ZESZYTY NAUKOWE POLITECHNIKI BIAŁOSTOCKIEJ 2008
Informatyka – Zeszyt 3

Tomasz Grześ[1], Valery Salauyou[1]

ALGORYTMY KODOWANIA STANÓW WEWNĘTRZNYCH AUTOMATU SKOŃCZONEGO DO MINIMALIZACJI POBORU MOCY

Streszczenie: Kodowanie stanów wewnętrznych automatu skończonego jest jednym z ważniejszych procesów podczas syntezy automatu. Zastosowanie odpowiedniego algorytmu pozwala m.in. obniżyć pobór mocy. W artykule skoncentrowano się na algorytmach minimalizujących pobór mocy. Przeprowadzono badania nad algorytmem kodowania kolumnowego, opisanego w pracy [1] oraz nad dwoma algorytmami opracowanymi przez autorów: sekwencyjnym [7] oraz rafinacyjnym. Badania przeprowadzono na standardowych układach testowych, opracowanych w Microelectronics Center of North Carolina [9]. Wyniki badań wykazują znaczące zmniejszenie poboru mocy układów zakodowanych z wykorzystaniem algorytmu sekwencyjnego w porównaniu z poborem z wykorzystaniem algorytmu kodowania kolumnowego (średnio o 12%); zastosowanie algorytmu rafinacyjnego pozwoliło obniżyć moc średnio o kolejny 1%.

Słowa kluczowe: Automat skończony, kodowanie stanów, obniżanie poboru mocy

1. Wstęp

Problem kodowania stanów wewnętrznych ma niebagatelne znaczenie podczas implementacji automatu skończonego w układzie cyfrowym (np. w strukturze programowalnej). Dzięki zastosowaniu odpowiednich algorytmów istnieje możliwość lepszego wykorzystania struktury, jak również obniżenia poboru mocy przez układ, co jest szczególnie ważne przy konstruowaniu systemów cyfrowych mobilnych, zasilanych bateryjnie, jak również dla zwiększenia wydajności i szybkości systemu.

Moc pobieraną przez układ CMOS można obliczyć z przedstawionego poniżej równania [5]:

$$P_a = \frac{1}{2} \cdot \frac{V_{DD}^2}{T_{cycle}} \cdot N_a \cdot C_a \tag{1}$$

[1] Wydział Informatyki, Politechnika Białostocka, Białystok

gdzie: V_{DD} – napięcie zasilające układ; T_{cycle} – czas, względem którego rozpatrywana jest aktywność przełączania (czas obliczania średniej liczby zmian stanu); C_a – pojemność wyjściowa elementu a; N_a – aktywność przełączania (średnia liczba zmian stanu wyjścia w czasie trwania jednego cyklu T_{cycle}) elementu a.

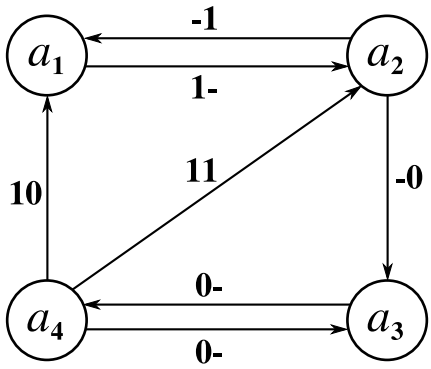
Zmniejszenie mocy dokonuje się poprzez zminimalizowanie wartości występujących po prawej stronie równania (1). Jednakże zmniejszenie V_{DD} oraz C_a wymaga zmiany technologii wykonania układu scalonego. Wartość T_{cycle} wynika z wymaganej prędkości pracy układu (zależy od częstotliwości pracy) i jest dla danej aplikacji stała. Jedynym parametrem, który można zminimalizować w sposób programowy, bez ingerowania w fizyczną budowę układu, jest aktywność przełączania N_a.

W układach sekwencyjnych aktywność przełączania jest ściśle związana ze sposobem kodowania stanów wewnętrznych. Zmiana stanu z a_i na a_j powoduje zmianę wartości kodu stanu zapamiętanego w przerzutnikach. Każda zmiana stanu przerzutnika powoduje wydzielenie mocy. W związku z tym zmiana sposobu kodowania będzie wpływała na aktywność przełączania, a co za tym idzie na ilość pobieranej mocy. Przykładami algorytmów kodowania stanów wewnętrznych, prowadzących do obniżenia poboru mocy są: algorytmy z prac [6] i [4], w których zastosowano wyżarzanie (ang. *annealing*) oraz algorytm z pracy [1], gdzie zastosowano kodowanie kolumnowe (ang. *column-based*).

2. Moc w układach sekwencyjnych

Dany jest automat skończony $F = \{A, X, Y, \phi, \psi, a_1\}$, gdzie A – zbiór stanów wewnętrznych, X – zbiór wektorów wejściowych, Y – zbiór wektorów wyjściowych, ϕ – funkcja przejść, ψ – funkcja wyjść, a_1 – stan początkowy. Dodatkowo zbiór stanów wewnętrznych A składa się ze skończonej liczby M stanów $A = \{a_1, a_2, \ldots, a_M\}$, gdzie M – liczba stanów automatu. Automat można przedstawić graficznie w postaci grafu przejść. Na rys. 1 przedstawiono graf przejść przykładowego automatu skończonego. Węzłami grafu są stany automatu (a_1, a_2, a_3 oraz a_4), natomiast krawędzie odpowiadają przejściom między stanami, które następują po pojawieniu się danego wektora wejściowego (np. przejście ze stanu a_2 do a_3 nastąpi, gdy na wejściu pojawi się wartość 00 lub 10). Na rysunku pominięto przejścia w ten sam stan, gdyż nie powodują one wydzielenia mocy w układzie.

Automaty skończone można opisywać za pomocą dyskretnych łańcuchów Markowa [2], co pozwala obliczać wartości prawdopodobieństw przejścia między stanami. Prawdopodobieństwa statyczne, określające prawdopodobieństwo znalezienia automatu w określonym stanie w chwili $t \to \infty$, można obliczyć korzystając z równań Chapmana-Kołmogorowa [8]. Zakładając zerowy współczynnik korelacji pomiędzy



Rysunek 1. Graf przejść przykładowego automatu skończonego

wartościami prawdopodobieństw wektorów wejściowych dla każdego stanu a_i automatu skończonego $(1 \leqslant i \leqslant M)$ można zapisać układ równań (2):

$$P(a_i) = \sum_{k=1}^{M} P(a_k) \cdot P(I_{ki}) \tag{2}$$

gdzie: $P(a_i)$ – prawdopodobieństwo statyczne stanu a_i; $P(a_k)$ – prawdopodobieństwo statyczne stanu a_k; $P(I_{ki})$ – prawdopodobieństwo pojawienia się wektora wejściowego I_{ki}, który spowoduje przejście automatu ze stanu a_k do stanu a_i [8].

Układ równań (2) jest liniowo zależny (dowolne równanie może zostać wyprowadzone z $M-1$ pozostałych), dlatego jedno z równań należy zastąpić równaniem (3), wynikającym z definicji prawdopodobieństwa:

$$\sum_{i=1}^{M} P(a_i) = 1 \tag{3}$$

Wybór równania, które należy zastąpić równaniem (3), nie wpływa na wynik obliczeń. Rozwiązaniem układu M równań (2) oraz (3) jest wektor prawdopodobieństw statycznych stanów automatu skończonego.

Prawdopodobieństwo zmiany stanu z a_i na a_j, pod warunkiem, że automat znajduje się w stanie a_i, jest zależne od prawdopodobieństwa pojawienia się na wejściu wektora I_{ij}, który spowoduje zmianę stanu z a_i na a_j i można je wyliczyć korzystając z równania (4):

$$P(a_i \rightarrow a_j) = P(a_i) \cdot P(I_{ij}) \tag{4}$$

gdzie: $P(a_i \to a_j)$ – prawdopodobieństwo zmiany stanu z a_i na a_j; $P(a_i)$ – prawdopodobieństwo statyczne stanu a_i; $P(I_{ij})$ – prawdopodobieństwo pojawienia się wektora wejściowego I_{ij}, który spowoduje zmianę stanu automatu z a_i na a_j.

Wyliczone prawdopodobieństwa pozwalają obliczyć aktywność przełączania przerzutników, a co za tym idzie moc wydzieloną w układzie [3].

Graf przejść automatu jest grafem skierowanym, jednakże zmiana stanu automatu z a_i na a_j wymaga przełączenia takiej samej liczby przerzutników, co zmiana stanu z a_j na a_i. Pozwala to przekształcać graf skierowany w nieskierowany. Wagi poszczególnych krawędzi będą miały wartość [1]:

$$w_{i,j} = P(a_i \to a_j) + P(a_j \to a_i) \tag{5}$$

Przykład

Dla grafu przejść z rys. 1 równania (2) będą miały postać:

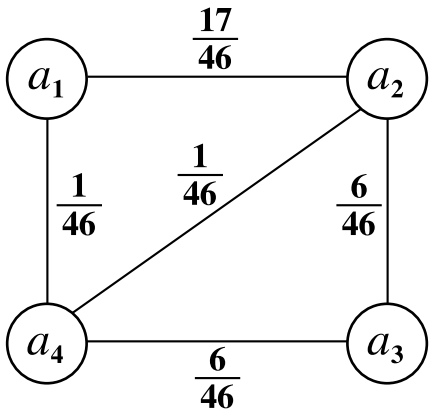
$$\begin{cases} P(a_1) = P(I_{21}) \cdot P(a_2) + P(I_{41}) \cdot P(a_4) \\ P(a_2) = P(I_{12}) \cdot P(a_1) + P(I_{42}) \cdot P(a_4) \\ P(a_3) = P(I_{23}) \cdot P(a_2) + P(I_{43}) \cdot P(a_4) \\ P(a_4) = P(I_{34}) \cdot P(a_3) \end{cases} \tag{6}$$

Jeśli założyć prawdopodobieństwo pojawienia się jedynki na każdym z wejść równe $\frac{1}{2}$, prawdopodobieństwa pojawienia się wektorów wejściowych będą równe: $P(I_{12}) = P(I_{21}) = P(I_{23}) = P(I_{34}) = P(I_{43}) = \frac{1}{2}$ oraz $P(I_{41}) = P(I_{42}) = \frac{1}{4}$. Podstawiając wartości do układu równań (6) oraz zastępując pierwsze równanie równaniem (3) otrzymamy:

$$\begin{cases} P(a_1) + P(a_2) + P(a_3) + P(a_4) = 1 \\ P(a_2) = \frac{1}{2} \cdot P(a_1) + \frac{1}{4} \cdot P(a_4) \\ P(a_3) = \frac{1}{2} \cdot P(a_2) + \frac{1}{2} \cdot P(a_4) \\ P(a_4) = \frac{1}{2} \cdot P(a_3) \end{cases} \tag{7}$$

Rozwiązaniem układu równań (7) są prawdopodobieństwa statyczne automatu: $P(a_1) = \frac{11}{23}$, $P(a_2) = \frac{6}{23}$, $P(a_3) = \frac{4}{23}$, $P(a_4) = \frac{2}{23}$.

Wyliczone wartości pozwalają obliczyć prawdopodobieństwo zmiany stanu. Podstawiając je do równania (4) otrzymujemy: $P(a_1 \to a_2) = \frac{11}{46}$, $P(a_2 \to a_1) = \frac{6}{46}$, $P(a_2 \to a_3) = \frac{6}{46}$, $P(a_3 \to a_4) = \frac{4}{46}$, $P(a_4 \to a_1) = \frac{1}{46}$, $P(a_4 \to a_2) = \frac{1}{46}$, $P(a_4 \to a_3) = \frac{2}{46}$. Wagi krawędzi grafu będą miały wartości: $w_{1,2} = \frac{17}{46}$, $w_{1,4} = \frac{1}{46}$, $w_{2,3} = \frac{6}{46}$, $w_{2,4} = \frac{1}{46}$, $w_{3,4} = \frac{6}{46}$. W związku z tym graf przejść przybierze postać przedstawioną na rys. 2.



Rysunek 2. Graf przejść automatu skończonego przekształcony do grafu nieskierowanego

3. Sformułowanie problemu

Implementując automat skończony w postaci układu sekwencyjnego, każdemu stanowi a_i należy przypisać kod c_i. W związku z tym zbiorowi stanów automatu A odpowiada zbiór C kodów stanów automatu skończonego, $C = \{c_1, c_2, \ldots, c_M\}$. Każdy kod musi być ortogonalny z wszystkimi pozostałym kodami. Liczba bitów kodu N może być wartością z zakresu $[\text{int}\log_2 M, M]$.

Niech c_i^l oznacza l-ty bit kodu stanu a_i. Odległość Hamminga $H(c_i, c_j)$, określona jest jako liczba bitów w kodach c_i i c_j, znajdujących się na tych samych pozycjach mających przeciwne wartości, może być opisana wzorem (8):

$$H(c_i, c_j) = \sum_{l=1}^{N} c_i^l \oplus c_j^l \tag{8}$$

Problem znalezienia sposobu kodowania, który doprowadzi do minimalizacji poboru mocy można przedstawić w postaci zadania całkowitoliczbowego programowania liniowego:

$$Min\left(\sum_{i,j=1}^{M} w_{i,j} \cdot H(c_i, c_j)\right) \tag{9}$$

przy ograniczeniach:

$$\sum_{l=1}^{N} c_i^l \oplus c_j^l \geqslant 1; \forall c_i, c_j; c_i \neq c_j \tag{10}$$

Dokładne rozwiązanie zadania opisanego równaniami (9) i (10) może być niewykonalne, szczególnie dla dużych automatów. Dlatego stosuje się rozwiązania heurystyczne.

4. Algorytmy kodowania stanów wewnętrznych

4.1 Algorytm kodowania kolumnowego

Algorytm kodowania kolumnowego bazuje na pojęciu klasy nierozróżnialności kodów stanów wewnętrznych [1]. Kody stanów mające identyczne kody częściowe należą do tej samej klasy nierozróżnialności (11):

$$c_i, c_j \in C_{k,l} \iff c_i^m = c_j^m, \forall 1 \leqslant m \leqslant l \tag{11}$$

Kodowanie zakończy się sukcesem tylko wtedy, gdy liczba kodów w każdej klasie jest ograniczona zależnością (12):

$$|C_{k,l}| \leqslant 2^{N-l} \tag{12}$$

Wykorzystując klasy nierozróżnialności można zakodować stany wewnętrzne „bit po bicie", bez konieczności wykonywania operacji na całym kodzie. W trakcie przypisywania wartości bitów do kolejnych kodów stanów automatu należy przestrzegać ograniczenia (12).

Poniżej przedstawiono algorytm działający na podstawie opisanych powyżej zależności, bazowany na algorytmie opisanym w [1].

1. Stwórz tablicę przejść, zawierającą pary stanów i wagi grafu odpowiadające parze stanów.
2. Punkty 3-7 wykonaj kolejno dla $l = 1 \ldots N$.
3. Posortuj tablicę przejść w porządku malejącym względem wagi.
4. Dla każdej pary stanów a_i oraz a_j z tablicy przejść wykonaj punkty 5-6.
5. Jeżeli bity kodów c_i^l oraz c_j^l nie zostały przypisane, wykonaj poniższe operacje.
   - Wybierz bit do przypisania w taki sposób, aby po przypisaniu suma wag stanów sąsiednich do a_i oraz a_j, których l-te bity kodów różnią się od l-tych bitów kodów stanów a_i oraz a_j, była jak najmniejsza.
   - Jeżeli przypisanie do obu stanów wybranego bitu nie spowoduje przekroczenia maksymalnych liczności klas, to przypisz tę samą wartość do c_i^l oraz c_j^l. W przeciwnym wypadku przypisz wartości przeciwne.
6. Jeżeli jeden z bitów kodów c_i^l lub c_j^l nie został przypisany, wykonaj poniższe operacje.

- Wybierz bit do przypisania, równy l-temu bitowi w przypisanym kodzie.
- Jeżeli przypisanie wybranego bitu nie spowoduje przekroczenia maksymalnych liczności klasy, to przypisz jego wartość do bitu, który nie został jeszcze przypisany (c_i^l lub c_j^l). W przeciwnym wypadku przypisz wartość przeciwną.

7. Dostosuj tablicę przejść korzystając z wzoru (13).

$$nowa(w_{i,j}) = stara(w_{i,j}) \cdot (H(c_i, c_j) + 1) \tag{13}$$

8. Koniec.

Przykład

Dla grafu z rys. 2 możemy stworzyć następującą tablicę przejść (dla ułatwienia każda waga została pomnożona przez 46, co nie wpływa na wynik kodowania): $\{a_1, a_2, 17\}$, $\{a_1, a_4, 1\}$, $\{a_2, a_3, 6\}$, $\{a_2, a_4, 1\}$, $\{a_3, a_4, 6\}$.

Najpierw kodujemy pierwszy bit kodu. Po posortowaniu tablica przybierze postać: $\{a_1, a_2, 17\}$, $\{a_3, a_4, 6\}$, $\{a_2, a_3, 6\}$, $\{a_1, a_4, 1\}$, $\{a_2, a_4, 1\}$. Bierzemy pierwszą parę stanów z tablicy: a_1 oraz a_2. Ponieważ oba stany nie mają przypisanego pierwszego bitu kodu, wybieramy taki sam bit do przypisania, np. 0 (żadne stany nie mają przypisanego żadnego bitu kodu, więc wartość może być dowolna). Przypisanie bitu do kodów stanów a_1 oraz a_2 nie spowoduje przekroczenia liczności klas (maksymalna liczność klasy na podstawie ograniczenia (12) wynosi 2).

Następnie wybieramy do przypisania parę a_3 oraz a_4. Podobnie jak w poprzednim przypadku, do obu stanów nie zostały przypisane kody, więc wybieramy taką samą wartość bitu do przypisania. Żeby suma wag kodów połączonych z kodami a_3 oraz a_4, których pierwsze bity mają różne wartości, była jak najmniejsza, należy wybrać wartość bitu równą 1. Przypisanie bitu do kodów stanów a_3 oraz a_4 nie spowoduje przekroczenia liczności klas.

Otrzymaliśmy następujące kody cząstkowe: $c_1 = 0$, $c_2 = 0$, $c_3 = 1$, $c_4 = 1$.

Dostosowując tablicę przejść zgodnie ze wzorem (13), otrzymamy nową tablicę: $\{a_1, a_2, 17\}$, $\{a_3, a_4, 6\}$, $\{a_2, a_3, 12\}$, $\{a_1, a_4, 2\}$, $\{a_2, a_4, 2\}$.

Kodujemy drugi bit kodu. Po posortowaniu tablica przybierze postać: $\{a_1, a_2, 17\}$, $\{a_2, a_3, 12\}$, $\{a_3, a_4, 6\}$, $\{a_1, a_4, 2\}$, $\{a_2, a_4, 2\}$. Bierzemy pierwszą parę stanów z tablicy: a_1 oraz a_2. Ponieważ oba stany nie mają przypisanego drugiego bitu kodu, wybieramy taki sam bit do przypisania, np. 0 (żadne stany nie mają przypisanego drugiego bitu kodu, więc wartość może być dowolna). Jednakże przypisanie bitu do kodów stanów a_1 oraz a_2 spowoduje przekroczenie liczności klas (maksymalna liczność klasy wynosi 1). W związku z tym do drugiego bitu kodu stanu a_1 przypisujemy 0, a do kodu stanu a_2 przypisujemy 1.

Następnie wybieramy parę a_2 oraz a_3. Kod stanu a_2 ma już przypisany drugi bit, więc należy przypisać bit do kodu stanu a_3. Do przypisania wybieramy wartość 1 (taką samą, jak drugi bit w kodzie stanu a_2). Takie przypisanie nie spowoduje przekroczenia liczności klas. Na koniec wybieramy parę kodów a_3 oraz a_4. Podobnie jak w poprzednim przypadku, kod stanu a_3 ma już przypisaną wartość. Jednakże wybranie takiego samego bitu (1) spowoduje przekroczenie liczności klasy, więc do przypisania wybieramy wartość 0.

Po zakończeniu kodowania otrzymamy następujące kody stanów: $c_1 = 00$, $c_2 = 01$, $c_3 = 11$, $c_4 = 10$.

4.2 Algorytm sekwencyjny

W algorytmie sekwencyjnym kodowanie zależy od przypisanych uprzednio kodów stanów wewnętrznych automatu. Algorytm wykorzystuje funkcję γ określającą sumę mocy wydzielonej przez automat przy przejściu ze stanu a_i do dowolnego stanu (14):

$$\gamma(c_i) = \sum_{j=1}^{M} w_{i,j} \cdot H(c_i, c_j) \tag{14}$$

Funkcja pozwala wyznaczyć kod przypisywany do stanu, aby wydzielana moc była jak najmniejsza [7].

Niech K^N oznacza zbiór wszystkich wartości kodów stanów o długości N bitów. Liczność zbioru K^N wynosi 2^N. Wartość N należy do przedziału $[\text{int}\log_2 M, M]$.

Poniżej przedstawiono algorytm sekwencyjny kodowania stanów automatu skończonego.

1. $C = \{\emptyset\}$, $K^N = \{k_1, \ldots, k_{2^N}\}$
2. Wybierz dwa stany a_i i a_j, dla których $w_{i,j}$ jest największe.
3. Dla stanów a_i oraz a_j przypisz dowolne kody c_i oraz c_j ze zbioru K^N, dla których $H(c_i, c_j) = 1$. Usuń wartości przypisanych kodów ze zbioru K^N.
4. Powtarzaj kroki 5-6, aż wszystkie stany będą zakodowane.
5. Wybierz ze zbioru A stan a_i, któremu nie przypisano kodu i dla którego suma wag krawędzi grafu łączących stan a_i ze stanami, dla których kody są już przypisane, osiąga wartość maksymalną.
6. Wybierz ze zbioru kodów K^N wartość kodu c_i, dla którego funkcja γ ma wartość minimalną. Usuń wartość kodu ze zbioru K^N.
7. Koniec.

Przykład

Na początku zbiory $C = \{\emptyset\}$ oraz $K^2 = \{00, 01, 10, 11\}$.

Wartość $w_{i,j}$ jest największa dla pary stanów a_1 oraz a_2. Dlatego przypisujemy do nich kody, np. $c_1 = 00$, $c_2 = 01$ (odległość Hamminga dla tej pary kodów wynosi 1). Kody te usuwamy ze zbioru K^2. W związku z tym $C = \{c_1, c_2\}$ oraz $K^2 = \{10, 11\}$

Następnie wybieramy kolejny stan do zakodowania. Dla stanu a_3 suma wag krawędzi grafu łączących stan a_3 ze stanami, dla których kody są już przypisane (czyli a_1 oraz a_2), wynosi $\frac{12}{46}$, natomiast dla stanu a_4 wynosi $\frac{8}{46}$. Do kodowania wybieramy stan a_3.

Ze zbioru K^2 wybieramy kod o najmniejszej wartości funkcji γ. W naszym przypadku będzie to kod 11. Przypisujemy go do kodu stanu a_3 i usuwamy ze zbioru K2. W związku z tym $C = \{c_1, c_2, c_3\}$ oraz $K^2 = \{10\}$

Na koniec do stanu a_4 zostanie przypisany kod 10.

Po zakończeniu kodowania otrzymamy następujące kody stanów: $c_1 = 00$, $c_2 = 01$, $c_3 = 11$, $c_4 = 10$.

4.3 Algorytm rafinacyjny

Algorytm rafinacyjny pozwala „poprawić" wyniki kodowania przeprowadzonego za pomocą innego algorytmu. Jego działanie polega na zamianie kodów stanów w taki sposób, aby zmniejszyć wartość mocy.

Poniżej przedstawiono pseudokod algorytmu rafinacyjnego kodowania stanów wewnętrznych automatu skończonego.

```
bez_zmiany := 0;

REPEAT

  zmiana := FALSE;
  FOR i := 1 TO M
    stan_a := ai;
    FOR kod := 0 TO 2^N - 1
      stan_b := Znajdz_Stan(kod);
      moc_przed := Oblicz_Moc();
      IF stan_b
        Zamien_Kody(stan_a, stan_b);
        IF moc_przed < Oblicz_Moc()
          Zamien_Kody(stan_a, stan_b);
        ELSE
          zmiana := TRUE;
```

```
        ENDIF
      ELSE
        stary_kod :=  stan_a.kod;
        stan_a.kod := kod;
        IF moc_przed < Oblicz_Moc()
          stan_a.kod := stary_kod;
        ELSE
          zmiana := TRUE;
        ENDIF
      ENDIF
    NEXT
  NEXT

  IF zmiana
    bez_zmiany := 0;
  ELSE
    bez_zmiany := bez_zmiany + 1;
  ENDIF

UNTIL bez_zmiany > epsilon;
```

Zastosowane funkcje: `Znajdz_Stan` – wyszukuje stan, któremu przypisano podany kod; `Oblicz_Moc` – zwraca wartość mocy wydzielonej przez układ przy bieżącym kodowaniu; `Zamien_Kody` – zamienia kody stanów.

Wartość `epsilon` jest ustalana przed wykonaniem algorytmu i określa maksymalną ilość przebiegów bez zmiany kodowania.

5. Badania eksperymentalne

Badania przeprowadzono wykorzystując standardowe testy (benchmark) [9]. Obliczenia zostały wykonane przy założeniu następujących wartości: C = 3pF, f = 5MHz, V_{DD} = 5V oraz $P(x_i = 1)$ = 0,5.

Wyniki zebrane w tab. 1. przedstawiają wyniki badań porównawczych nad algorytmami kodowania kolumnowego oraz sekwencyjnym. Dodatkowo obliczono wartość mocy dla kodowania binarnego (gdzie kod odpowiadał numerowi stanu) oraz „one-hot" (gdzie pozycja jedynki w kodzie odpowiadała numerowi stanu). Kolumna „Benchmark" zawiera nazwę układu testowego, w kolumnie „Stany" umieszczono liczbę stanów układu testowego, kolumna „Kod" określa liczbę bitów kodu, który został wykorzystany w kodowaniu, poza kodowaniem „one-hot", gdzie długość kodu jest równa liczbie stanów automatu. Następne kolumny zawierają wyniki obliczeń dla

czterech sposobów kodowania stanów. Dla każdego sposobu kodowania podano wartości obliczonej mocy („P_X" w mW, gdzie indeks „X" oznacza metodę kodowania: B – binarne, O – one-hot, K – kolumnowe, S – sekwencyjne). Na koniec w wierszu oznaczonym „Średnia" obliczono średnią wartość mocy.

Tablica 1. Wyniki badań eksperymentalnych algorytmów kodowania stanów wewnętrznych

Benchmark	**Stany**	**Kod**	$\mathbf{P_B}$	$\mathbf{P_O}$	$\mathbf{P_K}$	$\mathbf{P_S}$
bbara	10	4	62,14	83,48	56,26	52,77
bbtas	6	3	134,51	166,3	83,15	83,15
beecount	7	3	113,28	169,56	108,92	89,42
dk14	7	3	239,67	308,59	207,28	223,65
dk16	27	5	401,14	360,8	377,53	309,09
dk27	7	3	290,18	375	223,21	223,21
dk512	15	4	298,55	375	319,75	238,84
donfile	24	5	324,22	281,25	265,63	222,66
ex1	20	5	259,04	238,56	157,7	138,55
ex5	9	4	231,89	246,79	176,4	159,29
modulo12	12	4	171,88	187,5	93,75	93,75
opus	10	4	150,2	243,89	133,38	133,38
pma	24	5	252,5	199,3	105,55	104,76
s1	20	5	329,3	274,19	250,74	200,98
s1a	20	5	329,3	274,19	250,74	200,98
s27	6	3	192,75	255,25	168,33	168,33
s8	5	3	62,27	65,65	33,9	33,9
train11	11	4	101,9	124,32	86,96	63,52
Średnia			219,15	234,98	172,18	152,24

Z zestawienia przedstawionego w tab. 1 wynika, że najlepsze wyniki uzyskano po zastosowaniu algorytmu rafinacyjnego (średnia wartość mocy: 152,24), natomiast algorytm kodowania kolumnowego dał średnie wyniki gorsze o prawie 20 mW (średnia wartość mocy: 172,18). Najlepsze wyniki uzyskano przy zastosowaniu kodowania sekwencyjnego dla 17 z 18 testów. W 7 z 18 przypadków najlepsze wyniki osiągnięto przy zastosowaniu kodowania kolumnowego. Najgorsze wyniki dało zastosowanie kodowania „one-hot". Jeśli porównać dwa najlepsze algorytmy (kodowanie kolumnowe i sekwencyjne) okazuje się, że kodowanie sekwencyjne jest lepsze w 11 przypadkach, w 6 daje takie same wyniki, natomiast w 1 gorsze (dk14) w porównaniu do kodowania kolumnowego. Stosunek średniej wartości mocy algorytmu sekwencyjnego do algorytmu kodowania kolumnowego wyniósł 0,88.

W tab. 2 przedstawiono wyniki działania algorytmu rafinacyjnego. Algorytm miał za zadanie poprawić wyniki kodowania: binarnego, kolumnowego oraz sekwencyjnego. Kolumna „Benchmark" zawiera nazwę układu testowego, w kolumnach „Binarne", „Kolumnowe" oraz „Sekwencyjne" zebrano wyniki działania algorytmu rafinacyjnego przy zadaniu początkowego kodowania z użyciem algorytmu odpowiednio: binarnego, kodowania kolumnowego oraz sekwencyjnego. Kolumny: „P_{przed}" zawierają wartość mocy przed wykonaniem algorytmu rafinacyjnego, „P_{po}" zawiera wartość po wykonaniu algorytmu rafinacyjnego, natomiast „P_{po}/P_{przed}" stosunek wartości mocy przed wykonaniem algorytmu rafinacyjnego do wartości po wykonaniu algorytmu rafinacyjnego.

Tablica 2. Wyniki badań eksperymentalnych algorytmu rafinacyjnego

Benchmark	**Binarne**			**Kolumnowe**			**Sekwencyjne**		
	P_{przed}	P_{po}	P_{po}/P_{przed}	P_{przed}	P_{po}	P_{po}/P_{przed}	P_{przed}	P_{po}	P_{po}/P_{przed}
bbara	62,14	55,34	0,89	56,26	53,49	0,95	52,77	52,77	1
bbtas	134,51	83,15	0,62	83,15	83,15	1	83,15	83,15	1
beecount	113,28	89,42	0,79	108,92	89,42	0,82	89,42	89,42	1
dk14	239,67	207,28	0,86	207,28	207,28	1	223,65	207,28	0,93
dk16	401,14	300,95	0,75	377,53	303,24	0,8	309,09	290,41	0,94
dk27	290,18	223,21	0,77	223,21	223,21	1	223,21	223,21	1
dk512	298,55	223,21	0,75	319,75	215,40	0,67	238,84	236,61	0,99
donfile	324,22	226,56	0,7	265,63	214,84	0,81	222,66	207,03	0,93
ex1	259,04	136,43	0,53	157,70	133,29	0,85	138,55	138,55	1
ex5	231,89	163,61	0,71	176,40	159,29	0,9	159,29	159,29	1
modulo12	171,88	93,75	0,55	93,75	93,75	1	93,75	93,75	1
opus	150,2	133,32	0,89	133,38	133,32	1	133,38	133,32	1
pma	252,50	104,76	0,41	105,55	105,18	1	104,76	104,28	1
s1	329,30	204,28	0,62	250,74	208,45	0,83	200,98	198,65	0,99
s1a	329,30	204,28	0,62	250,74	208,45	0,83	200,98	198,65	0,99
s27	192,75	166,23	0,86	168,33	166,23	0,99	168,33	166,23	0,99
s8	62,27	33,9	0,54	33,9	33,9	1	33,9	33,9	1
train11	101,9	63,52	0,62	86,96	63,52	0,73	63,52	63,52	1
Średnia	219,15	150,73	0,69	172,18	149,75	0,9	152,24	148,89	0,99

Z tab. 2 wynika, że algorytm rafinacyjny w większości przypadków (40 z 54) poprawił sposób kodowania uzyskany za pomocą pozostałych algorytmów. Największa poprawa wystąpiła dla kodowania binarnego (0,69 mocy przed wykonaniem algorytmu). Najmniejszą poprawę uzyskano dla kodowania sekwencyjnego (0,99). Jed-

nakże w przypadku każdej metody stosowanie algorytmu rafinacyjnego pozwoliło na poprawę wyników.

Zastosowanie algorytmu rafinacyjnego dało najlepsze wyniki średnie dla kodowania sekwencyjnego (148,89), jednakże niewiele gorsze wyniki uzyskano dla kodowania kolumnowego (149,75). Najgorsze wyniki otrzymano dla kodowania binarnego (150,73), aczkolwiek różnica pomiędzy najlepszym a najgorszym wynikiem (czyli pomiędzy kodowaniem sekwencyjnym a binarnym) nie przekroczyła 2%.

6. Podsumowanie i wnioski

Przeprowadzone badania pokazały znaczne różnice w ilości mocy pobieranej przez układ sekwencyjny, dla którego kodowanie przeprowadzono algorytmem kodowania kolumnowego i algorytmem sekwencyjnym. Wykazano również duży wpływ algorytmu rafinacyjnego na wyniki uzyskane za pomocą analizowanych algorytmów.

Zastosowanie algorytmu sekwencyjnego dało średnio 1,12 razy mniejsze wartości mocy w porównaniu z zastosowaniem algorytmu kodowania kolumnowego. Dodatkowo, przy zastosowaniu algorytmu rafinacyjnego wartość mocy została zmniejszona średnio o 1%.

Działanie algorytmu rafinacyjnego było szczególnie widoczne w przypadku ustawienia jako początkowego kodowania binarnego. W takim przypadku moc zmniejszyła się po zastosowaniu algorytmu średnio do 0,68 wartości przed wynkonaniem algorytmu. Jednakże rezultat różni się niewiele (o ok. 2%) – w zależności od początkowego kodowania.

Najniższą moc średnią osiągnął algorytm rafinacyjny (148,89 mW), który może stanowić alternatywę dla pozostałych algorytmów. Dodatkowe zmniejszenie mocy może zostać osiągnięte m.in. poprzez zwiększenie liczby bitów kodu z zakresu $[\text{int}\log_2 M, M]$.

Literatura

[1] Benini L., DeMicheli G.: State Assignment for Low Power Dissipation, IEEE Journal on Solid-state Circuits, Vol. 30, No. 3 (1995), pp. 259-268.

[2] Freitas A. T., Oliveira A. L.: Implicit Resolution of the Chapman-Kolmogorov Equations for Sequential Circuits: An Application in Power Estimation, Proceedings of Design, Automation and Test in Europe Conference and Exhibition (DATE) 2003, pp. 10764-10769.

[3] Grześ T., Salauyou V.: Metody obliczania mocy w układach cyfrowych, „Pomiary, Automatyka, Kontrola" nr 7bis (2006), str. 101-102.

[4] Koegst M., Franke G., Feske K.: State Assignment for FSM Low Power Design, Proceedings of the Conference on European Design Automation, Geneva 2003, pp. 28-33.

[5] Pedram M.: Power simulation and estimation in VLSI circuits, "The VLSI Handbook", Edited by W-K. Chen, The CRC Press and the IEEE Press, 1999.

[6] Roy K., Prasad S. C.: Circuit Activity Based Logic Synthesis for Low Power Reliable Operations, IEEE Transactions on VLSI Systems, Vol. 1, No. 4 (1993), pp. 503-513.

[7] Salauyou V., Grzes T.: FSM state assignment methods for low-power design, Proceedings of 6th International Conference on Computer Information Systems and Industrial Management Applications (CISIM'2007), IEEE Computer Society, pp. 345-348.

[8] Tsui C.-Y., Monteiro J., Pedram M., Devadas S., Despain A. M., Lin B.: Power Estimation Methods for Sequential Logic Circuits, IEEE Transactions on VLSI Systems, Vol. 3, No. 3 (1995), pp. 404-416.

[9] Yang S.: Logic Synthesis and Optimization Benchmarks User Guide: Version 3.0, Technical Report, Microelectronics Center of North Carolina, 1991, 43 p.

FINITE STATE MACHINES STATE ASSIGNMENT ALGORITHMS FOR POWER MINIMIZATION

Abstract: State assignment for a finite state machine (FSM) is an important process in logic synthesis of the sequential circuits in programmable devices. Using the proper algorithm provides among other things the reduction of the power dissipation. In this paper we focused on the algorithms that reduce power dissipation. The analysis of the column based algorithm (described in [1]) as well as two algorithms proposed by authors: sequential [7] and iterational was made. Experiments were made on standard benchmarks, researched in Microelectronics Center of North Carolina [9]. Obtained results showed significant reduction of the power dissipation when using the sequential algorithm (12% in comparison with the column-based algorithm). Iterational algorithm improves the results by additional 1%.

Keywords: finite state machine, state assignment, low-power design