Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Metody kódování stavů synchronních automatů

Plzeň 2013 Monika Kovářová

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracovala samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 6. května 2013

Monika Kovářová

Poděkování

Tímto bych chtěla poděkovat doc. Ing. Vlastimilu Vavřičkovi, CSc. za materiály, které mi poskytl pro mojí práci a za čas, který mi věnoval na konzultacích.

Abstract

Methods of coding states of synchronous machines

This paper describes the various methods of coding states for synchronous finite state machines. These methods are divided into two basic groups, according to their computational complexity and given set of rules. These groups are basic and advanced algorithms. Advanced algorithms are further divided into heuristic and genetic algorithm group. The aim of this study is to compare several methods and identify those that best minimize the number of components needed for building the circuit. Further, this work deals with the DAG algorithm and examines the impact of used technological parameters on coding. The methods are applied to set of machines that have different number of inputs, outputs and next states.

Obsah

1 Úvod

V dnešním světě jsme obklopeni vědou a technikou. Snad každý již minimálně jednou použil automat na kávu, pračku nebo myčku. Tyto „stroje“ a mnoho dalších řadíme mezi automaty. Vykonávají danou sekvenci kroků podle zvoleného programu. Aby byla možná implementace, je potřeba navrhnout strukturu a jednoznačně stanovit každý krok chodu automatu, tzn. určit stavy, vstupy a výstupy. Tyto stavy je potřeba kódovat, aby bylo možné obvod realizovat pomocí elektronických součástek (AND, OR, NOT, klopné obvody, ...). Tato bakalářská práce se týká obecných automatů a popisuje metody pro kódování stavů.

Metody kódování stavů se zabývají problémem stavového přiřazení tj. přiřazení binárního kódu vnitřním stavům konečného automatu. Kódování stavů patří mezi známé problémy, které jsou zařazeny do kategorie NP-úplné. Neumíme najít „nejlepší“ algoritmus, proto se snažíme dosáhnout alespoň „dobrého“ kódování. Hledáme metody, které účinně minimalizují počet součástek potřebných pro realizaci obvodu [2].

Pokud je nalezen alespoň „dobrý“ algoritmus, zmenší se počet klopných obvodů potřebných pro daný automat, redukuje se počet hradel nutných pro implementaci, zmenší se vyžadovaný výkon a sníží se doba zpoždění obvodu [18].

Součástí této práce je i měření účinnosti metod pro kódování stavů a obsahuje výsledky cen [I] obvodů daných automatů pro metody binárního kódování, Grayovo kódování, DAG kódování a výstupního kódování.

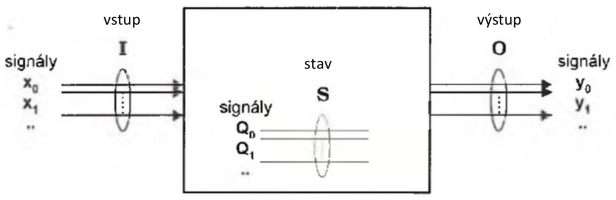
[I] cena je v dalším textu chápana jako počet vstupů do el. součástek potřebných pro realizaci obvodu daného automatu

2 Teorie automatů

Bakalářská práce je zaměřena na synchronní konečné automaty [4], tzn. změna stavů nastává až při působení synchronizačního (hodinového) impulzu CLK. Konečný automat nad abecedou Σ rozumíme uspořádanou pětici $A = (Q, \Sigma, \delta, q_0, F)$[5], kde

- Q je konečná neprázdná množina stavů (stavový prostor),
- Σ je neprázdná konečná množina vstupních symbolů (vstupní abeceda),
- δ je zobrazení $\delta: Q \times \Sigma \longrightarrow Q$ nazýváme přechodová funkce,
- $q_0 \in Q$ je počáteční stav (iniciální stav),
- $F \longrightarrow$ je cílová (finální) množina koncových stavů.

Konečný automat je abstraktním modelem sekvenčního obvodu. Stavy chápeme jako signály, které mohou nabývat dvou hodnot (0 a 1), a kterých je vždy konečný počet. Každou z kombinací označíme kvůli úspornému zápisu symbolem, např. písmenem. Obrázek 2.1 ukazuje automat se vstupními signály x_i, výstupní signály z_i a vnitřními signály Q_i. Vstup je označen jako I, výstup jako O a stav jako S. Jednotlivé konkrétní kombinace hodnot vstupů označíme jako I_0,I_1, atd. Obdobně budou označeny kombinace hodnot výstupů O_0,O_1, atd., a stavů S_0, S_1, atd. [14]



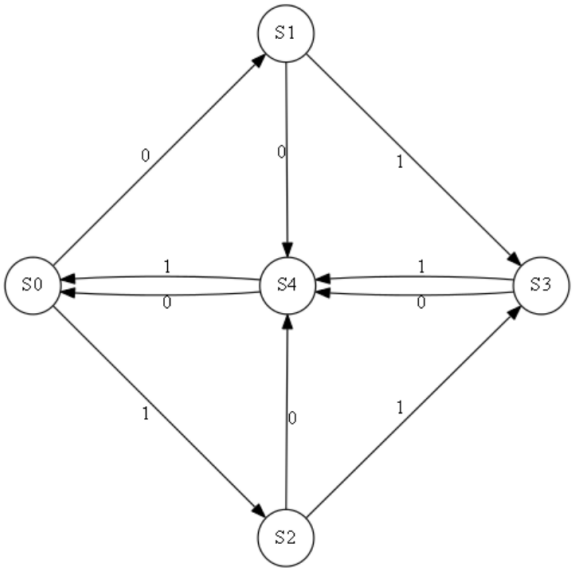
Obrázek 2.1: Signály a stavy v sekvenčním obvodu

Závislost následujícího stavu na současném stavu a vstupu vyjadřuje přechodová funkce: $S^t + 1 = f(S^t, I^t)$, kde symbol $S^t + 1$ značí následující stav, S^t značí současný stav, I^t značí současný výstup [14].

Výstup konečného automatu může být generován několika způsoby. Každý z nich je definován příslušnou výstupní funkcí. V obecném případě je výstup

funkcí vstupu a současného stavu : $O^t = g(S^t, I^t)$ [14]. Způsob odvození výstupu nemá vliv na kódování stavů.

Konečný automat lze reprezentovat různými způsoby: grafem (příklad uveden na obrázku 2.2), stavovou tabulkou (příklad uveden na obrázku 2.3), soustavou rovnic.



Obrázek 2.2: Graf

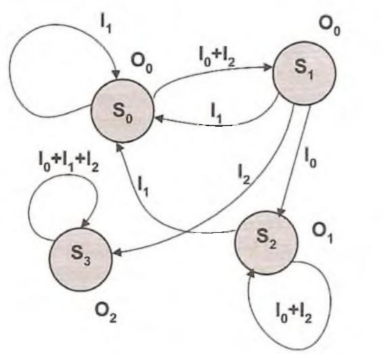
stavy	následníci		výstup	
	$I_0 = 0$	$I_0 = 1$	Z = 0	Z = 1
S0	S1	S2	0	0
S1	S4	S3	1	1
S2	S4	S3	1	0
S3	S4	S4	0	1
S4	S0	S0	0	0

Obrázek 2.3: Stavová tabulka

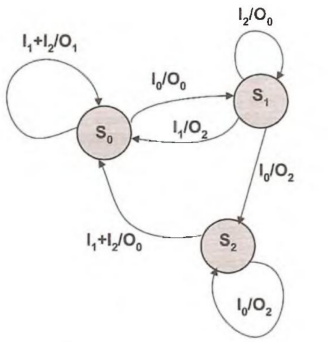
2.1 Dělení automatů

Pevný automat lze dělit z hlediska výstupu do dvou skupin: Mooreho (příklad Mooreho automatu je znázorněn na obrázku 2.4) a Mealyho (příklad je

uveden na obrázku 2.5).



Obrázek 2.4: Mooreho automat



Obrázek 2.5: Mealyho automat

2.1.1 Mooreho automat

Mooreho automat si můžeme představit jako zařízení s konečným počtem stavů, které pracují na základě vstupních symbolů. Každý stav má definovanou právě jednu hodnotu na výstupu. Automat dále musí mít definovaný výchozí stav, ve kterém se nachází před zadáním prvního vstupního symbolu a pravidla pro přechody mezi jednotlivými stavy. Výstup Mooreho automatu závisí pouze na vnitřním stavu, tedy: $O^t = g(S^t)$. Změna výstupu nastává pouze při vstupu hodinového signálu (CLK) [14].

Formální definice: Konečný automat typu Moore je uspořádaná šestice

$(X,\ Y,\ S,\ S_0,\ \delta,\ \omega)$, kde

- *X:* $|X|<\infty$ je konečná vstupní abeceda (množina vstupních symbolů),
- *Y:* $|Y|<\infty$ je konečná výstupní abeceda (množina výstupních symbolů),
- *S:* $|S|<\infty$ je konečná množina vnitřních stavů,
- S_0*:* $S_0 \in S$ je výchozí vnitřní stav,
- δ*:* $S \times X \longrightarrow S$ je přechodová funkce,
- ω*:* $S \longrightarrow Y$ je výstupní funkce.

2.1.2 Mealyho automat

Mealyho automat je konečný automat, jehož výstup je spojen s přechody mezi stavy. Výstup je generován na základě příchozího vstupu i momentálního stavu, ve kterém se automat nachází, tzn. $O^t = g(S^t, I^t)$. Stavový diagram automatu má ke každému přechodu přiřazenou nejen vstupní hodnotu, kterou je přechod aktivován, ale i výstupní hodnotou, která je při aktivaci přechodu vygenerována. Změna výstupu může nastat v libovolném čase, bez ohledu na hodinový signál CLK [14].

Formální definice: Mealyho stroj je uspořádaná šestice

(Z, Q, Y, ϕ*,* ψ*, q)*, kde:

- $Z = z_1, z_2, \ldots, z_n$ je konečná vstupní abeceda,
- $Q = q_1, q_2, \ldots, q_n$ je neprázdná konečná množina stavů proměnlivých v čase,
- $Y = y_1, y_2, \ldots, y_n$ je konečná výstupní abeceda,
- $\phi = g(t+1) = \phi[q(t), z(t)]$ je přechodová funkce,
- $\psi = y(t) = \psi[q(t), z(t)]$ je výstupní funkce, kde záleží na stavu, ve kterém se automat nachází a zároveň i na vstupním signálu,
- g je počáteční stav z množiny Q.

2.1.3 Pravdivostní tabulka

Logickou funkci lze zapisovat pravdivostní tabulkou. Tabulka obsahuje pouze logické hodnoty 0, 1 nebo neurčité stavy (často označovány symboly „X“ nebo „-“). Velikost tabulky je dána počtem všech vstupních proměnných a počtem výstupních funkcí. To znamená, že tabulka bude mít tolik řádků, kolik je počet všech kombinací stavů výstupních proměnných, které mohou nastat. Počet těchto kombinací se vypočítá jako 2^n , kde n je počet vstupních proměnných. Příklad pro funkci tří proměnných $f(x, y, z)$ a její negaci $\overline{f}(x, y, z)$ je uveden v tabulce 2.1.

Tabulka 2.1: Pravdivostní tabulka

DEK	x	y	z	f(x,y,z)	$\overline{f}$(x,y,z)
0	0	0	0	1	0
1	0	0	1	1	0
2	0	1	0	0	1
3	0	1	1	1	0
4	1	0	0	1	0
5	1	0	1	0	1
6	1	1	0	1	0
7	1	1	1	0	1

Zhuštěný zápis pravdivostní tabulky 2.1 je (vyjmenujeme ty řádky, kde $f = 1$): $f(x, y, z) = \sum m(0, 1, 3, 4, 6)$ [I] , symbol $\sum$ naznačuje, že funkce f je vyjádřena v součinovém (disjunktním) tvaru. Libovolnou logickou funkci lze vyjádřit jako součet elementárních logických funkcí m_i, které se nazývají *mintermy*. Výsledný součet mintermů se nazývá *úplná součtová normální forma, zkráceně ÚDNF* a získáme ji, pokud vybereme jen ty mintermy, které odpovídají řádkům s funkční hodnotou 1 [14].

2.1.4 Minimalizace logické funkce

Způsobů minimalizace logické funkce je několik a provádí se s využitím: Booleovy algebry, DeMorganových zákonů, Karnaughovy mapy a krychlového komplexu.

[I] Pokud by tabulka obsahovala i neurčité stavy zapíší se do hranatých závorek do součinového tvaru, př. $f(x, y, z) = \sum m(0, 1, 3, [4, 6])$

Obdobně jako v běžné algebře i v **Booleově algebře** platí komutativní, asociativní a distributivní zákon. Používání závorek se neliší od používání v běžné algebře.

DeMorganovy zákony jsou:

$\overline{(a+b)} = \overline{a} \cdot \overline{b}$

$\overline{(a \cdot b)} = \overline{a} + \overline{b}$

Karnaughova mapa je jeden z možných zápisů logické funkce. Tuto mapu použijeme při minimalizaci nebo při analýze. Jejím principem je zobrazení pravdivostní tabulky do dvourozměrné mapy, díky které je možno vyhledat slučitelné termy (oblasti, které jsou zcela nezávislé na jedné ze vstupních proměnných a ty pak zapisujeme jako minimalizovanou funkci) [14]. Ukázka minimalizace pomocí Karnaughovy mapy bude uvedena v textu (kapitola 2.2).

Krychlovým komplexem $C(\lambda)$ funkce $\lambda(x_1, x_2, ..., x_m)$ rozumíme množinu vrcholů (0-krychlí) m-měrné jednotkové krychle, event. množinu podkrychlí m-měrné jednotkové krychle, na jejichž vrcholech nabývá funkce λ hodnotou 1 či není definována (0 či není definována nebo 0 či 1) [5].

2.1.5 Hlavní pravidla pro tvorbu a úpravy logických výrazů

Logické výrazy upravujeme podle dvou kritérií. Buď se jedná o zjednodušení ve smyslu snížení počtu písmen ve výsledném výrazu, nebo se jedná o úpravu do takového tvaru, který vyhovuje číslicovému obvodu (součástce), který je již k dispozici [14]. V této práci budeme uvažovat pouze první případ. Předpokládejme tedy, že máme k dispozici všechny potřebné obvody.

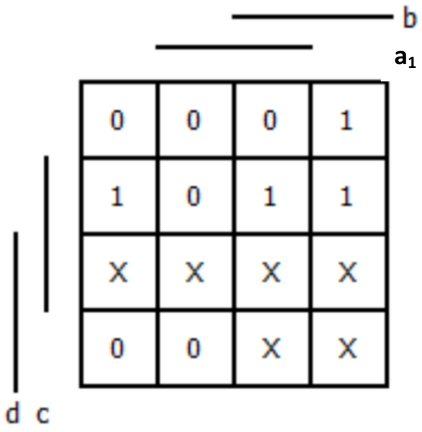
2.2 Ukázkový příklad

Příklad výpočtu ceny logického automatu, tedy počet AND-OR vstupů požadovaných v dvoustupňovém provedení každého vstupního paměťového prvku vstupní rovnice.

1. krok: Ze zakódované stavové tabulky 2.2 (červeně označené sloupce) sestrojíme Karnaughovu mapu (obrázek 2.6).

Tabulka 2.2: Zakódovaná stavová tabulka pro Karnaughovu mapu

stavy	následující stavy		výstupy	
	$I_0 = 0$	$I_0 = 1$	Z_0	Z_1
dcb	$\overline{a}_1\overline{a}_2\overline{a}_3$	$a_1a_2a_3$		
000	0 0 1	0 1 0	0	0
001	1 0 0	0 1 1	1	1
010	1 0 0	0 1 1	1	0
011	1 0 0	1 0 0	0	1
100	0 0 0	0 0 0	0	0



Obrázek 2.6: Karnaugova mapa pro první stavovou proměnnou a_1

Logická funkce pro první stavovou proměnnou:

$f_1(d, c, b, a_1) = \sum m(2, 4, 6, 7[10, 11, 12, 13, 14, 15])$

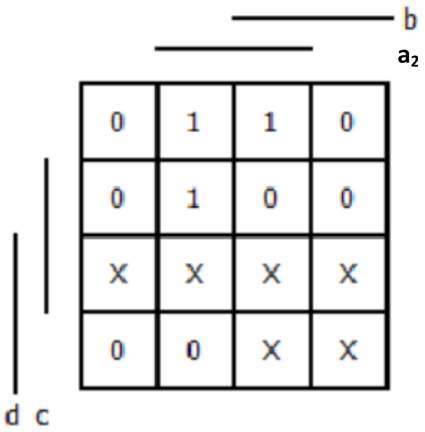
ÚDNF pro první stavovou proměnnou:

$f_1(d, c, b, a_1) = \overline{a}\mathrm{b} + \overline{a}\mathrm{c} + \mathrm{bc}$

2. krok: Stejný způsobem pokračujeme i pro ostatní sloupce následujících stavů (obrázek 2.7 a 2.8)

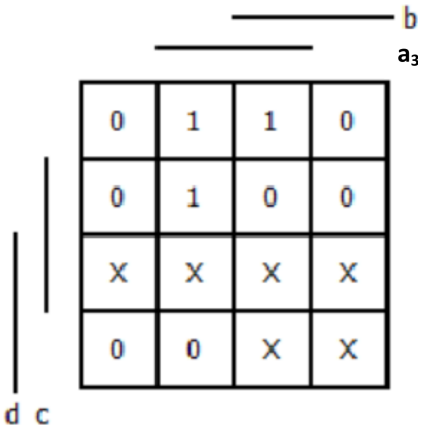
Logická funkce pro druhou stavovou proměnnou:

$f_2(d, c, b, a_2) = \sum m(1, 3, 5[10, 11, 12, 13, 14, 15])$



Obrázek 2.7: Karnaugova mapa pro druhou stavovou proměnnou a_2

ÚDNF pro druhou stavovou proměnnou: $f_2(d, c, b, a_2) = \mathrm{a}\overline{b}\overline{d} + \mathrm{a}\overline{c}\overline{d}$



Obrázek 2.8: Karnaugova mapa pro třetí stavovou proměnnou a_3

Logická funkce pro třetí stavovou proměnnou:

$f_3(d, c, b, a_3) = \sum \mathrm{m}(0,3,5[10,11,12,13,14,15])$

ÚDNF pro třetí stavovou proměnnou: $\mathrm{f(d,c,b,a)} = \overline{a}\overline{b}\overline{c}\overline{d} + ab\overline{c} + a\overline{b}c$

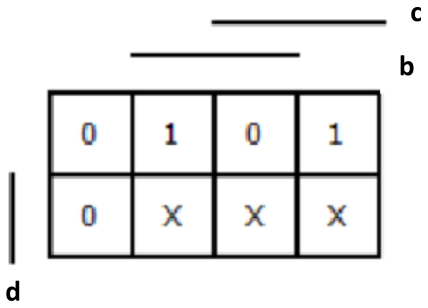
3. krok: Dále vytvoříme mapu i pro výstupy (obrázky 2.9 a 2.10).

Logická funkce pro výstup Z_0: $f_4(d, c, b) = \sum \mathrm{m}(1,2[5,6,7])$

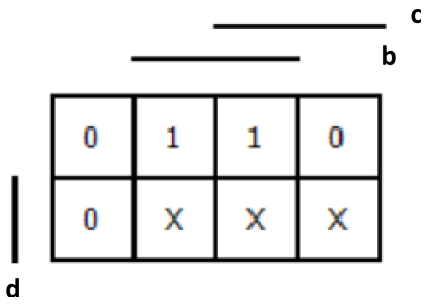
ÚDNF pro výstup Z_0: $f_4(d, c, b) = \mathrm{b}\overline{c} + \overline{b}\mathrm{c}$

Logická funkce pro výstup Z_1: $f_5(d, c, b) = \sum \mathrm{m}(1,3[5,6,7])$

ÚDNF pro výstup Z_1: $f_5(d, c, b) = \text{b}$



Obrázek 2.9: Karnaugova mapa pro první výstup



Obrázek 2.10: Karnaugova mapa pro druhý výstup

4. krok: Vypočítáme celkovou cenu obvodu ze všech uvedených forem f_i (tzn. počet vstupů do log. členů AND a OR), viz tabulka 2.3 .

Tabulka 2.3: Cena obvodu

ÚDNF	**cena obvodu**	
$f_1(d,c,b,a_1) = \overline{a_1}b + \overline{a_1}c + bc$	$2+2+2+3$	9
$f_2(d,c,b,a_2) = a_2\overline{bd} + a_2\overline{c}\overline{d}$	$3+3+2$	8
$f_3(d,c,b,a_3) = \overline{a_3}\overline{b}\overline{c}\overline{d} + a_2b\overline{c} + a_3\overline{b}c$	$4+3+3+3$	13
$f_4(d,c,b) = b\overline{c} + \overline{b}c$	$2+2+2$	6
$f_5(d,c,b) = b$	0	0
	$\sum$	**36**

3 Kódování stavů logických automatů

Problém kódování stavů, s minimálními náklady na cenu obvodu není zcela vyřešen. Metody, které jsou prozatím navrženy, mají jeden nebo více z následujících nedostatků [9]:

- jsou velmi pracné,
- vyžadují velké množství „ručního šití“ metod pro každý problém, tzn. není možné implementovat na digitálním počítači,
- jsou použitelné pouze pro malé obvody.

3.1 Počet různých kódování

Nechť je dán konečný automat M, který má r stavů. Předpokládáme, že pro zakódování stavů použijeme s bitů, kde s je nejmenší celé číslo, vyhovující podmínce $2^s \geq r$ [19]. Kolik různých přiřazení stavů můžeme dosáhnout u tohoto automatu? Vzorec je poměrně jednoduchý, ale výsledky jsou překvapivé [2]:

$$A = C_2^r = \frac{2^s!}{(2^s - r)!}$$

Pokud zvažujeme ekvivalentní přiřazení stavu, pak počet různých kódování podle definice McCluksey je roven (pro obvody používající klopné obvody SR, JK a T) [19]:

$$A1 = \frac{(2^s - 1)!}{(2^s - r)!s!}$$

S další definicí přišli Weiner-Smith a Harrison (pro obvody typu D) [19]:

$$A2 = \frac{(2^s)!}{(2^s - r)!s!}$$

V následující tabulce 3.1 je vidět, kolik různých kódování stavů můžeme dosáhnout [19].

Tabulka 3.1: Počet různých kódování

r	s	A	A1	A2
2	1	2	1	2
3	2	24	3	12
4	2	24	3	12
5	3	6 720	140	1 120
6	3	20 160	420	1 120
7	3	40 320	840	6 720
8	3	40 320	840	6 720
9	4	4 151 347 200	10 810 800	172 972 800
10	4	29 059 430 400	75 675 600	1 210 809 600
11	4	174 356 582 400	454 053 600	7 264 857 600
12	4	871 782 912 000	2 270 268 000	36 324 288 000
13	4	3 487 131 648 000	9 081 072 000	114 529 715 200
14	4	10 461 394 944 000	27 243 216 000	435 891 456 000
15	4	20 922 789 888 000	54 486 432 000	871 782 720 000
16	4	20 922 789 888 000	54 486 432 000	871 782 720 000

3.2 Účinnost metod kódování stavů

Pro porovnávání metod kódování stavů je potřeba určit metriku, podle které budeme porovnávat jednotlivé metody. Tři základní způsoby porovnávání účinnosti metod jsou:

- efektivita z hlediska výpočetní složitosti (výsledek v reálném čase, složitost algoritmu),
- minimální plocha (tzn. nejmenší počet součástek potřebných pro daný automat),
- nejmenší cena (tzn. nejmenší počet vstupních členů).

V této práci bude využíván pouze třetí způsob porovnávání (nejmenší cena). To znamená, že bude počítán počet vstupů do logického členu AND a počet vstupů do logických členu OR. Součet těchto vstupních členů bude pro jednoduchost označován jako „cena obvodu“.

3.3 Základní metody kódování

Tyto metody nerespektují strukturu automatu (tzn. neberou v úvahu následující stavy, předchozí stavy ani přechody stavů), a proto může být toto kódování velmi neefektivní z hlediska ceny obvodu (viz kapitola **6 Výsledky kódování**). Na druhou stranu nejsou algoritmicky náročné.

3.3.1 Binární kódování

Binární kódování pracuje na principu očíslování jednotlivých stavů tak, jak jsou na číselné ose. Každý stav je reprezentován jedním číslem v binární podobě (obrázek 3.1). Výhodou tohoto kódování je kromě rychlosti i minimální délka slova potřebná pro zakódování automatu. Počet potřebných bitů odpovídá $\lceil log_2 N \rceil$, kde N je počet stavů konečného automatu [12].

s_0	000
s_1	001
s_2	011
s_3	111
s_4	110
s_5	100

Obrázek 3.1: Binární kódování

3.3.2 Grayovo kódování

Grayův kód je specifický tím, že se dvě sousední slova (stavy) liší pouze v jednom bitu (tj. jejich Hammingova vzdálenost se rovná 1). Použití tohoto kódování pro některé obecné automaty může znamenat, že nebudeme mít minimální počet paměťových prvků. Ukázka Grayova kódování je na obrázku 3.2 [14].

S_0	000
S_1	001
S_2	011
S_3	111
S_4	110
S_5	100

Obrázek 3.2: Grayovo kódování

3.3.3 One-hot kódování

One-hot kódování přiřazuje každému stavu právě jednu logickou 1 v binárním řetězci o délce odpovídající počtu stavů. Výhodou tohoto kódování je, že Hammingova vzdálenost se rovná 2 mezi libovolnými dvěma stavy konečného automatu (obrázek 3.3). Díky této vlastnosti je tento kód vhodný pro rozsáhlé automaty s mnoha stavy (v praxi s více než osmi). Nevýhodou je velký počet klopných obvodů, potřebných pro realizaci obvodu (počet KO se rovná počtu stavů daného automatu). Někdy se pro zmenšení celkové vzdálenosti mezi stavy (tedy „přepínací“ aktivitou, switching activity) používá jako počáteční stav řetězec obsahující samé 0, což umožňuje nejen zkrátit řetěz o jeden bit, ale také sníží Hammingovu vzdálenost na 1 mezi prvním stavem a libovolným stavem [13] .

S_0	000001
S_1	000010
S_2	000100
S_3	001000
S_4	010000
S_5	100000

Obrázek 3.3: One-hot kódování

3.3.4 2-hot kódování

2-hot kódování je specifická varianta **m-n kódování**. Je tedy možné zakódovat $\binom{n}{2}$ stavů, v optimálním případě je možné udržet Hammingovu vzdálenost

mezi všemi přechody na vzdálenosti 2 (obrázek 3.4) [10].

S_0	10000
S_1	10001
S_2	10010
S_3	10100
S_4	11000
S_5	01000

Obrázek 3.4: 2-hot kódování

3.3.5 Zero-hot kódování

Jak již název napovídá, nejedná se o nic jiného, než opačné kódování k *one-hot*, tj. místo logické 1 určuje stav logická 0 mezi jedničkami (obrázek 3.5). Stejně jako u *one-hot* je možné zmenšit délku slova použitím samých 1 jako startovního stavu [10].

S_0	111110
S_1	111101
S_2	111011
S_3	110111
S_4	101111
S_5	011111

Obrázek 3.5: Zero-hot kódování

3.3.6 Johnsonovo kódování

Pro zakódování je potřeba $q = \lceil \frac{N}{2} \rceil$ bitů, kde N je počet stavů automatu. První stav je zakódován binární kódem 0. U dalších stavů se od konce slova postupně mění 0 na 1 do doby, než slovo obsahuje samé jedničky. U zbývajících stavů se obdobně mění jedničky na nuly (obrázek 3.6). Následující

stavy v tabulce za sebou se opět liší jedním bitem (tzn. že jejich Hammingova vzdálenost je rovná 1) [15]. Jedná se tedy o upravenou variantu Grayova kódu.

S_0	000
S_1	001
S_2	011
S_3	111
S_4	110
S_5	100

Obrázek 3.6: Johnsonovo kódování

3.3.7 Kódování m-n

Toto kódování vyžaduje délku slova n a počet logických 1, které jsou ve slově obsaženy, je m. Jednotlivé kódy jsou permutacemi slova obsahujícího n jedniček a *(m-n)* nul. Parametry m a n musí být zvoleny tak, aby jimi bylo možné všechny stavy pokrýt (obrázek 3.7). Maximální počet stavů, které lze zakódovat, odpovídá kombinačnímu číslu $\binom{m}{n}$, proto m a n musí být zvoleno tak, aby $\binom{m}{n} \geq N$ [10].

S_0	00111
S_1	01110
S_2	11100
S_3	01011
S_4	10011
S_5	01101

Obrázek 3.7: Kódování m-n

3.3.8 Náhodné kódování

Z názvu je zřejmé, že toto kódování se neřídí žádnými pravidly (obrázek 3.8). Délka stavu se řídí podle vztahu $2^s \geq r$, kde s je počet bitů (délka stavu) a

r je celkový počet stavů konečného automatu [10].

S_0	100
S_1	011
S_2	001
S_3	000
S_4	101
S_5	010

Obrázek 3.8: Náhodné kódování

3.3.9 Metoda postupného procházení

Tato metoda postupně vyzkouší všechny možné kombinace binárních kódů (viz kapitola **5.1 Počet různých kódování**). Pro každý případ vypočítá cenovou náročnost obvodu a zapamatuje si nejlepší řešení, kterým poté zakóduje konečný stavový automat. Vzhledem k časové složitosti je metoda určena pouze pro automaty s malým počtem stavů (do osmi stavů) [15].

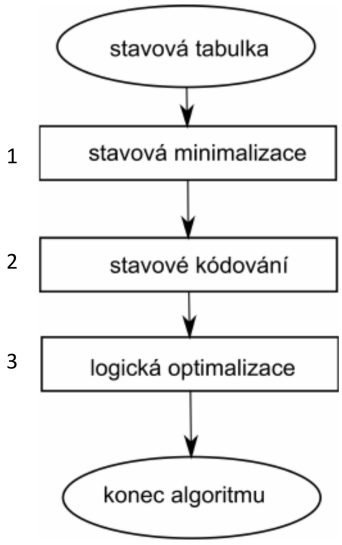
3.4 Pokročilé metody kódování

Nalezení nejlepšího stavového přiřazení synchronního obvodu je důležité při snižování ceny a složitosti obvodu v automatech. Tento problém SAP (**S**tate **A**ssignment **P**roblem, nalezení souvislosti mezi stavy a kódováním) patří do širší skupiny kombinatorických optimalizačních problémů a zároveň patří do skupiny NP-úplný [3].

Pro všechny druhy kódování platí obecný postup pro přiřazení binárních kódů stavům (obrázek 3.9) [11]:

1. ze stavové tabulky určíme ekvivalentní stavy a provedeme eliminaci redundantních stavů,
2. kódujeme všechny stavy unikátním binárním kódem,

3. použijeme nevyužité binární kódy pro neurčité stavy (pouze v případě , že ve fázi zjednodušování funkce se ukáží jako užitečné).



Obrázek 3.9: Obecný postup pro kódování stavů

Mezi pokročilé metody kódování řadíme:

- heuristické algoritmy (například MUSTANG, DAG),
- genetické algoritmy (jsou srovnatelné nebo lepší než heuristické algoritmy).

3.5 Heuristické algoritmy

Heuristické algoritmy mají dva hlavní cíle. Nalézt algoritmus, který

- nalezne výsledek v reálném čase,
- povede alespoň k dobrému výsledku.

.

Heuristických algoritmů je celá řada a většina metod jsou varianty těchto bodů:

- určení sousedních stavů (vážených párů),
- generování omezujících podmínek (stavy, které jsou ve stejné krychli),
- maximalizovat počet společných krychlí (vážený součet).

3.5.1 DAG kódování

DAG je zkratka z anglického slova The **D**esired **A**djacency **G**raph, neboli **graf sousednosti**[3]. DAG je neorientovaný, vážený, plně propojený graf, který má jako uzly stavy z FSM. Pro nízké náklady na cenu obvodu je nutné, aby se minimalizovala vzdálenost mezi stavy, které jsou pevně spojeny. Opět pro lepší pochopení je algoritmus vysvětlen na ukázkovém příkladu, kde obrázek 3.10 mám ukazuje přechody mezi jednotlivými stavy a výstupy, které jsou přiřazeny každému stavu:

	následníci		výstup	
stavy	$I_0 = 0$	$I_0 = 1$	Z = 0	Z = 1
S0	S1	S2	0	0
S1	S4	S3	1	1
S2	S4	S3	1	0
S3	S4	S4	0	1
S4	S0	S0	0	0

Obrázek 3.10: Příklad stavové tabulky

Postup pro vytvoření DAG kódování je následující:

1.krok : Vytvoříme si *množinu následníků* (z obrázku 3.10). *Pozn.: zapisujeme pouze množiny, které mají 2 nebo více následníků.*

$\mathrm{S}(S_0) = S_1, S_2$

$\mathrm{S}(S_1) = S_3, S_4$

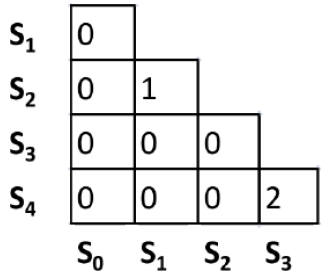
$\mathrm{S}(S_2) = S_3, S_4$

Z této množiny vytvoříme dolní trojúhelníkovou matici následníků. Začínáme nulami, které jsou přiřazeny všem hranám. Hraně (S_a, S_b) připočítáváme 1, jestliže S_a a S_b jsou prvky množiny následníků. Tímto vznikne **Graf sousednosti (DAG) následníků** (obrázek 3.11).

2.krok: Vytvoříme si *množinu předchůdců* (z původního obrázku 3.10).

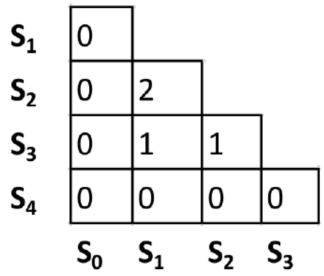
P(S_4, $I_0 = 0$) = S_1, S_2, S_3

P(S_3, $I_1 = 1$) = S_1, S_2



Obrázek 3.11: Graf sousednosti (DAG) následníků

Jako v předchozím kroku, opět vytvoříme dolní trojúhelníkovou matici, tentokrát matici předchůdců. Začínáme nulami, které jsou přiřazeny všem hranám. Hraně (S_a, S_b) připočítáme 1, jestliže S_a a S_b jsou prvkem množiny předchůdců stavu. Vznikne **Graf sousednosti (DAG) předchůdců** (obrázek 3.12).



Obrázek 3.12: Graf sousednosti (DAG) předchůdců

3.krok: Vytvoříme si *množinu výstupů*.

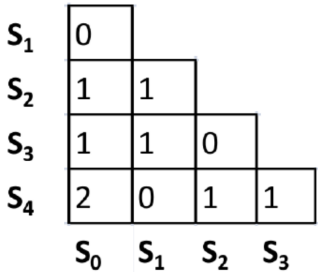
O(Z_0) = (S_0, S_3, S_4), (S_1, S_2)

O(Z_1) = (S_0, S_2, S_4), (S_1, S_3)

Pozn.: Stavy, které mají ve sloupci Z_0 výstup 0, jsou v první množině závorek. Stavy, jejichž výstup ve sloupci Z_0 je 1, jsou v druhé množině závorek. Stejnými kroky postupujeme i pro výstup Z_1.

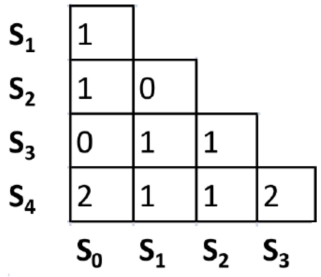
Z této množiny vytvoříme dolní trojúhelníkovou matici výstupů. Začínáme nulami, které jsou přiřazeny všem hranám. Hraně (S_a, S_b) připočítá-

váme 1, jestliže S_a a S_b jsou prvky množiny následníků. Tímto vznikne **Graf sousednosti (DAG) výstupů** (obrázek 3.13).



Obrázek 3.13: Graf sousednosti (DAG) výstupů

4.krok: Vytvoříme si poslední matici: matici přechodů. Začínáme nulami, které jsou přiřazeny všem hranám. Hraně (S_a, S_b) připočítáváme 1, jestliže existuje hrana vedoucí z S_a do S_b (obrázek 3.14).



Obrázek 3.14: Graf sousednosti (DAG) přechodů

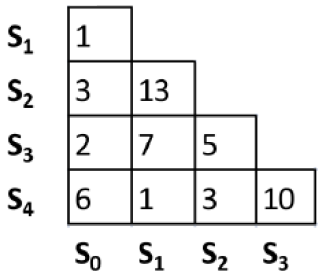
5.krok: V tomto kroku spojíme všechny matice z předchozích kroků. Ovšem každá matice má jinou váhu, kterou určil Amaral v roce 1990 [2].

DAG = 3*následník + 4*předchůdce + 2*výstupní + 1*přechody

Tyto váhy v jisté míře ovlivňují výsledné kódování. Proto tyto konstanty byly změněny a porovnány (viz kapitola Závěr). Algoritmus byl tedy rozšířen na:

- DAG1 = 1*následník + 1*předchůdce + 1*výstupní + 1*přechody
- DAG2 = 2*následník + 1*předchůdce + 3*výstupní + 4*přechody
- DAG3 = 4*následník + 3*předchůdce + 1*výstupní + 2*přechody
- DAG4 = 2*následník + 2*předchůdce + 1*výstupní + 1*přechody

Po vynásobení (původními konstantami) a sečtení matic nám vznikne **konečná matice grafu sousednosti DAG** (obrázek 3.15).



Obrázek 3.15: Konečná matice grafu sousednosti DAG

6.krok: Z konečné matice DAG (obrázek 3.15) vypočítáme váhu stavu S_a sečtením hodnot přiřazeným hranám $(S_a, S_?)$. Získaný váhový vektor vyjadřuje, kterému stavu má být dána v procesu kódování přednost. Tento krok zopakujeme pro všechny stavy.

7.krok: Pokud již máme určené váhy každého stavu, následuje vybrání stavu s největší váhou a přidělení tomuto stavu kód v binární hodnotě 0. Může se ovšem stát, že dojde ke shodě maximálních vah. V tomto případě budeme hledat maximální hodnotu v matici DAG (obrázek 3.15) u shodných vah stavů.

8.krok: Když již máme přidělený binární kód 0, nalezneme stav s nejsilnější vazbou k prvému stavu a přidělíme binární kód 1.

9.krok: Dále vytvoříme z obrázku 3.16 **neúplnou tabulkou přiřazení (IAT)**. Každá buňka tabulky obsahuje hodnotu, která se počítá podle vztahu:

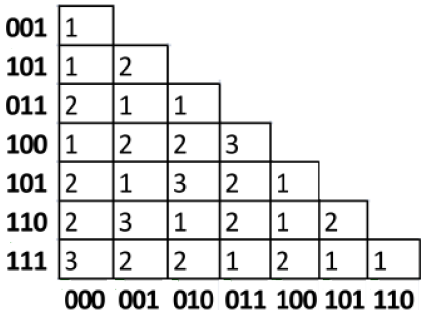
$\sum_{i=0}^{s-1} \sum_{j=0}^{s-1} D(S_i, S_j) DAG_{ij}$

Hodnota buňky (S_0, S_2) tedy je: $D(S_0, S_1)DAG_{01} + D(S_0, S_2)DAG_{02} =$

$= D(010, 001)DAG_{01} + D(010, 000)DAG_{02} =$

$= 2 \times 1 + 1 \times 3 = 2 + 3 = 5$

Pozn.: Pozici DAG_{01} vybereme z tabulky DAG (obrázek 3.15), pozice $D(010, 001)$ je hodnota z tabulky vzdálenosti prvků binárního kódu (obrázek 3.16), neboli Hammingova vzdálenost.



Obrázek 3.16: Vzdálenost prvků binárního kódu

Hodnoty v ostatních polích se počítají obdobně (tabulka 3.2). Po vyplnění neúplné tabulky sečteme hodnoty v každé řádce. Stav, který vykazuje největší sumu v tabulce, nejvíce přispívá k ceně. Proto by měl být kódován prioritně. V neúplné tabulce přiřazení hledáme řádku s maximálním řádkovým součtem a vybereme buňku s minimální vahou. Pokud v tomto kroku nastane rovnost, vybereme náhodně.

Tabulka 3.2: Neúplná tabulka přiřazení

	000	001	010	011	100	101	110	111	∑
S_0	-	-	5	7	5	7	9	11	44
S_1	-	-	-	-	-	-	-	-	-
S_2	-	-	-	-	-	-	-	-	-
S_3	-	-	19	17	19	17	31	29	132
S_4	-	-	5	7	5	7	9	11	44

9.krok: Nyní začneme počítat neúplnou tabulku přiřazení od začátku pro nezakódované stavy. Takto postupujeme dokud nezakódujeme všechny stavy (obrázek 3.17).

3.5.2 Output-Based encoding (Výstupní kódování)

Tento druh kódování spočívá v jednoduchém principu. K přiřazení kódu stavům využívá jejich výstupy. Nicméně může být automat, který má stejný výstup pro dva nebo více stavů (viz obrázek 2.3). Každý stav musí být zakódován jinou hodnotou, proto algoritmus přidá další bit (pro dva stejné výstupy). Prvnímu stavu se přidá nejvyšší bit 0 a druhému stavu přiřadíme 1. Pokud tedy budeme mít stavy S_0, S_4 a hodnota jejich výstupu je 00. Tyto stavy zakódujeme binárním kódem 000 pro stav S_0 a 100 pro stav S_4 [16].

S_0	100
S_1	001
S_2	000
S_3	011
S_4	010

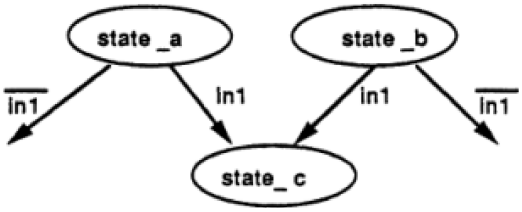
Obrázek 3.17: Kodování stavů podle DAG

3.5.3 TOOL1

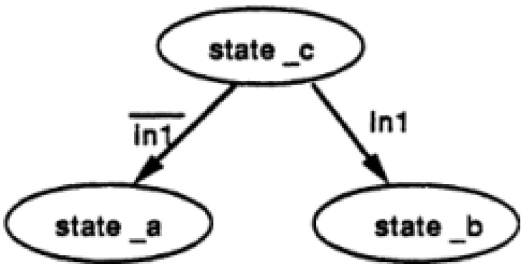
Toto kódování se řídí podle dvou pravidel. Ta určují, které stavy se budou lišit pouze v jednou bitu (nebo-li jejich Hammingova vzdálenost bude rovna jedné) [13]:

1. Stavy, které mají společné následníky (nebo-li další stavy). Příklad je uveden na obrázku 3.18, kde stavy *state_a* a *state_b* budou mít Hammingovu vzdálenost rovnou 1.
2. Následníci, kteří mají stejný původní stav. Na obrázku 3.19 to jsou stavy *state_a* a *state_b*.

Tato pravidla nám vytvoří tzv. omezující matici. Každému omezení (1.pravidlo a 2.pravidlo) je přiřazen faktor hmotnosti, který určuje, jakému pravidlu je dána přednost. Nakonec je vybráno kódování, které nejlépe odpovídá těmto heuristickým pravidlům [13].



Obrázek 3.18: 1. pravidlo



Obrázek 3.19: 2. pravidlo

3.5.4 TOOL2

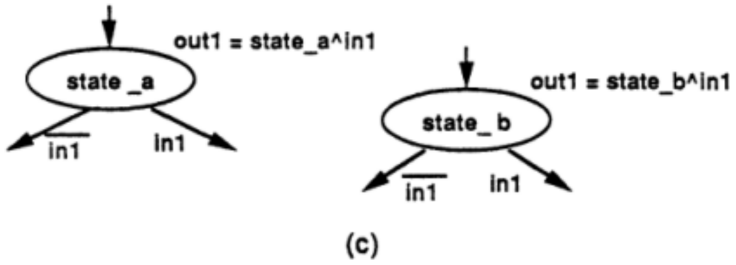
Tento druh kódování je rozšíření metody TOOL1. TOOL2 má celkem 4 pravidla, podle kterých se kódují stavy automatů [13]:

1. totožné s 1.pravidlem metody TOOL1,
2. totožné s 2.pravidlem metody TOOL1,
3. stavy se společnými podmíněnými výstupy sousedí pouze v případě, když výstupy jsou podmíněny stejným vstupem (obrázek 3.20),
4. stavy se společnými nepodmíněnými výstupy sousedí pouze v případě, když nastane situace z obrázku 3.21.

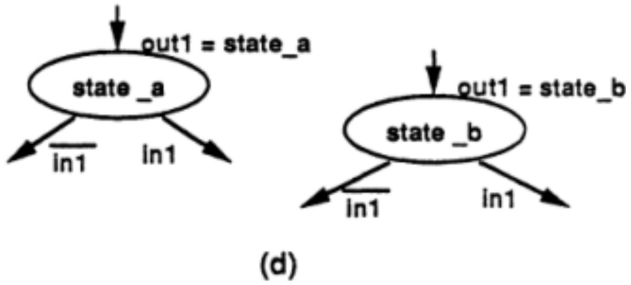
Pozn:

- **podmíněný výstup** - výstup, který je závislý na současném stavu a aktuální vstupní kombinaci,
- **nepodmíněný výstup** - závisí pouze na současném stavu.

Jak již bylo uvedeno v metodě TOOL1, každá podmínka má svůj hmotnostní faktor. Podmínka 3 a 4 by měla mít větší prioritu, a proto větší hmotnostní faktor. Tyto podmínky slouží pro vytvoření podmínkové matice, která slouží pro vybrání vhodného kódování stavů [13].



Obrázek 3.20: 3. pravidlo



Obrázek 3.21: 4. pravidlo

3.5.5 Mustang

Tato kódovací technika je založena na minimalizaci společné krychle. Cílem algoritmu je najít kódování, které maximalizuje počet společných krychlí. Dříve než bude uveden postup algoritmu Mustang [8], je potřeba seznámit se s několika pojmy:

Symbolický implikant: představuje přechod od jednoho nebo více stavů do dalšího stavu podle vstupů.

Stavová skupina: seskupení stavů, které mají stejné vstupy a stejné následníky.

Krychle: pokud stavovou skupinu zakódujeme binárním kódem tak, že všechny stavy ze stavové skupiny se budou lišit pouze v jednom bitu, pak tato skupina může být realizována jako krychle.

FSM (konečný stavový automat) je zde reprezentován dvěma rovnocennými strukturami:

1. Graf přechodových stavů *G(V, E, W(E))*, kde *V* je množina vrcholů

odpovídající všem stavům, $|| E || = N_s$ mohutnosti stavů FSM hrany v_i, v_j a $W(E)$ je sada popisků každé hrany, každý popisek nese informaci o hodnotě vstupu.

2. Přechodová stavová tabulka *T(I, S, O)*, kde I jsou vstupy, S označuje sadu stavů a O jsou výstupy.

Tabulka má tolik řádků jako hran stavového grafu a tolik sloupců jako $N_i + N_o + 2$, kde N_i je počet bitů pro zakódování vstupů a N_o počet bitů pro zakódování výstupů.

Globální strategie algoritmu Mustang

Naším cílem je vybudovat graf $G_M(V, E_M, W(E_M))$, kde V je komunikace mezi stavy FSM, E_M je kompletní sada hran (každý uzel je spojen ve všemi uzly v FSM) a $W(E_M)$ představuje zisky, které jsou dosaženy tím, že kódování všech stavů má co nejmenší Hammingovu vzdálenost (tyto zisky jsou nezávisle vypočítávány z výčtu přímých vztahů mezi vstupy, stavy a výstupy).

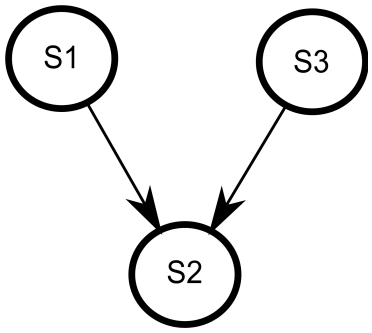
Algoritmus Mustang se vytvoří pomocí dvou struktur:

1. **Fanout-oriented**: maximalizuje velikost nejčastějších krychlí před optimalizací.
2. **Fanin-oriented**: maximalizuje počet výskytů největších společných krychlí před optimalizací.

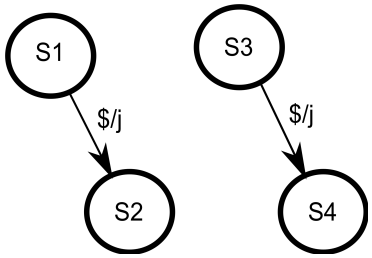
Fanout-oriented algoritmus

Algoritmus sestaví kompletní graf $G_M(V, E_M, W(E_M))$ s prázdnou hranovou vahou $W(E_M)$. K této váze poté přičítáme konstanty (n nebo *1*, kde n je počet bitů potřebných pro kódování).

Příklad: pokud nastane situace z obrázku 3.22, přičteme k $W(E_M(S1, S3))$ konstantu n, pokud budeme mít stejný výstup j (viz obrázek 3.23), přičteme k $W(E_M(S1, S3))$ konstantu 1. Pro všechny ostatní situace se váhy nemění.



Obrázek 3.22: 1.Fanout-orientovaný algoritmus



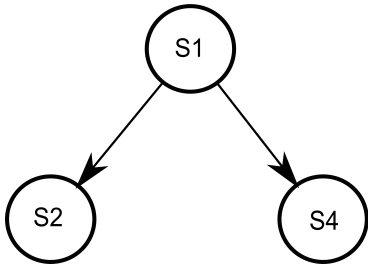
Obrázek 3.23: 2.Fanout-orientovaný algoritmus

Fanin-oriented algoritmus

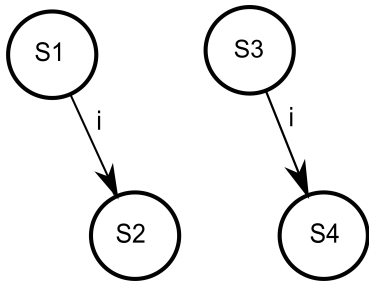
Jako algoritmus *Fanout-oriented*, bude *Fanin-oriented* algoritmus počítat hranové váhy.

Příklad: k počáteční prázdné hranové váze se přičte *n/2* k $W(E_M(S2, S4))$ (znázorněno na obrázku 3.24) a konstanta 1 se přičte pro

$W(E_M(S2, S4))$.



Obrázek 3.24: 1.Fanin-orientovaný algoritmus



Obrázek 3.25: 2.Fanin-orientovaný algoritmus

Algoritmus vkládání

Algoritmus vkládání přiřazuje skutečné (binární) kódy podle algoritmu *Fanout-oriented* a *Fanin-oriented*. Tento problém je klasická kombinatorická optimalizace, zvaná vkládaní grafů (*graph embedding*) G_M (bude popsán pseudokódem níže). G_M musí být vloženo do booleovské krychle tak, aby sousední stavy splňovaly heuristický algoritmus *wedge clusting*. Tento algoritmus používá k přiřazení kódu v G_M minimalizaci $\sum_{i=0}^{N_s} \sum_{j=i+1}^{N_s} we(e_M(v_i, v_j)) * dist(enc(v_i), (v_j))$, kde v_k jsou vrcholy v G_M, $we(e_M(v_i, v_j))$ je váha hrany e mezi vrcholy v_i a v_j, $enc(v_k)$ je kódování vrcholu a funkce $dist()$ vrací vzdálenost mezi dvěma binárními kódy.

Grafy generované *Fanout-oriented* a *Fanin-oriented* algoritmem mají určitou strukturu. V těchto grafech jsou malé skupiny stavů, které jsou pevně propojeny, tzn. interně (hrany mezi stavy ve stejné skupině a vyznačují se velkou váhou) a stavy, které jsou připojeni slabě, tzn. externě (hrany mezi stavy, které nejsou ve stejné skupině a vyznačují se nízkou váhou). Heuristika využívá povahu grafu tím, že se pokouší najít jeho uskupení, a podle vah přiřadí stavům v rámci každé skupiny jedinečný kód. Algoritmus vkládání probíhá následujícím způsobem (popsáno pseudokódem):

$GG = G_M$

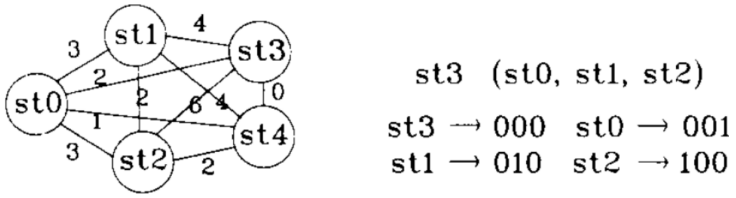
while (GG není prázdný) {

zvolte $v_l \in GG$, $y_i \in GG$ tak $\sum_{i=l}^{N_b} we(e_M(v_l, v_i))$ *bylo maximum*

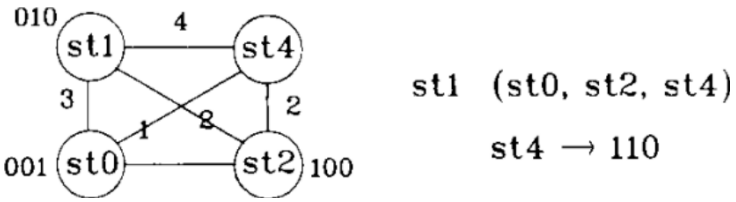
zakódujte y_i a v_l *nevyužitými kódy s minimální vzdáleností kódů*

$GG = GG = v_l$ }

Algoritmus vkládání znázorníme na obrázku 3.26 pomocí 5 stavů, které mají být kódovány pomocí 3 bitů. Pro začátek si zvolíme jeden stav např. *st3*. Tento stav má 3 hrany vedoucí ke stavům *st0, st1, st2* (hrana ke stavu *st4* má nulovou váhu, proto se do grafu nezapočítává). Stav *st3* zakódujeme nejnižším binárním kódem 000. Ostatní stavy budeme kódovat tak, aby měly Hammingovu vzdálenost rovnou 1 od stavu *st3*. Pro zakódování stavu *st4* si upravíme graf (obrázek 3.27). Zjistíme, že stav *st4* má hrany ke stavů *st0, st1,st2*. Proto stav *st4* zakódujeme tak, aby jeho Hammingova vzdálenost byla 1 od stavů *st0, st1,st2*.



Obrázek 3.26: a) Příklad algoritmu vkládání



Obrázek 3.27: b) Příklad algoritmu vkládání

3.5.6 Genetické algoritmy

GA (genetické algoritmy) se snaží pomocí evolučního principu biologie nalézt řešení složitých problémů. Tyto algoritmy jsou založeny na dědičnosti, mutaci, přirozeného výběru a křížení [1].

Obecný postup pro kódování stavů GA je, že začneme s populací náhodných jedinců, kteří jsou vybráni pro použití křížení (2 rodiče vygenerují potomka). Mutace náhodně přenese informace na potomka, které se zpět vloží do populace. Když populace dosáhne dané velikosti (obvykle dvakrát oproti původní), je dokončena jedna generace. Pro snížení velikosti populace (do původní) se použije výběrové řízení a opět může začít nová generace. Všechny výběry se provádí podle pravděpodobnosti a vhodnosti každého jednotlivce. Některé úvahy provedené *Whitley* jsou platné i pro SAP [3].

4 Implementace kódovacího algoritmu DAG

Praktická část této práce se zabývá vývojem programu, který umožňuje zakódovat stavy výše uvedenými metodami. Jeho výstup byl použit pro srovnání ceny obvodů testovaných automatů. Pro rychlý a snadný vývoj bylo využito programovacího jazyka C♯ a vývojového prostředí Microsoft Visual Studio 2010.

Metoda DAG je algoritmicky náročnější než ostatní metody, které byly využity pro porovnávání cen obvodů. Proto kódování stavů metodou DAG bylo naprogramováno, u ostatních metod bylo kódování stavů provedeno ručně.

4.1 Třída Stav

Tato třída uchovává informace o jednotlivých stavech a obsahuje privátní proměnné:

- název stavu (původní stav),
- následující stavy,
- výstupy.

4.1.1 Metoda ToString

Data pro tento program jsou zadána v textovém souboru jako sada znaků. Tyto znaky je třeba separovat a zařadit do příslušných polí. První znak řádku je vždy původní stav, který je oddělen od následujících stavů znakem „ ; “. Následující stavy se mezi sebou oddělují znakem „ , “. Mezi posledním následujícím stavem a výstupem je znak „ ; “. Konec řádky je značen „ ; “. Tímto způsobem nám vzniknou pole pro původní stavy, následující stavy a pole výstupu, se kterými budeme dále pracovat.

4.2 Třída StavovaTrida

Stavová třída obsahuje privátní proměnné:

- pole všech původních stavů
- mocnina (určuje kolik bitů bude potřeba k zakódování stavů)

V této třídě najdeme všechny metody používané pro výpočet matic, které potřebujeme k určení kódu stavů.

4.2.1 Metoda VypoctiMaticiVystupu

Matice výstupu se získá z množiny výstupních oddílů. Ty získáme tak, že z každého výstupu vezmeme první bit, pokud se rovná 0 zapíšeme do prvního pomocného pole odpovídající původní stav. Pokud se rovná 1 zapíšeme odpovídající stav do druhého pomocného pole. Tyto množiny poté zapíšeme do matice výstupů pomocí jedniček pro dané indexy.

4.2.2 Metoda VypoctiMaticiNasledniku

Tato metoda si vytvoří vazby mezi následujícími stavy daného původního stavu. A to tak, že postupně projde všechny původní stavy, zjistí všechny jejich následníky, a pokud následník není obsažen v dané množině, zapíše se do množiny následujících stavů. Tyto množiny se poté zapíšou do matice následníku, tzn. do matice se na daný index přičte jednička.

4.2.3 Metoda VypoctiMaticiPredchudcu

Matici předchůdců vytvoří metoda tak, že projde postupně všechny následující stavy a zjistí jejich původní stavy, které zapíše do pole předchůdců. Prvky tohoto pole se zapíší do matice stejným způsobem, jak tomu bylo u matice následníků.

4.2.4 Metoda VypoctiMaticiPrechodu

Zde si metoda vytvoří vazby mezi původními stavy a následujícími stavy. Vezme si první stav, zjistí jeho prvního následníka a přičte jedničku na dané místo v matici. Vezme další následující stav (pokud nějaký je) a opět přičte jedničku. Takto postupuje dokud neprojde všechny následující stavy původního stavu. Tento postup opakuje pro všechny stavy a zapisuje do jedné společné matice

4.2.5 Metoda VypoctiMaticiDAG

Matice DAG se počítá podle zadaného klíče, který určuje jakou hodnotou se budou násobit matice výstupů, matice předchůdců, matice následníků a matice přechodů. Klíčů je celkem 5 a podle výběru uživatele se rozhodne, jaké budou konstanty pro násobení matic.

4.2.6 Metoda HammingovaVzdalenost

K zakódování všech stavů, potřebujeme určitý počet bitů. Podle tohoto počtu metoda udělá všechny variace s opakováním pro prvky 0 a 1. Dále se vytvoří dolní trojúhelníková matice a do ní se podle indexů zapíše, o kolik bitů se navzájem liší tyto variace. To se provede tak, že každá variace se bude procházet bit po bitu a porovnávat s ostatními prvky, pokud hodnota daných bitů se bude lišit, přičte 1 do matice vzdáleností.

4.2.7 Metoda VypoctiPoleVah

Metoda pracuje s hodnotami, které má uložené v matici DAG. Sečte řádku a sloupec pro daný stav a uloží si hodnotu do pole pod indexem daného stavu. Takto projde všechny původní stavy a zapíše do pole, které se na konci cyklu seřadí sestupně podle hodnot. Stav, který má nejvyšší hodnotu (na nulté pozici v poli) je zakódován binárním číslem 0. Pokud se stane, že nejvyšší hodnota v poli není jediná, zjistí největší hodnoty z matice DAG pro tyto stavy s nejvyšší váhou. Stav, kterému náleží největší hodnota z matice DAG (a také nejvyšší váha v poli) je zakódován binárním číslem 0. Dále projdeme všechny buňky matice DAG, které náleží tomu stavu (s binárním kódem 0),

zjistíme nejvyšší hodnotu a pozici v matici. Tato pozice nám označí další stav, který budeme kódovat binárním kódem 1.

4.2.8 Metoda TabulkaPrirazeni

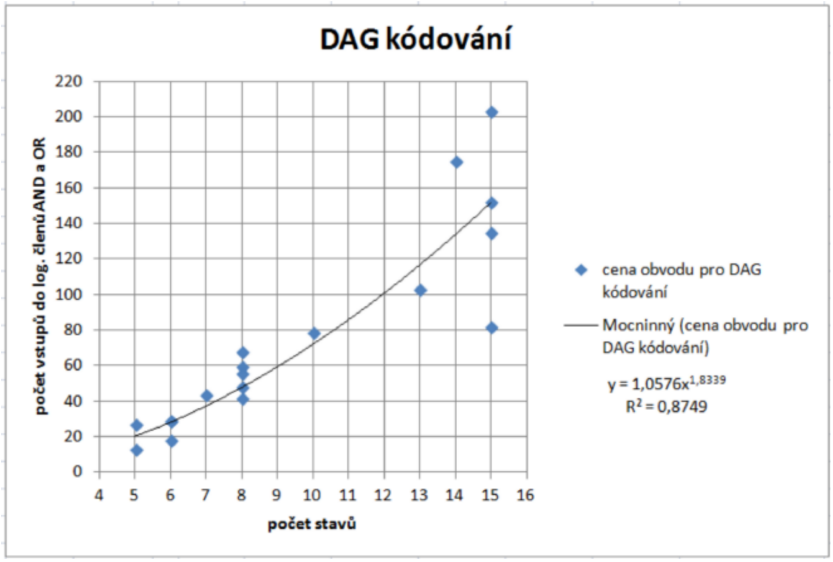
Metoda pracuje s maticí DAG a maticí vzdálenosti. Řádky matice jsou označeny indexy původních stavů a sloupce jsou označeny binárními kódy. Dva stavy v tuto chvíli jsou zakódované binárním kódem 0 a 1, proto první dva sloupce se inicializují na nulu. Příslušné řádky, které odpovídají zakódovaným stavům se také inicializují na nulu. Ostatní buňky tabulky obsahují hodnotu, která vychází ze vzorce $\sum_{i=0}^{s-1}\sum_{j=0}^{s-1} D(S_i, S_j)DAG_{ij}$, kde $D(S_i, S_j)$ jsou hodnoty z matice vzdáleností a DAG_{ij} jsou hodnoty z DAG matice. Tímto způsobem metoda vyplní celou tabulku a sečte sumu jednotlivých řádků. Zjistí, které řádce patří největší suma, v této řádce poté najde nejmenší hodnotu, která odpovídá danému sloupci s binární hodnotou. Řádka (s největší sumou) určila, jaký stav se bude kódovat a sloupec (nejnižší hodnota v řádce) určil, jakým kódem se tento stav bude kódovat. Pokud program nalezne více prvků s nejnižší nebo nejvyšší hodnotou, vybere vždy první nalezený. Binární hodnotu stavu zapíšeme do pole a vymažeme všechny hodnoty v tabulce přiřazení. Metoda si inicializuje buňky na nulu: za 1. pro řádky, které odpovídají již zakódovaným stavům a za 2. pro sloupce, jejichž kódy již byly použity. Tento proces se opakuje, dokud všechny stavy nejsou zakódovány a uloženy v poli.

5 Porovnání metod kódování

Pro lepší přehlednost bude každý druh kódování zobrazen pomocí grafu (kódování DAG obr. 5.1, kódování DAG1 obr. 5.2, kódování DAG2 obr. 5.3, kódování DAG3 obr. 5.4, kódování DAG4 obr. 5.5, binární kódování obr. 5.7, Grayův kód obr. 5.6 a výstupní kódování obr. 5.8). Pomocí mocninné spojnice trendu je v grafech vidět, že cena obvodu roste stálou rychlostí. Rovnice spojnice trendu je ve tvaru $y = c \times^b$, kde c a b jsou konstanty. Hodnota spolehlivosti grafu se značí R^2 (čím víc se tato hodnota blíží k jedné, tím víc se křivka spolehlivější vůči datům).

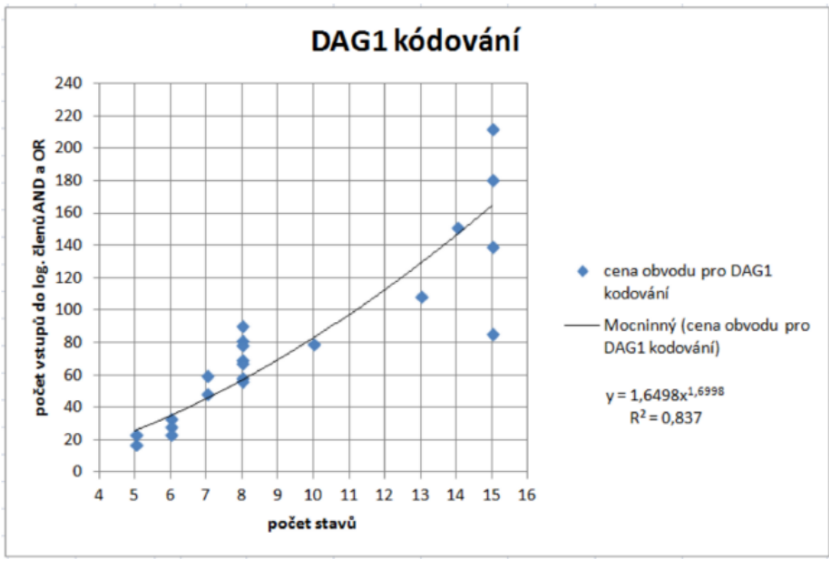
Dále je zobrazena tabulka všech dosažených výsledků (cen obvodů) 21 testovaných automatů (tabulka 5.4), kde jsou barevně označeny nejlepší výsledky (tzn. nejmenší cena) pro daný automat.

Poznámka: obrázky všech testovaných automatů jsou uvedeny v příloze

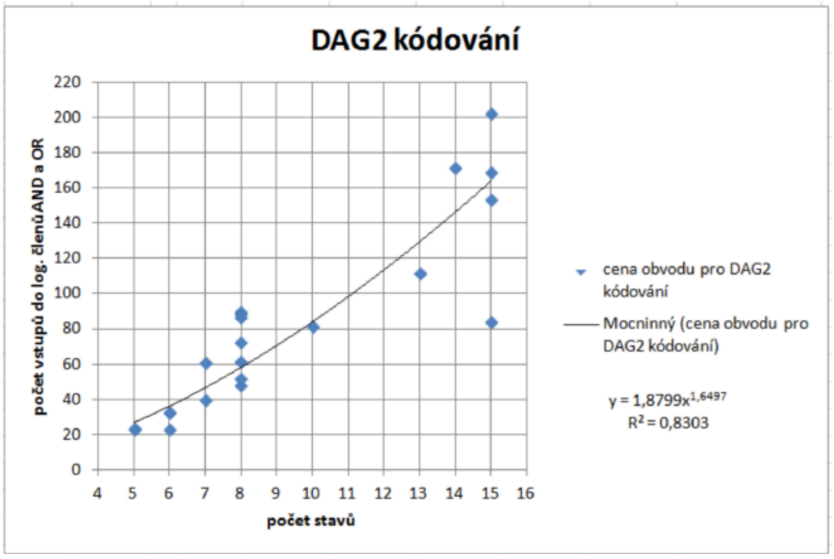


Obrázek 5.1: Výsledky kódování DAG

Z tabulky 5.1 lze vyčíst, které kódování má největší úspěšnost (co se týče hodnoty ceny) pro testované automaty. Tato tabulka pracuje pouze s třemi nejlepšími výsledky (ostatní výsledky cen zahazuje). Nejlepší výsledek (nejnižší cena) se násobí konstantou 3, druhý nejlepší se násobí konstantou 2 a třetí nejlepší se násobí konstantou 1. Pomocí těchto konstant se vypočítá účinnost daného kódování pro všechny testované automaty.

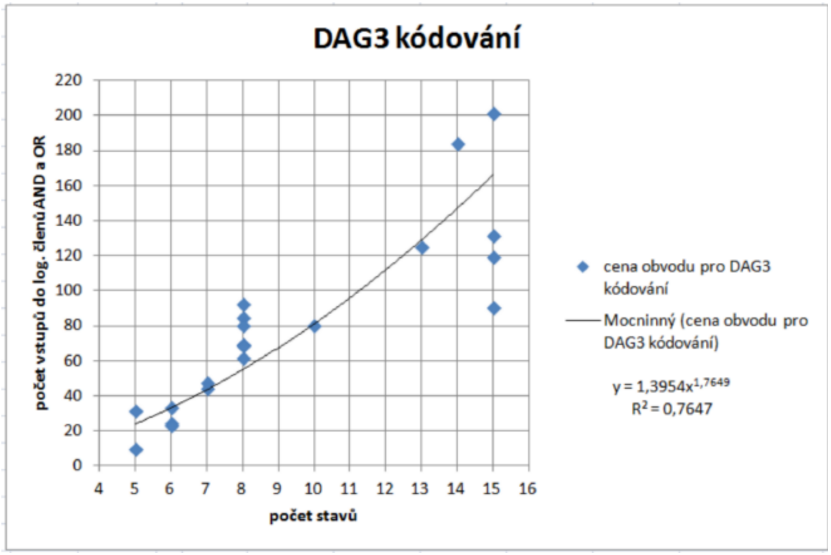


Obrázek 5.2: Výsledky kódování DAG1

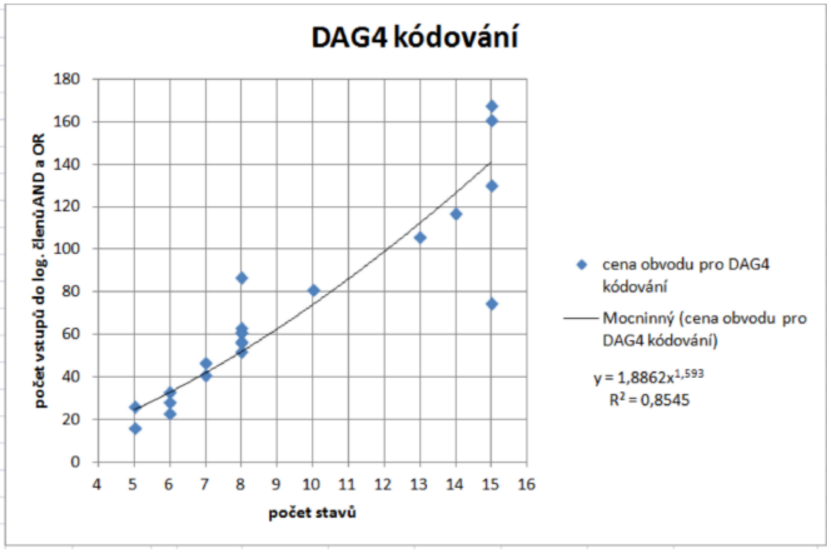


Obrázek 5.3: Výsledky kódování DAG2

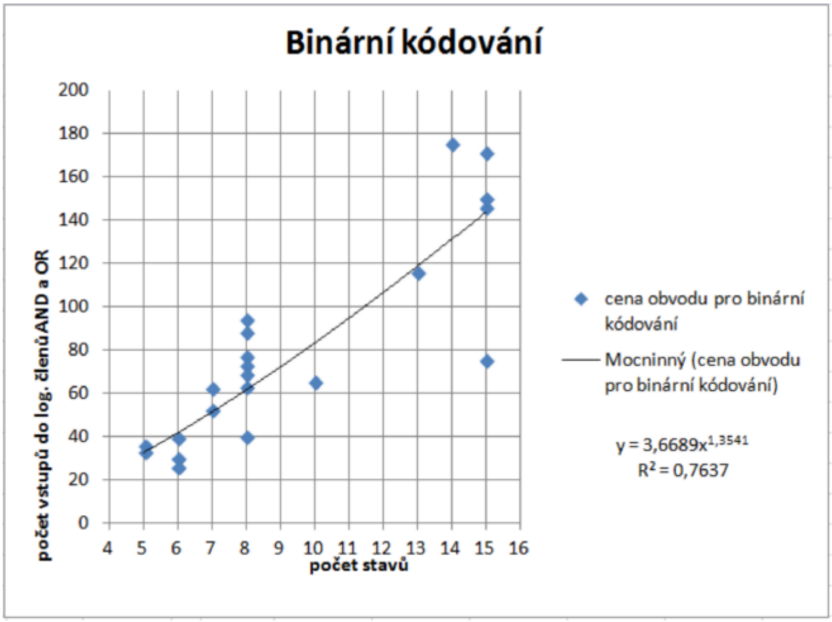
Dále rozdělíme tabulku na malé (do osmi stavů) a velké (minimálně devět stavů) automaty a přepočítáme úspěšnost použitých kódování (tabulky 5.2 a 5.3). Zhodnocení těchto výsledků je uvedeno v kapitole **6 Závěr**.



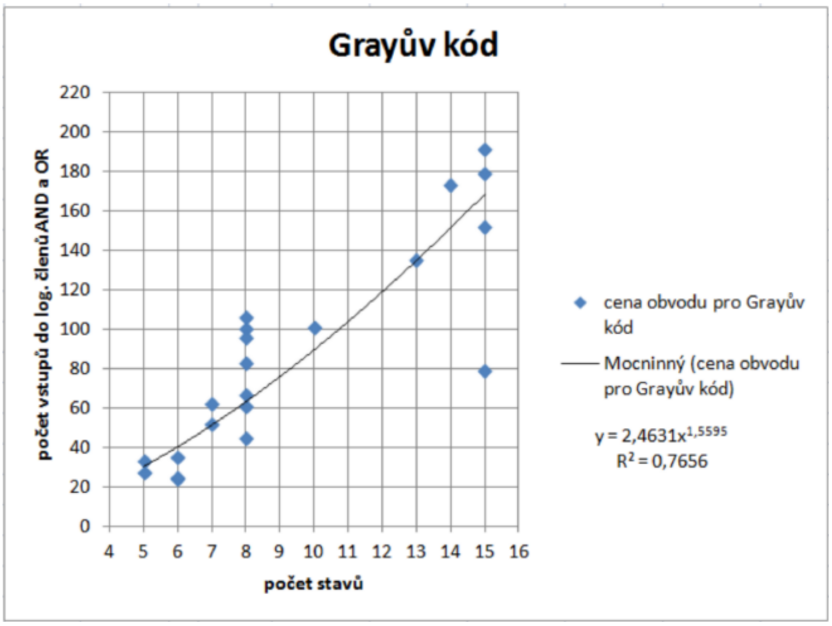
Obrázek 5.4: Výsledky kódování DAG3



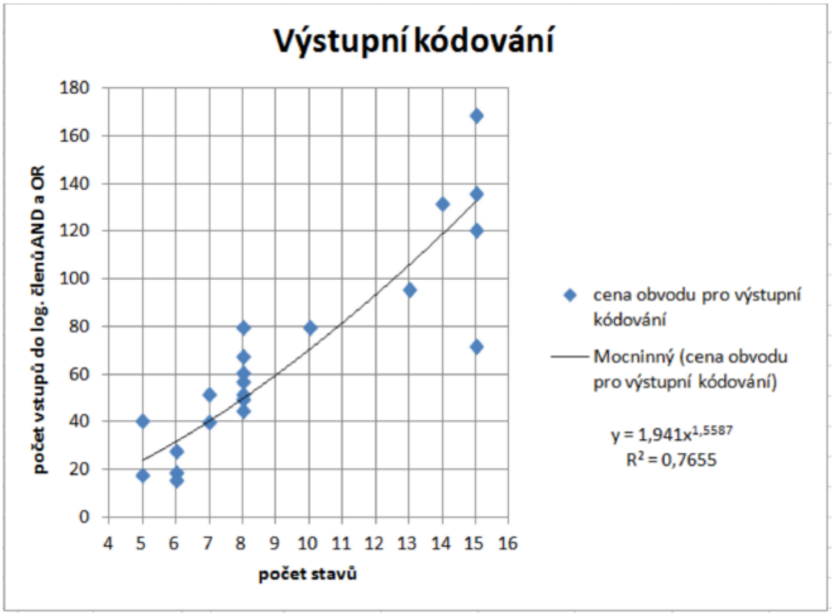
Obrázek 5.5: Výsledky kódování DAG4



Obrázek 5.6: Výsledky binárního kódování



Obrázek 5.7: Výsledky Grayova kódu



Obrázek 5.8: Výsledky výstupního kódování

Tabulka 5.1: Účinnost kódování pro všechny automaty

druh kódování	účinnost [%]
DAG	46.03
DAG1	5.13
DAG2	5.13
DAG3	11.97
DAG4	30.16
BIN	5.13
GRAY	11.13
OUTPUT	61.90

Tabulka 5.2: Účinnost kódování pro automaty s max. počtem stavů 8

druh kódování	účinnost [%]
DAG	64.29
DAG1	28.57
DAG2	26.19
DAG3	28.57
DAG4	47.62
BIN	11.90
GRAY	14.29
OUTPUT	57.14

Tabulka 5.3: Účinnost kódování pro automaty s min. počtem stavů 9

druh kódování	účinnost [%]
DAG	19.50
DAG1	14.29
DAG2	9.52
DAG3	47.62
DAG4	42.86
BIN	19.50
GRAY	4.76
OUTPUT	66.67

Tabulka 5.4: Výsledky kódování

název automatu	počet stavů	počet následníků	druhy kódování							
			DAG	DAG[1]	DAG[2]	DAG[3]	DAG[4]	BIN	GRAY	OUTPUT
Hektor	5	2	13	17	24	9	16	36	27	18
Ponorka	5	2	27	23	23	31	26	33	33	41
Špagetka	6	2	29	33	33	33	33	39	35	28
Kocour	6	2	29	28	33	24	28	30	24	19
Včela	6	2	18	23	23	23	23	26	25	16
Bobina	7	2	53	48	40	47	41	52	52	52
Brejlovec	7	2	44	59	61	44	47	62	62	40
Adéla	8	2	48	58	48	61	52	40	45	52
Raketa	8	2	42	69	52	69	61	63	67	50
Mandarinka	8	2	60	67	62	69	63	77	61	45
Esmeralda	8	3	68	90	90	92	87	94	83	57
Plecháč	8	3	57	81	89	84	57	73	100	61
Princezna	8	3	68	78	73	80	61	88	96	80
Kredenc	8	3	56	56	87	68	56	69	106	68
Delfín	10	2	79	79	82	80	81	65	101	80
Rohlík	13	2	103	108	112	125	106	116	135	96
Žralok	14	2	175	151	172	184	117	175	173	132
Sysel	15	2	82	85	84	90	75	75	79	72
Žehlička	15	2	135	139	154	119	130	146	152	121
Banán	15	3	203	180	169	201	168	171	179	169
Dvojče	15	3	152	212	203	131	161	150	191	136

nejlepší výsledek
2. nejlepší výsledek
3. nejlepší výsledek

6 Závěr

Předem je třeba poznamenat, že nelze určit, pro který druh automatů (čítač, obecný automat,...) je jedno kódování lepší než druhé. A ani neexistuje kódování, které nám 100% zaručí nejnižší cenu obvodu u všech automatů.

Pro svou práci jsem automaty rozdělila na dva druhy: malé automaty (do 8 stavů) a velké automaty (minimálně 9 stavů) a vypočítala jejich ceny obvodů a účinnosti pro kódování: *DAG, DAG1, DAG2, DAG3, DAG4, binární kódování, Grayův kód, výstupní kódování*. Kde *DAG1, DAG2, DAG3, DAG4* je upravený algoritmus DAG.

Ve skupině pro malé automaty se ukázalo, že nejlepším kódováním je *DAG* s úspěšností 64.29%. Druhým nejlepším přiřazením kódu je *výstupní kódování* s úspěšností 57.14%. Je třeba podotknout, že algoritmus *DAG* je na výpočet složitější než *výstupní kódování*. Nejhůře dopadlo *binární kódování a Grayův kód*, kde účinnost nepřesáhla ani 15%.

U velkých automatů je nejlepším přiřazením kódu *výstupní kódování* s účinností 66.67%. Druhý nejlepším kódováním je *DAG3* s účinností 47.62%. Nejhorší kódování pro velké automaty je *Grayův kód* s účinností 4.76%.

Dalším porovnáním je kódování *DAG, DAG1, DAG2, DAG3, DAG4*. Pro malé automaty nejlepším kódováním je *DAG*, druhým nejlepším je *DAG4* s úspěšností 47.62%. Úspěšnost ostatních algoritmů nepřesáhla 30%. Pro velké automaty je nejlepším algoritmem *DAG3* s úspěšností 47.62%, druhým nejlepším je *DAG4* s úspěšností 42.86%. Původní algoritmus *DAG* pro velké automaty nepřesáhl úspěšnosti kódování ani 20%.

Přehled zkratek

- **CLK**- [*Clock*] synchronizační hodinový pulz
- **SAP**- [*StateAssignmentProblem*] problém kódování stavů
- **DAG**- [*DesiredAdjacencyGraf*] kódování stavů podle grafu sousednosti
- **FSM**- [*FiniteStateMachine*] konečný stavový automat
- **IAT**- [*IncompletAssignmentTable*] neúplná tabulka přiřazení
- **GA**- [*GeneticAlgorithm*] genetické algoritmy
- **BIN**- binární kódování
- **GRAY**- Grayovo kódování
- **OUTPUT**- výstupní kódování
- **KO**- klopný obvod
- **GM**- [*graphembedding*] vkládání grafů

Seznam obrázků

Seznam tabulek

Literatura

[1] Amaral, J.: Designing genetic algorithms for the state assignment problem. *IEEE Wplore*, 1995.

[2] Amaral, J. N.; Cunha, W. C.: State assignment algorithm for incompletely speciffied finite state machines. In *VLSI Design Methodologies*, SPIE Proceedings, listopad 1990.

[3] Amaral, J. N.; Tumer, K.; Ghosh, J.: Designing Genetic Algorithms for the State Assignment Problem. *IEEE Transactions on Systems*, prosinec 1995: s. 623–628.

[4] Bambušková, K.: *Konstrukce konečného automatu III.* Diplomová práce, FEI, VŠB-TU Ostrava, Ostrava, 2005.

[5] Bokr, J.: *Logické obvody a automaty.* Katedra informatiky a výpočetní techniky, 2003.

[6] Bukka: Bmin - Program Visualizer of Boolean minimization. http://bukka.eu/cs/bmin/, online; citace 28.2.2011.

[7] Bukka: Bmin - Visualizer of Boolean minimization. http://bukka.eu/cs/bmin/1.0.1, online; citace 28.2.2011.

[8] Devadas, S.: MUSTANG: State Assignment of Finite State Machines Targeting Multilevel Logic Implementations. *Computer-Aided Design of Integrated Circuits and Systems*, prosinec 1988: s. 1290–1300.

[9] Dolotta, T.: The Coding of Internal States of Sequential Circuits. *Electronic Computers, IEEE Transactions*, říjen 1964: s. 549 – 652.

[10] Egert, J.: *Algoritmy pro kódování stavů konečného automatu.* Diplomová práce, České vysoké učení technické v Praze, červen 2007.

[11] Ghosh, A.: Estimation of Average Swiching Activitz in Combinational and Sequential Circuits. Technická zpráva, Mitsubishi Electronics America, 1992.

[12] Hartmais, J.: On the State Assignment Problem foe Sequential Machines. *Electronic Computers, IRE Transactions*, červen 1961: s. 157–165.

[13] Makki, R.: Analysis and Characterization of State Assignment Techniques for Sequential Machines. *VLSI Design*, 1994: s. 81–88.

[14] Pinker, J.: *Číslicové systmy a jazyk VHDL*. 2006.

[15] Prokeš, M.: *Diplomová práce (FEL code)*. Diplomová práce, České vysoké učení technické v Praze, Praha, 2001.

[16] Sentovich, E.: Sequential circuit design using synthesis and optimization. *VLSI in Computers and Processors*, říjen 1992: s. 328–333.

[17] Spinellis, D.: Graphviz - Graph Visualization Software. http://graphviz.org/, online; citace 28.2.2011.

[18] Story, J.: Optimum State Assignment for Synchronous Sequential Circuits. *IEEE Transactions on Computers*, prosinec 1972: s. 1365–1373.

[19] Vavřička, V.: Logické systemy, 2011, sekvenční obvody - Optimalizace konečných automatů.

Přílohy

Uživatelská příručka

Program na kódování stavů spustíme pomocí aplikace *Projekt1.exe*. Konzole nás vyzve k zadání názvu stavové tabulky, která je zadána v textovém souboru. Aby program mohl správně pracovat, je potřeba mít stavovou tabulku ve správném formátu. První sloupec stavové tabulky jsou původní stavy, dále následující stavy a výstupy. Každý sloupec je oddělen znakem „;“. Pokud máme více následujících stavů oddělujeme je znakem „,“. Jednotlivé bity výstupu také oddělujeme znakem „,“. Konec řádky značíme „;“. Formát stavové tabulky může vypadat:

puvodníStav;nasledujicíStav1,následujícíStav2;výstup1,výstup0;

Příklad stavové tabulky:

S0;S1,S2;0,0;

S1;S4,S3;1,1;

S2;S4,S3;1,0;

S3;S4,S4;0,1;

S4;S0,S0;0,0;

Po zadání názvu stavové tabulky si vybereme druh kódování (obrázek. 1):

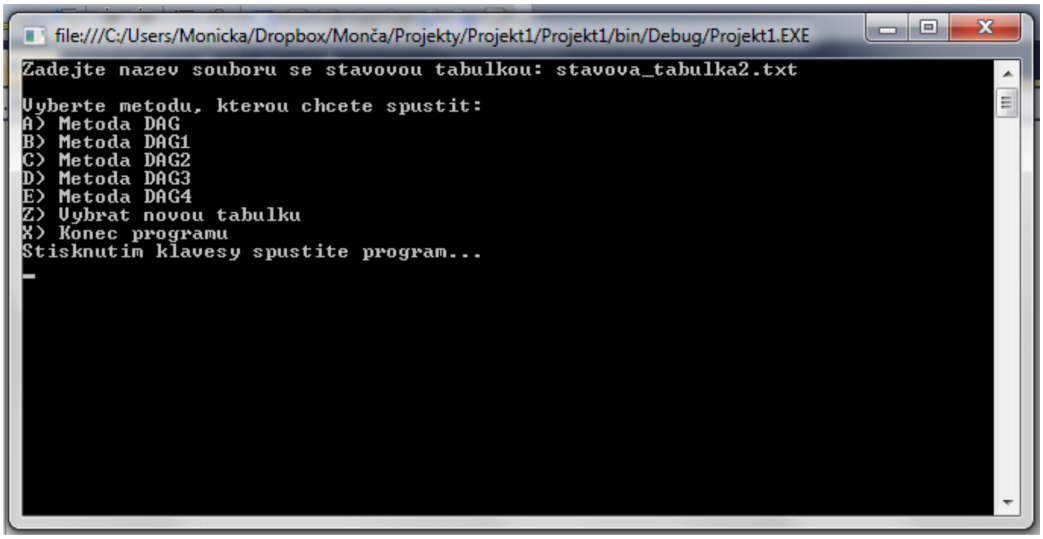
a) metoda DAG

b) metoda DAG1

c) metoda DAG2

d) metoda DAG3

e) metoda DAG4



Obrázek 1: uživatelská konzole

Podle zvolené metody se provede druh kódování a zapíše se do souboru kodovani.csv. V souboru jsou uloženy potřebné matice pro výpočet kódování, indexy stavů, váhy stavů a kódování daného stavu.

Pokud si nevybereme ani jeden druh kódování je možné volbou „x“ ukončit program, popřípadě volbou „z“ zvolit si jinou stavovou tabulku.

Dále v bakalářské práci byl použit program na minimalizaci logických funkcí **Bmin**. Manuál k tomuto programu nalezneme na internetových stránkách http://bukka.eu/cs/bmin/1.0.1 [7] a program je možné stáhnout z adresy http://bukka.eu/cs/bmin/ [6].

Testovací automaty

Nejprve pro vykreslení grafů testovaných automatu byl použit program *Graphviz - Graph Visualization Software* [17], bohužel pro velkém automaty výstup toho programu nebyl ideální (grafy automatů byly velmi nepřehledné). Proto jsem zvolila zápis automatu pomocí tabulky přechodu a výstupů.

Zde jsou uvedeny všechny testovací automaty (vlevo je vždy tabulka přechodů a vpravo výstupy stavů daného automatu).

stavy	následující stavy	
	$I_0 = 0$	$I_0 = 1$
s0	s1	s2
s1	s4	s3
s2	s4	s3
s3	s4	s4
s4	s0	s0

stavy	výstupy
s0	00
s1	11
s2	10
s3	01
s4	00

Obrázek 2: Hektor

stavy	následující stavy	
	$I_0 = 0$	$I_0 = 1$
s0	s1	s1
s1	s2	s3
s2	s3	s4
s3	s4	s4
s4	s2	s1

stavy	výstupy
s0	111
s1	100
s2	001
s3	110
s4	011

Obrázek 3: Ponorka

stavy	následující stavy	
	$l_0 = 0$	$l_0 = 1$
s0	s1	s2
s1	s2	s3
s2	s3	s4
s3	s4	s5
s4	s5	s0
s5	s0	s1

stavy	výstupy
s0	000
s1	100
s2	110
s3	111
s4	101
s5	001

Obrázek 4: Špagetka

stavy	následující stavy	
	$l_0 = 0$	$l_0 = 1$
s0	s1	s1
s1	s2	s3
s2	s4	s4
s3	s5	s5
s4	s1	s3
s5	s4	s4

stavy	výstupy
s0	011
s1	001
s2	100
s3	110
s4	111
s5	101

Obrázek 5: Kocour

stavy	následující stavy	
	$l_0 = 0$	$l_0 = 1$
s0	s5	s5
s5	s6	s7
s6	s9	s9
s7	s8	s8
s8	s9	s9
s9	s8	s8

stavy	výstupy
s0	111
s5	000
s6	001
s7	100
s8	110
s9	011

Obrázek 6: Včela

stavy	následující stavy	
	$I_0 = 0$	$I_0 = 1$
s1	s2	s3
s2	s4	s3
s3	s4	s2
s4	s5	s6
s5	s7	s6
s6	s7	s5
s7	s5	s6

stavy	výstupy
s1	001
s2	010
s3	100
s4	111
s5	101
s6	011
s7	000

Obrázek 7: Bobina

stavy	následující stavy	
	$I_0 = 0$	$I_0 = 1$
s0	s1	s6
s1	s2	s0
s2	s3	s1
s3	s4	s2
s4	s5	s3
s5	s6	s4
s6	s0	s5

stavy	výstupy
s0	000
s1	011
s2	100
s3	111
s4	110
s5	101
s6	001

Obrázek 8: Brejlovec

stavy	následující stavy	
	$I_0 = 0$	$I_0 = 1$
s0	s6	s6
s6	s7	s8
s7	s9	s10
s8	s11	s12
s9	s10	s10
s10	s9	s11
s11	s10	s12
s12	s11	s11

stavy	výstupy
s0	011
s6	001
s7	100
s8	110
s9	111
s10	101
s11	010
s12	000

Obrázek 9: Adéla

stavy	následující stavy	
	$I_0 = 0$	$I_0 = 1$
s0	s1	s1
s1	s2	s3
s2	s3	s1
s3	s4	s7
s4	s5	s3
s5	s6	s4
s6	s7	s5
s7	s1	s5

stavy	výstupy
s0	1000
s1	1011
s2	1100
s3	1111
s4	1110
s5	1101
s6	1001
s7	1010

Obrázek 10: Raketa

Obrázek 11: Mandarinka

stavy	následující stavy		
	$I_0 = 00$	$I_0 = 01$	$I_0 = 10$
s0	s1	s1	s1
s1	s4	s5	s6
s2	s7	s1	s1
s3	s2	s4	s5
s4	s6	s7	s1
s5	s1	s2	s3
s6	s4	s5	s7
s7	s1	s1	s2

stavy	výstupy
s0	110
s1	101
s2	100
s3	011
s4	010
s5	001
s6	000
s7	111

Obrázek 12: Esmeralda

stavy	následující stavy		
	$I_0 = 00$	$I_0 = 01$	$I_0 = 10$
s0	s1	s1	s1
s1	s7	s6	s5
s2	s4	s3	s1
s3	s1	s2	s5
s4	s5	s6	s7
s5	s1	s2	s3
s6	s4	s5	s7
s7	s1	s2	s3

stavy	výstupy
s0	1110
s1	1101
s2	1100
s3	1011
s4	1010
s5	1001
s6	1000
s7	0111

Obrázek 13: Plecháč

stavy	následující stavy		
	$I_0 = 00$	$I_0 = 01$	$I_0 = 10$
s0	s8	s8	s8
s8	s14	s13	s12
s9	s11	s10	s8
s10	s14	s13	s12
s11	s12	s13	s14
s12	s13	s14	s11
s13	s11	s12	s14
s14	s8	s9	s10

stavy	výstupy
s0	0110
s8	0101
s9	0100
s10	0011
s11	0010
s12	0001
s13	0000
s14	1111

Obrázek 14: Princezna

stavy	následující stavy		
	$I_0 = 00$	$I_0 = 01$	$I_0 = 10$
s0	s8	s8	s8
s8	s9	s10	s11
s9	s12	s13	s14
s10	s8	s9	s11
s11	s12	s13	s14
s12	s8	s9	s10
s13	s11	s12	s14
s14	s8	s9	s10

stavy	výstupy
s0	110
s8	101
s9	100
s10	011
s11	010
s12	001
s13	000
s14	111

Obrázek 15: Kredenc

stavy	následující stavy	
	$I_0 = 0$	$I_0 = 1$
s0	s1	s5
s1	s2	s3
s2	s3	s4
s3	s4	s4
s4	s2	s1
s5	s6	s7
s6	s9	s9
s7	s8	s8
s8	s9	s9
s9	s8	s8

stavy	výstupy
s0	1111
s1	0000
s2	1110
s3	0001
s4	1101
s5	0010
s6	1100
s7	0011
s8	1011
s9	0100

Obrázek 16: Delfín

stavy	následující stavy	
	$I_0 = 0$	$I_0 = 1$
s0	s1	s6
s1	s2	s3
s2	s4	s4
s3	s5	s5
s4	s1	s3
s5	s4	s4
s6	s7	s8
s7	s9	s10
s8	s11	s12
s9	s10	s10
s10	s9	s11
s11	s10	s12
s12	s11	s11

stavy	výstupy
s0	1001
s1	0101
s2	1010
s3	0100
s4	1011
s5	0011
s6	1100
s7	0010
s8	1101
s9	0001
s10	1110
s11	0000
s12	1111

Obrázek 17: Rohlík

stavy	následující stavy	
	$I_0 = 0$	$I_0 = 1$
s0	s5	s4
s1	s10	s9
s2	s10	s11
s3	s12	s7
s4	s8	s13
s5	s0	s6
s6	s10	s1
s7	s13	s2
s8	s11	s7
s9	s10	s4
s10	s9	s7
s11	s4	s1
s12	s13	s0
s13	s9	s7

stavy	výstupy
s0	0000
s1	0001
s2	0010
s3	0011
s4	0100
s5	1111
s6	1110
s7	1101
s8	1100
s9	1011
s10	1010
s11	1001
s12	1000
s13	0111

Obrázek 18: Žralok

stavy	následující stavy	
	$I_0 = 0$	$I_0 = 1$
s0	s1	s2
s1	s3	s4
s2	s5	s6
s3	s7	s8
s4	s9	s10
s5	s11	s12
s6	s13	s14
s7	s0	s0
s8	s0	s0
s9	s0	s0
s10	s0	s0
s11	s0	s0
s12	s0	s0
s13	s0	s0
s14	s0	s0

stavy	výstupy
s0	000
s1	001
s2	010
s3	100
s4	111
s5	101
s6	011
s7	000
s8	000
s9	000
s10	000
s11	000
s12	000
s13	000
s14	000

Obrázek 19: Sysel

stavy	následující stavy	
	$I_0 = 0$	$I_0 = 1$
s0	s1	s8
s1	s3	s3
s2	s3	s1
s3	s4	s7
s4	s5	s3
s5	s6	s4
s6	s7	s5
s7	s1	s5
s8	s10	s9
s9	s10	s11
s10	s8	s9
s11	s14	s13
s12	s10	s8
s13	s11	s14
s14	s13	s8

stavy	výstupy
s0	1000
s1	1011
s2	1100
s3	1111
s4	1110
s5	1101
s6	1001
s7	1010
s8	0011
s9	0100
s10	0111
s11	0110
s12	0101
s13	0001
s14	0010

Obrázek 20: Žehlička

stavy	následující stavy		
	$I_0 = 00$	$I_0 = 01$	$I_0 = 10$
s0	s1	s8	s1
s1	s4	s5	s6
s2	s7	s1	s1
s3	s2	s4	s5
s4	s6	s7	s1
s5	s1	s2	s3
s6	s4	s5	s7
s7	s1	s1	s2
s8	s9	s10	s11
s9	s12	s13	s14
s10	s8	s9	s11
s11	s12	s13	s14
s12	s8	s9	s10
s13	s11	s12	s14
s14	s8	s9	s10

stavy	výstupy
s0	1110
s1	1101
s2	1100
s3	1011
s4	1010
s5	1001
s6	1000
s7	0111
s8	0101
s9	0100
s10	0011
s11	0010
s12	0001
s13	0000
s14	1111

Obrázek 21: Banán

stavy	následující stavy		
	$I_0 = 00$	$I_0 = 01$	$I_0 = 10$
s0	s1	s8	s1
s1	s7	s6	s5
s2	s4	s3	s1
s3	s1	s2	s5
s4	s5	s6	s7
s5	s1	s2	s3
s6	s4	s5	s7
s7	s1	s2	s3
s8	s14	s13	s12
s9	s11	s10	s8
s10	s14	s13	s12
s11	s12	s13	s12
s12	s13	s14	s11
s13	s11	s12	s14
s14	s8	s9	s10

stavy	výstupy
s0	1110
s1	1101
s2	1100
s3	1011
s4	1010
s5	1001
s6	1000
s7	0111
s8	0101
s9	0100
s10	0011
s11	0010
s12	0001
s13	0000
s14	1111

Obrázek 22: Dvojče