná.

Návrh vnútorného kódu je riešený pomocou priameho kódu, keďže dĺžka prechodu je maximálne dve preklopenia preklápacích obvodov a veľkosť pamäťovej časti je menší, ako pri návrhu vnútorného kódu pomocou multikódov.

3 Špecifikácia

Cieľom diplomovej práce je vytvoriť nový softvérový prostriedok, ktorý zostaví optimálny vnútorný kód vzhľadom na počet preklápacích obvodov v pamäťovej časti a vzhľadom na dĺžku prechodu medzi dvoma príslušnými stavmi. Softvérový systém má poskytovať možnosť overenia, či je daný vnútorný kód v zhode s požadovaným automatom.

3.1 Opis funkčnosti programového systému

Navrhnutý a implementovaný softvérový systém je rozdelený na dve časti. Prvou je obslužný program, ktorý načítava vstupy a zabezpečuje komunikáciu s vysokovýkonným výpočtovým systémom. Druhou časťou je program, ktorý sa vykonáva na vysokovýkonnom výpočtovom systéme.

Obslužný program poskytuje možnosť vložiť zadanie automatu pomocou súboru, vo formáte VHDL, alebo vo formáte definovanom pre prácu na vysokovýkonnom výpočtovom systéme. Automat vo formáte VHDL je možné opísať pomocou softvérového návrhového prostriedku HDL Designer. Vnútorný kód je vypočítaný na počítačovom systéme s jedným jadrom, alebo na vysokovýkonnom výpočtovom systéme, kde je riešenie rozdelené medzi viaceré výpočtové uzly. Prechodové funkcie sú vypočítané z vytvoreného vnútorného kódu stavov automatu, a je možné výpočet spustiť na vysokovýkonnom výpočtovom systéme. Keďže počet možných vnútorných kódov a prechodových funkcií pre jeden automat je vysoký, je potrebné vypočítať počet logických členov pre každú možnosť vnútorného kódu a následne vybrať reprezentáciu automatu, ktorá potrebuje najmenej logických členov. Systém na základe budiacich funkcií a výstupných funkcií vytvorí opis obvodu v jazyku VHDL, ktorý je možné následne otestovať pomocou simulačného prostriedku ModelSim, ktorý je určený na testovanie navrhnutých logických obvodov.

3.2 Ohraničenie programového systému

Implementačné obmedzenia poskytujú potrebné požiadavky na vývoj systému, kde je vyvinutý programový systém prevádzkovaný. Funkčné obmedzenia uvádzajú ohraničenia

na vstupy do programového systému, ako aj na funkciu programového systému a jeho výstupy.

3.2.1 Implementačné obmedzenia

Softvérový systém je vyvíjaný na dvoch platformách. Prvá časť, ktorá spravuje vstupy do systému a zabezpečuje komunikáciu s vysokovýkonným prostriedkom, je vyvíjaná pod operačným systémom MS Windows 8. Obslužný program nemá vysoké požiadavky na výpočtový systém. Pre plnú funkčnosť obslužného programu postačuje bežne dostupný počítač s operačným systémom MS Windows 8. Je potrená aj prítomnosť ".*Net*" knižníc pre fungovanie obslužného programu. Pre prístup na vysokovýkonný výpočtový systém je potrebný prístup na internet a prístup ku samotnému vysokovýkonnému prostriedku cez protokol *"ssh"*.

Vysokovýkonný výpočtový systém musí byť pripojený na internet. Programové vybavenie vysokovýkonného výpočtového systému musí byť schopné komunikovať cez protokol *"ssh"* a musí zabezpečiť spúšťanie programov využívajúcich knižnicu *"mpi"*. Pre implementáciu vyvinutého návrhového systému bol využitý vysokovýkonný výpočtový systém vo výpočtovom stredisku Slovenskej akadémie vied v Bratislave. Vysokovýkonný výpočtový systém **Aurel** má nasledujúce parametre:

- Systém: IBM Power 775.
- Počet výpočtových jadier: 3072.
- Pamäť: 24TB.
- Výpočtová sieť: 5-48GB/s Internal Optical Links, 10GB/s Ethernet prepojenie s úložiskom dát.
- Kapacita externého úložiska dát: 600TB.
- Operačný systém: AIX.
- Komunikačné prostredie: MPI.

3.2.2 Funkčné obmedzenia

Implementácia navrhnutého systému bude mať tieto funkčné obmedzenia:

1. Vstupom obslužného programu je opis automatu, ktorý je vo formáte VHDL, alebo vo formáte určenom pre vysokovýkonný výpočtový systém. Opísaný automat je fundamentálny automat prvého stupňa a výstupy automatu neobsahujú funkčné hazardy.

2. Vstupom pre program, ktorý sa vykonáva na vysokovýkonnom výpočtovom systéme, je opis automatu v upravenom formáte, ktorý je v zjednodušenej podobe oproti opisu automatu vo formáte VHDL, a je vytvorený obslužným programom.

3. Po dokončení programu, ktorý sa vykonáva na vysokovýkonnom výpočtovom systéme, je výsledok uložený priamo v systéme. Ak je spustený obslužný program, výsledok sa posiela obslužnému programu.

4. Prechodové funkcie sú navrhnuté pre preklápacie obvody JK.

5. Výstupom obslužného programu je opis architektúry asynchrónneho digitálneho obvodu, ktorý reprezentuje zadaný automat, a je vo formáte VHDL.

4 Formálny opis navrhnutej metódy

Nech **A**= (X, S, Y, δ, λ) je konečný stavový automat podľa *definície 2.1,* **A** je tiež fundamentálny automat prvého rádu podľa *definície 2.3* a **Q** je neprázdna podmnožina stavov **s ⊆ S** automatu **A**. Potom dichotómia **π** je definovaná definíciou 4.1.

Definícia 4.1

Dichotómia π automatu **A** s množinou stavov **Q** je dvojprvková množina, ktorej prvky sú z Q, pričom platí:

- $\pi = \{Q_i ; Q_j\}, Q_i \cap Q_j = \varnothing$,
- $\pi = \{Q_i ; Q_j\} = \{Q_j ; Q_i\}$.

Definícia 4.2

Množina **D** je množina všetkých dichotómií π, ktoré sú vytvárané pomocou týchto troch pravidiel:

1. Ak existuje prechod zo stavu $\mathbf{s_i}$ do stavu $\mathbf{s_j}$ a zároveň existuje prechod zo stavu $\mathbf{s_k}$ do stavu $\mathbf{s_l}$ pri vstupnom slove **X**, potom dichotómia $\pi = \{\{\mathbf{s_i}, \mathbf{s_j}\}; \{\mathbf{s_k}, \mathbf{s_l}\}\}$je pridaná do množiny dichotómií **D**, pričom i, j, k, l sú ľubovoľné indexy stavov **s**, ktoré sú z množiny **Q**.
2. Ak existuje prechod zo stavu $\mathbf{s_i}$ do stavu $\mathbf{s_j}$, a zároveň existuje stabilný stav $\mathbf{s_k}$ pri vstupnom slove **X**, potom dichotómia $\pi = \{\{\mathbf{s_i}, \mathbf{s_j}\}; \{\mathbf{s_k}\}\}$ je pridaná do množiny dichotómií **D**, pričom i, j, k, l sú ľubovoľné indexy stavov **s** z množiny **Q**.
3. Ak existuje stabilný stav $\mathbf{s_i}$, a zároveň existuje stabilný stav $\mathbf{s_j}$ pri vstupnom slove **X**, potom dichotómia $\pi = \{\{\mathbf{s_i}\}; \{\mathbf{s_j}\}\}$ je pridaná do množiny dichotómií **D**, pričom i, j, sú ľubovoľné indexy stavov **s** z množiny **Q**.

Q je neprázdna množina stavov **s ⊆ S** z fundamentálneho automatu **A**.

Definícia 4.3

Množina **M ⊆ D** je minimálna množina dichotómií π, v ktorej je každý prvok jedinečný.

Pre **M =** $\{\pi_1,\pi_2 \ldots\ldots \pi_n\}$ platí:

- $\pi = \{\{s_i, s_j\}; \{s_k, s_l\}\} = \{\{s_j, s_i\}; \{s_k, s_l\}\} = \{\{s_i, s_j\}; \{s_l, s_k\}\} =$
  $= \{\{s_j, s_i\}; \{s_l, s_k\}\} = \{\{s_k, s_l\}; \{s_i, s_j\}\} = \{\{ s_l, s_k \}; \{ s_i, s_j \}\} =$
  $= \{\{ s_k, s_l \}; \{ s_j, s_i \}\} = \{\{ s_l, s_k \}; \{ s_j, s_i \}\},$
- $\pi = \{\{ s_i, s_j \}; \{ s_k \}\} = \{\{ s_j, s_i \}; \{ s_k \}\} = \{\{ s_k \}; \{ s_i, s_j \}\} = \{\{ s_k \}; \{ s_j, s_i \}\},$
- $\pi = \{\{ s_i \}; \{ s_k \}\} = \{\{ s_k \}; \{ s_i \}\}.$

s sú stavy z **S** fundamentálneho automatu **A**.

Definícia 4.4

Nech **C =** $\{(\tau, P)\}$ je množina, kde $\boldsymbol{\tau} = \{Q_i ; Q_j\}$je úplná dichotómia, pre ktorú platí $Q_i \cup Q_j = \mathbf{S}$, **S** - množina všetkých stavov automatu **A** a i, j, sú indexy dvoch podmnožín stavov z Q.

P je množina dichotómií π, pričom $\pi \in$ **M,** ktoré pokrýva dichotómia **τ**. Dichotómia **τ**, je vytvorená pomocou *pravidla 2.1,* z **M**.

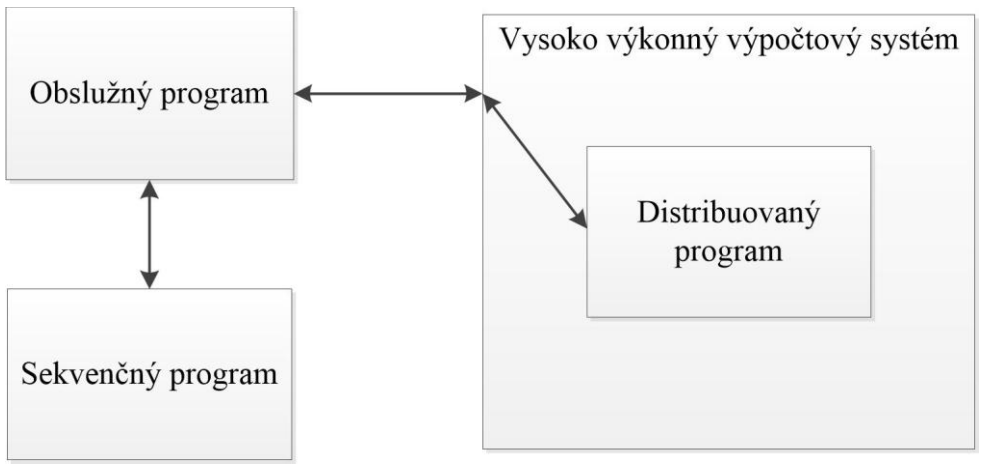
Definícia 4.5

Nech **T =** $\{(\tau, P)\}$ je množina, pre ktorú platí **T** $\subseteq$ **C** a zároveň $\bigcup_{i=1}^{n} P_i = \mathbf{M}$ pričom **n** je počet dvojíc (τ, P) v **T, M** je minimálna množina dichotómií a **C** je množina úplných dichotómií, potom množina **T** je množina dvojíc, z ktorej sa prepisom vytvorí zakódovanie stavov.

5 Návrh programového systému

Návrh implementácie návrhového programového systému je rozdelený do troch hlavných častí. V prvej časti je uvedený návrh sekvenčného algoritmu, ktorý sa vykonáva na jednoprocesorovom systéme. V nasledujúcej časti je uvedený návrh algoritmu, ktorý je vykonávaný na vysokovýkonnom výpočtovom systéme. V poslednej tretej časti je uvedený návrh obslužného programu, ktorý spracováva vstupy a výstupy pre program vykonávaný na vysokovýkonnom výpočtovom systéme, a taktiež komunikuje s týmto systémom.

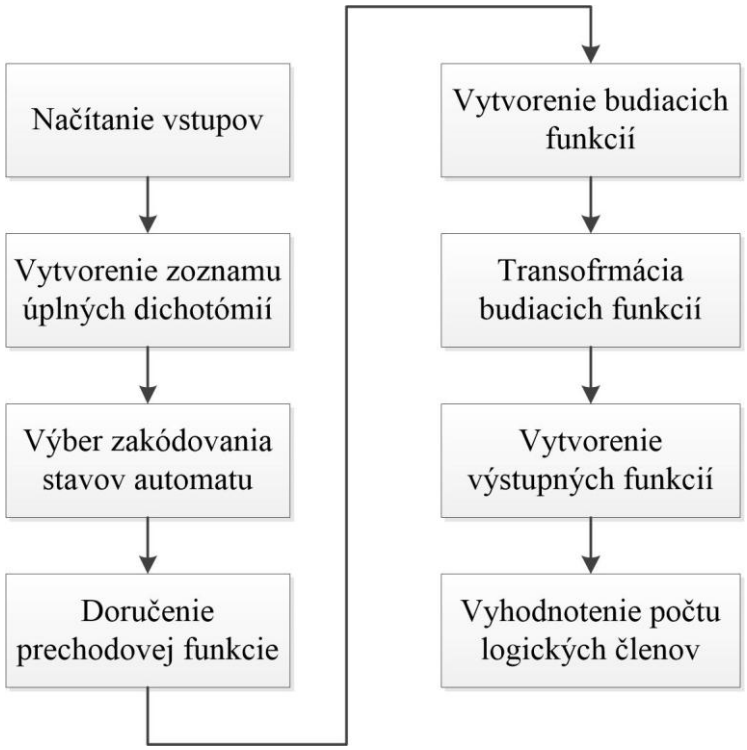
Na obrázku 5.1 je uvedená architektúra celého programového systému. Zahŕňa výpočet sekvenčným algoritmom, výpočet na distribuovanom systéme a obslužný program, ktorý spracováva vstupy a výstupy.



Obrázok 5.1 Architektúra programového systému

5.1 Návrh pre jednoprocesorový systém

Programový prostriedok pre návrh asynchrónnych sekvenčných obvodov implementuje algoritmus, ktorý sa skladá zo ôsmich hlavných častí ako je ukázané na obrázku 5.2.



Obrázok 5.2 Algoritmus pre návrh asynchrónnych sekvenčných obvodov

5.1.1 Načítanie vstupov

Prvá časť algoritmu má za úlohu načítať vstupné hodnoty (opis automatu) zo súboru, ktorý bude zadaný v preddefinovanom formáte. Vo vstupnom súbore je opísaný fundamentálny automat prvého stupňa typu **Moore**, ktorý musí byť bez funkčných hazardov. Vstupom je textový súbor a výstupom je reprezentácia automatu v programe. Súčasťou načítania vstupu je aj kontrola automatu, ktorá určí, či sú pre všetky možné vstupy zadefinované cieľové stavy.

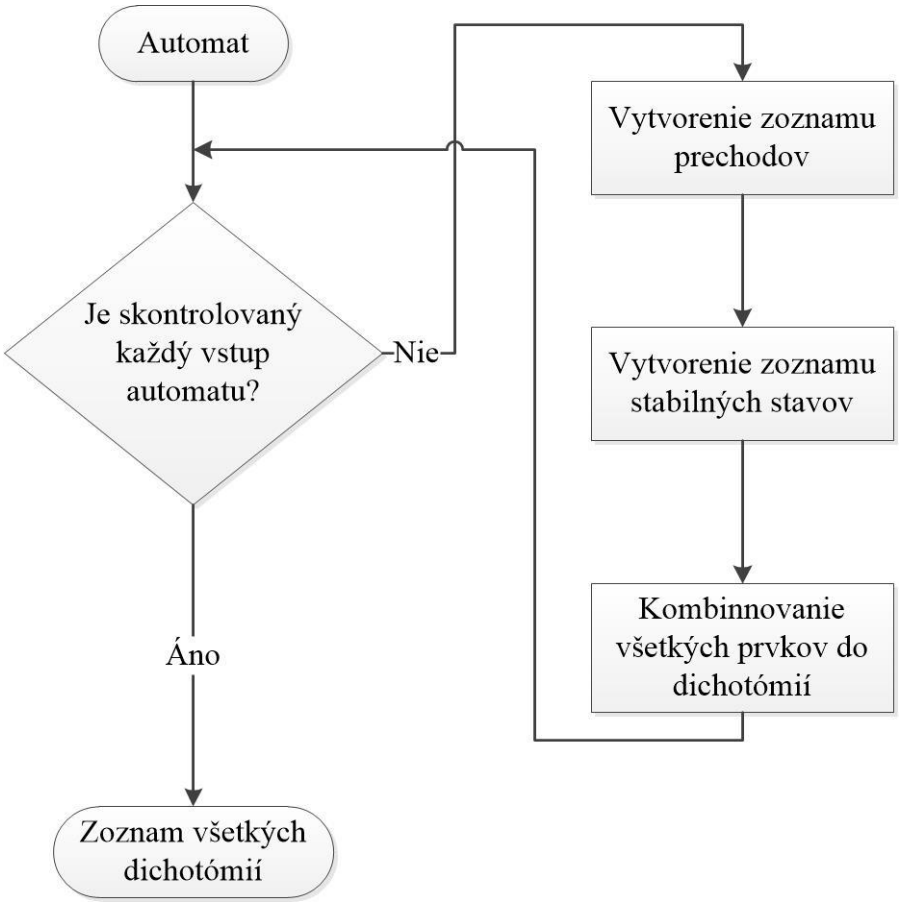
5.1.2 Vytvorenie zoznamu úplných dichotómií

V druhej časti algoritmu sa z načítaného automatu vytvorí množina úplných dichotómií **C**, z ktorej je možné následne vybrať zakódovanie stavov automatu. Vstup, s ktorým algoritmus pracuje, je prechodová časť automatu, teda časť, v ktorej sú definované prechody medzi stavmi. Výstupom z algoritmu je zoznam úplných dichotómií, združený

so zoznamom, ktorý definuje pokrytia z minimálneho zoznamu dichotómií. Samotný algoritmus je rozdelený do troch častí ako je uvedené na obrázku 5.3.

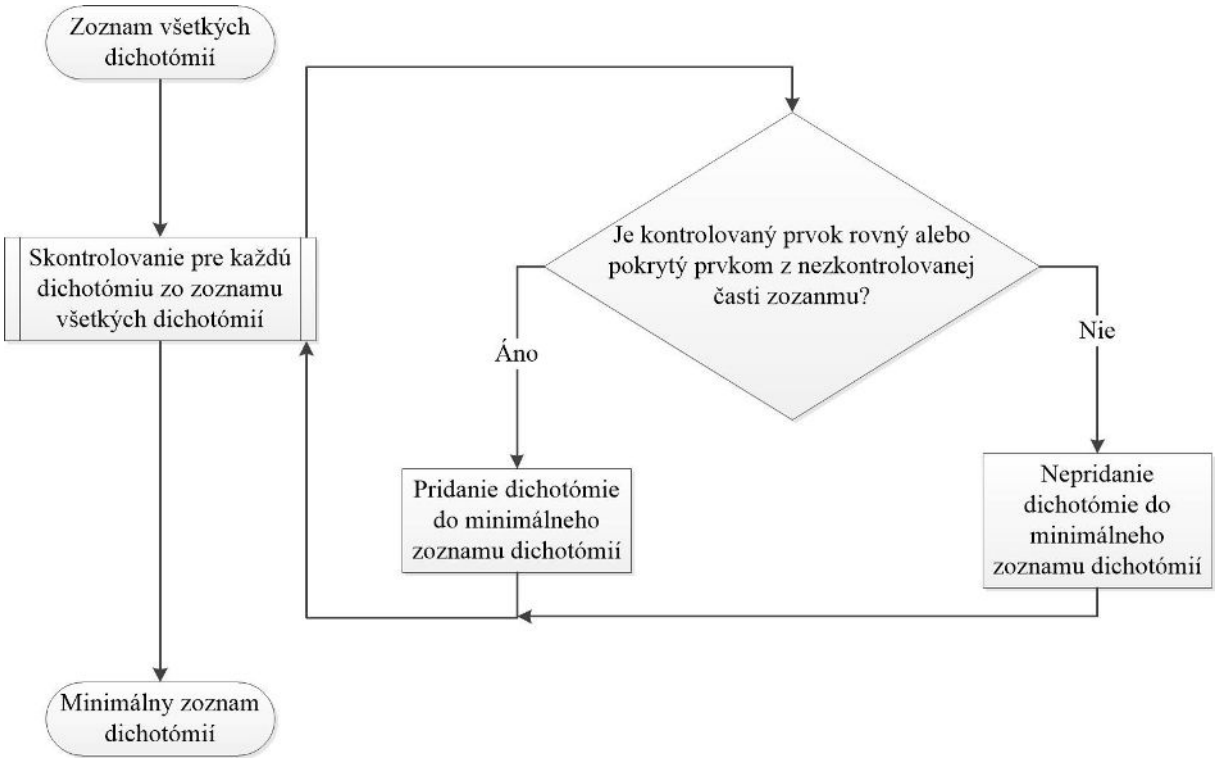
Obrázok 5.3 Algoritmus vytvorenia zoznamu úplných dichotómií

1. Vytvorenie zoznamu všetkých dichotómií - pre každý možný vstup načítaného automatu sa vytvára zoznam prechodov a zoznam stabilných stavov. Následne sa vytvárajú všetky možné kombinácie, pre ktoré platia pravidlá o vytváraní dichotómie. Potom sa vytvorené dichotómie pridajú do jedného zoznamu. Vstupom do prvej časti je automat a výstupom je zoznam všetkých dichotómií. Použitý algoritmu je uvedený na obrázku 5.4.



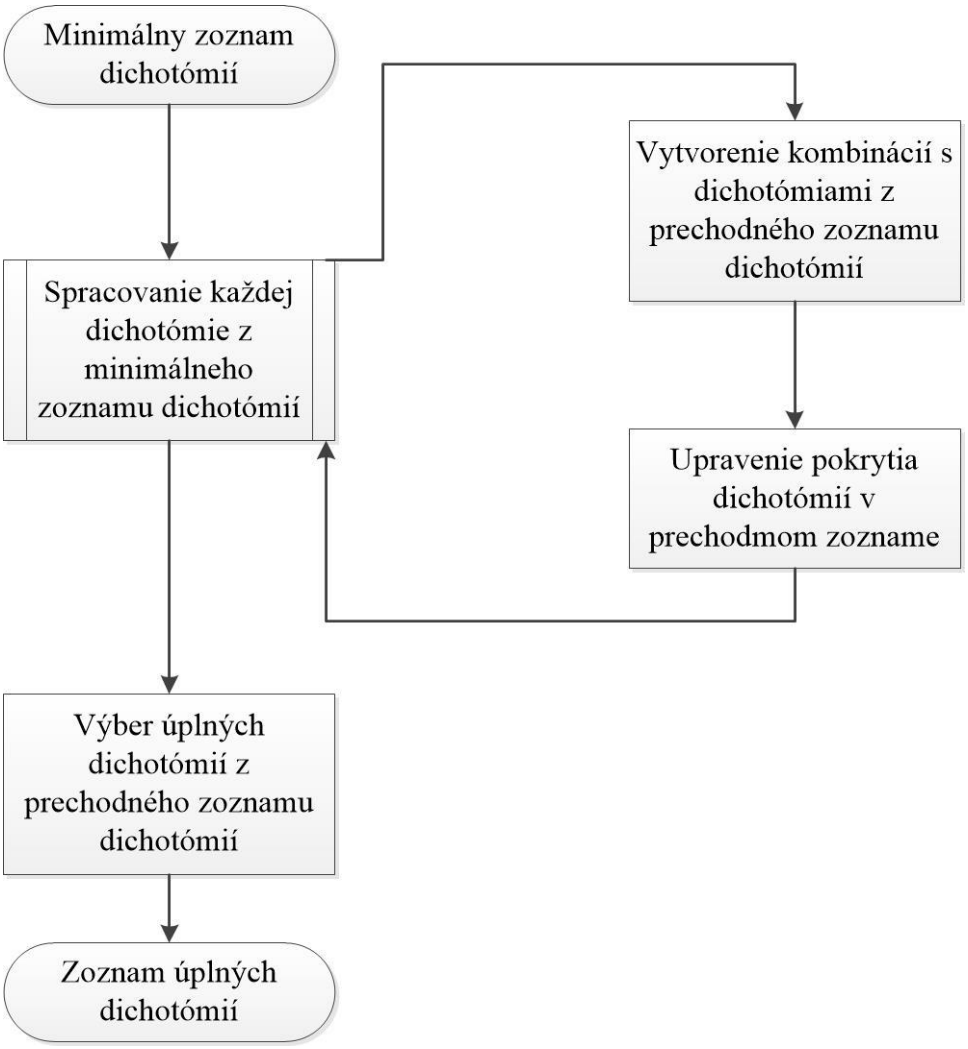
Obrázok 5.4 Vytvorenie zoznamu všetkých dichotómií

2. Vytvorenie minimálneho zoznamu dichotómií - pre každý prvok v zozname všetkých dichotómií sa zistí pokrytie danej dichotómie. Ak neexistuje dichotómia zo zoznamu neskontrolovaných dichotómií, ktorá by pokrývala kontrolovanú dichotómiu, tak sa kontrolovaná dichotómia pridá do minimálneho zoznamu dichotómií. Vstupom je zoznam všetkých dichotómií, a výstupom je minimálny zoznam dichotómií, ktorý pokrýva všetky dichotómie a nenachádzajú sa v ňom žiadne dve rovnaké dichotómie. Použitý algoritmu je uvedený na obrázku 5.5.



Obrázok 5.5 Vytvorenie minimálneho zoznamu dichotómií

3. Vytvorenie zoznamu úplných dichotómií - vytváranie úplných dichotómií prebieha pomocou kombinácií všetkých minimálnych dichotómií. Pre každú možnú kombináciu spojenia minimálnych dichotómií do úplnej dichotómie sa vypočíta pokrytie z minimálneho zoznamu dichotómií pomocou úplnej dichotómie. Vstupom je minimálny zoznam dichotómií, výstupom zoznam úplných dichotómií ako aj pokrytie jednotlivých úplných dichotómií. Vytvorenie zoznamu úplných dichotómií je na obrázku 5.6.



Obrázok 5.6 Vytvorenie zoznamu úplných dichotómií

5.1.3 Výber zakódovania stavov automatu

Zakódovanie stavov v automate sa vyberie v tretej časti algoritmu. Zo zoznamu úplných dichotómií **C** sa vyberú podmnožiny **T,** v ktorých je minimálny počet dvojíc τ a P, kde zjednotenie všetkých množín P zo všetkých dvojíc danej podmnožiny **T** dáva pokrytie všetkých stavov. Následne pre každú podmnožinu **T** sú vytvorené všetky kombinácie zakódovania stavov. Počet týchto kombinácií je 2^S, kde S je počet dvojíc (τ , P) v podmnožine **T.** Vstupom do tretej časti programu je zoznam všetkých úplných dichotómií s príslušným pokrytím. Výstupom je zoznam s viacerými možnosťami zakódovania stavov v automate.

5.1.4 Doručenie prechodovej funkcie

Štvrtá časť algoritmu pre každú vybranú možnosť zakódovania stavov vytvorí nové záchytné stavy, ktoré ošetria kritické súbehy. Cieľom je pokryť každú možnú cestu prechodu pre každý prechod v automate.

Pre každý prechod so začiatočným stavom **s** a koncovým stavom **e**, v ktorom je Hammingova vzdialenosť kódu **s** a kódu **e** rovná dvom, vytvorí dvojicu záchytných stavov $\mathbf{p_1}$ a $\mathbf{p_2}$. Pre kód $\mathbf{p_1}$platí, že Hammingova vzdialenosť s kódom **s** a Hammingova vzdialenosť s kódom **e** je rovná jednej. Pre kód $\mathbf{p_2}$ platí, že Hammingova vzdialenosť s kódom **s** a Hammingova vzdialenosť s kódom **e** je rovná jednej. Zároveň platí, že kód $\mathbf{p_1}$a kód $\mathbf{p_2}$ je rozdielny. Prechody zo stavov $\mathbf{p_1}$ a $\mathbf{p_2}$ sú zhodné so stavom **s** a výstupy stavov $\mathbf{p_1}$ a $\mathbf{p_2}$ sú tak isto zhodné so stavom **s**. Rozdiel je len kód daných stavov.

Vstupom pre štvrtú časť algoritmu je automat a príslušné zakódovanie stavov v automate. Výstupom je rozšírený automat, ktorý obsahuje aj všetky potrebné záchytné stavy. Výstupom je aj kompletný zoznam zakódovania všetkých stavov v rozšírenom automate.

5.1.5 Vytvorenie budiacich funkcií

Vytvorenie logických funkcií pre vstupnú kombinačnú časť obvodu je úlohou piatej časti algoritmu. Budiace funkcie pre pamäťovú časť sú vo formáte **UDNF**[3]. Budiace funkcie opisujú vstupné logické funkcie do preklápacích obvodov. Vstupom do piatej časti algoritmu je rozšírený automat spolu so zoznamom zakódovaných stavov. Výstupom sú budiace funkcie pre preklápacie obvody typu D.

5.1.6 Transformácia budiacich funkcií

Vytvorené budiace funkcie z piatej časti algoritmu sú kombinačnou časťou pre preklápacie obvody typu SR. V šiestej časti algoritmu sú vytvorené budiace funkcie pre preklápacie obvody typu JK. Transformované funkcie pre preklápacie obvody typu JK nie je potrebné ošetrovať proti statickým hazardom. Transformácia prebieha pomocou transformačnej tabuľky, ktorá je uvedená v tabuľke 5.1.

Tabuľka 5.1 Vytvorenie zoznamu úplných dichotómií

Aktuálny stav	J	K	Opis	Stav po prechode
0	0	X	bez zmeny	0
0	1	X	nastav 1	1
1	X	1	nastav 0	0
1	X	0	bez zmeny	1

Vstupom do šiestej časti algoritmu sú budiace funkcie vo formáte **UDNF.** Výstupom sú transformované budiace funkcie pre preklápacie obvody typu JK.

5.1.7 Vytvorenie výstupných funkcií

Siedma časť algoritmu vytvorí funkcie pre výstupnú kombinačnú logiku. Výstupné funkcie sú vytvárané podľa stavov, v ktorých sú výstupné premenné v logickej hodnote jeden. Pre každú výstupnú premennú je vytvorená jedna výstupná funkcia.

Vstupom do siedmej časti algoritmu je rozšírený automat zo štvrtej časti algoritmu a výstupom sú výstupné funkcie vo formáte **UDNF**[3].

5.1.8 Vyhodnotenie počtu logických členov

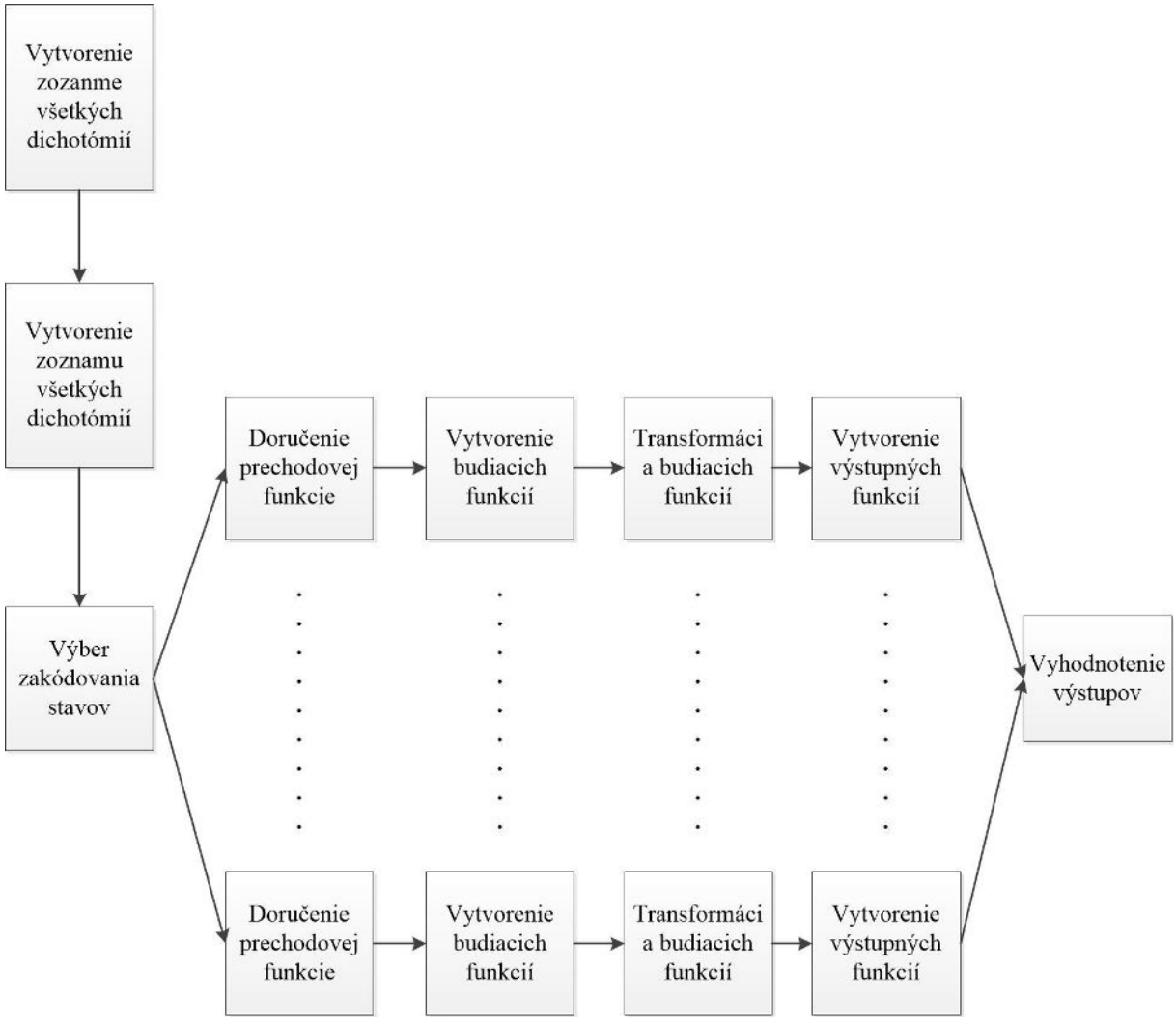
Posledná časť algoritmu vyhodnocuje, ktoré základné zakódovanie stavov v pôvodnom automate na konci poskytuje najmenej logických členov. Po spočítaní logických členov vyhodnotí ich počet a výslednou reprezentáciou automatu je to, ktoré ma najmenej logických členov.

Vstupom do ôsmej časti algoritmu sú vytvorené budiace funkcie a prislúchajúce výstupné funkcie. Výstupom súbor s reprezentáciou automatu s najnižším počtom logických členov v kombinačnej časti.

5.2 Návrh pre vysokovýkonný výpočtový systém

Distribuované riešenie vychádza zo sekvenčného riešenia. V sekvenčnom riešení existujú časti, ktoré je možné paralelizovať, čiže navrhnúť tak, aby sa vykonávali súbežne.

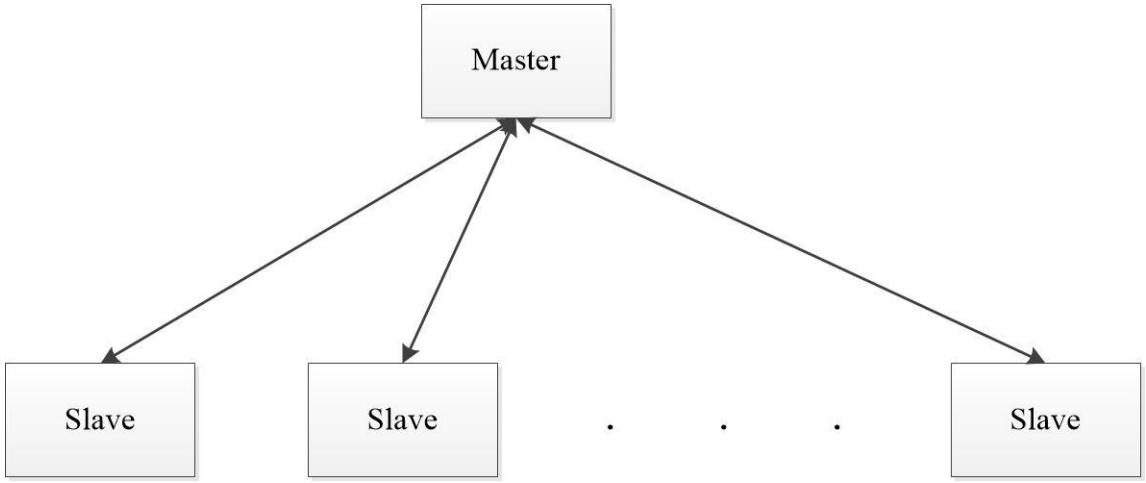
Paralelné spracovanie je možné z dôvodu nezávislosti určitých sekvencií programu na výstupných údajoch predchádzajúcej časti programu. Súčasťou distribuovaného riešenia je aj systém komunikácie, pomocou ktorého sa rozosielajú údaje a výsledky v celom programovom riešení. Je dôležité, aby čas komunikácie a výpočtu bol v čo najlepšom pomere, v prospech času výpočtu. Preto je potrebné dobre zvoliť granularitu paralelizmu. Bola zvolená vysoká granularita, pri ktorej sa paralelne vykonávajú veľké časti programu a komunikácia je veľmi nízka. Je potrebná len na začiatku a na konci vykonávania podprogramu. Rozdelenie sekvenčného algoritmu je znázornené na obrázku 5.7.



Obrázok 5.7 Rozdelenie sekvenčného programu pre paralelné počítanie

Pri návrhu algoritmu pre vysokovýkonný výpočtový systém je použitá architektúra **Master - Slave**, kde časť **Master** vykonáva všetky časti algoritmu, ktoré je potrebné vykonať pred distribuovaním výpočtu na programy typu **Slave.** Preto je možné rozdeliť návrh na dve

časti, návrh pre časť **Master** a návrh pre časť **Slave.** Použitá architektúra je všeobecne uvedená na obrázku 5.8.



Obrázok 5.8 Architektúra Master - Slave

5.2.1 Návrh časti Master

Návrh programu, ktorý sa vykonáva na vysokovýkonnom výpočtovom systéme len jedenkrát, je zobrazený na obrázku 5.9.

Obrázok 5.9 Návrh programu typu Master

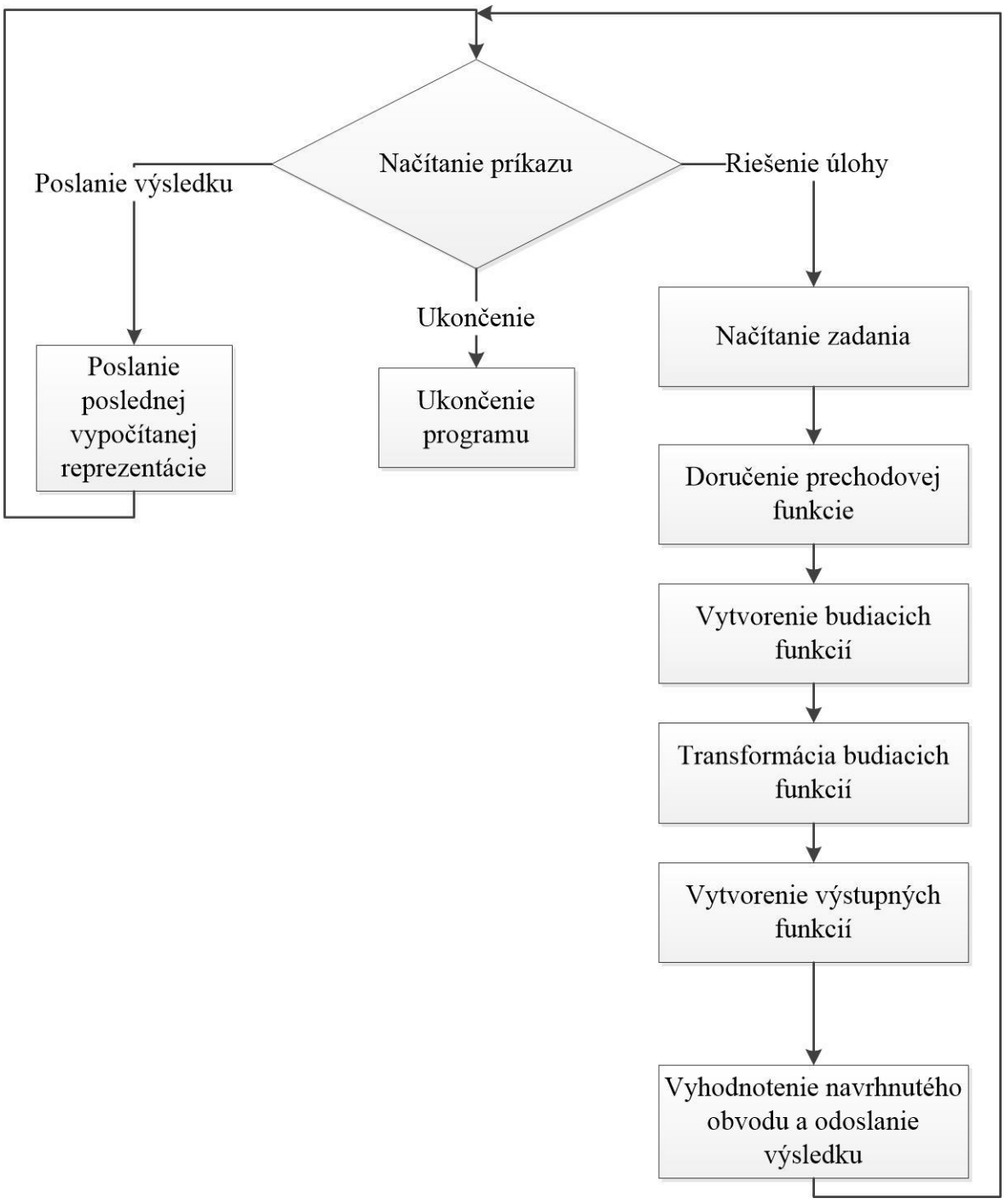
Master zabezpečuje načítanie informácií zo vstupu, pripravuje informácie pre jednotlivé programy typu **Slave**, zabezpečuje komunikácie medzi jednotlivými programami typu **Slave** a programom typu **Master.** Po ukončení zbiera výsledky a vyhodnocuje ich. Ako možno vidieť na obrázku 5.9, program je rozdelený do viacerých častí:

1. Načítanie automatu zo súboru - prvá časť programu **Master** načíta automat zo súboru. Táto časť je zhodná s prvou časťou algoritmu pri sekvenčnom riešení, ktorá je opísaná v časti 5.1.1.
2. Vytvorenie zoznamu úplných dichotómií - vytvorenie zoznamu úplných dichotómií je totožné s druhou časťou algoritmu opísanou v časti 5.1.2.
3. Výber zakódovania stavov - príprava údajov pre distribuované počítanie je ukončená treťou časťou, kde je vybraných veľa možných kódov. Výber zakódovania stavov je opísaný v časti 5.1.3.

4. Rozoslanie úloh - komunikácia medzi uzlami vykonávajúcimi výpočet je zabezpečená v štvrtej časti algoritmu. Na začiatku sa pošle opis automatu všetkým výpočtovým uzlom, na ktorých sa vykonáva program typu **Slave**. Následne sa pošlú konkrétne úlohy pre určitý výpočtový uzol. Ak už programy spracovali všetky úlohy, tak sa výpočtovému uzlu pošle príkaz na ukončenie. Následne sa čaká na príjem výsledku z uzlov, do ktorých boli poslané konkrétne úlohy. Vstupom je zoznam zakódovaných stavov a opis automatu. Výstupom sú príkazy pre je jednotlivé výpočtové uzly.
5. Zozbieranie výsledkov a ich vyhodnotenie - po prijatí výsledkov z uzlov sa vykoná vyhodnocovanie, čiže sa porovnajú počty logických členov vo výsledných funkciách. Ak je ich počet najmenší, tak sa vyžiada od výpočtového uzla aj opis funkcií, inak sa opis funkcií nebude vyžadovať. Zozbieranie výsledkov prebieha toľkokrát, aký je počet odoslaných výsledkov. Vstupom sú výsledky z výpočtových uzlov, výstupom je logický obvod reprezentujúci vstupný automat s najmenším počtom logických členov.
6. Zapísanie výsledku do súboru - posledná časť vykonáva rovnakú funkciu ako ôsma časť sekvenčného algoritmu a je opísaná v kapitole 5.1.8.

5.2.2 Návrh časti Slave

Časť **Slave** zabezpečuje výpočet na viacerých výpočtových uzloch v systéme. Je spustená viacnásobne a každé vykonávanie programu má svoje identifikačné údaje.



Obrázok 5.10 Návrh programu typu Slave

Ako je uvedené na obrázku 5.10, program je vykonávaný v slučke dovtedy, kým dostáva údaje, a je rozdelený do týchto častí:

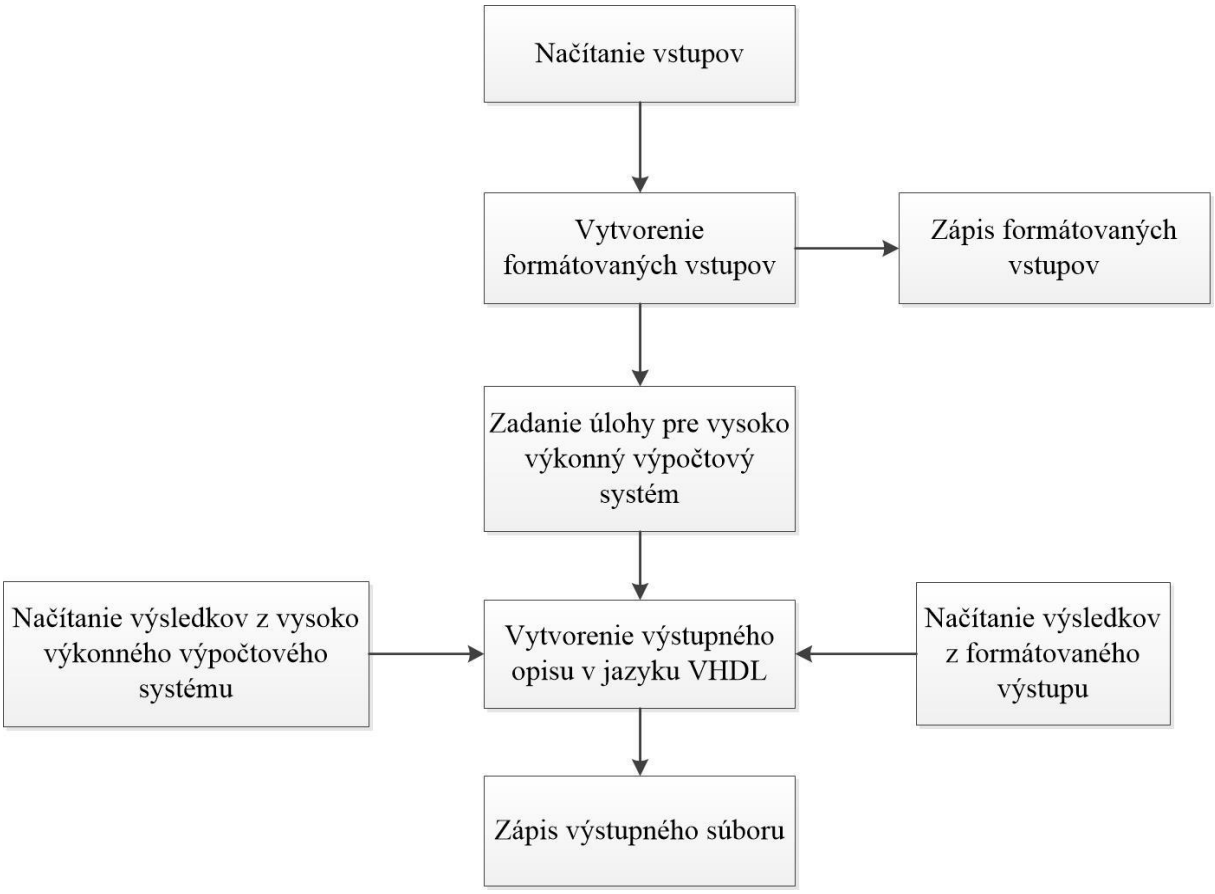
1. Načítanie zadania - v prvej časti sa vykoná načítanie automatu. Následne program čaká na jeden z troch príkazov od programu **Master.**

- Prvým príkazom je príkaz na ukončenie programu - po jeho prijatí sa program výpočtového uzla ukončí.
- Druhý príkaz pozostáva z dvoch častí, v prvej je samotný príkaz a v druhej je doručená úloha, ktorá sa má vykonať. Po prijatí úlohy sa pokračuje vo vykonávaní programu.
- Po prijatí tretieho príkazu program odošle posledný kompletný výsledok.

2. Doručenie prechodovej funkcie - vstupom druhej časti algoritmu je zadanie načítané v prvej časti a výstupom je rozšírený automat so zakódovanými stavmi. Táto časť je totožná so štvrtou časťou sekvenčného programu, ktorá je opísaná v časti 5.1.4.
3. Vytvorenie budiacich funkcií - budiace funkcie sú vytvorené v tretej časti algoritmu. Táto časť je totožná s piatou časťou sekvenčného algoritmu, opísanou v časti 5.1.5.
4. Transformácia budiacich funkcií - prepis funkcií pre preklápacie obvody JK je rovnaký ako v šiestej časti sekvenčného algoritmu, ktorý je uvedený v časti 5.1.6.
5. Vytvorenie výstupných funkcií - v piatej časti algoritmu sa vytvoria výstupné funkcie na základe upraveného automatu. Postup je podrobne opísaný v siedmej časti sekvenčného algoritmu, ktorý je uvedený v časti 5.1.7.
6. Odoslanie výsledku - posledná časť algoritmu pre **Slave**, pozostáva vo vyhodnotení vytvoreného opisu, spočítaní jeho členov a odoslaní výsledku. Po odoslaní výsledku sa celý algoritmu opakuje od prvého bodu, až kým nie je poslaný ukončovací príkaz.

5.3 Návrh obslužného programu

Obslužný program ja navrhnutý pre spracovávanie vstupov a výstupov pre programy, ktoré vykonávajú výpočtovú funkciu. Taktiež zabezpečuje komunikáciu so serverom, ktorý vykonáva program na vysokovýkonnom výpočtovom prostriedku. Architektúra obslužného programu je na obrázku 5.11.



Obrázok 5.11 Architektúra obslužného programu

5.3.1 Načítanie vstupov

Prvá časť obslužného programu zabezpečuje načítanie vstupov z formátu **VHDL**. Vstupom je automat, ktorý môže byť graficky opísaný v programe **HDL Designer** a následne pre grafickú reprezentáciu automatu vytvorený súbor vo formáte **VHDL**, ktorý je vstupom obslužného programu. Vstupný súbor vo formáte **VHDL** musí spĺňať nasledujúce kritéria:

1. Opísaný automat je bez funkčných hazardov.
2. Opísaný automat je fundamentálny automat prvého stupňa.
3. Opísaný automat je typu **Moore**.
4. Opis každého stavu vo vstupnom automate má uvedené prechody, respektíve stabilné stavy pre všetky kombinácie vstupov a sú v opise uvedené v rovnakom poradí.
5. Hodnoty výstupných premenných sú definované pre každý stav.

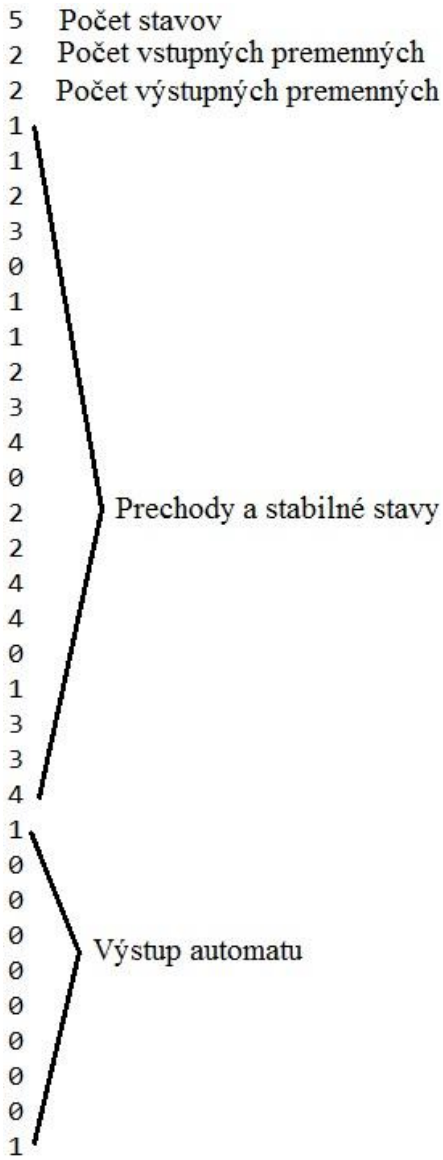
Načítanie vstupu prebieha v načítaní **VHDL** súboru ako textového súboru, kde je každý riadok osobitne spracovaný. Vstupom prvej časti algoritmu je súbor vo formáte **VHDL**, výstupom je reprezentácia automatu v pamäti programu.

5.3.2 Vytvorenie formátovaných vstupov

V druhej časti algoritmu sa vytvárajú vstupy, ktoré sú v čo najjednoduchšom tvare, a je možné ich načítať pomocou programu pre vysokovýkonný výpočtový systém ako aj sekvenčným programom. Samotný formátovaný vstup je textom v preddefinovanom formáte, ktorý je nasledovný:

1. V prvom riadku je hodnota **N**, ktorá označuje počet stavov.
2. V druhom riadku je hodnota **I**, ktorá označuje počet vstupných premenných.
3. V treťom riadku sa nachádza hodnota **O,** ktorá značí počet výstupných premenných.
4. Nasledujú prechody alebo stabilné stavy pre každý vstup automatu – N - tice, ktorých počet je $2^{\mathbf{I}}$.
5. Na záver formátovaného súboru je **O N -** tíc, ktoré reprezentuje výstupy automatu.

Príklad textového súboru zapísaný v navrhnutom formáte je na obrázku 5.12.



Obrázok 5.12 Formátovaný vstupný súbor

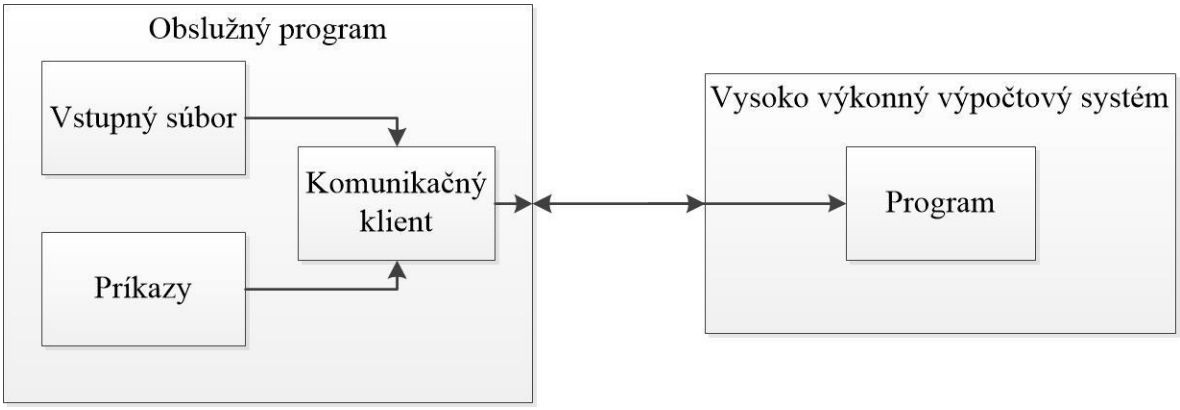
Vstupom druhej časti obslužného programu je vnútorná štruktúra, ktorá opisuje automat a výstupom je text v navrhnutom formáte.

5.3.3 Zápis formátovaných vstupov

Výstupný modul pre zápis formátovaných vstupov zapíše do zvoleného textového súboru opis automatu v navrhnutom formáte. Vstupom je text, v ktorom je opis automatu v navrhnutom formáte a výstupom je textový súbor, v ktorom je uložený opis automatu.

5.3.4 Zadanie úlohy pre vysokovýkonný výpočtový systém

Poslanie úlohz a spustenie príslušných programov na vysokovýkonnom výpočtovom prostriedku je úlohou štvrtej časti algoritmu. Pomocou komunikačného klienta sa pošle vstupný súbor v navrhnutom formáte a spustí sa program na vysokovýkonnom výpočtovom systéme. Postupnosť krokov je na obrázku 5.13.



Obrázok 5.13 Zadanie úlohy pre vysokovýkonný výpočtový systém

Vstupom je súbor s opisom automatu v preddefinovanom formáte a súbory potrebné pre vytvorenie spojenia. Výstupom je prenesený súbor na vysokovýkonnom výpočtovom systéme a spustenie programu na vysokovýkonnom výpočtovom systéme.

5.3.5 Načítanie výsledkov z formátovaného výstupu

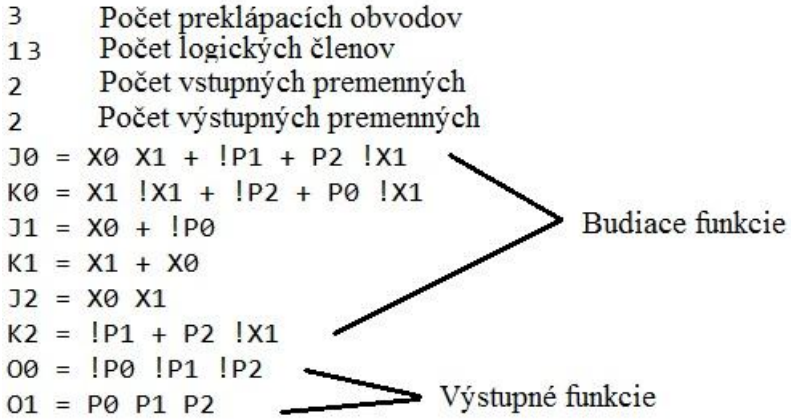
Výsledky z programov sú zapisované vo formátovanom výstupe, ktorým je textový súbor s preddefinovanou štruktúrou. Štruktúra súboru je táto:

1. V prvom riadku je počet preklápacích obvodov **P**.
2. V druhom riadku je počet logických členov, **potrebných** pre realizáciu automatu.
3. V treťom riadku je počet vstupných premenných.
4. Vo štvrtom riadku je počet výstupných funkcií **F**.
5. Nasleduje 2.**P** riadkov, kde prvým riadkom dvojice je opis funkcie pre vstup **J** z preklápacieho obvodu a druhým riadkom je jeho vstup **K**. Každý riadok je v nasledujúcom formáte NX = Z, kde
    - N je označenie vstupu preklápacieho obvodu. Nadobúda hodnotu J alebo K.

- X je poradové číslo preklápacieho obvodu.
- Z je budiaca funkcia preklápacieho obvodu.

6. Na koniec súboru je **F** riadkov, ktoré opisujú výstupné funkcie pre navrhnutú realizáciu automatu. Sú vo formáte NX = Z, kde
   - N je označenie výstupnej funkcie. Nadobúda hodnotu O.
   - X je poradové číslo výstupnej premennej.
   - Z je výstupná funkcia.

Príklad výstupného súboru a jeho rozdelenie je na obrázku 5.14.



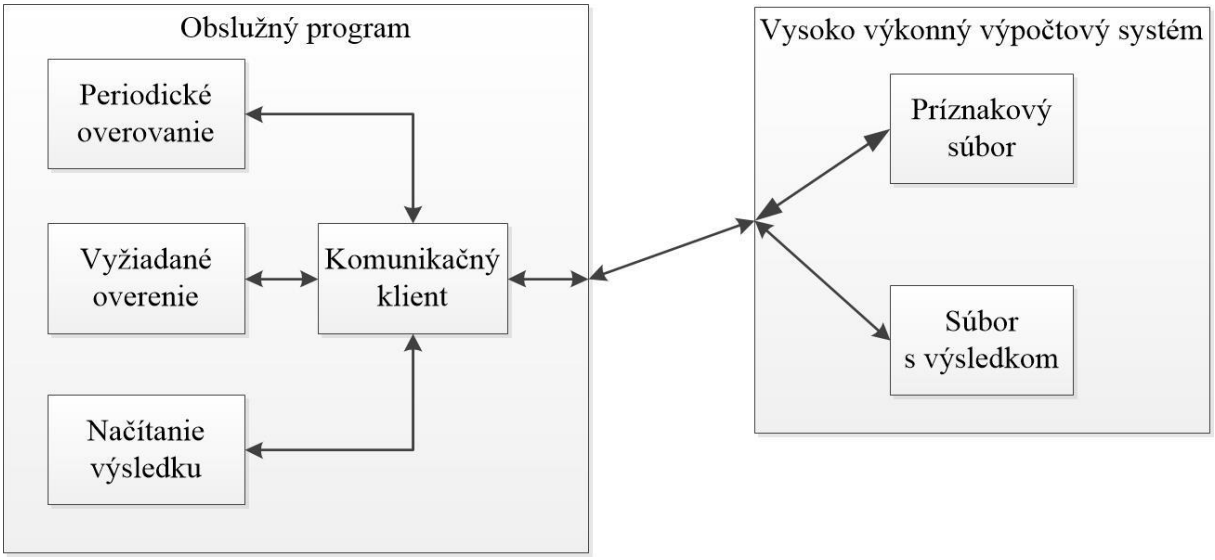
Obrázok 5.14 Formátovaný výstupný súbor

Načítanie výsledkov do vnútorných štruktúr programu je úlohou piatej časti programu. Vnútorné štruktúry programu reprezentujú budiace a výstupné funkcie navrhnutej reprezentácie automatu.
Vstupom je súbor s preddefinovanou štruktúrou. Výstupom sú vnútorné štruktúry, ktoré reprezentujú navrhnutý automat.

5.3.6 Načítanie výsledkov z vysokovýkonného výpočtového systému

Ukončenie výpočtu na vysokovýkonnom výpočtovom systéme nie je možné časovo určiť pri posielaní úlohy. Ukončenie výpočtu je závislé na vyťaženosti prostriedkov na vysokovýkonnom výpočtovom systéme, preto šiesta časť algoritmu má dve možnosti zistenia výsledku. Môže kontrolovať vysokovýkonný výpočtový systém v pravidelných intervaloch alebo môže skontrolovať systém na žiadosť používateľa. Ak je

na vysokovýkonnom výpočtovom systéme príznakový súbor, ktorý označuje ukončenie výpočtu, tak prebehne prenos výsledku do obslužného programu. Následne je automaticky načítaný výstupný súbor pomocou časti pre načítanie výsledkov, ktorá je uvedená v časti 5.3.5. Komunikácia a spôsob načítania výsledkov je uvedený na obrázku 5.15.



Obrázok 5.15 Načítanie výsledkov z vysokovýkonného výpočtového systému

Vstupom do tejto časti sú súbory potrebné pre komunikáciu s vysokovýkonným výpočtovým systémom. Výstupom je vnútorná štruktúra, ktorá reprezentuje budiace funkcie a výstupné funkcie navrhnutého opisu.

5.3.7 Vytvorenie výstupného opisu v jazyku VHDL

V predposlednej časti obslužného programu sú spracované vnútorné štruktúry, ktoré opisujú navrhnutú realizáciu asynchrónneho sekvenčného obvodu. Budiace aj výstupné funkcie sú pretransformované do textovej podoby vo formáte **VHDL**, aby bolo možné otestovať funkčnosť návrhu, alebo aby bolo možné návrh ďalej spracovávať. Formát **VHDL** pre realizáciu asynchrónneho sekvenčného obvodu sa skladá z dvoch hlavných častí:

1. Opis použitých komponentov - kde sa opíše správanie sa alebo architektúra použitých častí obvodu, v našom prípade preklápacie obvody typu JK**.**
2. Zapojenie jednotlivých komponentov - opis zapojenia signálov do vstupov a výstupov obvodu. Zapojenie je rozdelené do štyroch častí:

- Zapojenie vstupných premenných a výstupov z preklápacích obvodov do budiacich funkcií.
- Zapojenie budiacich funkcií do preklápacích obvodov.
- Zapojenie výstupov preklápacích obvodov do výstupných funkcií.
- Zapojenie výstupných funkcií na výstupné premenné.

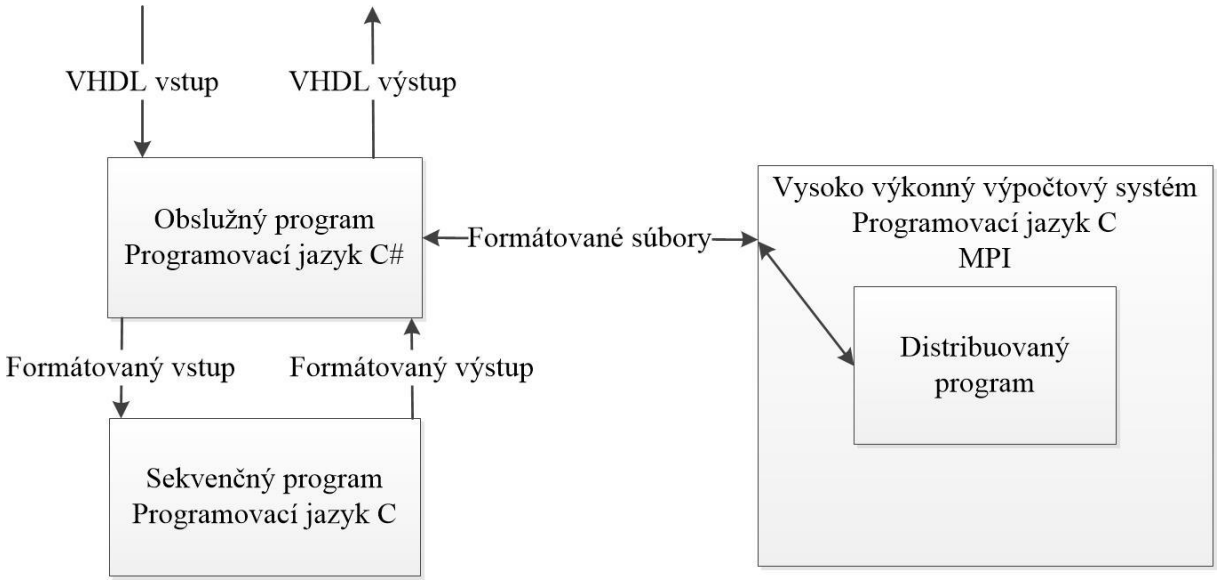
Vstupom sú vnútorné štruktúry, ktoré opisujú architektúru navrhnutého asynchrónneho sekvenčného obvodu. Výstupom je text vo formáte **VHDL.**

5.3.8 Zápis výstupného súboru

Výstupná časť pre zápis výstupného súboru zapíše text vo formáte **VHDL** do zvoleného súboru. Vstupom je text vo formáte **VHDL**, v ktorom je opísaná architektúra navrhnutého asynchrónneho sekvenčného obvodu. Výstupom je súbor vo formáte **VHDL,** v ktorom je uložená navrhnutá architektúra.

6 Implementácia programového systému

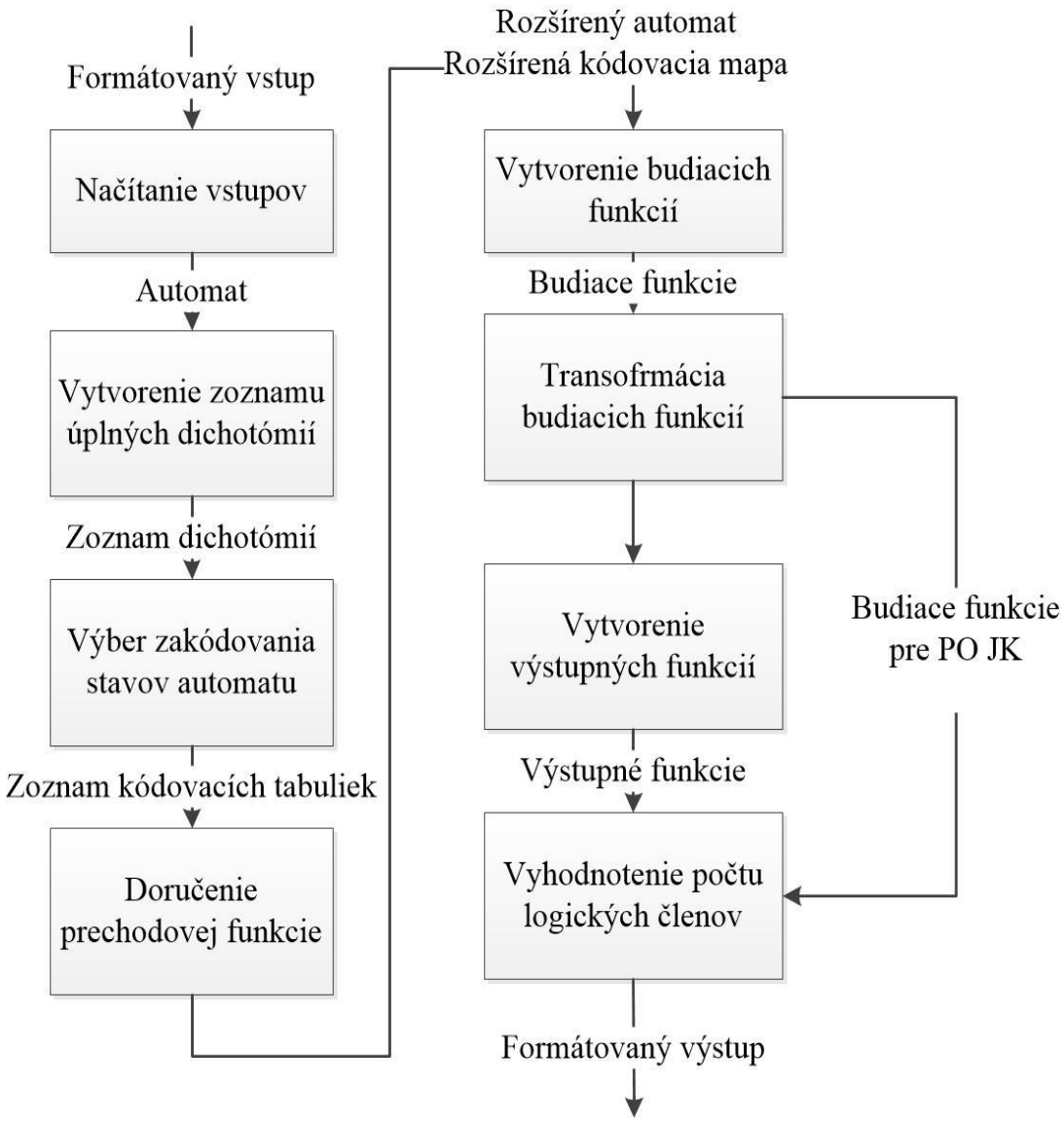
Implementácia programového systému opisuje realizáciu návrhu na jednotlivých platformách. Implementácia je rozdelená rovnako ako návrh do troch častí. V prvej je opísaná implementácia programu pre sekvenčný jednojadrový systém v jazyku C, v druhej časti je opísaná implementácia programu pre vysokovýkonný výpočtový systém, ktorá je v jazyku C s použitím knižnice **MPI** [8]**.** V poslednej časti je opísaná implementácia obslužného programu v jazyku C# s použitím komunikačných knižníc. Zloženie systému je uvedené na obrázku 6.1. Pre písanie zdrojových kódov je použitý MS Visual Studio 2012.



Obrázok 6.1 Zloženie programového systému z hľadiska implementácie

6.1 Implementácia programu pre jednojadrový systém

Program pre návrh asynchrónnych sekvenčných obvodov je implementovaný v jazyku C. Zdrojový kód je písaný v návrhovom prostredí MS Visual Studio 2012. Implementácia navrhnutého algoritmu pre jednojadrový systém sa skladá z ôsmich častí. Ich popis a tok štruktúr medzi nimi je uvedený na obrázku 6.2.



Obrázok 6.2 Implementácia sekvenčného programu

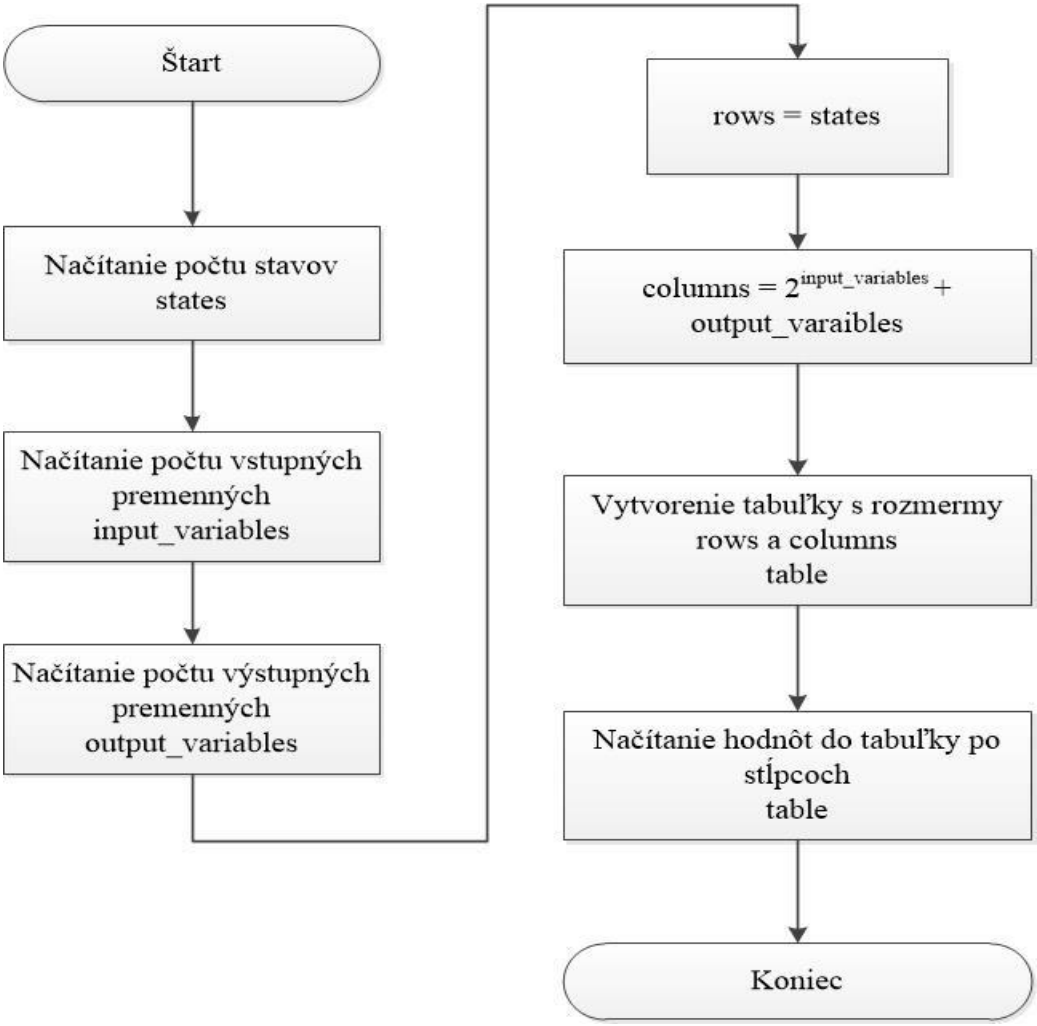
6.1.1 Realizácia načítania vstupov

Prvá časť implementovaného programu načíta vstupný súbor, ktorý je vo formáte definovanom v časti 5.3.2 a uloží ho do štruktúry, ktorá je opísaná v tabuľke 6.1.

Tabuľka 6.1 Štruktúra automatu

automat		
Názov	Typ	Opis
table	**int	Tabuľka, v ktorej sú uložené prechody, stabilné stavy a výstupné funkcie.
states	int	Počet stavov v automate.
input_variables	int	Počet vstupných premenných.
output_variables	int	Počet výstupných premenných.
rows	int	Počet riadkov v tabuľke.
columns	int	Počet stĺpcov v tabuľke.

Postup načítania automatu do štruktúry je na obrázku 6.3.



Obrázok 6.3 Načítanie automatu do štruktúry

6.1.2 Implementácia vytvorenia zoznamu úplných dichotómií

Vytvorenie zoznamu úplných dichotómií s ich pokrytím je uvedené na obrázku 6.4.

Obrázok 6.4 Vytvorenie zoznamu úplných dichotómií

Okrem štruktúry **automatu**, ktorý je vstupom do druhej časti algoritmu, sú použité tri nové štruktúry. Prvou štruktúrou je štruktúra **transition**, ktorá ukladá jeden prechod v automate. Je uvedená v tabuľke 6.2.

Tabuľka 6.2 Štruktúra transition

transition		
Názov	Typ	Opis
origin	int	Pôvodný stav prechodu.
goal	int	Cieľ prechodu.

Druhou použitou štruktúrou je štruktúra **dichotomy,** ktorá reprezentuje jednu dichotómiu aj so zoznamom pokrytia. Implementácia tejto štruktúry v tabuľke 6.3.

Tabuľka 6.3 Štruktúra dichotomy

dichotomy		
Názov	Typ	Opis
first_block	*int	Stavy v prvej časti dichotómie.
second_block	*int	Stavy v druhej časti dichotómie.
first_count	int	Počet stavov v prvej časti dichotómie.
second_count	int	Počet stavov v druhej časti dichotómie.
covers	*int	Pokryté dichotómie.
cover_count	int	Počet pokrytých dichotómií.
next	*dichotomy	Odkaz na ďalšiu dichotómiu.

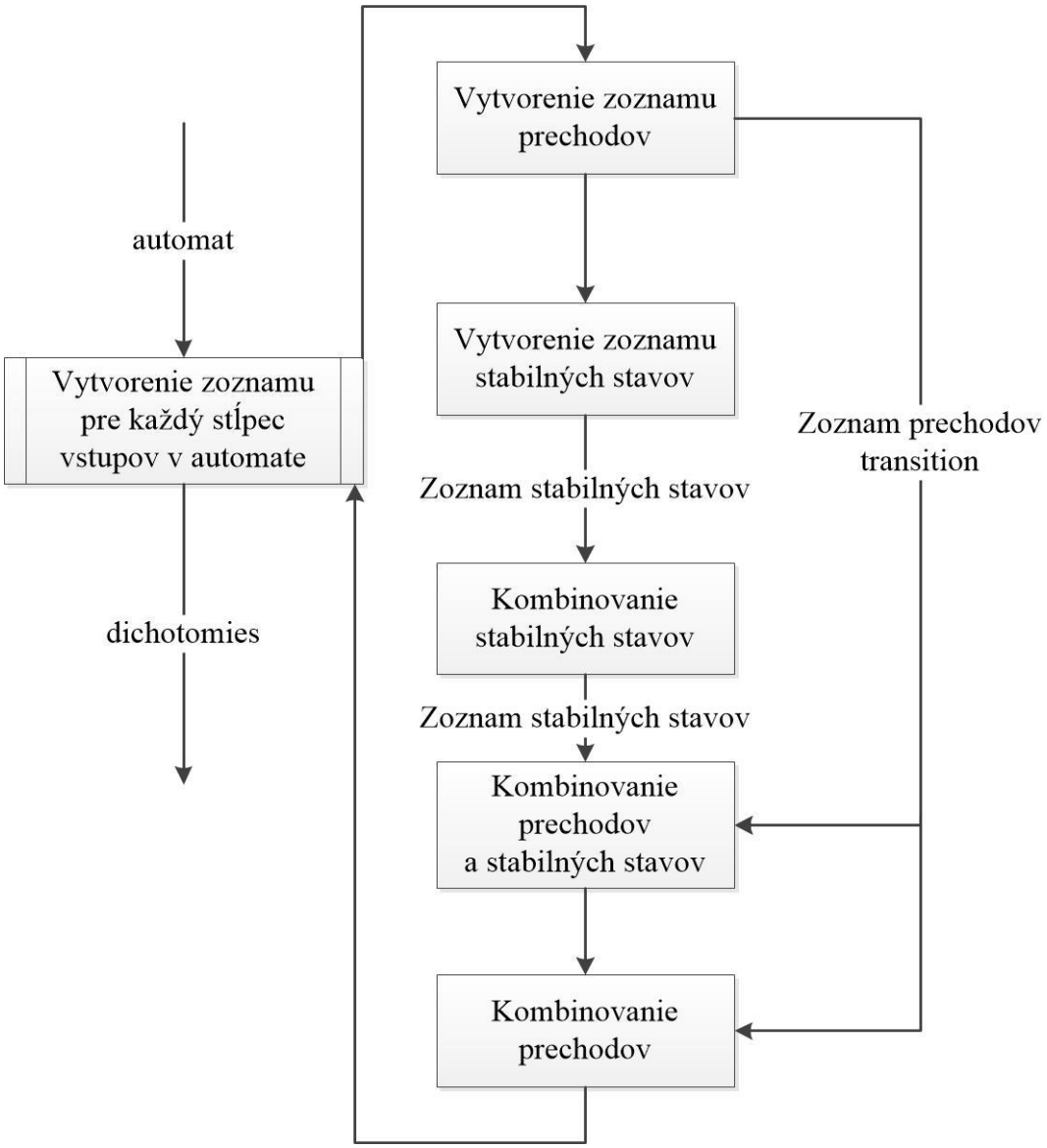
Treťou štruktúrou, ktorá je použitá v druhej časti algoritmu, je štruktúra **dichotomies.** Navrhnutá štruktúra je v tabuľke 6.4 a reprezentuje zoznam dichotómií. Keďže štruktúra **dichotomy** obsahuje odkaz na ďalšiu dichotómiu je možné reprezentovať zoznam dichotómií nie len pomocou poľa, ale aj pomocou spájaného zoznamu.

Tabuľka 6.4 Štruktúra dichotomies

dichotomies		
Názov	Typ	Opis
dichotomies	**dichotomy	Zoznam dichotómií.
count	int	Počet dichotómií v zozname.

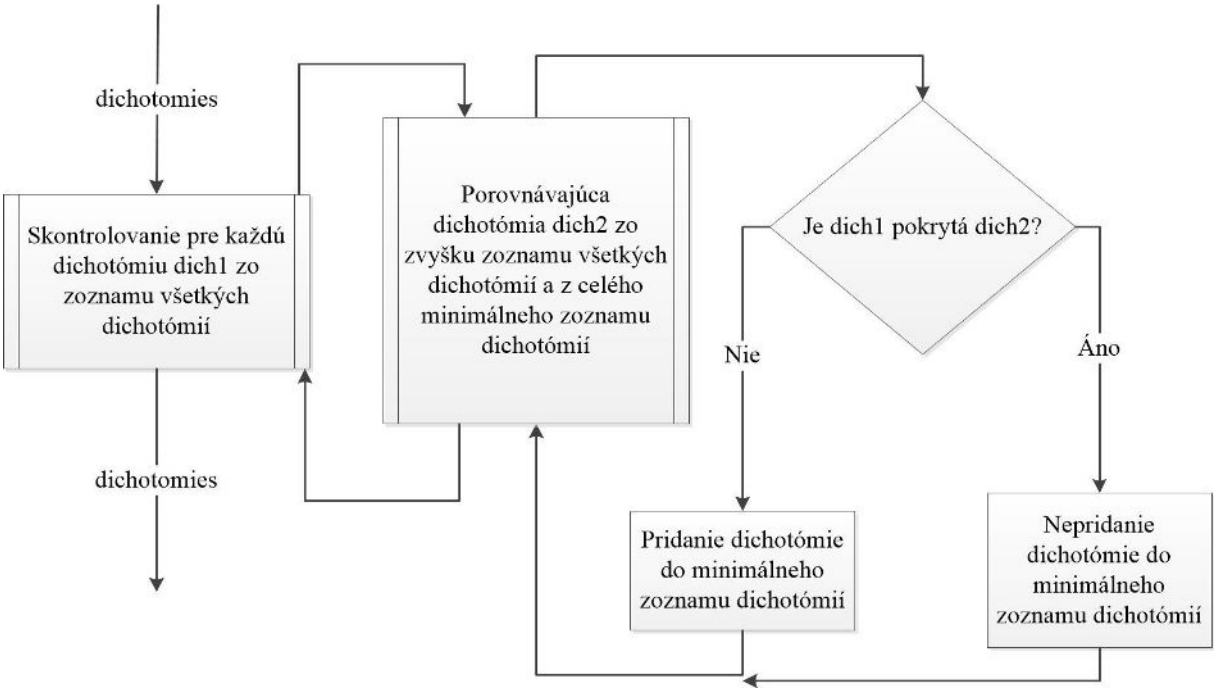
Jednotlivé kroky z obrázku 6.4, sú realizované takto:

1. Vytvorenie zoznamu všetkých dichotómií - z načítaného automatu sa pre každý stĺpec, ktorý reprezentuje jeden možný vstup, vytvoria dva zoznamy. V prvom zozname sú stabilné stavy a v druhom zozname sú uložené prechody v štruktúre **transition.** V nasledujúcich krokoch sú obidva zoznamy uvažované ako jedna množina stavov a prechodov. Z tejto množiny sú vytvorené kombinácie. Celý postup je na obrázku 6.5.



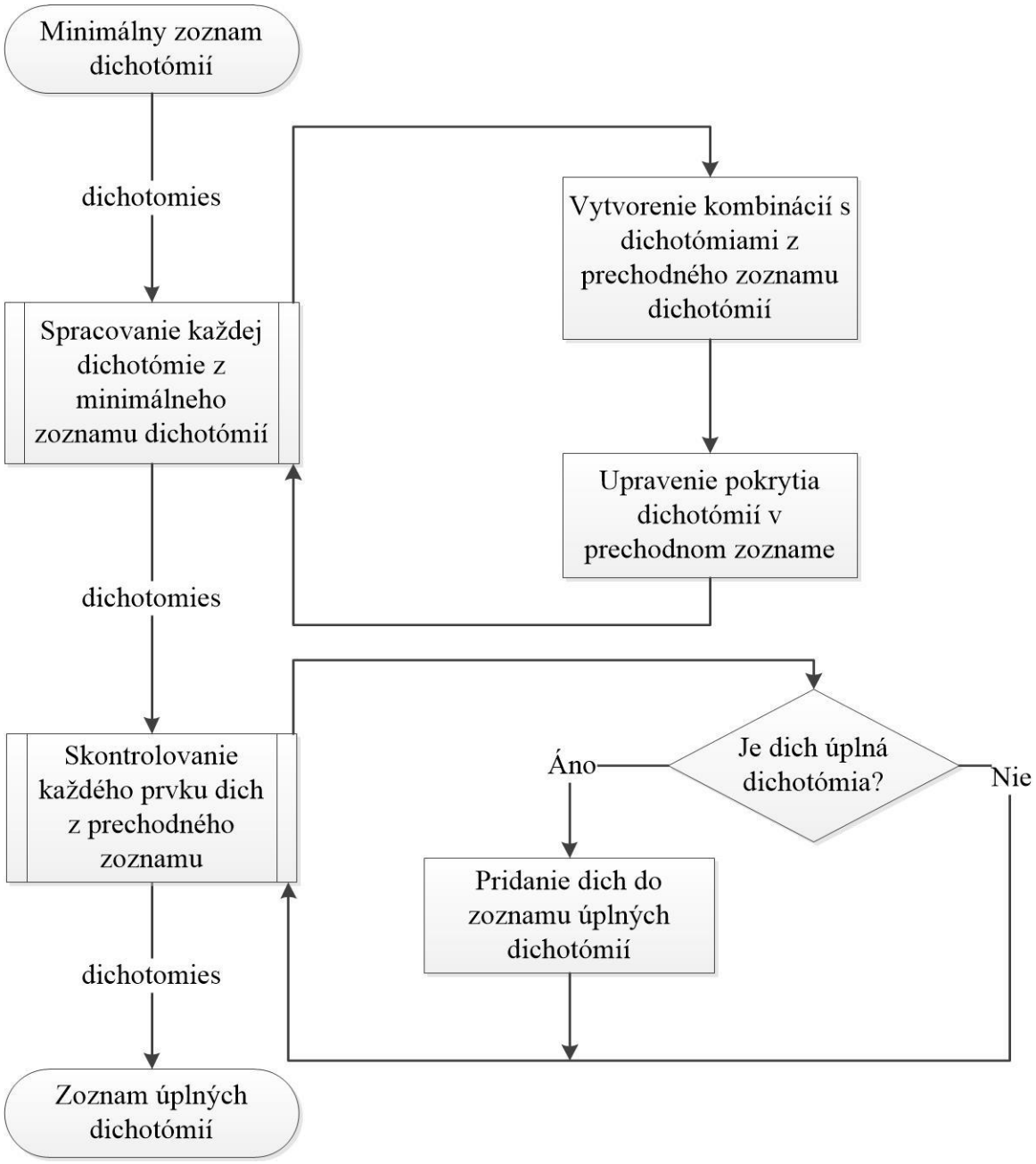
Obrázok 6.5 Realizácia vytvorenia zoznamu všetkých dichotómií

2. Vytvorenie minimálneho zoznamu dichotómií - každý prvok zo zoznamu všetkých dichotómií je porovnávaný so všetkými neskontrolovanými dichotómiami zo zoznamu všetkých dichotómií a takisto je porovnávaná zo všetkými prvkami, ktoré sú už pridané v minimálnom zozname dichotómií. Porovnávanie pozostáva v kontrole, či kontrolovaná dichotómia nie je pokrytá inou dichotómiou, ale či nie je totožná s inou dichotómiou. Ak je kontrolovaná dichotómia jedinečná, čiže pri kontrole nebolo zistené pokrytie alebo zhoda s inou dichotómiou, tak je pridaná do minimálneho zoznamu dichotómií. Realizácia je uvedená na obrázku 6.6.



Obrázok 6.6 Realizácia vytvorenie minimálneho zoznamu dichotómií

3. Vytvorenie zoznamu úplných dichotómií - zo vstupného minimálneho zoznamu dichotómií sa postupne vyberajú dichotómie a s vybratou dichotómiu sa rozširuje každý prvok v prechodnom zozname dichotómií. Ak vznikne po zlúčení nová dichotómia, je pridaná do prechodného zoznamu. Na záver sú prejdené všetky dichotómie v zozname prechodných dichotómií a vyberú sa tie, ktoré sú unikátne a úplné, čiže súčet počtu stavov v prvej a v druhej časti dichotómie je zhodný s celkovým počtom stavov v automate. Celý je postup je zobrazený na obrázku 6.7.



Obrázok 6.7 Implementácia vytvorenia zoznamu úplných dichotómií

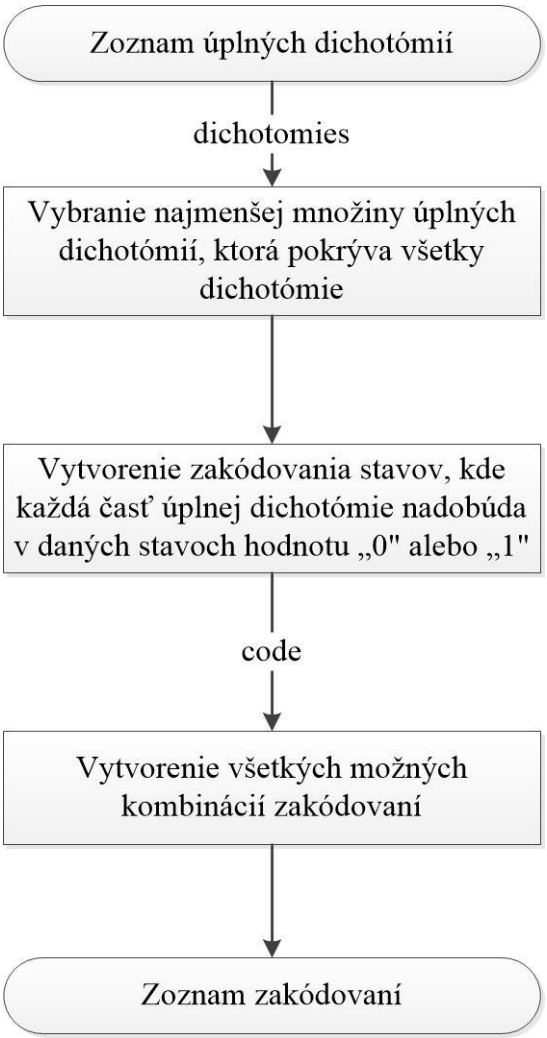
6.1.3 Realizácia výberu zakódovania stavov

Použiteľné zakódovania automatu sa vyberú v nasledujúcej časti algoritmu. Štruktúra, ktorá uchováva možné kódovania stavov je v tabuľke 6.5.

Tabuľka 6.5 Štruktúra codes

code		
Názov	Typ	Popis
coding	**int	Uložené zakódovanie.
code_length	int	Dĺžka kódu.

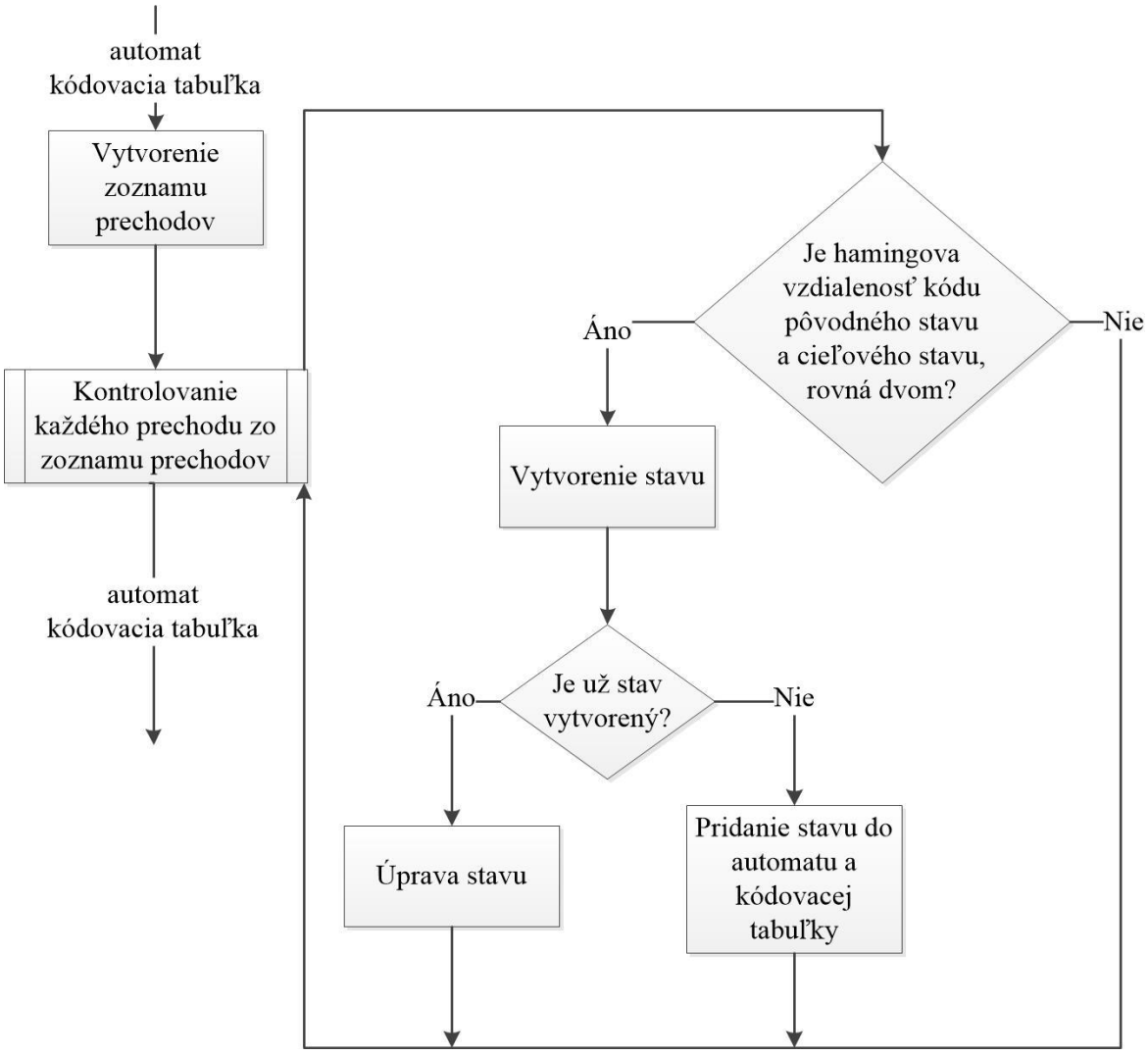
Na začiatku sa vyberie najmenší zoznam úplných dichotómií zo zoznamu úplných dichotómií. Pre vybraný zoznam platí, že všetky použité dichotómie pokrývajú všetky časti z minimálneho zoznamu dichotómií. Na základe vybraných dichotómií sa vytvoria zakódovania. Ich počet je 2^N, kde N je počet vybratých úplných dichotómií. Celý postup je uvedený na obrázku 6.8.



Obrázok 6.8 Realizácia zakódovania stavov automatu

6.1.4 Implementácia ošetrenia kritických súbehov

Ošetrenie kritických súbehov alebo doručenie prechodovej funkcie vytvorí rozšírený automat a zakódovanie pre rozšírený automat, kde rozdiely v kóde vo všetkých prechodoch sú v maximálne jednej stavovej premennej. Na vstupe je pôvodný automat a pôvodné zakódovanie stavov. Pre vstupný automat sa vytvorí zoznam všetkých prechodov so štruktúrou **transition,** opísanej v 6.1.2. Následne sa pre každý prechod zo zoznamu kontroluje kód stavu, kde prechod začína a kód stavu, kde prechod končí. Ak sa kódy menia v dvoch stavových premenných, tak sa pridajú dva pomocné stavy, ktorých zakódovanie je také, že pri prechodoch sa môže meniť maximálne jedna stavová premenná. Prechodový predpis stavu obsahuje len cieľový stav prechodu, a to pri tom vstupe, pri ktorom sa prechod aktivuje. Výstupná funkcia prechodu je zhodná so stavom, z ktorého začína prechod. Použitý podrobný algoritmus je na obrázku 6.9.



Obrázok 6.9 Implementácia doručenia prechodovej funkcie

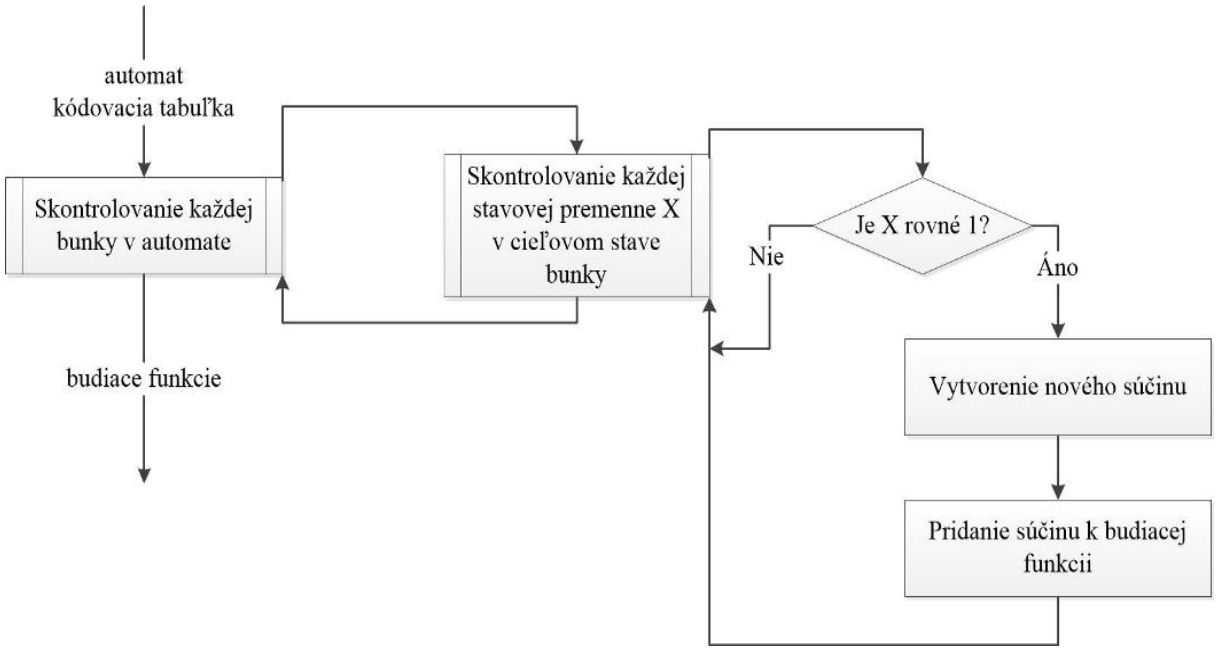
6.1.5 Implementácia vytvorenia budiacich funkcií pre pamäťovú časť

Budiace funkcie preklápacích obvodov sa vytvárajú do štruktúry **ff_function,** ktorá je uvedená v tabuľke 6.6. Použitá štruktúra **product** je zoznam premenných, ktorý reprezentuje jeden súčin.

Tabuľka 6.6 Štruktúra ff_function

ff_function		
Názov	Typ	Popis
products	**product	Zoznam súčinov.
count	int	Počet súčinov.

Počet budiacich funkcií ja rovnaký ako počet stavových premenných. Algoritmus použitý pri vytváraní budiacich funkcií je uvedený na obrázku 6.10.



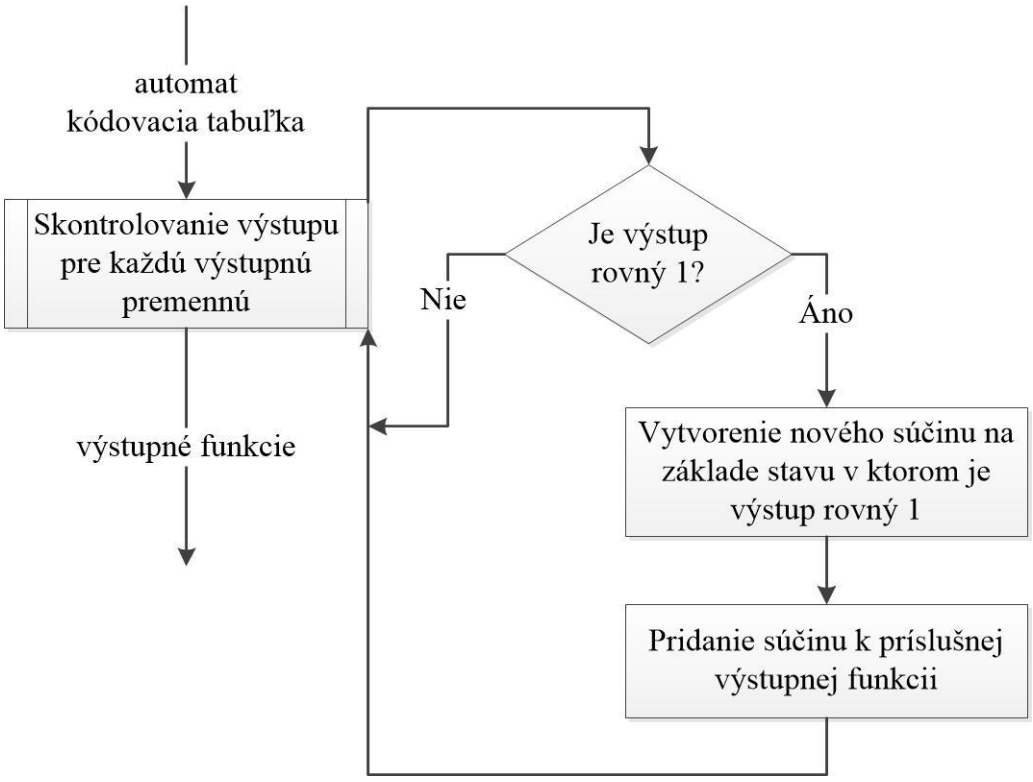
Obrázok 6.10 Realizácia vytvorenia budiacich funkcií

6.1.6 Realizácia transformácie budiacich funkcií pre preklápacie obvody typu JK

Prepis funkcií pre preklápacie obvody typu JK je podľa tabuľky 5.1. Budiace funkcie sú ukladané do štruktúry **ff_function**, ktorá je uvedená v časti 6.1.5. Počet budiacich funkcií pre preklápacie obvody typu JK je dvojnásobný, oproti počtu pôvodných budiacich funkcií. Je to preto lebo preklápacie obvody typu JK, majú dva vstupy. Výstupom je zoznam budiacich funkcií reprezentovaný zoznamom štruktúra **ff_function.**

6.1.7 Implementácia vytvorenia výstupných funkcií

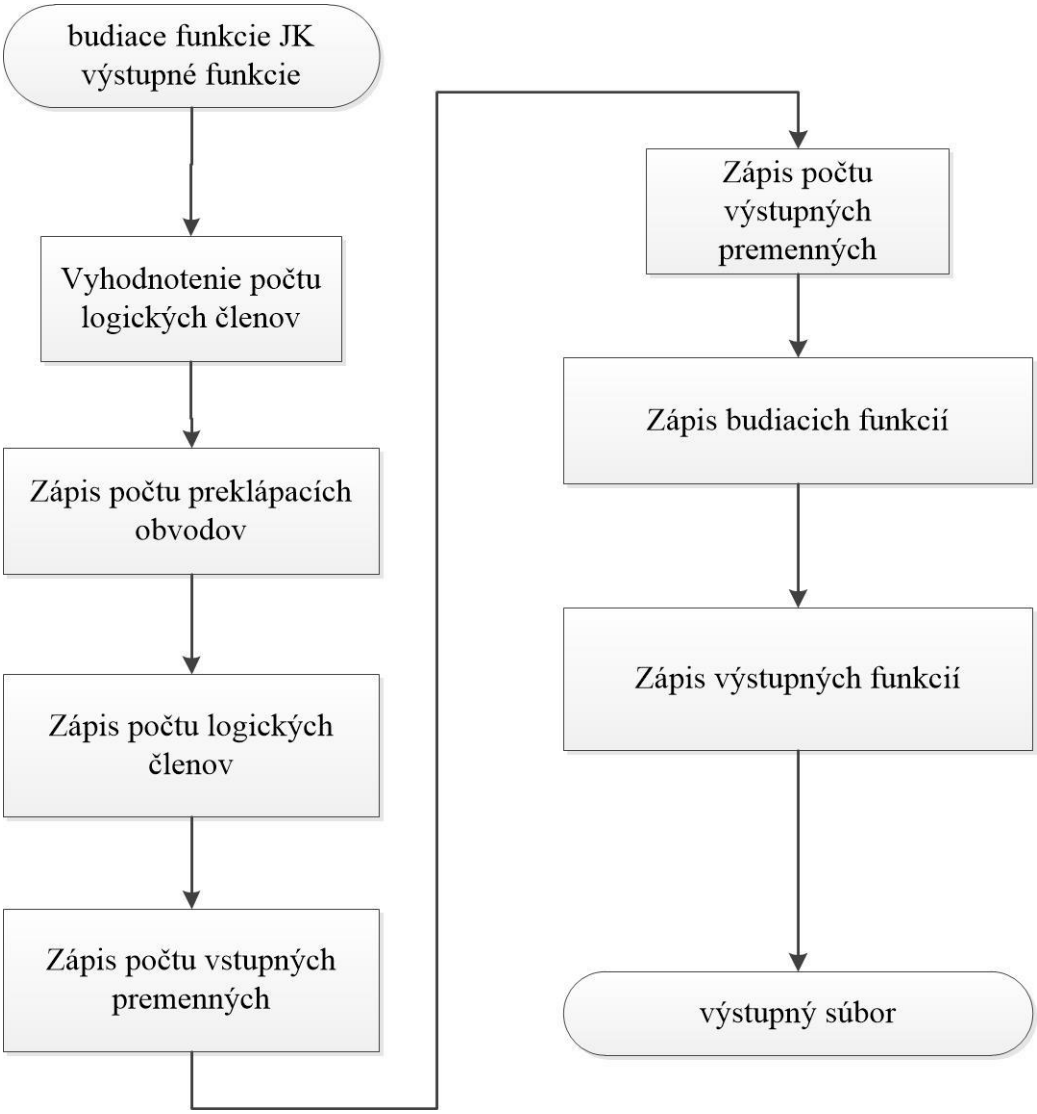
Predposledná časť programu vytvorí výstupné funkcie, ktoré opisujú realizáciu výstupnej kombinačnej logiky. Výstupné funkcie sú vytvárané na základe zakódovanie stavov, v ktorých príslušná výstupná premenná nadobúda logickú hodnotu jeden. Počet výstupných funkcií je zhodný s počtom výstupných premenných a na uloženie je použitá štruktúra **ff_function,** ktorá je opísaná v časti 6.1.5. Realizácia je uvedená na obrázku 6.11.



Obrázok 6.11 Vytvorenie výstupných funkcií

6.1.8 Implementácia vyhodnotenia výsledku a zápis výstupu

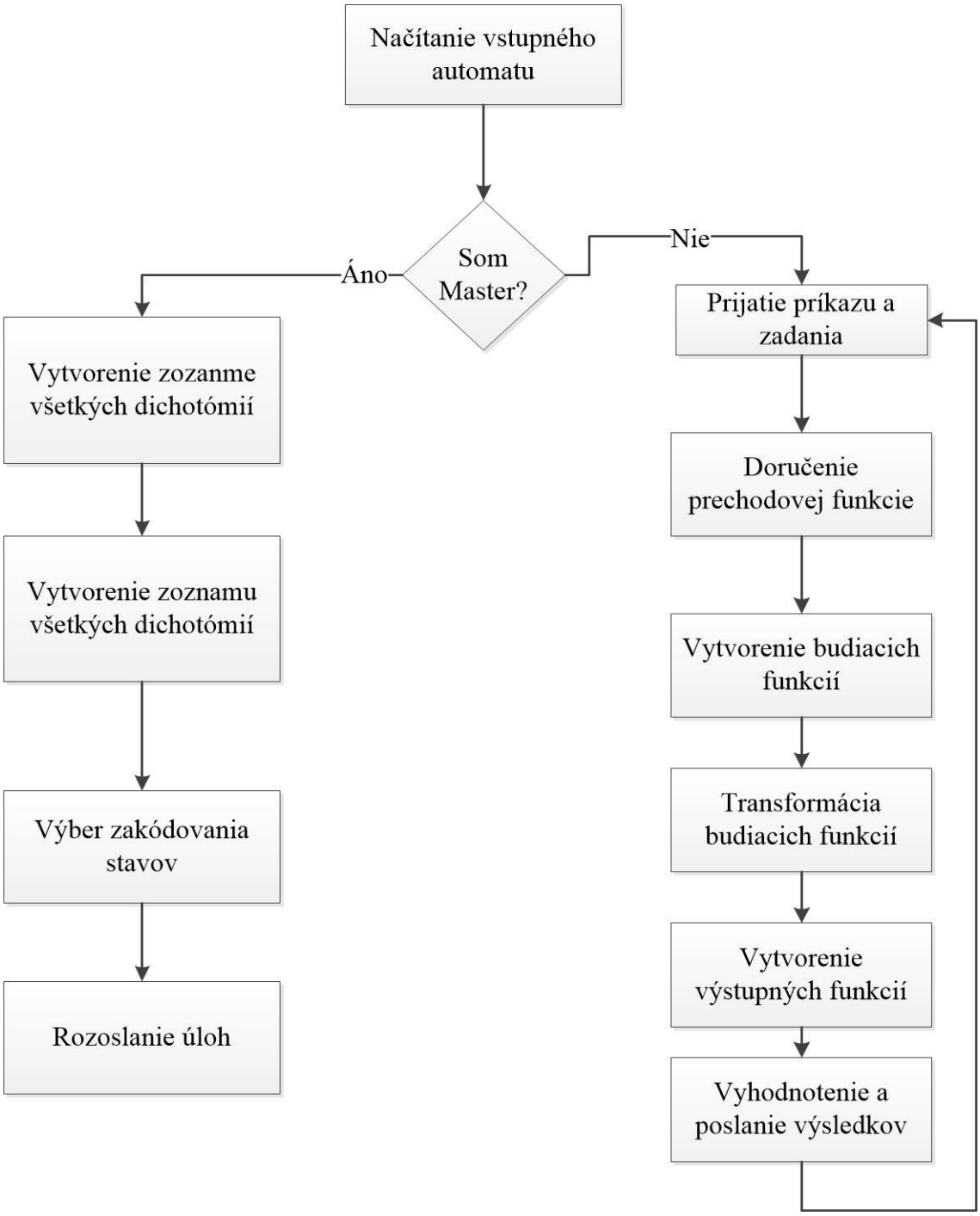
V poslednej časti programu sa spočíta počet použitých logických členov. Ak je počet najmenší, tak sa zapíše výsledná zostava funkcií. Toto vyhodnotenie prebieha pre všetky navrhnuté zakódovania, aby sa našla reprezentácia s najnižším počtom preklápacích obvodov. Vstupom do poslednej časti sú reprezentácie automatu pomocou funkcií v štruktúrach **ff_function**. Po nájdení reprezentácie s najmenším počtom logických členov je potrebné vytvoriť výstupný súbor, ktorý je vo formáte opísanom v kapitole 5.3.5. Postup vytvorenia výstupného súboru je uvedený na obrázku 6.12.



Obrázok 6.12 Realizácia vytvorenia výstupného súboru

6.2 Implementácia programu pre vysokovýkonný výpočtový systém

Program, implementovaný na vysokovýkonnom výpočtovom systéme, využíva architektúru **Master - Slave** a pre komunikáciu využíva komunikačné prostredie **MPI**. Program je jeden a na začiatku vykonávania programu sa rozhodne, či daný výpočtový uzol bude pracovať ako **Master** alebo ako **Slave.** Po spustený programu a inicializácii komunikačného prostredia **MPI** sa priradí každému programu unikátne identifikačné číslo. Po uistení unikátneho identifikačného čísla sa program rozhodne, či bude pracovať ako **Master** alebo ako **Slave.** Ak je identifikačné číslo rovné 0, tak program bude vykonávať časť **Master** a ak je identifikačné číslo iné, tak bude program pracovať ako **Slave.** Zloženie programu je uvedené na obrázku 6.13.



Obrázok 6.13 Realizácia programu na vysokovýkonnom výpočtovom systéme

6.2.1 Komunikačné prostredie MPI

Komunikačné prostredie **MPI** *("message passing interface")* je systém, ktorý poskytuje prostredie na komunikáciu medzi výpočtovými uzlami v distribuovanom počítaní na vysokovýkonných výpočtových systémoch. Posúvanie správ je spôsob komunikácie

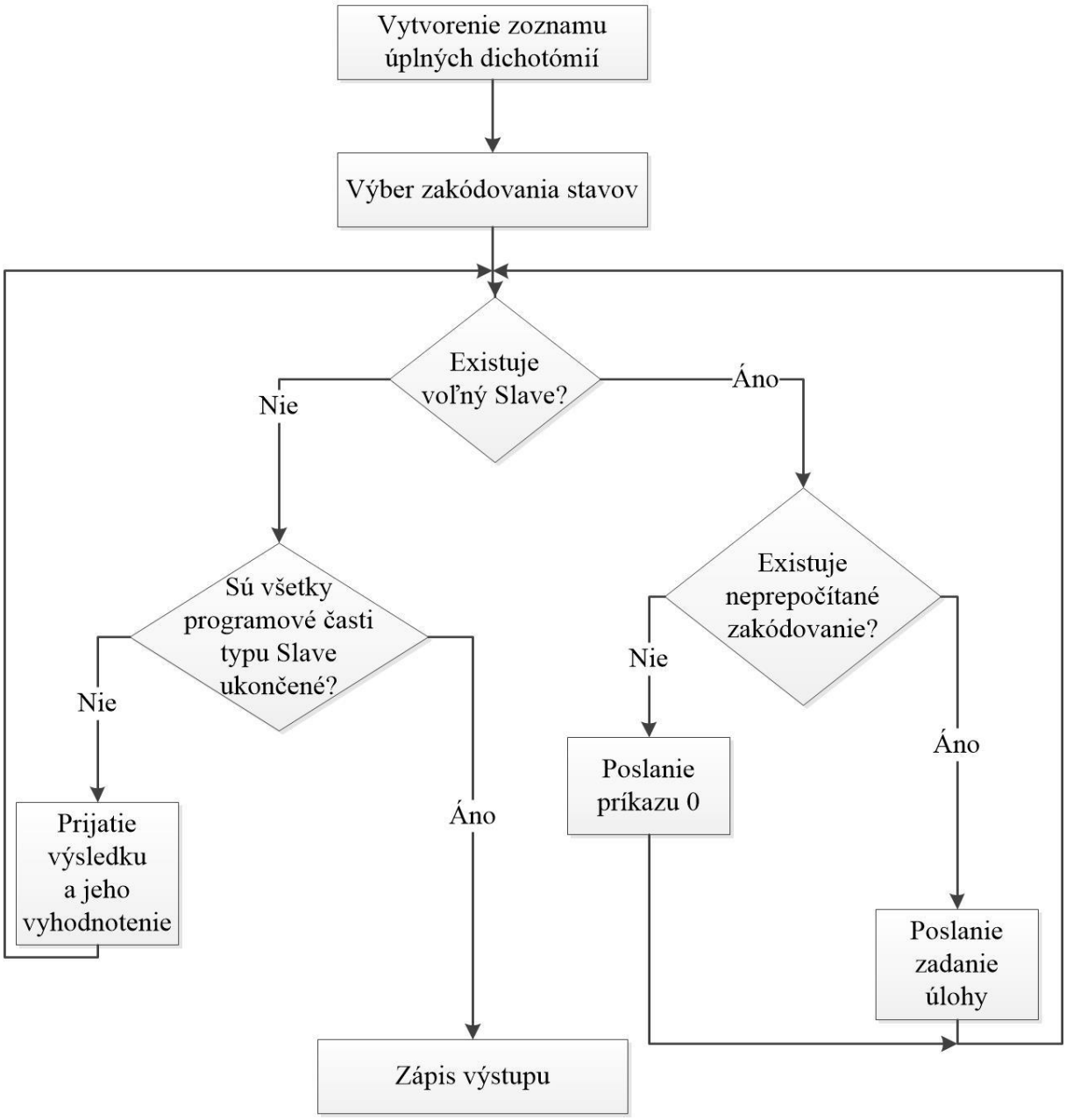
používaný v paralelných vysokovýkonných systémoch. Cieľom **MPI** je vytvorenie štandardu pre písanie programov, ktoré využívajú posúvanie správ. Systém má byť efektívny, flexibilný a multiplatformový. Kompletný zoznam cieľov je uvedený nižšie:

- Návrh programového prostredia pre distribuované systémy.
- Efektívna komunikácia - zabránenie viacnásobnému kopírovaniu rovnakých údajov v pamäti.
- Implementácia v heterogénnom prostredí - implementovať **MPI** na viacerých vývojových platformách a umožniť komunikáciu medzi nimi.
- Umožnenie spoľahlivej komunikácia - používateľ nemôže riešiť chyby v doručení správ. Používateľ má zabezpečené správne doručenie správy.
- Návrh systému, ktorý môže byť jednoducho implementovaný viacerými výrobcami vysokovýkonných výpočtových systémov.
- Navrhnutý systém musí byť nezávislý na jazyku, v ktorom je **MPI** implementované.

Štandard **MPI** je určený pre používateľov na písanie distribuovaných programov pre vysokovýkonné výpočtové systémy. **MPI** poskytuje jednoduchý, vysokovýkonný systém pre posúvanie správ [8].

6.2.2 Implementácia časti programu Master

Implementácia programovej časti **Master**, ktorý sa vykonáva na vysokovýkonnom výpočtovom systéme práve jedenkrát, je uvedená na obrázku 6.14.



Obrázok 6.14 Implementácia programovej časti Master

Úlohou programovej časti **Master** je vypočítať možné zakódovania stavov a potom rozdeliť úlohy medzi ostatné uzly, na ktorých sa vykonáva programová časť **Slave.** Po dokončení úlohy programovou časťou **Slave**, **Master** vyhodnotí výsledky a výslednú reprezentáciu zapíše do súboru vo formáte, ktorý je definovaný v kapitole 5.3.5. Ako je uvedené na obrázku 6.14, programová časť **Master** je rozdelená do viacerých častí:

1. Vytvorenie zoznamu úplných dichotómií - vstupom do druhej časti programu je štruktúra **automat.** Výstupom je zoznam úplných dichotómií, ktoré sú uložené v štruktúre **dichotomies.** Táto časť programu je opísaná v časti 6.1.2.
2. Zakódovanie stavov - výber zakódovania stavov v automate je úlohou tretej časti programu. Je vybraných veľa kombinácií zakódovania stavov. Táto časť programu je totožná s treťou časťou programu sekvenčného riešenia, ktorá je opísaná v časti 6.1.3.
3. Rozoslanie úloh - zadanie činnosti pre podprogramy type **Slave.** Kým nie sú spracované všetky pripravené kódovania stavov v automate, tak posielaj úlohy. Typy príkazov sú uvedené v tabuľke 6.7.

Tabuľka 6.7 Typy príkazov

Príkaz		
Označenie	Typ	Popis
0	int	Ukonči podprogram.
1	int	Čakaj na zadanie posielam.
2	int	Pošli celý posledný výsledok.

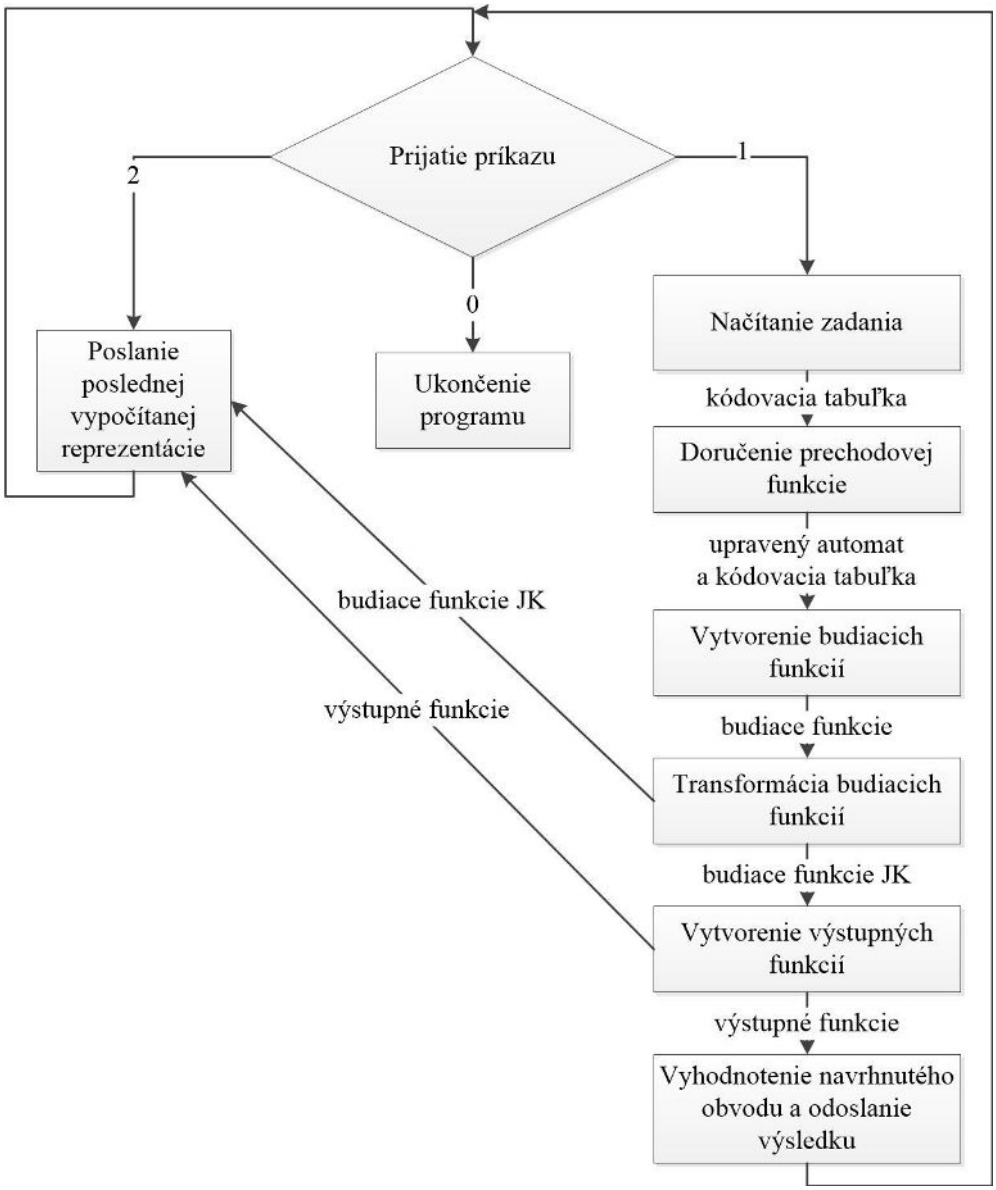
Po zadní príkazu typu 1, pošle **Master** dva údaje typu int o počte riadkov a počte stĺpcov v kódovacej tabuľke a následne pošle dvojrozmerné pole, ktoré reprezentuje kódovaciu tabuľku.

Po zadaní príkazu typu 2, program **Master** čaká na prijatie reprezentácie automatu, ktorá je v textovom formáte.

4. Prijatie výsledkov a ich vyhodnotenie - po prijatí výsledku od niektorého podpgoramu typu **Slave** program vyhodnotí, či je počet logických členov v riešení najmenší ak áno, nasleduje štvrtá časť podprogramu s príkazom 2. Keď sú zozbierané všetky výsledky od podprogramov typu **Slave,** tak sa im pošle ukončujúci príkaz.
5. Zapísanie výsledku do súboru - posledná časť programu, zapíše realizáciu automatu s najmenším počtom logických obvodov do výstupného súboru podľa formátu, ktorý je opísaný v kapitole 5.3.5. Posledná časť je implementovaná rovnako ako posledná časť sekvenčného programu, ktorá je opísaná v kapitole 6.1.8.

6.2.3 Implementácia časti programu Slave

Programová časť **Slave** zabezpečuje výpočet výslednej reprezentácie automatu logickými obvodmi. Je spustená mnohonásobne na viacerých výpočtových uzloch na vysokovýkonnom výpočtovom systéme. Každý program, v ktorom sa vykonáva programová časť **Slave,** má svoje identifikačné číslo v komunikačnej množine **MPI,** ktoré je rozdielne od 0. Každý podprogram môže posielať správy cez systém **MPI.** Adresovateľným programom je len program, pod ktorým beží programová časť **Master.** Programy s programovou časťou **Slave** sa nemôžu adresovať. Ako je znázornené na obrázku 6.15, celý podprogram **Slave** sa vykonáva dovtedy, kým nepríde príkaz na ukončenie typu 0.



Obrázok 6.15 Realizácia programovej časti Slave

Jednotlivé časti realizovanej programovej časti **Master** sú:

1. Načítanie zadania - na začiatku sa načíta automat zo vstupného súboru ešte pre vykonávaním cyklu programovej časti **Slave.** Následne sa čaká na správu od programu s programovou časťou **Master**, ktorý zadá úlohu.

    - Ak príde príkaz 0 , potom sa program ukončí.
    - Ak príde príkaz 1, program bude načítavať úlohu, čiže prijme dva čísla, kde prvé číslo označuje počet riadkov a druhé počet stĺpcov. Následne načíta kódovaciu mapu do štruktúry **code** a program pokračuje vo vykonávaní druhou časťou.
    - Po prijatí príkazu 2, posledné budiace funkcie a výstupné funkcie zapíše do textu, vo formáte opísanom v časti 5.3.5, a pošle ho programu s programovou časťou **Master.** Následne opäť čaká na príkaz od programovej časti **Master.**

2. Doručenie prechodovej funkcie - Druhá časť je totožná so štvrtou časťou sekvenčného algoritmu, ktorá je opísaná v kapitole 6.1.4.
3. Vytvorenie budiacich funkcií - vytvorenie budiacich funkcií je úlohou tretej časti algoritmu. Výstupom sú štruktúry, ktoré reprezentujú budiace funkcie pamäťovej časti. Tretia časť je totožná s piatou časťou sekvenčného programu, ktorá je opísaná v kapitole 6.1.5.
4. Transformácia budiacich funkcií - prepis budiacich funkcií pre preklápacie obvody JK je totožný s prepisom opísaným v šiestej časti sekvenčného programu, ktorá je uvedená v kapitole 6.1.6.
5. Vytvorenie výstupných funkcií - piata časť programovej časti **Slave** na vysokovýkonnom výpočtovom systéme, je rovnaká ako siedma časť sekvenčného algoritmu, ktorá je opísaná v kapitole 6.1.7.
6. Odoslanie výsledku - posledná časť programu pre programovú časť **Slave**, spočíva v spočítaní počtu logických členov navrhnutej reprezentácie rozšíreného automatu. Po spočítaní počtu logických členov sa hodnota pošle späť programu s programovou časťou **Master**. Po dokončení šiestej časti sa znovu prejde na prvú časť.

7 Overenie programových systémov

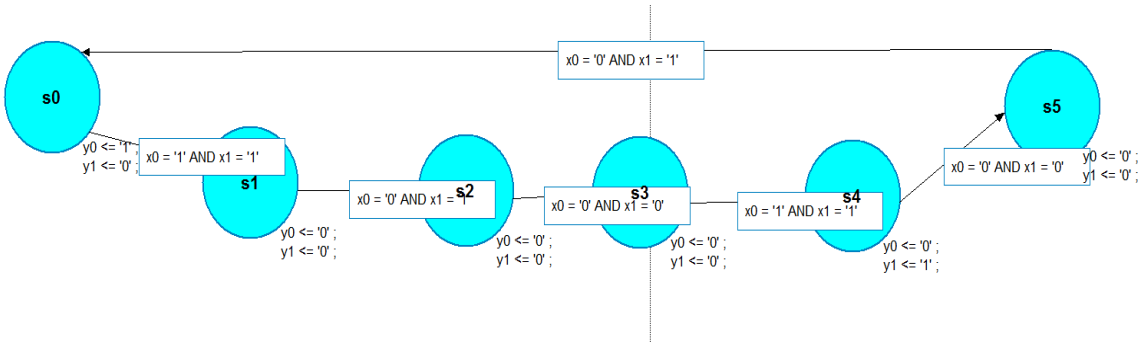
Overenie programového systému prebiehalo vo viacerých fázach. Pomocou logovacích výstupov z programov boli overené čiastkové výstupy, ktoré sa zhodovali s očakávanými výstupmi. Záverečné testovanie je možné rozdeliť do dvoch častí. Prvou časťou je testovanie funkčnosti, v ktorej je overená správnosť výsledku a preukázané správne správanie sa navrhnutej realizácie automatu. V druhej časti sa porovnáva výkon jednoprocesorového programu s výkonom programu pre vysokovýkonný výpočtový systém.

7.1 Testovanie funkčnosti

Testovanie funkčnosti bolo vykonané v dvoch častiach. V prvej časti boli otestovaný jednotlivé časti programu osobitne. Pomocou pomocných výstupov programu bola overená funkčnosť každej implementovanej časti programu. V druhej časti bol otestovaný celý programový systém.

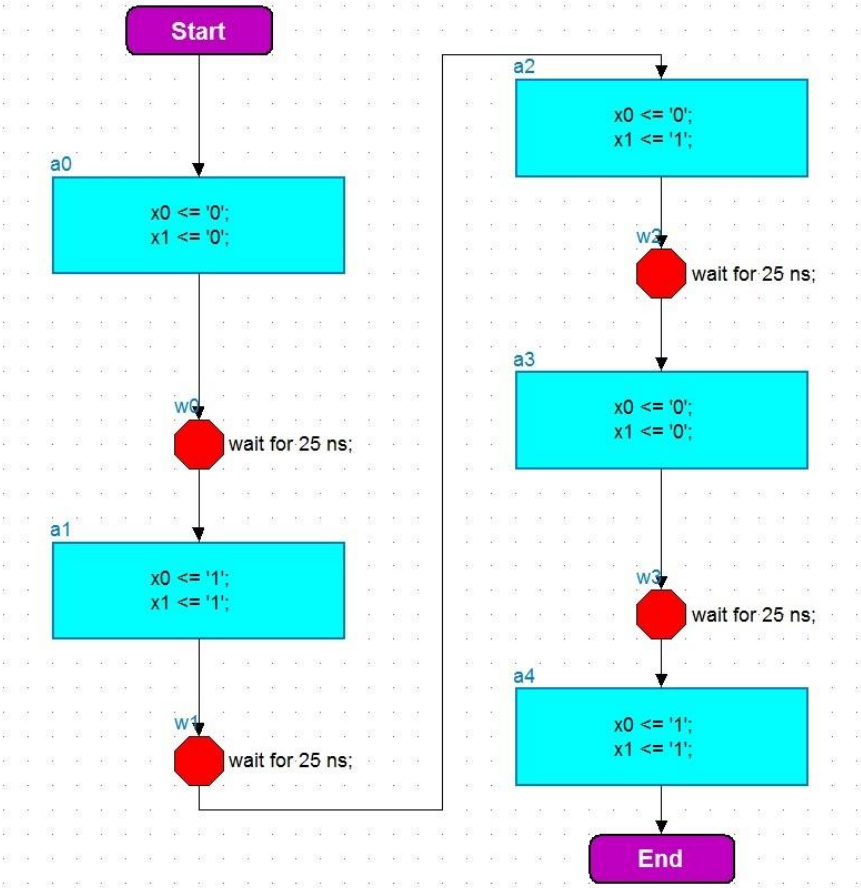
7.1.1 Metóda testovania

Testovanie celého systému prebiehalo pomocou prístupu čierna skrinka (*black box*), ktorý neberie do úvahy medzivýsledky a postupné spracovávanie. Na základe vstupných údajov očakáva výstupné údaje. Ak sú reálne výstupné údaje zhodné s očakávanými výstupnými údajmi, tak test prebehol úspešne, ak nie sú zhodné, test neprebehol úspešne.
Poskytnutým vstupom je súbor vo formáte **VHDL**, ktorý je vytvorený návrhovým systémom **HDL Designer.** Na obrázku 7.1 je automat, ktorý je vstupom testovania funkčnosti. Automat musí mať pre každý prechod uvedené všetky vstupné premenné a každý stav musí mať uvedené všetky výstupné premenné.



Obrázok 7.1 Grafický opis vstupného automatu

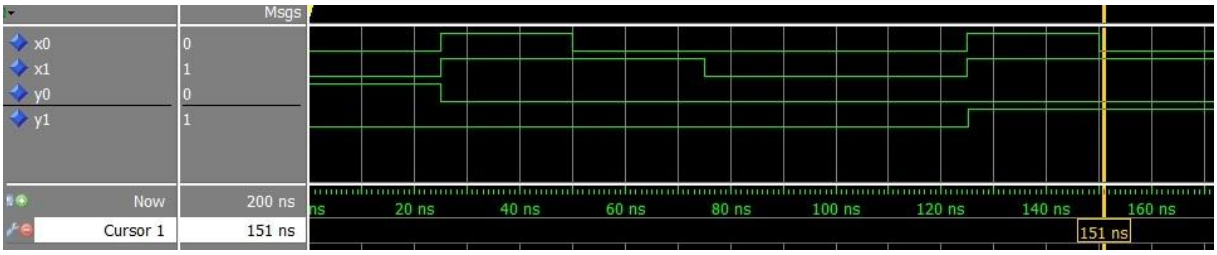
Po vytvorení vstupného súboru v návrhovom systéme **HDL Designer** je spustený obslužný program, ktorý načíta súbor s opisom automatu a vytvorí formátovaný vstup, ktorý sa následne pošle na vysokovýkonný výpočtový systéme a spustí sa program na vysokovýkonnom výpočtovom systéme. Po získaní výsledkov sa súbor s výsledkami načíta do obslužného programu a vytvorí sa končený výstupný súbor s opisom automatu vo formáte **VHDL.** Následne je opis odsimulovaný v simulačnom prostredí **ModelSim.** Pre výslednú realizáciu automatu je vytvorené testovacie okolie, ktoré simuluje vstupy do automatu. Testovacie okolie pre automat uvedený na obrázku 7.1 je opísané na obrázku 7.2.



Obrázok 7.2 Testovacie prostredie

7.1.2 Experimentálne výsledky

Výsledný opis architektúry, bol vložený do testovacieho prostredia a správanie sa navrhnutého automatu je na obrázku 7.3.



Obrázok 7.3 Správanie navrhnutého automatu

Správanie sa navrhnutej architektúry automatu sa zhoduje s opisom správania sa automatu zo zadania. V počiatočnom stave ak je vstup rozdielny od vstupného vektora (0,0), tak sa stav nemení. Ak príde na vstup vektor (1,1), tak automat prejde do ďalšieho stavu a upraví sa aj výstupný vektor, ktorý zo stavu (1,0) prejde do stavu (0,0).

7.2 Testovanie efektívnosti

Testovanie efektívnosti pozostáva z meraní času, za ktorý navrhnutý programový systém poskytne výsledok. Následne je vypočítané zrýchlenie vzhľadom na veľkosť automatu a počet uzlov, na ktorých je spustená programová časť **Slave**.

7.2.1 Metóda testovania

Zrýchlenie je definované podľa vzťahu (7.1).

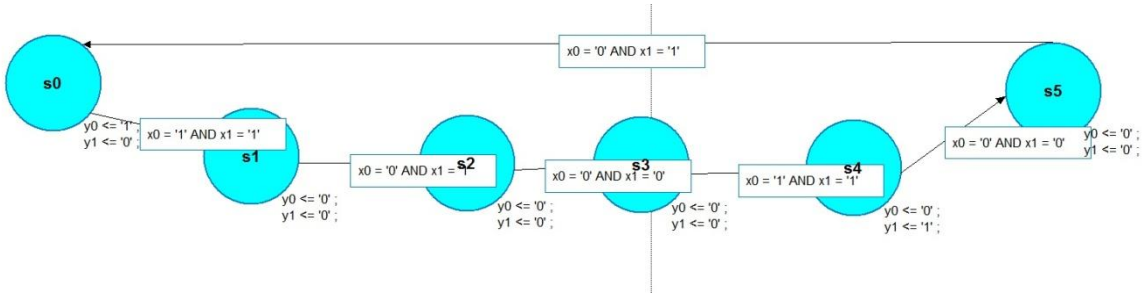
$$Zrýchlenie = \left(\frac{t_1}{t_n} - 1\right) * 100 \tag{7.1}$$

t_1 - čas výpočtu na jednoprocesorovom systéme

t_n - čas výpočtu na vysokovýkonnom výpočtovom systéme s n uzlami

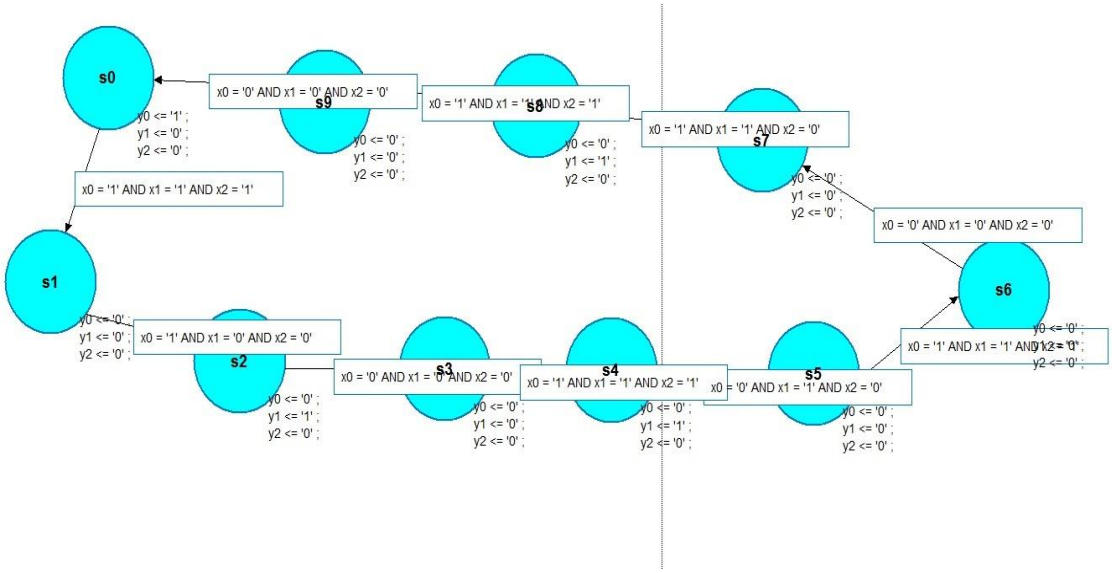
Každé meranie je vykonané päť krát a následne je vypočítaný priemer z týchto meraní.

Pre meranie zrýchlenia vzhľadom na veľkosť automatu boli vytvorené tri automaty s rozdielnou veľkosťou. Veľkosť automatu je pre potreby testovania určená ako veľkosť štruktúry, ktorá je vytvorená po načítaní automatu z formátovaného vstupu.
Prvý automat je opísaný na obrázku 7.4, a veľkosť tabuľky je 36.



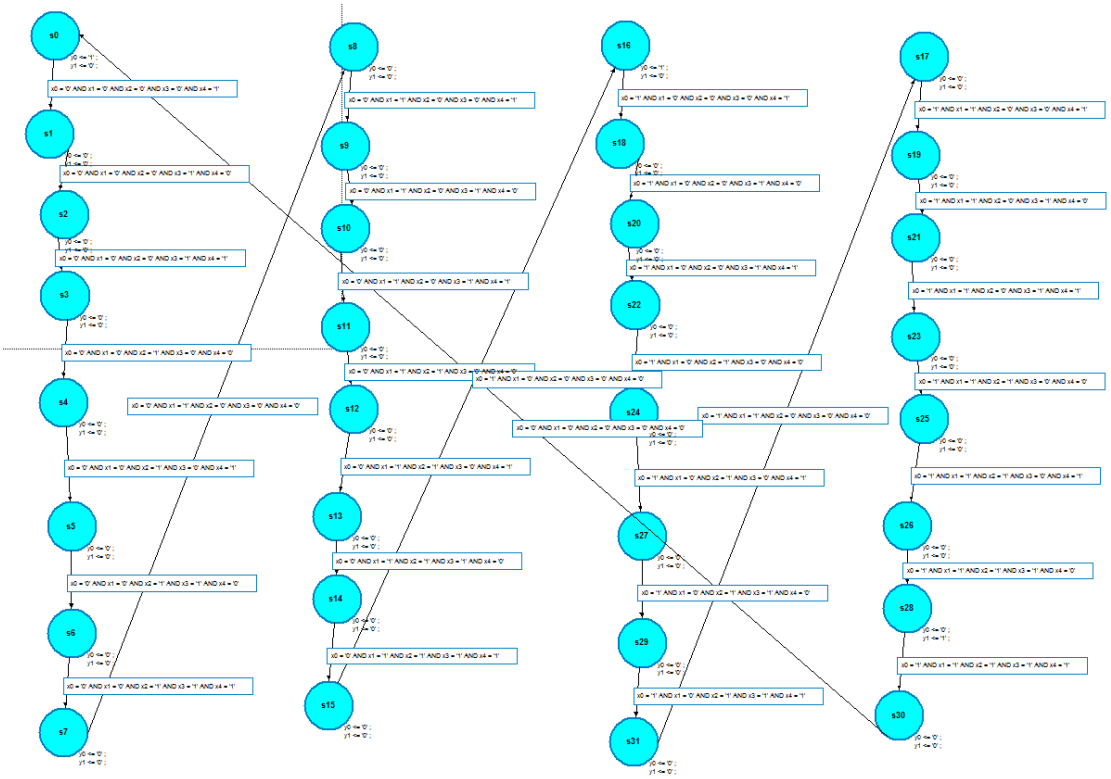
Obrázok 7.4 Prvý automat s veľkosťou 36

Druhý automat opísaný na obrázku 7.5, má veľkosť 120.



Obrázok 7.5 Druhý automat s veľkosťou 120

Tretí automat je uvedený na obrázku 7.6, jeho veľkosť je 576.



Obrázok 7.6 Tretí automat s veľkosťou 576

Zvolené počty uzlov pre meranie zrýchlenia vzhľadom na počet uzlov, sú: 2, 8, 32, 128, 512.

7.2.2 Experimentálne výsledky

Namerané hodnoty pre prvý automat sú v tabuľke 7.1 v sekundách.

Tabuľka 7.1 Namerané hodnoty pre prvú veľkosť úlohy

	Počet výpočtových uzlov					
	1	2	8	32	128	512
Meranie 1	0,68	2,18	1,71	1,85	2,13	3,15
Meranie 2	0,55	2,10	1,69	1,75	2,12	3,26
Meranie 3	0,79	2,16	1,86	1,62	2,10	3,01
Meranie 4	0,63	2,12	1,54	1,82	2,09	2,89
Meranie 5	0,96	2,11	2,01	1,93	2,03	3,05
Priemer	0,72	2,13	1,76	1,79	2,09	3,07

Výpočtový čas s narastajúcim počtom výpočtových uzlov narastá, čo je spôsobené malou veľkosťou úlohy. Veľa času sa spotrebuje na komunikáciu medzi uzlami, hoci uzly nemajú čo počítať.

Namerané hodnoty pre druhý automat sú uvedené v tabuľke 7.2 v sekundách. Vidíme zníženie výpočtového času s narastajúcim počtom uzlov.

Tabuľka 7.2 Namerané hodnoty pre druhú veľkosť úlohy

	Počet výpočtových uzlov					
	1	2	8	32	128	512
Meranie 1	10,60	9,16	5,23	2,36	1,61	1,35
Meranie 2	11,80	9,14	4,13	2,86	1,34	1,40
Meranie 3	10,56	9,52	4,81	2,45	1,45	1,32
Meranie 4	12,80	8,96	4,56	2,63	1,23	1,26
Meranie 5	10,96	8,17	4,69	2,40	1,45	1,45
Priemer	11,34	8,99	4,68	2,54	1,42	1,36

Namerané hodnoty pre tretí automat sú uvedené v tabuľka 7.3 v sekundách. S narastajúcim počtom uzlov sa výpočtový čas značne zmenšuje z dôvodu dostatočného množstva úlohy pre paralelné počítanie.

Tabuľka 7.3 Namerané hodnoty pre tretiu veľkosť úlohy

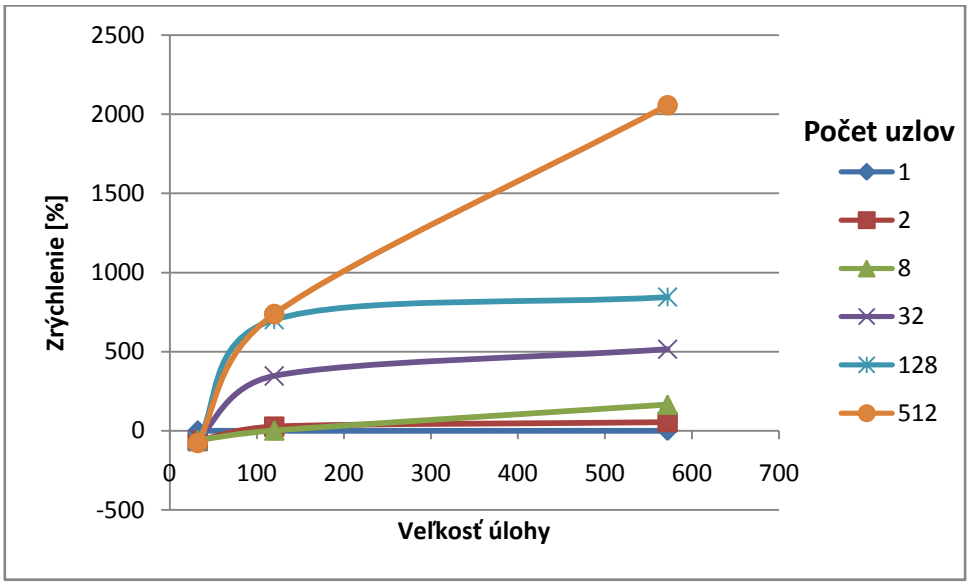
	Počet výpočtových uzlov					
	1	2	8	32	128	512
Meranie 1	3652	2010	1124	532	361	154
Meranie 2	3241	2364	1036	561	345	145
Meranie 3	3361	2247	1698	548	326	161
Meranie 4	3451	2145	1241	491	381	158
Meranie 5	3324	2269	1342	635	391	172
Priemer	3406	2207	1288	553	361	158

Vypočítané zrýchlenia sú uvedené v tabuľke 7.4 v percentách. Zrýchlenie pre najmenšiu veľkosť problému je záporné. Je to spôsobené veľkou réžiou pre komunikáciu a malým množstvom výpočtu, ktorý sa môže počítať paralelne. Ako zložitosť automatu rastie, rastie aj zrýchlenie.

Tabuľka 7.4 Vypočítané zrýchlenia

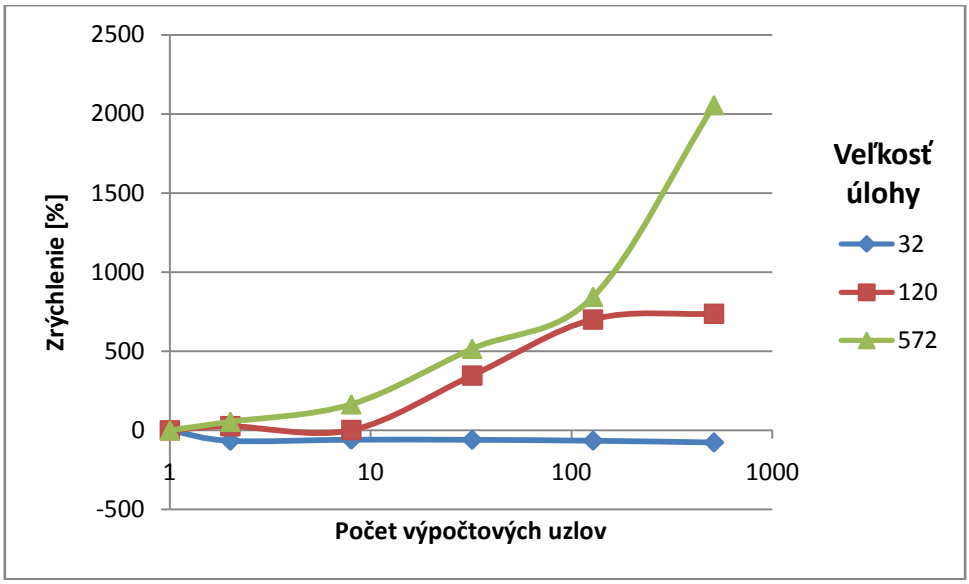
Zložitosť automatu	Počet výpočtových uzlov					
	1	2	8	32	128	512
32	0	-66	-59	-60	-66	-76
120	0	26	1	347	701	737
572	0	54	164	515	844	2056

Vývoj zrýchlenia v závislosti od veľkosti úlohy je uvedený v grafe na obrázku 7.7. Zrýchlenie rastie pre rovnaký počet výpočtových uzlov s veľkosťou úlohy. To znamená, že so zväčšovaním veľkosti úlohy, rýchlejšie rastie množstvo paralelne počítaných výpočtov.



Obrázok 7.7 Veľkosť zrýchlenia v závislosti od veľkosti úlohy

Závislosť zrýchlenia od počtu výpočtových uzlov je uvedená v grafe na obrázku 7.8. Negatívne zrýchlenie je pri nízkej veľkosti úlohy spôsobený malým počtom výpočtov, ktoré by mohli byť počítané paralelne. Pre väčšie veľkosti úloh zrýchlenie rastie s počtom uzlov, na ktorých prebieha výpočet.



Obrázok 7.8 Veľkosť zrýchlenia v závislosti od počtu výpočtových uzlov

8 Záver

Na základe analýzy metód syntézy asynchrónnych sekvenčných obvodov bola vybratá realizácia pomocou priameho kódu. Metóda návrhu pomocou priameho kódu bola formálne opísaná, čo umožnilo rozdeliť návrh algoritmu do viacerých samostatných častí. Navrhnutý programový systém sa skladá z troch častí. Prvá časť je sekvenčný algoritmus pre jednoprocesorový výpočtový prostriedok, ktorý je implementovaný v jazyku C. Druhá časť programového systému je navrhnutá pre vysokovýkonný výpočtový systém pomocou architektúry **Master - Slave**. Implementácia druhej časti je tiež v jazyku C, s použitím komunikačného prostredia **MPI.** Obslužný program tvorí tretiu časť programového systému. Nestará sa len o vytváranie formátovaných vstupov a výstup, ale aj o komunikáciu medzi jednotlivými časťami programového systému.

Overenie funkčnosti programového systému bolo vykonané pomocou testovacieho prístupu „čierna skrinka“, kde výstupy z programového systému boli zhodné s predpokladmi. Následná simulácia v simulačnom prostredí bola úspešná. Experimentálne merania zrýchlenia programu na vysokovýkonnom výpočtovom systéme oproti programu na jednoprocesorovom systéme ukázali, že čas výpočtu klesá so zvyšujúcim sa počtom výpočtových uzlov. Zrýchlenie je obmedzené časťou implementovaného algoritmu, ktorá sa nedá paralelizovať a podľa Amdahlovho zákona [9] je obmedzené určitou hranicou, aj keby sa zvyšoval počet výpočtových uzlov do nekonečna. Ak ale s pridávaním uzlov sa zväčšujú aj úlohy, čiže vstupný automat, hranica zrýchlenia sa zvýši. Znamená to, že zrýchlenie je vyššie pri väčšom automate, čo bolo ukázané pri experimentálnych meraniach. Z experimentálnych meraní je možné vidieť, že časť, ktorá nebola paralelizovaná je dosť veľká, keďže zrýchlenie s rastúcim počtom výpočtových uzloch rastie pomaly pri malých úlohách.

Návrh programového systému je možné ešte vylepšiť podrobnou analýzou použitých algoritmov a následne optimalizáciou implementácie navrhnutých algoritmov. Napríklad, zredukovanie zoznamov, ktoré treba skontrolovať, pri vytváraní budiacich funkcií.
Zmenšením granularity paralelnej časti programového systému je možné lepšie rozdistribuovať výpočet medzi viaceré uzly. Nevýhodou zníženia granularity je väčšia réžia posielania správy medzi jednotlivými uzlami.
Funkcionalitu obslužného programu je možné rozšíriť pridaním modulu pre transformáciu automatu typu Melay do automatu typu Moore a naopak.

Literatúra

[1] SPARSO, J., FURBER, S.: Principles of Asynchronous Circuit Design: A Systems Prespective. Kluwer Academic Publishers, Boston/Dordrecht/London, 2001, 337 p.

[2] UNGER, S.H.: Asynchronous Sequential Circuits. J. Wiley Int., New York 1969, 290 p.

[3] FRIŠTACKÝ, N., KOLESÁR,M. a i.: LOGICKÉ SYSTÉMY . Syntéza asynchrónnych sekvenčných obvodov. Bratislava SVŠT 1986, p. 458-484. ISBN 80-05-00414-1

[4] MOČOL, M.: Programový systém podporujúci syntézu asynchrónnych sekvenčných obvodov. Bakalárska práca FIIT STU, 2005, 18 p.

[5] KUDLAČÁK, F.: Programový systém podporujúci syntézu asynchrónnych sekvenčných obvodov. Bakalárska práca FIIT STU, 2011, 45 p.

[6] JÁNEŠ, V., DOUŠKA, J : Logické systémy. ČVUT Praha 2001. ISBN 80-01-018180, 299 p.

[7] CORTADELLA, J.: Asynchronous circuit verification and synthesis with Petri nets. *Workshop on Hardware Design and Petri Nets (HWPN'98),* Lisbon, 1998, 10 p.

[8] MPI: A Message-Passing Interface Standard. [Online; 14 máj 2013] Prístupné na: http://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf

[9] AMDAHL, G.: Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities. *AFIPS Conference Proceedings (30),* USA, 1967, p. 483-485.

Plán práce

Letný semester 2011/2012

- Analýza stavu problémovej oblasti.
- Analýza metód pre návrh asynchrónnych sekvenčných obvodov.
- Analýza podobných riešení.
- Podrobná špecifikácia.

Zimný semester 2012/2013

- Špecifikácia požiadaviek na programový systém.
- Formálny opis implementovanej metódy.
- Návrh programu pre jednojadrový systém.
- Návrh programu pre vysokovýkonný výpočtový systém.
- Návrh obslužného programu.
- Implementácia jednojadrového programu.
- Implementácia programu pre školský vysokovýkonný výpočtový systém - prototyp.

Letný semester 2012/2013

- Implementácie programu pre školský vysokovýkonný výpočtový systém.
- Implementácia programu pre vysokovýkonný výpočtový systém na SAV.
- Implementácia obslužného programu.
- Testovanie funkčnosti programového systému.
- Testovanie efektívnosti programového systému.
- Dokončenie dokumentácie a vyhodnotenie meraní.