Zeszyty Naukowe Wydziału Elektrotechniki i Automatyki Politechniki Gdańskiej Nr 22

XVI Seminarium
ZASTOSOWANIE KOMPUTERÓW W NAUCE I TECHNICE' 2006
Oddział Gdański PTETiS
Referat nr 24

PORÓWNANIE WYBRANYCH ALGORYTMÓW SZEREGOWANIA ZADAŃ DLA POTRZEB REDUKCJI POBORU MOCY CYFROWYCH UKŁADÓW CMOS

Władysław SZCZEŚNIAK [1] Łukasz WŁODARCZYK [2], Piotr SZCZEŚNIAK[3] Maciej PIECHÓWKA[1]

1. Politechnika Gdańska, ul. G. Narutowicza 11/12, 80-952 Gdańsk
   tel: (058) 347 21 78 fax: (058) 341 61 32 e-mail: {wlad, macpi}@eti.pg.gda.pl
2. Lufthansa Systems Poland sp. z o.o., ul. Długie Ogrody 8, 80-765 Gdańsk
   tel. (058) 3265 546 fax: (058) 3265 599 e-mail: lukasz.wlodarczyk@lhsystems.pl
3. Ośrodek Badawczo Rozwojowy Centrum Techniki Morskiej, ul. Dickmana 62
   81-109 Gdynia
   tel: (058) 666 53 45 fax: (058) 666 53 04 e-mail: piotr.szczesniak@ctm.gdynia.pl

Szeregowanie zadań stosowane dla potrzeb redukcji poboru mocy cyfrowych układów CMOS prowadzi do problemów NP trudnych. Stąd też brakuje analitycznych algorytmów gwarantujących uzyskanie optymalnego rozwiązania w akceptowalnym czasie.
Praca prezentuje porównanie jakości rozwiązań wyznaczonych heurystycznymi algorytmami szeregowania zadań stosowanych na etapie syntezy wysokiego poziomu cyfrowych układów CMOS, które zostały uzyskane dla wybranego zbioru przykładów testowych. Przedstawione porównanie dotyczy również czasów obliczeń dla wybranych algorytmów szeregowania zadań.

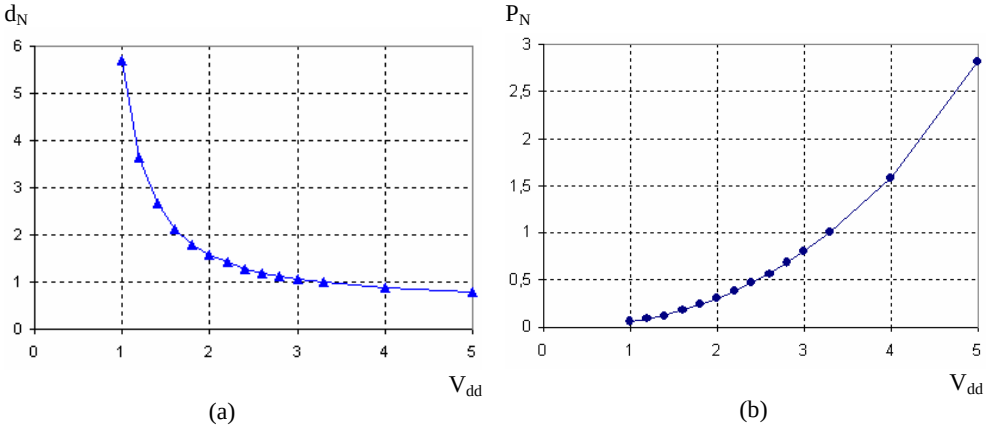
1. WPROWADZENIE

Poziom poboru mocy systemów VLSI jest bardzo ważnym kryterium projektowym. Jest on szczególnie ważny w urządzeniach przenośnych, zasilanych z baterii.
Potrzeba uwzględnienia powyższego kryterium w procesie projektowania układów scalonych stawia przed projektantem nowe zadania.
Niniejsza praca jest próbą porównania wyników uzyskanych przy pomocy różnych algorytmów szeregowani zadań zastosowanych do redukcji mocy pobieranej ze źródła zasilającego przez cyfrowe systemy VLSI.

Jedną z metod redukcji poboru mocy cyfrowych układów VLSI jest obniżenie napięcia zasilającego wybranych jednostek funkcjonalnych (f_u) bez pogarszania wydajności/przepustowości całego systemu. Zależności opóźnienia jednostki funkcjonalnej od na-

Recenzent: Dr inż. Walerian Gruszczyński: - Wydział Elektroniki, Telekomunikacji i Informatyki Politechniki Gdańskiej

pięcia zasilającego oraz wynikająca z tego redukcja poboru mocy zostały przedstawione na rys. 1.



Rys. 1. Przykładowa zależności unormowanego opóźnienia d_N (a) i unormowanej mocy pobieranej ze źródła P_N (b) jednostki funkcjonalnej od napięcia zasilającego V_{dd}.

2. ZDEFINIOWANIE PROBLEMU

Celem niniejszej pracy jest porównanie uzyskanych rozwiązań redukcji poboru mocy przy pomocy czterech algorytmów szeregowania zadań zaimplementowanych w systemie SAPR (System of Algorithmic Power Reduction). Definicję przyjętego modelu szeregowania zadań i szczegółowy opis formalny problemu rozwiązywanego algorytmami prezentowanymi w niniejszej pracy przedstawiono [1].

Wykorzystano tutaj takie przyporządkowanie funktorów do jednostek funkcjonalnych, aby możliwe było spowolnienie niektórych z nich bez pogorszenia wydajności całego układu. System ów zakłada wykorzystywanie jednostek spowolnionych pracujących 2, 4, 8... razy wolniej.

W celu opisania danych wejściowych stosuje się graf przepływu danych (ang. *data flow graph – DFG*) . Graf *DFG (V,E)* jest grafem skierowanym, w którym zbiór wierzchołków $V=\{v_i\}$ reprezentuje funktory natomiast zbiór krawędzi $E=\{e_{lm}\}$ reprezentuje wartości (zmienne i stałe), przekazywane pomiędzy funktorami.

Dla zadanego grafu DFG, w wyniku syntezy wysokiego poziomu (HLS), uzyskuje się harmonogram uszeregowania zadań *SG*, w którym każdy wierzchołek *DFG* jest przyporządkowany do odpowiedniej jednostki funkcjonalnej i cyklu zegarowego w taki sposób, że wszystkie zależności od wyników wcześniejszych operacji są zachowane. Każda kolumna *SG* reprezentuje jedną jednostkę funkcjonalną, a każdy wiersz - kolejny takt zegara. Zależności pomiędzy operacjami powodują, że nie wszystkie jednostki funkcjonalne mają przypisany wierzchołek grafu *DFG* dla wszystkich taktów zegara, przez co ich wykorzystanie nie jest pełne.

Jednostki funkcjonalne, które posiadają takty zegarowe do których nie jest przyporządkowany żaden funktor mogą być spowolnione. Spowolnienie jednostki funkcjonalnej powoduje zwiększenie liczby taktów zegara wymaganych na wykonanie operacji związanej z wierzchołkiem grafu *DFG*. Pozwala to na zmniejszenie napięcia zasilania spowolnionej jednostki funkcjonalnej. Przy spowalnianiu jednostek funkcjonalnych można założyć dopuszczalne zwiększenie długości harmonogramu. Wynik algorytmu jest tym lepszy, im

więcej jednostek funkcjonalnych zostało w nim spowolnionych, przy jak najmniejszym zwiększeniu liczby taktów, potrzebnych na wykonanie operacji zadanych grafem *DFG*.

Poniżej zaprezentowano skrótowe omówienie pięciu algorytmów szeregowania zadań dla potrzeb redukcji mocy w układach CMOS.

3. OMÓWIENIE WYBRANYCH ALGORYTMÓW

3.1. Algorytm BF

Algorytm BF (ang. *Brute Force*) [5] charakteryzuje się tym, iż w każdym kroku spowalnia wybraną jednostkę funkcjonalną i układa harmonogram na nowo przy pomocy algorytmu ASAP (ang. *As Soon As Possibile*) ze zmodyfikowanym zbiorem zasobów. Zmiana jest cofana, jeśli harmonogram nadmiernie się wydłużył, lub odnotowano brak zmniejszenia mocy pobieranej przez układ. W takim wypadku jednostka, którą próbowano spowolnić, oznaczana jest jako niepodatna na dalsze spowolnienia.

Wybór jednostki funkcjonalnej dokonywany jest za pomocą metody zachłannej. Spowalniana jest jednostka, dla której poziom redukcji rozpraszanej mocy uzyskany ze spowolnienia jest największy. Sprawdzenie to odbywa się w czasie $O(m)$.

3.2. Algorytm IIOI

Przedstawiony w pracy [4] algorytm IIOI (ang. *Insert Idle Operations with Interchanging*) pracuje na gotowym harmonogramie uzyskanym przez algorytm ASAP. W każdym kroku na podstawie analizy harmonogramu wybiera on jednostkę funkcjonalną, którą następnie próbuje spowolnić.

W działaniu algorytmu wyróżniamy dwie fazy. Pierwszą jest faza wstępna, w której za pomocą szybkiej analizy harmonogramu odrzucana jest część jednostek funkcjonalnych, która nie może być spowolniona. Sprawdzenie to dokonywane jest za pomocą testu liczby pustych cykli w harmonogramie dla danej jednostki funkcjonalnej. Jeśli jest ich zbyt mało by pomieścić narzut czasowy wynikający ze spowolnienia wszystkich uszeregowanych na niej zadań, algorytm nie będzie starał się takiej jednostki spowolnić.

Druga faza polega na wybraniu jednostki funkcjonalnej najlepiej nadającej się do spowolnienia. Wybór dokonywany jest spośród tych jednostek, które nie zostały odrzucone w pierwszej fazie. Za pomocą opisanej poniżej wagi algorytm odnajduje zadania najlepiej podatne na spowolnienie i stara się umieścić je na jednej jednostce funkcjonalnej, a dokonuje tego poprzez zamianę zadań między sobą w harmonogramie. Za pomocą kolejnej wagi wybiera najlepiej nadającą się do spowolnienia jednostkę funkcjonalną. Spowolnieniu ulegają wszystkie uszeregowane na niej zadania. Ma to wpływ na zmianę harmonogramu nie tylko tej jednostki, ale także na poddrzewa grafu przepływu danych rozpoczynające się w spowolnionych zadaniach. Wszystkie dotknięte zmianą zadania muszą być ponownie uszeregowane, tak aby zachować zdefiniowaną relację kolejności.

Długość nowo uzyskanego harmonogramu nie może przekroczyć długości oryginalnego, referencyjnego harmonogramu wyznaczonego dla jednostek niespowolnionych. Jeśli warunek ten zostanie naruszony, wszystkie zmiany muszą być cofnięte, a wybrana jednostka jest ignorowana przy kolejnych próbach spowolnienia.

Waga, która określa podatność zadania do spowolnienia, zdefiniowana jest w sposób następujący (dla zadania v_i):

$$f_{vi} = \frac{i_{vi} + o_{vi} + (f_{avi} - s_{vi} n_i)}{p_i + 1}, \tag{1}$$

gdzie:

- i_{vi} - liczba różnych wejść systemowych które są komponentami tego zadania,
- o_{vi} - liczba różnych wyjść systemowych, które są produktami tego zadania,
- f_{avi} - liczba cykli zegara począwszy od zakończenia zadania v_i do końca harmonogramu,
- s_{vi} - liczba operacji tego samego typu jak zadanie v_i w poddrzewie rozpoczynającym się w zadaniu v_i,
- n_i - liczba cykli zegara potrzebna by wykonać zadanie v_i,
- p_i - etykieta priorytetu algorytmu zadania v_i.

Niektóre z tych wartości nie zmieniają się w trakcie działania algorytmu (jak i_{vi}, o_{vi}, s_{vi}, p_i), a w omawianej implementacji wyznaczane są na początku działania algorytmu dla wszystkich zadań i przechowywane w pamięci. Dodatkowo, raz wyznaczona wartość f_{vi} dla zadania jest przechowywana tak długo, aż harmonogram nie zmieni się na tyle, aby trzeba było przeliczyć ją ponownie (zmiana wartości n_i lub f_{avi}). Znacznie przyspiesza to działanie algorytmu dla dużych, gęstych grafów.

Waga, według której wybierana jest jednostka funkcjonalna do spowolnienia wyznaczana jest na podstawie formuły:

$$f_{Ck} = \frac{i_{Ck} + o_{Ck} + s_{Ck} + P^n_{dCk} l_{Ck}}{p_{Ck} + l_{Ck} + n_{Ck}}, \tag{2}$$

gdzie:

- C_k - k-ta jednostka funkcjonalna w zbiorze zasobów.
- i_{Ck} - liczba różnych wejść systemowych, które są komponentami zadań uszeregowanych na jednostce C_k,
- o_{Ck} - liczba różnych wyjść systemowych, które są produktami zadań uszeregowanych na jednostce C_k,
- s_{Ck} - suma numerów cykli zegara, w których rozpoczynają się zadania uszeregowane na jednostce C_k,
- P^n_{dCk} - znormalizowana wartość mocy pobieranej przez jednostkę C_k,
- l_{Ck} - liczba zadań uszeregowanych na jednostce C_k,
- p_{Ck} - suma etykiet priorytetów zadań uszeregowanych na jednostce C_k,
- n_{Ck} - liczba cykli zegara potrzebną by wykonać zadanie na jednostce C_k.

Po zakończeniu obu faz algorytm przechodzi do kolejnej iteracji i powtarza obie omówione fazy. Jednostki funkcjonalne oznaczone jako ignorowane nie są brane pod uwagę w kolejnych przebiegach. Algorytm kończy działanie, gdy zabraknie jednostek funkcjonalnych podatnych na spowolnienie.

3.3. Algorytm ewolucyjny z populacją EL2k3

Implementacja operatorów krzyżowania i mutacji dla algorytmu EL2k3 wykorzystuje opis zaprezentowany w pracy [2]. Osobnikiem populacji jest zbiór etykiet spowolnień jednostek funkcjonalnych oraz etykiet spowolnień zadań.

W fazie inicjalizacji wartości spowolnień osobników są generowane losowo. Aby zapewnić istnienie w populacji osobnika, który może wygenerować harmonogram o długości harmonogramu referencyjnego, jeden z osobników ma ustawione wartości spowolnień na minimalne.

Etapem selekcji w *n*-elementowej populacji jest zmodyfikowana selekcja turniejowa. Dokonywanych jest *2n* wyborów losowych par z bieżącej populacji. Z każdej pary osobników wyłaniany jest zwycięzca o mniejszej wartości funkcji przystosowania opisanej przez:

$$f = x \cdot cs_i + (1-x)\, p_i , \tag{3}$$

gdzie:

- cs_i - długość harmonogramu utworzonego przez *i*-tego osobnika,
- p_i - wartość mocy pobieranej przez jednostki funkcjonalne w harmonogramie utworzonym przez *i*-tego osobnika,
- x - współczynnik określany jako *trade-off parametr* (ang. parametr wyważenia).

Współczynnik *x* określa, jak ważne jest w danym obiegu pętli to, żeby tworzony przez najlepszych osobników harmonogram nie był wydłużony w stosunku do harmonogramu referencyjnego uzyskanego w fazie inicjalizacji. Współczynnik *x* wyznaczany jest dla każdego cyklu ewolucyjnego jako:

$$x = \frac{z_k}{z_k + 1}, \tag{4}$$

Rekurencyjnie obliczany współczynnik z_k jest zależny od tego, czy w cyklu *k* osobnik o najmniejszym poborze mocy generował harmonogram o poprawnej długości czy nie. Formuła określająca kolejne wartości współczynnika z_k jest następująca:

$$z_{k+1} = \begin{cases} z_k \cdot 1.5 \Leftrightarrow cs_{\min} \le cs_{ref} \\ z_k / 1.5 \Leftrightarrow cs_{\min} > cs_{ref} \end{cases}, z_0 = 1, \tag{5}$$

gdzie:

- $cs_{\min}$ - długość harmonogramu wygenerowanego z osobnika o najmniejszym poborze mocy,
- cs_{ref} - długość referencyjnego harmonogramu.

Mały współczynnik *x* spowoduje wydłużanie harmonogramów generowanych przez niektórych osobników, co sprzyja poszukiwaniu nowych rozwiązań i wychodzeniu z minimów lokalnych. Duży współczynnik *x* spowoduje, że rozwiązania generujące zbyt długie harmonogramy będą odrzucone.

Aby zapewnić przetrwanie najlepszego osobnika w populacji, jest on bezwarunkowo przenoszony nie tylko do kolejnej fazy, ale i do kolejnego cyklu ewolucyjnego. Najlepszy osobnik to taki, który generuje harmonogram o najmniejszym poborze mocy, lecz nieprzekraczający długości harmonogramu referencyjnego.

W fazie krzyżowania wybierane są losowe pary z populacji uzyskanej po etapie selekcji i z każdej z nich konstruowany jest nowy osobnik dziedziczący cechy rodziców. Sami rodzice są po tej operacji odrzucani. Z *2n* różnych par rodziców tworzonych jest *n* potomków. Losowy z nich zostaje odrzucony, by ustąpić miejsce najlepszemu osobnikowi poprzedniej iteracji.

Nowy osobnik przejmuje cechy każdego z rodziców z równym prawdopodobieństwem. Dla każdej cechy (każdego spowolnienia jednostki funkcjonalnej i zadania) wykonywana jest symetryczna próba wyłaniająca rodzica, od którego potomek ją odziedziczy.

Faza mutacji polega na przyspieszaniu lub spowalnianiu losowo wybranych jednostek i zadań. Dla każdej cechy wykonywany jest rzut, który decyduje, czy będzie ona mutowana. Jeśli tak, z równym prawdopodobieństwem zmniejszany lub zwiększany jest współ-

czynnik spowolnienia. Prawdopodobieństwo mutacji spowolnienia jednostki funkcjonalnej wynosi 0.05, natomiast zadania – 0.015. Obie te wielkości są konfigurowalnymi parametrami populacji i można zmienić je przed uruchomieniem algorytmu. Najlepszy osobnik nie bierze udziału w fazie mutacji.

Faza naprawy polega na dobraniu współczynników spowolnienia zadań osobnika tak, by na podstawie jego cech można było zbudować harmonogram. Jest wielce prawdopodobne, że w wyniku krzyżowania i mutacji niektóre z etykiet spowolnień zadań zostały ustawione na wartość, która nie ma swojego odpowiednika w etykietach spowolnień jednostek. W takim wypadku nie będzie można uszeregować tego zadania. Wartości etykiet spowolnień zadań są z równym prawdopodobieństwem zwiększane lub zmniejszane do najbliższej dostępnej wartości spowolnienia jednostki odpowiadającego typu.

3.4. Algorytm ewolucyjny z populacją naturalną

Populacja naturalna powstała w oparciu o populację opisaną powyżej. Modyfikacjami są etapy selekcji i krzyżowania – inicjalizacja, mutacja i naprawa zostały zaadaptowane bez zmian. Powodem stworzenia tej populacji była niewystarczająca wydajność algorytmów ewolucyjnych opartych o populację EL2k3 [5].

Etap selekcji został zastąpiony klasycznym modelem turniejowym – w każdej z $n/2$ dobranych losowo różnych par wyłaniany jest zwycięzca na podstawie następującej funkcji kryterialnej: jeśli oba osobniki z pary generują harmonogramy o tej samej długości, lub długości obu harmonogramów są mniejsze lub równe długości harmonogramu referencyjnego, wygrywa ten osobnik, którego jednostki funkcjonalne zużywają mniej mocy. Jeśli harmonogram tylko jednego z nich ma pożądaną długość, to niezależnie od poboru mocy jest on przenoszony do kolejnej fazy. Pomaga to przyspieszyć zbieżność populacji.

Etap krzyżowania polega na losowym sparowaniu pozostałych po fazie selekcji osobników i utworzeniu z każdej takiej pary dwóch potomków. Dla jednego z osobników wykonywany jest symetryczny rzut, aby ustalić, od którego z rodziców odziedziczona będzie dana cecha. Drugi z potomków dziedziczy tą cechę od drugiego rodzica. Operacja ta jest powtarzana dla wszystkich cech niesionych przez osobników, a w wyniku krzyżowania uzyskuje się populację o pierwotnym rozmiarze.

Sposób przeprowadzenia faz selekcji i krzyżowania automatycznie przenosi najlepszego osobnika do kolejnego cyklu – nie trzeba sztucznie go w nim umieszczać. Spodziewana jest większa wydajność algorytmów używających populacji naturalnej – w fazie selekcji budowanych jest cztery razy mniej nowych osobników niż w poprzednio opisanym algorytmie. Dodatkowo, opisana populacja nie wykonuje tylu operacji naprawy, co EL2k3. Po fazie krzyżowania procesowi naprawy podlega tylko jej nowo utworzona połowa. Spodziewany jest wzrost wydajności algorytmów korzystających z implementacji populacji naturalnej w stosunku do populacji EL2k3, choć przewidywana jest jednak strata na jakości rozwiązania. Populacja naturalna traci na elastyczności – proces selekcji ustawiony jedynie na harmonogramy o dozwolonej długości zawęża szanse na wychodzenie z minimów lokalnych.

4. WYNIKI EKSPERYMENTALNE [5]

Opis technologiczny zawiera możliwości spowolnień jednostek funkcjonalnych 2, 4 i 8 razy (identycznie jak w pracy [4]).

W tablicach wyników przyjęto oznaczenia:

P_r - procentowa wartość zredukowanej mocy w stosunku do niespo-

wolnionych jednostek funkcjonalnych,

t - czas obliczeń.

Tablice 1-6 przedstawiają uzyskane czasy oraz zyski w poborze mocy testowanych układów. Wyniki algorytmów, w których zastosowano metody losowe są średnią z trzech uruchomień. Dla algorytmów ewolucyjnych przyjęto stałą wielkość populacji równą 40. Przeprowadzono dodatkowe serie testów, w których zezwolono algorytmom na wydłużenie harmonogramu bazowego o 20 i 50%.

Wszystkie testy zostały wykonane dwukrotnie, dla dwóch różnych zestawów zasobów. Pierwszy zestaw (Tablice 1-3) został uzyskany przy pomocy standardowego algorytmu ASAP. Zestaw ten nie jest minimalnym zestawem gwarantującym uzyskaną długość harmonogramu. Drugi zestaw zasobów uzyskany został przy pomocy algorytmu MNP [3]. Algorytm ten redukuje liczbę jednostek funkcjonalnych potrzebnych do zrealizowania harmonogramu, bez jego wydłużenia. Jego zastosowanie w procesie HLS układów CMOS powoduje redukcję kosztów dzięki zmniejszeniu liczby wymaganych jednostek funkcjonalnych.

Tab. 1. Wyniki zbiorcze przeprowadzonych testów dla zasobów pozyskanych za pomocą klasycznego algorytmu ASAP oraz 0% wydłużenia harmonogramu.

Algorytm		BF		IIOI		NatPop		EL2k3	
Test	$\|V\|$	P_r [%]	t [s]	P_r [%]	t [s]	P_r [%]	t [s]	P_r [%]	t [s]
c432	160	22,0	2,5	25,8	1,3	24,9	19,2	**26,8**	23,3
c499	202	**3,5**	3,5	3,3	2,6	0,0	10,7	1,2	24,1
c880	357	45,4	4,3	3,8	2,2	54,0	72,8	**59,2**	75,2
c1355	514	**1,5**	5,0	1,2	8,9	0,4	25,4	0,6	37,7
c1908	718	54,5	5,4	3,2	5,4	58,8	411,6	**65,0**	440,4
c2670	997	35,2	9,2	6,7	8,3	56,0	455,2	**60,9**	649,6
c3540	1446	37,5	11,0	15,7	12,0	62,7	1997,6	**67,7**	2499,2
c5315	1994	52,1	34,0	8,7	22,5	74,4	5903,1	**78,7**	7171,6
c6288	2416	10,9	22,5	9,3	19,4	**28,3**	8926,6	26,7	3345,5
c7552	2978	54,7	73,4	15,7	44,7	65,7	11656,5	**68,1**	10884,8

Tab. 2. Wyniki zbiorcze przeprowadzonych testów dla zasobów pozyskanych za pomocą klasycznego algorytmu ASAP oraz 20% wydłużenia harmonogramu.

Algorytm		BF		IIOI		Natop		EL2k3	
test	$\|V\|$	P_r [%]	t [s]	P_r [%]	t [s]	P_r [%]	t [s]	P_r [%]	t [s]
c432	160	48,0	3,8	28,3	1,8	42,5	27,4	**49,4**	29,6
c499	202	20,0	4,3	13,2	3,0	28,9	35,9	**33,1**	38,0
c880	357	56,6	4,4	12,0	2,5	64,4	75,7	**73,1**	117,5
c1355	514	13,2	7,8	7,8	9,1	**38,0**	225,5	36,3	136,9
c1908	718	70,5	5,3	4,0	5,2	69,1	528,3	**74,8**	577,2
c2670	997	70,1	9,0	19,1	9,8	68,4	620,1	**74,2**	802,1
c3540	1446	46,0	11,8	23,3	16,0	73,0	2418,0	**75,9**	3318,4
c5315	1994	69,2	35,3	42,9	32,5	82,4	9374,4	**83,0**	6999,0
c6288	2416	10,4	28,0	13,1	22,9	**47,8**	9761,4	41,7	2636,8
c7552	2978	68,6	73,7	34,5	50,3	**74,8**	12370,9	34,9	2210,1

Tab. 3. Wyniki zbiorcze przeprowadzonych testów dla zasobów pozyskanych za pomocą klasycznego algorytmu ASAP oraz 50% wydłużenia harmonogramu.

Algorytm		BF		IIOI		NatPop		EL2k3	
test	$\|V\|$	P_r [%]	t [s]	P_r [%]	t [s]	P_r [%]	t [s]	P_r [%]	t [s]
c432	160	**75,5**	3,5	63,8	1,8	64,9	34,0	74,7	30,9
c499	202	48,7	3,7	45,2	4,3	54,3	49,8	**60,4**	54,5
c880	357	67,0	4,6	17,5	2,4	82,9	115,2	**85,2**	146,9
c1355	514	13,2	7,5	11,4	9,4	55,2	224,6	**59,7**	238,8
c1908	718	79,2	5,5	11,6	6,1	**82,2**	616,0	61,8	310,4
c2670	997	**81,7**	10,2	26,3	11,6	80,8	768,5	68,1	753,8
c3540	1446	64,9	11,8	33,7	15,2	**83,3**	3454,9	36,0	235,2
c5315	1994	81,8	44,4	49,3	31,4	**89,1**	11109,1	37,7	573,4
c6288	2416	10,4	24,4	20,0	23,9	**66,4**	9087,9	27,4	416,4
c7552	2978	69,4	71,8	41,9	54,3	**85,4**	18839,4	40,1	1250,4

Tab. 4. Wyniki zbiorcze przeprowadzonych testów dla zasobów pozyskanych za pomocą algorytmu MNP oraz 0% wydłużenia harmonogramu.

Algorytm:		BF		IIOI		NatPop		EL2k3	
test	$\|V\|$	P_r [%]	t [s]	P_r [%]	t [s]	P_r [%]	t [s]	P_r [%]	t [s]
c432	160	**2,1**	0,7	0,0	0,5	0,8	9,8	0,8	7,6
c499	202	0,0	2,9	0,0	1,9	0,0	9,4	0,0	11,1
c880	357	0,2	1,1	0,2	0,6	0,4	12,0	**0,6**	10,2
c1355	514	0,0	4,8	0,0	5,2	0,0	14,6	0,0	20,0
c1908	718	5,2	3,2	0,2	2,0	16,6	83,4	**17,2**	59,7
c2670	997	6,1	4,4	0,0	2,3	9,4	108,6	**9,8**	108,2
c3540	1446	**2,7**	5,5	0,0	1,7	2,1	71,9	2,4	70,8
c5315	1994	4,5	5,4	0,0	2,8	**5,1**	260,6	**5,1**	180,7
c6288	2416	**1,2**	5,0	0,0	3,7	0,0	21,4	0,0	21,8
c7552	2978	**3,5**	6,2	0,0	3,3	3,1	552,2	2,8	277,1

Tab. 5. Wyniki zbiorcze przeprowadzonych testów dla zasobów pozyskanych za pomocą algorytmu MNP oraz 20% wydłużenia harmonogramu.

Algorytm		BF		IIOI		NatPop		EL2k3	
test	$\|V\|$	P_r [%]	t [s]	P_r [%]	t [s]	P_r [%]	t [s]	P_r [%]	t [s]
c432	160	20,6	0,9	0,0	0,4	22,6	15,7	**23,1**	9,0
c499	202	18,5	4,4	11,0	2,1	24,6	23,5	**29,6**	27,5
c880	357	**20,2**	1,5	0,2	0,7	11,0	18,7	11,6	17,5
c1355	514	12,0	4,3	4,8	6,0	35,0	114,2	**35,5**	127,2
c1908	718	**48,9**	4,8	5,0	1,6	41,9	122,8	42,7	84,7
c2670	997	**27,8**	5,2	5,9	2,1	21,7	177,4	26,2	169,5
c3540	1446	**24,2**	5,0	2,2	2,2	15,3	166,5	17,0	125,4
c5315	1994	20,7	5,4	3,4	3,7	**24,4**	1033,4	19,5	322,7
c6288	2416	**22,4**	5,6	0,2	4,2	14,4	250,0	14,6	263,8
c7552	2978	**20,1**	6,7	1,4	3,5	18,7	1391,3	15,5	442,4

Tab. 6. Wyniki zbiorcze przeprowadzonych testów dla zasobów pozyskanych za pomocą algorytmu MNP oraz 50% wydłużenia harmonogramu.

Algorytm:		BF		IIOI		NatPop		EL2k3	
bench.	$\|V\|$	P_r [%]	t [s]	P_r [%]	t [s]	P_r [%]	t [s]	P_r [%]	t [s]
c432	160	**54,2**	0,9	30,9	0,6	45,9	13,5	50,1	12,0
c499	202	44,4	3,1	35,7	2,3	52,0	33,0	**58,8**	56,7
c880	357	**45,3**	2,0	4,5	0,7	31,7	35,4	36,3	25,0
c1355	514	12,1	4,3	11,8	7,8	54,3	157,3	**55,9**	190,1
c1908	718	**64,5**	4,3	8,9	1,7	50,8	125,6	46,2	108,0
c2670	997	**47,7**	4,1	8,5	2,7	44,8	226,2	46,2	216,2
c3540	1446	35,6	5,2	9,2	2,1	39,9	304,8	**41,5**	287,5
c5315	1994	43,1	4,9	9,9	3,4	41,5	862,7	**44,1**	496,7
c6288	2416	**51,8**	5,1	2,2	3,7	36,0	488,6	19,5	295,9
c7552	2978	**35,2**	7,0	2,1	4,9	32,6	2495,4	33,8	870,8

Testy systemu SAPR przeprowadzono na procesorze AMD Athlon XP 2000+ (32bit, 1667 MHz, 1GB RAM) korzystając z dystrybucji środowiska Sun Microsystems J2SE 1.4.2_09. Wykorzystano przykłady testowe ISCAS'85 i ISCAS'89 [6].

Zaprezentowane algorytmy pozwalają na redukcję poboru mocy dynamicznej układów scalonych CMOS na etapie HLS. W ten sposób osiągane jest zmniejszenie ich chwilowej i szczytowej temperatury pracy, zwiększenie niezawodności i obniżenie kosztów wytworzenia. W zależności od struktury grafu przepływu danych redukcja poboru mocy testowanych układów wahała się od 1.5 do 78.8% (dla c5315 oraz algorytmu ewolucyjnego) bez pogorszenia przepustowości układów. Wydłużenie harmonogramu (testowane były przypadki 20 i 50%) umożliwiło redukcję poboru mocy nawet do 89%.

Dla małych wydłużeń harmonogramu (Tablice 1, 4) lepsze wyniki uzyskiwano przy pomocy algorytmów genetycznych. Przy większych wydłużeniach (Tablice 2-3,5-6), wyniki generowane przez algorytm BF okazywały się lepsze. Ponadto przewagę swoją ukazywał wcześniej dla bardziej ograniczonych zasobów, uzyskanych przy pomocy algorytmu MNP. Uzyskanie wyników przy pomocy algorytmu IIOI zajmowało najmniej czasu, jednak wyniki te były średnio o 32% gorsze od najlepszego uzyskanego wyniku.

Zgodnie z oczekiwaniami, dla testów z liczbą zasobów zoptymalizowaną przy pomocy algorytmu MNP (Tablice 4-6) uzyskano mniejszą redukcję mocy. Wynosiła ona 27% wobec 61% dla liczby zasobów uzyskanej przy pomocy algorytmu ASAP (Tablice 1-3).

5. PODSUMOWANIE

Sformułowanie zagadnień projektowania układów cyfrowych w ramach teorii szeregowania zadań przynosi różne korzyści, ułatwiając przy tym badanie problemów optymalizacyjnych, wobec których staje konstruktor układów. Wiele z tych zagadnień okazuje się być uogólnieniami modeli znanych od dawna w literaturze z zakresu badań operacyjnych. Pozwala to przenosić na grunt elektroniki rozmaite wyniki negatywne oraz wnioskować o NP–trudności napotykanych zagadnień. Można również żywić nadzieję na podobną przekładalność rezultatów pozytywnych tj. metod i algorytmów heurystycznych sprawdzonych przy konstrukcjach harmonogramów. Niektóre z nich winny pozwalać zaadaptować się do

nowych zagadnień tworząc efektywne algorytmy wspomagające projektowanie układów cyfrowych CMOS. Próba takiego podejścia stanowi obiecujący obszar badawczy.

6. BIBLIOGRAFIA

1. Giaro K., Szcześniak W.: Formalizm i metody szeregowania zadań dla potrzeb redukcji poboru mocy cyfrowych układów CMOS, Zeszyty Naukowe Wydziału Elektrotechniki i Automatyki Nr 22, Gdańsk 2006, ISSN 1425-5766.
2. Kozieł S., Szcześniak W.: Reducing average and peak temperatures of VLSI CMOS circuits by means of evolutionary algorithm applied to high level synthesis, Microelectronics Journal, 2003, Elsevier, Vol.34 2003, s. 1167-1174, ISSN 0026-2692.
3. Szcześniak P., Szcześniak W.: Dobór optymalnej liczby jednostek funkcjonalnych dla realizacji syntezy wysokiego poziomu układów cyfrowych, Zeszyty Naukowe Wydziału Elektrotechniki i Automatyki Nr 21, Gdańsk 2005, s. 237-245, ISSN 1425-5766
4. Szcześniak W., Voss B., Theisen M., Becker J., Glesner M.: Influence of high-level synthesis on average and peak temperatures of CMOS circuits, Microelectronics Journal, vol. 32, Oct. 2001, s. 855-862, ISSN 0026-2692
5. Włodarczyk Ł.: Komputerowa weryfikacja wybranych algorytmów szeregowania zadań dla potrzeb redukcji poboru mocy układów cyfrowych. Praca dypl. Politechnika Gdańska 2005.
6 Collaborative Benchmarking Laboratory, ftp.cbl.ncsu.edu

7. PODZIĘKOWANIA

Autorzy artykułu dziękują Kol. Krzysztofowi Giaro za konstruktywne uwagi dotyczące prezentowanych zagadnień.

Niniejsza praca została w części sfinansowana przez Ministerstwo Nauki i Informatyzacji, grant nr 3T11B 015 27.

COMPARISON OF CHOSEN TASK SCHEDULING ALGORITHMS APPLIED AT THE STAGE OF HIGH LEVEL SYNTHESIS OF DIGITAL CMOS CIRCUITS

Task scheduling applied to reduction of power dissipated in digital CMOS circuits leads to a NP hard problem. Hence, there are no exist analytical algorithms which guarantee a optimal solution in an acceptable computational time. This paper presents a comparison of quality of solutions obtained for a chosen set of benchmarks with different task scheduling heuristic algorithms applied at the stage of high level synthesis of digital CMOS circuits. The presented comparison contains also calculation times for the implemented task scheduling algorithms.