Uniwersytet Zielonogórski
Instytut Informatyki i Elektroniki

Małgorzata Kołopieńczyk

Zastosowanie konwertera adresów do zmniejszenia rozmiaru pamięci mikroprogramowanego układu sterującego ze współdzieleniem kodów

Rozprawa doktorska

Promotor:
dr hab. inż. Larysa Titarenko

Zielona Góra, czerwiec 2007

Spis treści

1. Wstęp

Obecnie coraz częściej przy projektowaniu systemów cyfrowych bierze się pod uwagę takie kryteria jak efektywne wykorzystanie zasobów sprzętowych oraz niski pobór energii (szczególnie w przypadku urządzeń przenośnych). Problem optymalizacji rozmiaru pamięci jednostki sterującej jest zatem ciągle aktualnym zadaniem w informatyce. W przypadku zastosowania układów *FPGA* (ang. *Field Programmable Gate Array, FPGA*), jednym ze sposobów rozwiązania tego problemu jest zmniejszenie liczby bitów w adresie mikroinstrukcji (Bolton, 1991; DeMicheli, 1994; In-Cheol Park i in. 1994; Wang, Roy, 1999; Barkalov i in., 2005; Bomar, 2002).

W chwili obecnej układy cyfrowe mogą być implementowane z wykorzystaniem pojedynczego układu *VLSI* typu *System-on-a-chip* (*SoC*) (Grushnitsky i in., 2002; Maxfield, 2004). Przewiduje się, że do roku 2012 układy cyfrowe w swojej strukturze zawierać będą do około 1,3 miliarda tranzystorów (Brown, Vernesic, 2000; Iwai, 2004). Regułą jest, że bloki funkcjonalne układu *SoC* implementowane są z wykorzystaniem matryc programowalnych *FPGA* (Jenkins, 1994; Maxfield, 2004).

Klasyczna metoda implementacji jednostki sterującej jako skończonego automatu stanów pochłania zasoby układów programowalnych, które mogą być w efektywniejszy sposób wykorzystane przez inne bloki projektowanego systemu (Adamski, Barkalov, 2006; Barkalov i in. 2006).

W rozprawie zaproponowano realizację jednostki sterującej z wykorzystaniem mikroprogramowanego układu sterującego, który najbardziej pasuje dla implementacji algorytmów liniowych. Mikroprogramowany układ sterujący jest rozwinięciem modelu skończonego automatu stanów (Zieliński, 2003; Pochopień, 2004; Łuba, 2005; Kamionka-Mikuła i in., 2006). Taka organizacja jednostki sterującej umożliwia zachowanie minimalnego rozmiaru pamięci w porównaniu z dobrze znanymi strukturami opisanymi w literaturze (Baranov, 1994; Molski, 1986; Traczyk, 1986; Barkalov, Węgrzyn, 2006).

W pracy zaprezentowano sposób organizacji mikroprogramowanego układu sterującego, dla algorytmu sterowania opisanego liniową siecią działań, który umożliwia zmniejszenie rozmiaru pamięci jednostki sterującej w porównaniu z układem o strukturze podstawowej (Barkalov i in., 2006a, 2006b; Titarenko, Kołopieńczyk 2006).

1.1. Motywacja

Z przeglądu literatury przedmiotu wynika, że jednostka sterująca może być implementowana jako mikroprogramowany układ sterujący (ang. *Compositional Microprogram Control Unit, CMCU*), w którym blok pamięci *CM* (ang. *Control memory, CM*) przechowuje mikroprogram, natomiast układ kombinacyjny CC (ang. *Combinational Circuit, CC*) wyznacza adres mikroinstrukcji w przypadku gdy naturalny porządek wykonywania mikroinstrukcji nie jest zachowany.

Wykorzystując układy *FPGA* do implementacji jednostki sterującej należy pamiętać, że główną ich cechą jest wykorzystanie elementów *LUT* (ang. *Look-Up Table*) do realizacji funkcji logicznych. Ilość wejść elementu *LUT* jest ściśle ograniczona. To ograniczenie prowadzi do zastosowania dekompozycji funkcji boolowskich opisujących zachowanie jednostki sterującej (Baranov, 1994; DeMicheli, 1994; Barkalov, 1998; Brown, Vernesic, 2000; Gorzałczany, 2000, Łuba i in., 2002). Negatywnym skutkiem funkcjonalnej dekompozycji jest powstawanie układów wielopoziomowych, co prowadzi do zmniejszenia szybkości działania układu. Aby poprawić negatywne skutki dekompozycji ilość argumentów w implementowanej funkcji powinna zostać zmniejszona jak to tylko możliwe. W przypadku mikroprogramowanego układu sterującego problem ten może zostać rozwiązany poprzez zastosowanie metody współdzielenia kodów. W metodzie tej, dla układu opisanego siecią działań, adres mikroinstrukcji reprezentowany jest jako połączenie kodu łańcucha sieci działań z kodem elementu łańcucha.

Takie podejście ma sens tylko wówczas, gdy długość adresu mikroinstrukcji jest taka sama jak minimalna długość adresu w przypadku mikroprogramowanego układu sterującego o strukturze podstawowej (Barkalov, 2005; Barkalov i in., 2006c).

Jednakże istnieje wiele sieci działań, dla których warunek ten zostaje naruszony. W takim przypadku pamięć mikroprogramowanego układu sterującego zwiększa się w porównaniu z jej minimalną wartością dla układu o strukturze podstawowej (Barkalov i in., 2006b; 2006d).

W pracy doktorskiej proponowane jest nowe podejście umożliwiające wykorzystanie metody współdzielenia kodów niezależnie od charakterystyki interpretowanej sieci działań. Proponowane metody bazują na wprowadzeniu bloku konwertera adresów do mikroprogramowanego układu sterującego. Konwerter adresów przekształca adres mikroinstrukcji na adres o minimalnej pojemności bitowej, co umożliwia utrzymanie rozmiaru pamięci na możliwie minimalnym poziomie.

W rozprawie opisane zostaną metody umożliwiające zmniejszenie rozmiaru pamięci mikroprogramowanego układu sterującego ze współdzieleniem kodów, dla którego algorytm sterowania został opisany liniową siecią działań.

W praktyce często są wykorzystywane automaty współbieżne, dla których konieczne jest opracowanie efektywnych metod syntezy.

Najczęściej stosowaną metodą syntezy układu cyfrowego jest dekompozycja automatu współbieżnego na sekwencyjne automaty składowe (Adamski, 1982, 1998a; Bliński i in. 1994;). W rozprawie przyjęto najbardziej oczywistą ścieżkę projektowania, którą zaproponowano dla implementacji i optymalizacji tych automatów:

- utworzenie opisu działania układu cyfrowego za pomocą diagramu *SFC*,
- konwersja diagramu *SFC* na interpretowaną sieć Petriego,
- dekompozycja automatu współbieżnego opisanego siecią Petriego na sekwencyjne automaty składowe,
- przekształcenie powstałych automatów składowych na sieci działań (ang. *Flow-Chart of Algorithm, FCA*),
- implementacja jednostki sterującej opisanej siecią działań.

W wyniku przejścia z diagramu *SFC* do *FCA* można otrzymać sieci działań różnego typu. W zależności od rodzaju sieci działań wykorzystywane są różne struktury oraz metody optymalizacji opisanego w ten sposób układu.

W rozprawie skoncentrowano się na sieciach działań o charakterze liniowym. Dla tego typu sieci jednostka sterująca implementowana jest jako mikroprogramowany układ sterujący.

Proponowana metoda jest szczególnie przydatna w przypadku diagramu *SFC*, gdzie liniowe sekwencje współbieżne występują bardzo często (Adamski 1982; IEC 1992; Lewis 1995; Jiang, Holding 1996; Adamski, Chodań 2000).

Metoda opisu działania układu cyfrowego za pomocą diagramu *SFC* oraz sposób konwersji *SFC* na sieć Petriego została opisana w (Adamski, Chodań 2000; Węgrzyn, 2003). Sposób dekompozycji automatu współbieżnego na sekwencyjne automaty składowe można znaleźć w (Adamski, 1990, 1998). Metoda przejścia z automatów składowych na sieci działań wykracza poza zakres pracy.

1.2. Teza i cele pracy

Analizując aktualny stan badań postawiono następującą tezę pracy:

> **Wprowadzenie konwertera adresów w automatach pokrywających diagram SFC, które implementują liniowe, powiązane sieci działań, umożliwia zmniejszenie rozmiaru pamięci jednostki sterującej.**

Z tak przyjętej tezy głównej wynikają następujące cele o charakterze teoretycznym:

1. Opracowanie metody syntezy mikroprogramowanego układu sterującego z konwerterem adresów.
2. Opracowanie metody syntezy mikroprogramowanego układu sterującego z rozszerzonymi mikroinstrukcjami.
3. Opracowanie metody syntezy mikroprogramowanego układu sterującego z kodowaniem kolekcji mikrooperacji.
4. Opracowanie metody syntezy mikroprogramowanego układu sterującego z elementarnymi łańcuchami.
5. Opracowanie metody syntezy mikroprogramowanego układu sterującego z elementarnymi łańcuchami i konwerterem adresów.
6. Opracowanie metody syntezy mikroprogramowanego układu sterującego z elementarnymi łańcuchami i rozszerzonymi mikroinstrukcjami.
7. Opracowanie metody syntezy mikroprogramowanego układu sterującego z elementarnymi łańcuchami i kodowaniem kolekcji mikrooperacji.

Realizacja celów teoretycznych stanowi podstawę pod budowę autorskiego oprogramowania, zwanego dalej *fca2cmcu* (jest to skrótowiec utworzony od angielskich słów *Flow-Chart of Algorithm to Compositional Microprogram Control Unit, fca2cmcu*), umożliwiającego implementację jednostki sterującej za pomocą wybranych metod.

Ostatnim celem praktycznym jest przeprowadzanie implementacji przykładowych modeli w strukturach programowalnych – jako docelowe wybrano popularne i dostępne autorce układy programowalne *FPGA* firmy *Xilinx* (http1).

1.3. Struktura pracy

Praca została podzielona na siedem części. Rozdział pierwszy wprowadza do tematyki poruszanej w pracy. W rozdziale tym przedstawiona została teza i cele pracy oraz motywacja podjęcia się badań.

W rozdziale drugim zaprezentowano dwie metody modelowania układów cyfrowych: diagramy *SFC* i interpretowane sieci Petriego oraz sposób transformacji diagramu *SFC* do sieci Petriego. Przedstawiono również przykład dekompozycji automatu współbieżnego na sekwencyjne automaty składowe.

Rozdział trzeci przedstawia zagadnienia dotyczące metod projektowania układów sterujących w strukturach programowalnych. Omówione zostaną trzy sposoby syntezy układów sterujących.

Rozdział czwarty stanowi najistotniejszą część pracy, w której opisano oryginalne metody syntezy mikroprogramowanych układów sterujących umożliwiające zmniejszenie rozmiaru pamięci jednostki sterującej.

W rozdziale piątym przedstawiono opis autorskiego oprogramowania *fca2cmcu*. Omówiono jego budowę, działanie oraz format danych wejściowych i wyjściowych.

Rozdział szósty zawiera wyniki testów przeprowadzonych dla poszczególnych struktur mikroprogramowanych układów sterujących.

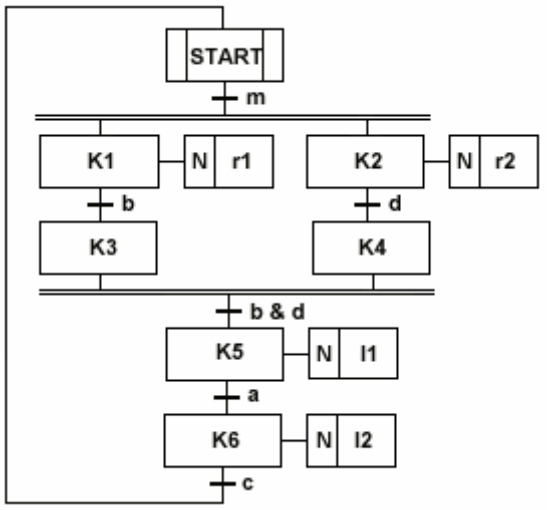
Rozdział siódmy stanowi podsumowanie pracy oraz wskazuje kierunki dalszych badań.

2. Modelowanie współbieżnych układów cyfrowych

W rozdziale drugim zostaną przedstawione sposoby modelowania współbieżnych układów cyfrowych z wykorzystaniem diagramów *SFC* (ang. *Sequential Function Chart*) i interpretowanych sieci Petriego (Halang, 1989; Bliński, 1994; Kozłowski i in., 1995; Lewis, 1995; Kalinowski, 1984) oraz przykład dekompozycji automatu współbieżnego opisanego siecią Petriego na sekwencyjne automaty składowe (Adamski, 1982, 1990).

2.1. Diagramy SFC

Diagram *SFC* jest językiem graficznym, objętym normą *IEC 1131* (IEC, 1992) z wykorzystaniem, którego można przedstawić w sposób współbieżny operacje sekwencyjne. Proces binarny jest reprezentowany za pomocą poprawnie zdefiniowanych kolejnych kroków połączonych z tranzycjami (Rys. 2.1). Elementy diagramu *SFC* są przydatnym narzędziem graficznym wykorzystywanym do opisu sterowników logicznych (Lewis, 1995).



Rys. 2.1. Przykładowy diagram *SFC*

Definicja 2.1. Diagram SFC (Halang, 1989)

Diagram *SFC* definiowany jest w postaci uporządkowanej czwórki $SFC = (S, T, L, I)$ gdzie:

- S – niepusty, skończony zbiór kroków;
- T – niepusty, skończony zbiór tranzycji;

- L – niepusty, skończony zbiór połączeń pomiędzy krokiem a tranzycją lub tranzycją a krokiem;
- $I \subset S$ – zbiór kroków inicjujących.

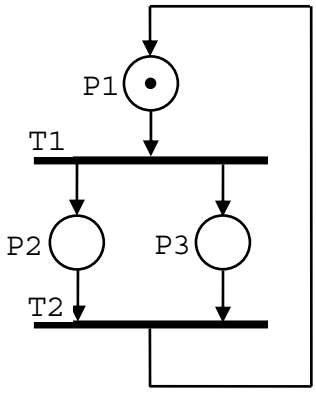
Zgodnie z normą *IEC 1131*, zbiór kroków inicjujących powinien być zbiorem jednoelementowym. Elementy zbiorów S i T są reprezentowane jako węzły grafu. Zbiór kroków inicjujących rozpoczyna proces i determinuje stan inicjujący procesu. Z każdym krokiem powiązana jest akcja, której wykonanie następuje wtedy, gdy krok jest aktywny. Z każdą tranzycją powiązany jest warunek boolowski: jeżeli krok poprzedzający tranzycję jest aktywny i warunek spełniony, to następuje dezaktywacja kroku poprzedzającego tranzycję i aktywacja kroku następującego po przejściu (IEC, 1992; Lewis, 1995).

Krok określający stan procesu, oraz tranzycja, określająca zmianę stanu wraz z warunkiem tej zmiany, są dwoma podstawowymi i niezbędnymi elementami diagramu. Wykorzystanie tych dwóch komponentów daje realny obraz działania sterownika. Dodatkowy składnik diagramu – blok działania, ukazuje szczegółowy, techniczny opis układu (Adamski 1998; Lewis, 1995). Pozwala on określić rodzaj akcji (kwalifikator), symbol zwrotnej zmiennej logicznej lub opisać zachowanie układu sterującego z wykorzystaniem języka *ST* (ang. *Structured Text*), *IL* (ang. *Instruction List*), a także reprezentację akcji boolowskiej (IEC, 1992; Adamski, Chodań, 2000).

2.2. Interpretowane sieci Petriego

Sieć Petriego jest dwudzielnym skierowanym grafem, w którym występują dwa rodzaje wierzchołków, zwane miejscami i tranzycjami (Peterson, 1981; Reisig, 1988; Murata, 1989; Adamski, 1990; 1998; Banaszak i in., 1993).

Graficznie sieć Petriego reprezentowana jest jako struktura złożona z prostokątów i kółek połączonych łukami (Rys. 2.2). Prostokąty reprezentują przejścia, kółka – miejsca, a łuki – relację przepływu między miejscami i przejściami oraz przejściami i miejscami. Każdemu miejscu w sieci może być przyporządkowany co najwyżej jeden żeton (marker, znacznik).



Rys. 2.2. Przykładowa sieć Petriego

Definicja 2.2. Interpretowana sieć Petriego

Interpretowaną siecią Petriego (z wejściami binarnymi i wyjściami binarnymi typu Moore'a i Mealego) nazywana jest uporządkowana szóstka:

$PN_{IO} = (PN, X, Y, \rho, \lambda, \gamma)$,

gdzie: PN to żywa i bezpieczna szkieletowa sieć Petriego;

X jest zbiorem stanów wejść; $X = \{X\}$;

Y jest zbiorem stanów wyjść; $Y = \{Y\}$;

- $\rho : T \to 2^X$ jest funkcją, która każdej tranzycji przyporządkowuje jednoznacznie podzbiór stanów wejść $X(T)$. Symbol 2^X oznacza zbiór wszystkich możliwych podzbiorów X;
- $\lambda : M \to Y$ jest funkcją, wyjść typu Moore'a, która każdemu znakowaniu sieci M przyporządkowuje jednoznacznie pewien stan wyjść $Y(M)$;
- $\gamma : (M \times X) \to Y$ jest funkcją wyjść typu Mealy'ego, która znakowaniu sieci M oraz stanowi wejść przyporządkowuje jednoznacznie podzbiór stanów wyjść Y.

Stan wejść charakteryzowany jest wektorem binarnym $X = (x_1, x_2, \ldots, x_n)$. Stan wyjść charakteryzowany jest wektorem binarnym $Y = (y_1, y_2, \ldots, y_m)$.

Dla uproszczenia definicji założono, że wszystkie wyjścia binarne automatu są określone w sposób jednoznaczny i przyjmują wartość *1* (gdy wyjścia są aktywne) lub *0* (gdy wyjścia są nieaktywne).

Miejsca sieci reprezentują stany lokalne sterownika, tranzycje ich zmiany. Wszystkie miejsca oznakowane w tym samym czasie, definiują stan globalny sterownika. Znaczniki

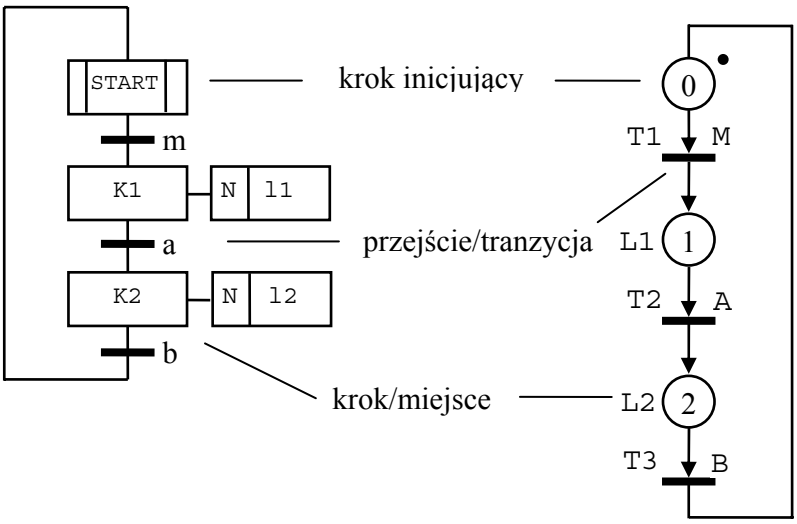
wyznaczą, które stany sterownika są aktywne w danym czasie. Oznakowanie początkowe reprezentuje stan początkowy układu. Każda tranzycja typu rozwidlenie (ang. *fork*), tzn. tranzycja posiadająca więcej niż jedno miejsce wyjściowe, jest początkowym punktem dla równoległych procesów, natomiast każda tranzycja typu złączenie (ang. *join*), tzn. tranzycja posiadająca więcej niż jedno miejsce wejściowe, synchronizuje procesy, które łączą się w tym punkcie. Do specyfikacji automatów współbieżnych wykorzystywane są najczęściej synchroniczne, interpretowane sieci Petriego (Kozłowski i in., 1995; Węgrzyn, 2003).

Możliwe jest wyznaczenie wszystkich stanów globalnych automatu współbieżnego, czyli wyznaczenie jego przestrzeni stanów lokalnych. Z punktu widzenia teorii sieci Petriego odpowiada to wyznaczeniu grafu znakowań sieci i przypisaniu wierzchołkom tego grafu nazw stanów równoważnego mu klasycznego automatu sekwencyjnego. Wyznaczanie wszystkich stanów globalnych automatu, nawet dla sieci stosunkowo prostych, staje się wyjątkowo pracochłonne (Adamski, Chodań, 2000).

2.3. Interpretowana sieć Petriego a diagram SFC

Sposób przedstawiania procesu sterowania za pomocą diagramu *SFC* nie odbiega znacznie od sposobu reprezentacji tego samego procesu za pomocą interpretowanej sieci Petriego (Adamski, Chodań, 2000; Węgrzyn, Chodań, 2000). Reguły modelowania *SFC* i sieci Petriego są podobne. Inna jest natomiast reprezentacja graficzna podstawowych elementów obu sieci tj. kroków, kroku początkowego, realizacji procesów współbieżnych.

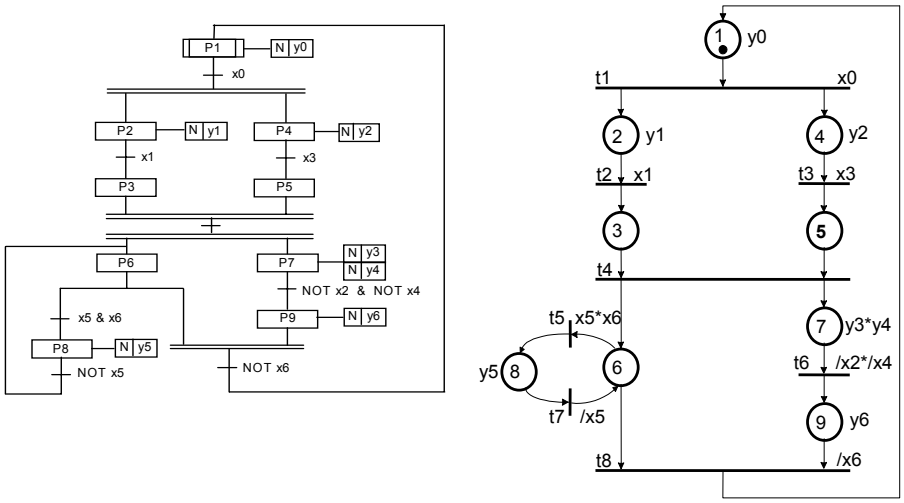
Podstawowe elementy graficzne diagramu *SFC* i sieci Petriego porównano na Rys. 2.3.



Rys. 2.3. Porównanie elementów graficznych diagramu *SFC* i sieci Petriego

W diagramach *SFC*, z każdym krokiem może być powiązany blok akcji, zawierający opis rodzaju akcji wykonywanej w chwili, gdy krok ten stanie się aktywny. Opis akcji może być podany w językach: *ST*, *FBD*, *LD* oraz *IL*, zdefiniowanych w normie *IEC:1131*. Jednakże takiej specyfikacji diagramów SFC nie można odwzorować bezpośrednio w sieciach Petriego.

Na Rys. 2.4 przedstawiono diagram *SFC* i odpowiadającą mu sieć Petriego (Adamski, 1998a). Przekształcenia dokonano zgodnie z regułami przedstawionymi na Rys. 2.3.



Rys. 2.4. Transformacja diagramu *SFC* na sieć Periego

2.4. Automat współbieżny

W odróżnieniu od klasycznego automatu sekwencyjnego, automat współbieżny znajduje się równocześnie w jednym lub kilku stanach wewnętrznych. Maksymalne zbiory równocześnie występujących stanów lokalnych definiują stany globalne automatu. Dowolny podzbiór równocześnie występujących stanów lokalnych nazywany jest stanem częściowym. W automatach współbieżnych rozpatruje się zamiast globalnych funkcji przejść lokalne relacje, wiążące ze sobą wewnętrzne stany częściowe, aktualne i następne, oraz odpowiednie stany wejść i wyjść automatu. Do opisu automatów współbieżnych wykorzystywane są metody tabelaryczne, symboliczne oraz grafowe. Interpretowana sieć Petriego jest obrazową formą przedstawienia automatu współbieżnego (Adamski, 1990).

2.5. Dekompozycja sieci Petriego na podsieci typu automatowego

Podczas projektowania może zaistnieć sytuacja, gdy rozmiar układu wykracza poza ramy narzucone rozmiarami sterownika. Dokonuje się wtedy jego podziału na szereg współpracujących ze sobą podukładów. Dekompozycja układu ułatwia proces projektowania, a jej wyniki mogą być wykorzystane do efektywnego kodowania lokalnych stanów wewnętrznych układu sterującego.

Dekompozycja równoległa jest szczególnie uzasadniona, gdy sterownik jest realizowany w postaci połączonych ze sobą mikrokontrolerów. W takim przypadku każdy z komunikujących się mikroprocesorów realizuje jedną z procedur sekwencyjnych.

Najczęściej stosowaną metodą syntezy układu cyfrowego jest dekompozycja automatu współbieżnego na sekwencyjne automaty składowe (Adamski, 1990). Podsieci typu automatowego to podsieci otrzymane po dekompozycji sieci Petriego, które zawierają tylko jeden marker.

Często stosowaną metodą rozbioru sieci Petriego na podsieci automatowe, jest metoda zapoczątkowana przez Andre (Albicki, 1980; Varadharajan, 1987). Metoda Andre polega na wykorzystaniu makrosieci, zawierającej wyłącznie tranzycje wielowejściowe i wielowyjściowe oraz makromiejsca typu automatowego. Makromiejsca te reprezentują sekwencyjne części sieci. Na podstawie makrosieci tworzy się jej graf znakowań i następnie na jego podstawie określa relacje współbieżności W między makromiejscami.

Kolejnym krokiem jest kolorowanie grafu współbieżności W. Poszczególne kolory wyznaczają zbiory makromiejsc niewspółbieżnych (sekwencyjnych). Każdy taki zbiór to pojedynczy automat sekwencyjny (Adamski 1982; Banaszak i in., 1993).

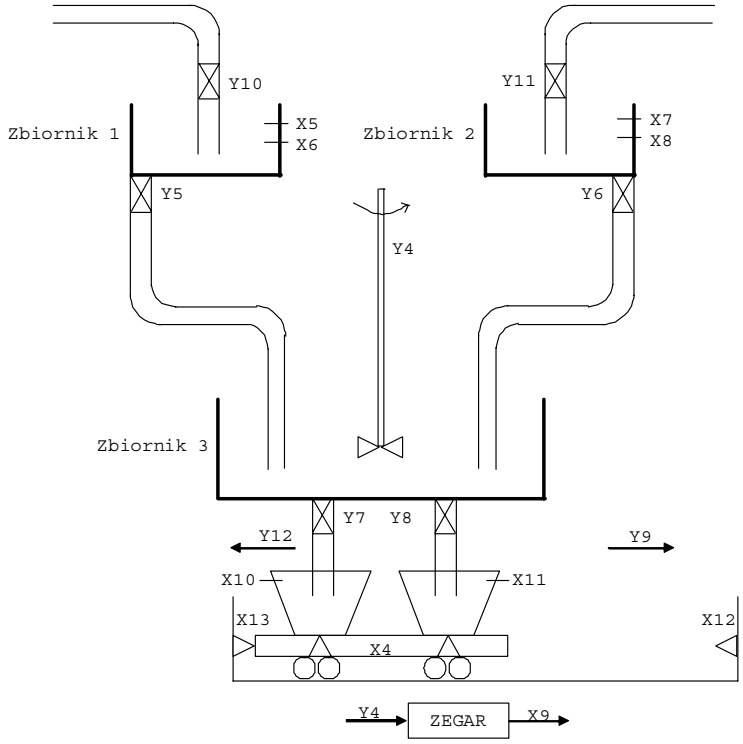
2.6. Przykład dekompozycji sieci Petriego na podsieci typu automatowego

Metoda dekompozycji sieci Petriego na podsieci typu automatowego zostanie przedstawiona na przykładzie procesu produkcji napojów (Adamski, 1990; Blanchard, 1979).

System sterowania produkcją napojów (Rys. 2.5) rozpoczyna działanie po naciśnięciu przycisku startowego $x1$. Wówczas rozpoczyna się napełnianie zbiorników 1 i 2 oraz dostarczanie pojemników na napój. Składniki dostarczane są do zbiornika 1 przez zawór $y10$, do zbiornika 2 przez zawór $y11$, aż do momentu ich napełnienia. Napełnienie zbiornika 1 wskazywane jest stanem czujnika $x5 = 1$, zbiornika 2 stanem czujnika $x7 = 1$.

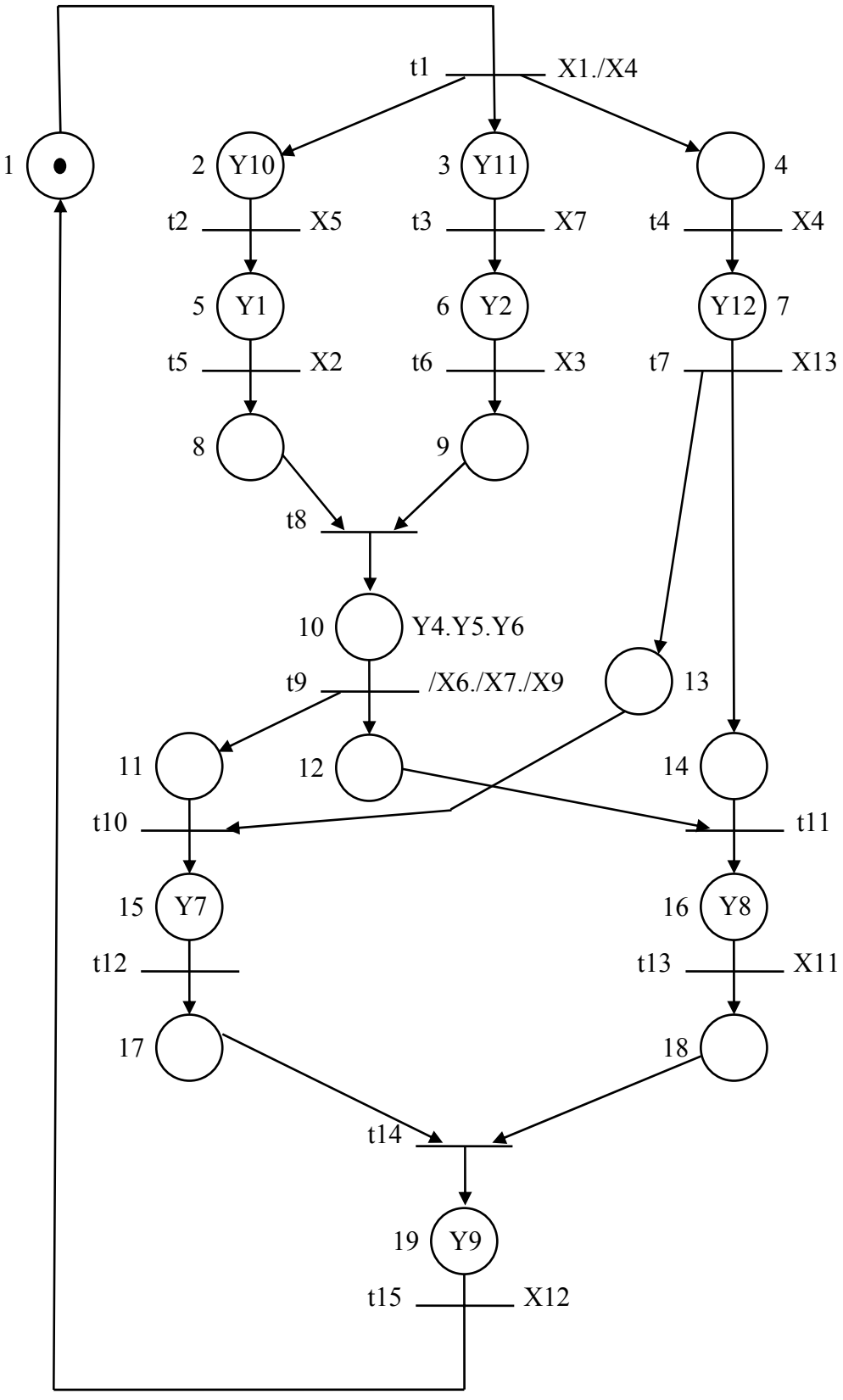
Dostarczanie pojemników odbywa się wózkiem pod wpływem sygnału *y12* i kończy się sygnałem *x13*. Gdy zbiornik *1* lub *2* jest napełniany, następuje przygotowywanie składników, zakończone odpowiednimi sygnałami *x2 = 1* (zbiornik *1*), *x3 = 1* (zbiornik *2*) i *x4 = 1* (pojemniki). Przygotowane składniki ulegają mieszaniu w pojemniku *3*, do którego wpuszczane są zaworami *y5* i *y6*. Zawory te zostają zamknięte po całkowitym opróżnieniu zbiorników *1*, *2* i wymieszaniu składników napoju, co sygnalizowane jest odpowiednio czujnikami *x6 = 0*, *x8 = 0*, *x9 = 0*. Jeżeli pojemniki są gotowe następuje niezależne od siebie napełnianie i zamykanie pojemników *5* i *6*, zasygnalizowane odpowiednio sygnałami *x10* i *x11*. Obydwa pojemniki równocześnie idą do pasteryzacji. Po zakończeniu operacji (*x12 = 1*) układ jest gotowy do pracy.

Uwaga: Mimo, że przyspieszyłoby to proces produkcyjny, zbiorniki *1* i *2* nie mogą być napełniane w trakcie mieszania w zbiorniku *3* oraz transportu produktu ze względu na ewentualne zepsucie się składników.



Rys. 2.5. Model rzeczywisty procesu produkcji napojów

Na Rys. 2.6 został przedstawiony model w postaci sieci Petriego, opisujący proces produkcji napojów. Sieć Petriego składa się z dziewiętnastu kroków opisanych w tabeli 2.1. Opis warunków, które muszą być spełnione, w celu realizacji tranzycji, oraz opis wyjść zamieszczony jest w tabeli 2.2.



Rys. 2.6. Interpretowana sieć Petriego opisująca działanie urządzenia do produkcji napojów

Tabela 2.1. Opis stanów lokalnych sterownika

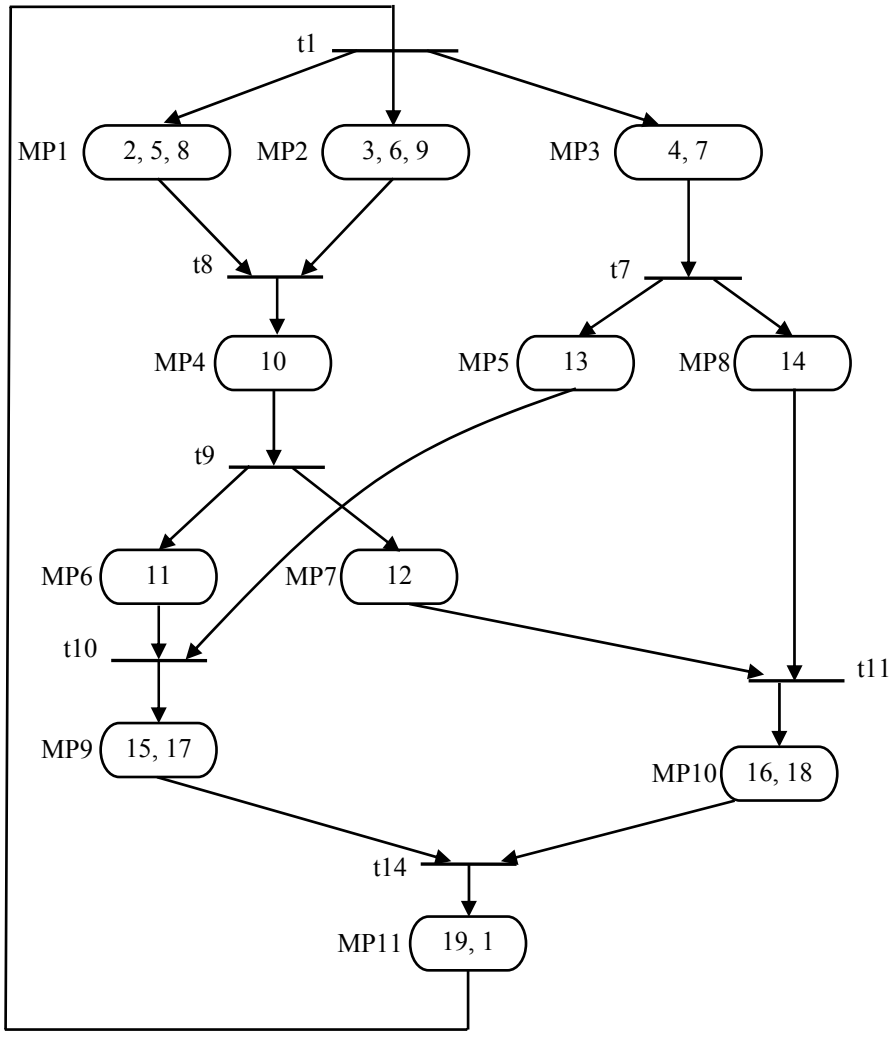
Stan lokalny	**Opis**
p1	Stan początkowy
p2	Napełniania zbiornika 1
p3	Napełniania zbiornika 2
p4	Załadowanie pojemników
p5	Przygotowanie składnika 1
p6	Przygotowanie składnika 2
p7	Wózek jedzie w lewo
p10	Otwarcie zaworów spustowych obu zbiorników i mieszanie składników
p15	Napełnienie pojemnika 1
p16	Napełnienie pojemnika 2
p19	Wózek jedzie w prawo
p8, p9, p11, p12, p13, p14, p17, p18	Oczekiwanie

Tabela 2.2. Opis wejść i wyjść sterownika

Sygnały wejściowe	**Sygnały wyjściowe**
x1 – start	y1 – przygotuj składnik 1
x2 – składnik 1 gotowy	y2 – przygotuj składnik 2
x3 – składnik 2 gotowy	y3 – załaduj pojemniki
x4 – pojemniki na wózku	y4 – mieszaj
x5 – górny poziom w zbiorniku 1 (x5=1)	y5 – otwórz zawór spustowy składnika 1
x6 – dolny poziom w zbiorniku 1 (x6=0)	y6 – otwórz zawór spustowy składnika 2
x7 – górny poziom w zbiorniku 2 (x7=1)	y7 – napełnij pojemnik 1
x8 – dolny poziom w zbiorniku 2 (x8=0)	y8 – napełnij pojemnik 2
x9- koniec mieszana (x9=0)	y9 – wózek jedzie w prawo
x10 – pojemnik 1 napełniony	y10 – napełnij zbiornik 1
x11 – pojemnik 2 napełniony	y11 – napełnij zbiornik 2
x12 – wózek w prawym skrajnym położeniu	y12 – wózek jedzie w lewo
x13 – wózek w lewym skrajnym położeniu	

2.6.1. Makrosieć

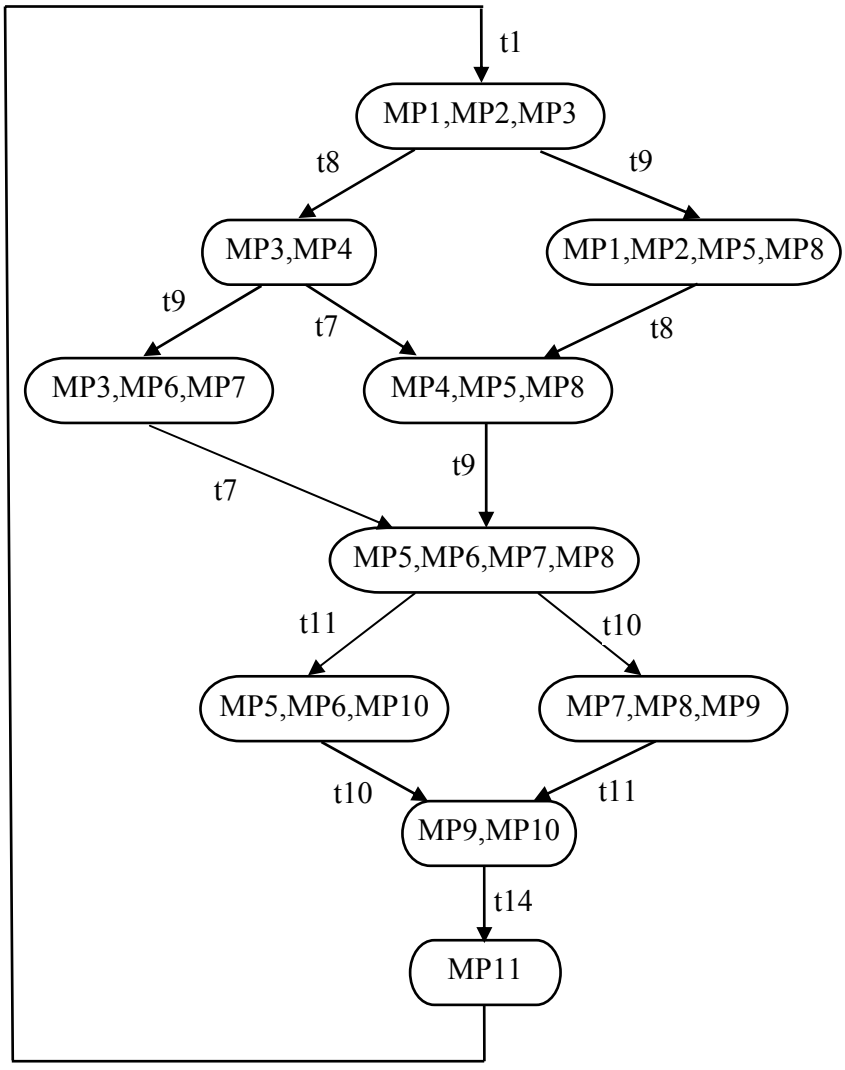
Makrosieć jest skondensowaną wersją danej sieci Petriego, zachowującą jej strukturę i właściwości. Zawiera jedynie tranzycje wielowejściowe i wielowyjściowe oraz miejsca zastępujące jednoznakowe fragmenty sieci. Do jednoznakowego fragmentu sieci wchodzą tranzycje o jednym miejscu wejściowym i jednym miejscu wyjściowym wraz z tymi miejscami, lub ciąg takich tranzycji. Rysunek 2.7 przedstawia makrosieć dla sieci Petriego z rysunku 2.6.



Rys. 2.7. Makrosieć opisująca działanie urządzenia do produkcji napojów

2.6.2. Graf znakowań

Wyznaczanie grafu znakowań polega na analizowaniu zmian oznakowania sieci w miarę odpalania gotowych tranzycji i zapisywaniu wszystkich możliwych stanów w formie grafu. Wierzchołki grafu zawierają zbiór miejsc oznakowanych w danym stanie, krawędzie odpowiadają odpalonym tranzycjom. Do utworzenia grafu znakowań niezbędne są struktura sieci i oznakowanie wstępne.

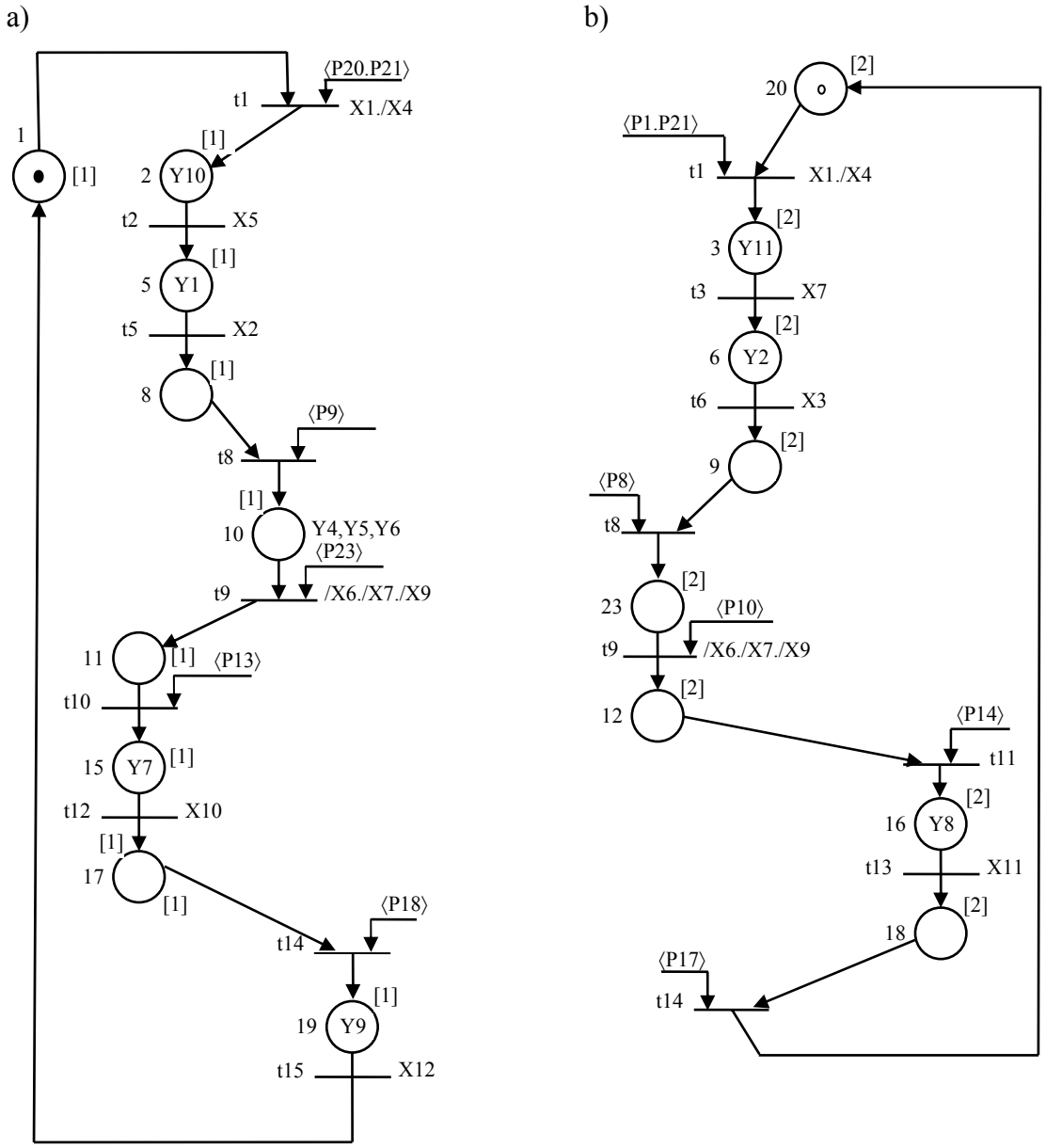


Rys. 2.8. Graf znakowań

Kolejnym krokiem jest kolorowanie grafu współbieżności W, które polega na przypisaniu wierzchołkom grafu niezorientowanego minimalnej liczby kolorów w taki sposób, aby dwa wierzchołki sąsiednie miały zawsze różne kolory. Liczba kolorów znana jest już na początku i jest równa maksymalnej liczbie makromiejsc zawartych we wierzchołku

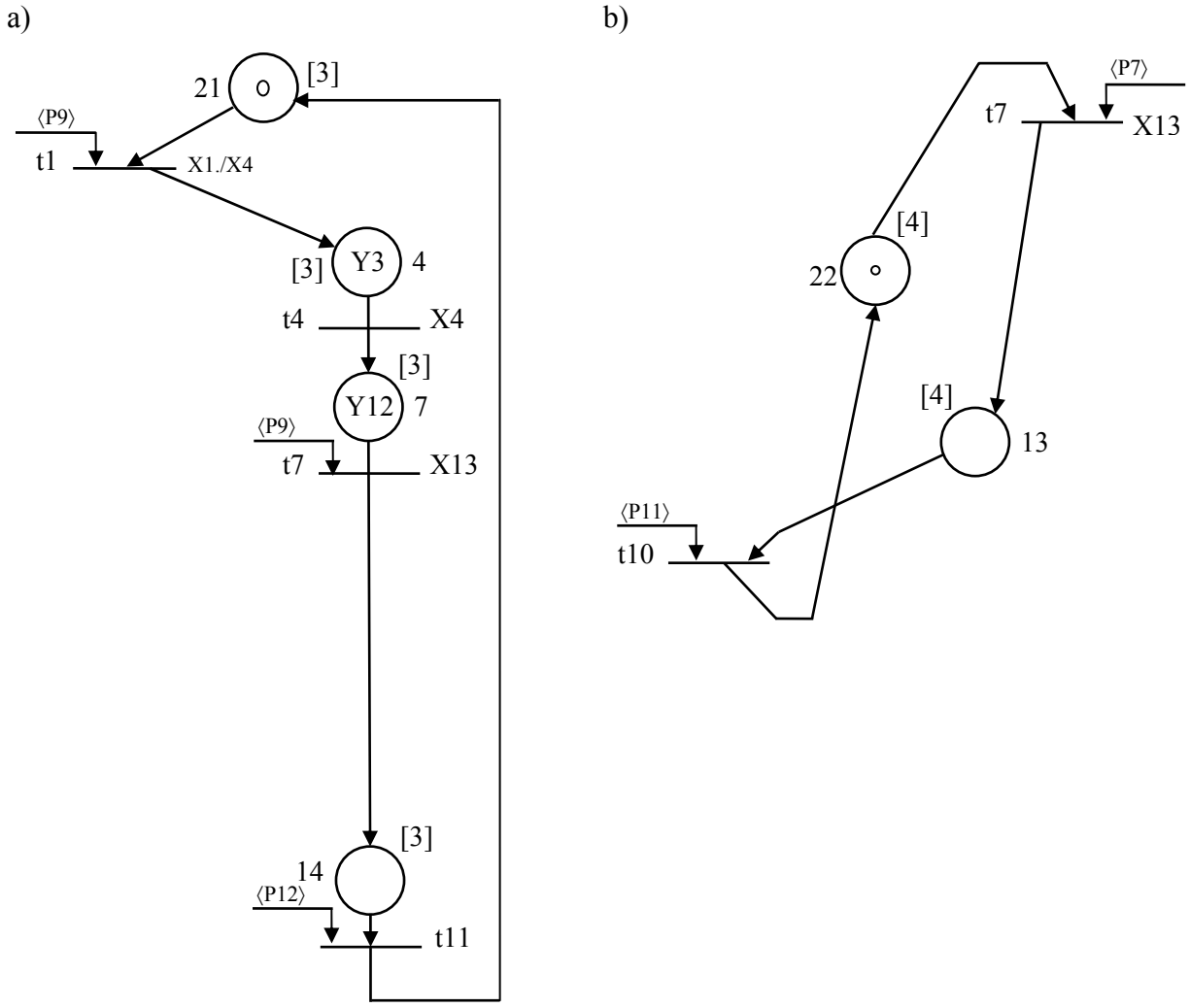
grafu znakowań. Poszczególne kolory wyznaczają zbiory makromiejsc niewspółbieżnych (sekwencyjnych). Każdy taki zbiór to pojedynczy automat sekwencyjny.

Jeżeli zbiór nie zawiera początkowo oznakowanego miejsca sieci, to dodaje się do niego dodatkowe miejsce spoczynkowe, zawierające znak (np. miejsce *p20* na Rys. 2.9b). Miejsce spoczynkowe bez znaku wprowadza się do podsieci zawierającej już znak, gdy nie jest ona spójna (np. miejsce *p23* na Rys. 2.9b).



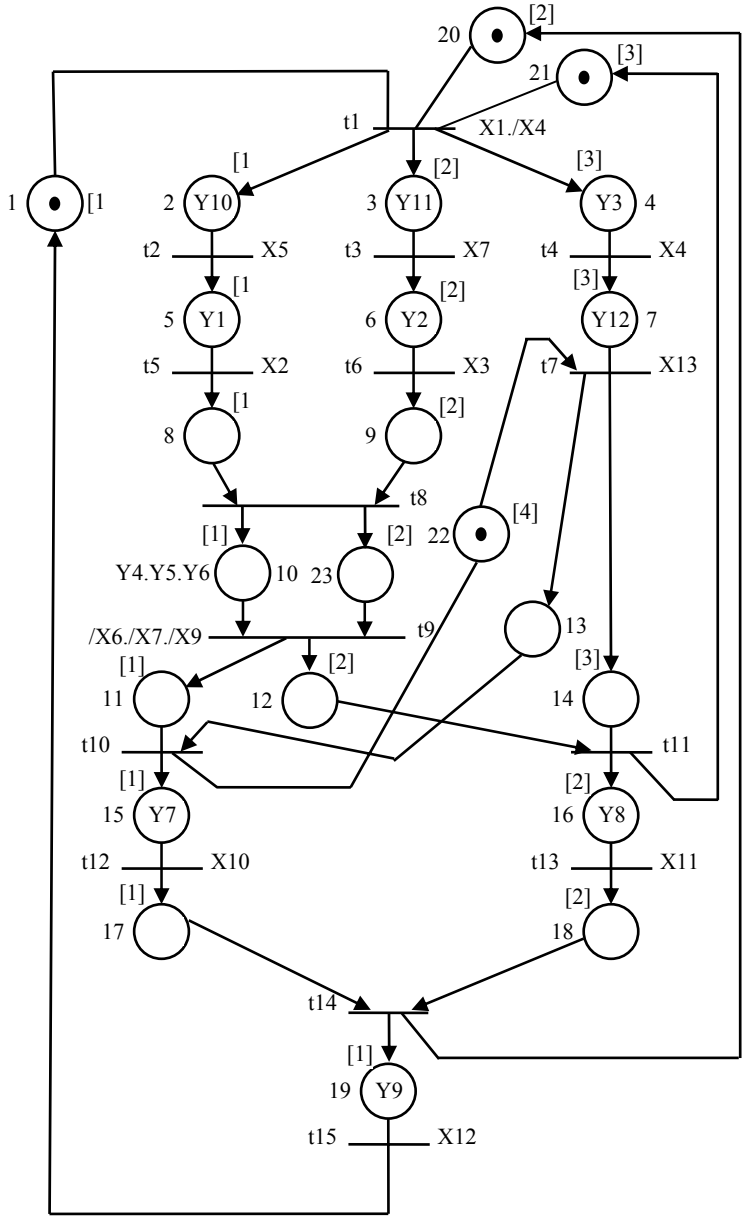
Rys. 2.9. Podsieci typu automatowego: a) podsieć opisująca zbiornik *1*, b) podsieć opisująca zbiornik *2*

Pierwsza podsieć (Rys.2.9a) opisuje działanie automatu realizującego zadania napełnienia zbiornika *1*, mieszania składników w zbiorniku *3*, napełnienia pojemnika *1* na wózku oraz odjazd wózka spod maszyny. Druga podsieć (Rys. 2.9b) opisuje działania automatu działającego analogicznie, tylko dla zbiornika *2*. Trzecia podsieć (Rys. 2.10a) opisuje załadowanie pojemników na wózek i jego podjazd pod maszynę. Aby automaty reprezentowane przez podsieci (Rys. 2.9a i Rys. 2.10a) mogły ze sobą współdziałać, konieczne było utworzenie automatu komunikacyjnego (Rys. 2.10b).



Rys. 2.10. Podsieci typu automatowego: a) podsieć opisująca działanie wózka, b) automat komunikacyjny

Na rysunku 2.11 pokazano sieć Petriego pokrytą podsieciami typu automatowego (Adamski, 1990; Banaszak i in., 1993; Blanchard, 1979).

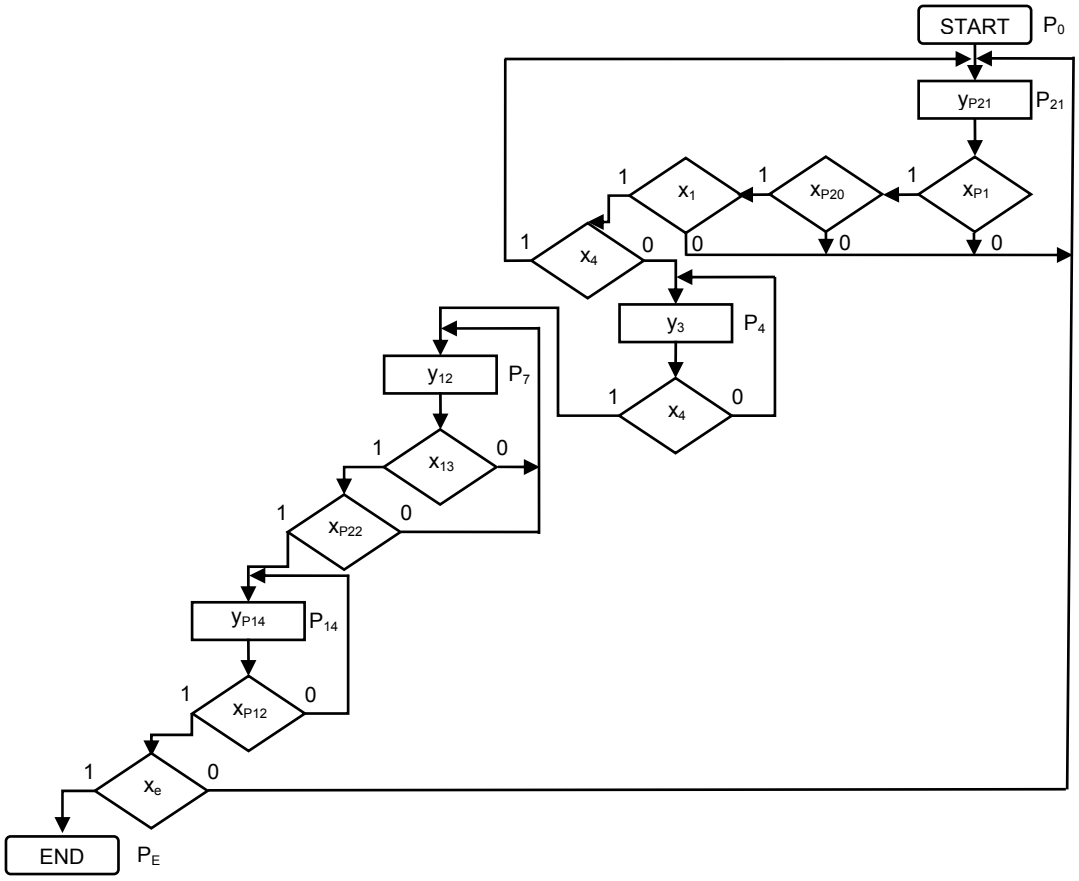


Rys. 2.11. Sieć Petriego pokryta podsieciami typu automatowego

2.7. Implementacja automatu współbieżnego jako mikroprogramowany układ sterujący

W wyniku dekompozycji automatu współbieżnego opisanego siecią Petriego (Rys. 2.6) otrzymano cztery sekwencyjne automaty składowe (Rys. 2.9, Rys. 2.10), które następnie przekształcono na odpowiadające im sieci działań. Każdy z tych automatów implementowany jest jako mikroprogramowany układ sterujący: *CMCU I* – automat realizujący napełniane zbiornika *1*; *CMCU II* - automat realizujący napełniane zbiornika *2*; *CMCU III* – automat realizujący działanie wózka; *CMCU IV* – automat komunikacyjny.

Dla potrzeb synchronizacji do układu dodano dodatkowe sygnały. Na rysunku 2.12 przedstawiono sieć działań dla podsieci typu automatowego z rysunku 2.10a.



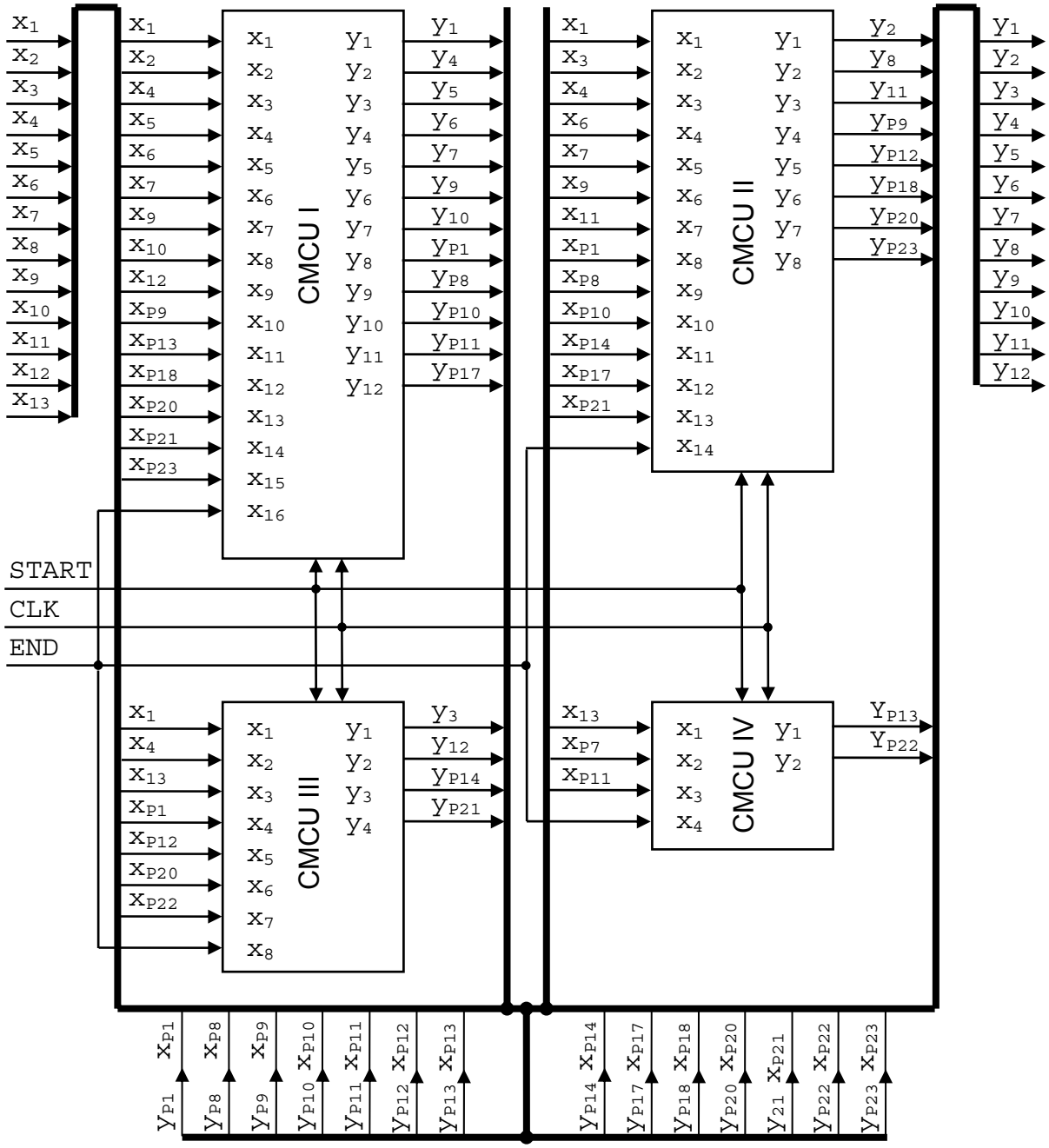
Rys. 2.12. Sieć działań

Opis warunków przejść dla sieci działań z rysunku 2.12 zamieszczono w tabeli 2.3.

Tabela 2.3. Tabela przejść sieci działań

Stan aktualny	**Warunek**	**Stan następny**	**Wyjście**
P21	$x_{P1}.x_{P20}.x_1./x_4$	P4	y_3
P21	$/x_{P1}+/x_{P20}+/x_1+x_4$	P21	y_{P21}
P4	x_4	P7	y_{12}
P4	$/x_4$	P4	y_3
P7	$x_{13}.x_{P22}$	P14	y_{P14}
P7	$/x_{13}+/x_{P22}$	P7	y_{12}
P14	x_{P12}	P21	y_{P21}
P14	$/x_{P12}$	P14	y_{P14}

Na rysunku 2.13 przedstawiono strukturę układu sterującego, implementującego powiązane liniowe sieci działań, powstałe w wyniku dekompozycji sieci Petriego z rysunku 2.6. Układ składa się z czterech mikroprogramowanych układów sterujących. Dla potrzeb synchronizacji poszczególnych sieci, do układu zostały dodane dodatkowe sygnały wewnętrzne (x_{Px}, y_{Px}).



Rys. 2.13. Struktura układu implementującego powiązane liniowe sieci działań

Pozostałe sieci działań dla podsieci typu automatowego z rysunków 2.9a, 2.9b, 2.10b oraz przykład implementacji zamieszczono w Dodatku A.

2.8. Podsumowanie

W rozdziale przedstawiono sposoby modelowania układów cyfrowych z wykorzystaniem diagramów *SFC* oraz interpretowanych automatowo sieci Petriego. Zaprezentowano przykład dekompozycji automatu współbieżnego na sekwencyjne automaty składowe implementowane jako mikroprogramowany układ sterujący.

Zarówno diagramy *SFC* jaki i sieci Petriego pozwalają w sposób jawny opisywać procesy współbieżne jak również skomplikowane sposoby koordynacji tych procesów. Zastosowanie dekompozycji sieci Petriego na sekwencyjne automaty składowe ułatwia proces projektowania oraz syntezę układu cyfrowego, przedstawionego w postaci tej sieci.

3. Metody projektowania układów sterujących

W rozdziale tym zostaną przedstawione metody projektowania układów sterujących w strukturach programowalnych. Omówione zostaną trzy sposoby syntezy układów sterujących. Pierwsze dwie metody bazują na realizacji jednopoziomowego oraz dwupoziomowego skończonego automatu stanów, trzecia to realizacja układu sterującego jako mikroprogramowany układ sterujący.

3.1. Wprowadzenie

Układ sterujący (Barkalov, 2003; Małysiak, Pochopień, 2002) jest jednym z najważniejszych elementów systemu cyfrowego. Klasyczna metoda implementacji jednostki sterującej w postaci skończonego automatu stanów (Baranov, 1994; Łuba, Zbierzchowski 2002; Łuba, 2003; Kamionka-Mikuła i in., 2004;Pochopień, 2005) często pochłania zasoby układów programowalnych (Grushnitsky i in., 2002), które mogą być w efektywniejszy sposób wykorzystane przez inne bloki projektowanego systemu. Jednym z rozwiązań tego problemu może być wykorzystanie dwupoziomowej struktury skończonego automatu stanów (ang. *Finite-State-Machine, FSM*) (Barkalov, 1994). Metoda bazuje na strukturalnej dekompozycji systemu funkcji wyjść automatu. W podejściu tym mikroinstrukcjom zostają przydzielone odpowiednie kody, zaś układ kombinacyjny automatu realizuje funkcje wzbudzeń przerzutników oraz generuje kody mikroinstrukcji. Mikroinstrukcje natomiast są dekodowane na podstawie kodu w układzie drugiego poziomu.

Innym sposobem, pozwalającym znacznie zmniejszyć ilość wykorzystanych zasobów sprzętowych jest implementacja jednostki sterującej jako mikroprogramowany układ sterujący (Barkalov, 1997; Benso i in., 2001; Bomar, 2002; Łuba, 2003). Składa się on z dwóch podstawowych jednostek: bloku zarządzającego oraz układu generującego i przechowującego mikroinstrukcje. Blok zarządzający to uproszczony skończony automat stanów, odpowiedzialny za wyznaczanie adresu mikroinstrukcji. Zbudowany jest z układu kombinacyjnego, generującego funkcję wzbudzeń oraz rejestru przechowującego wartość aktualnego stanu, w jakim znalazł się układ. Drugi blok stanowi licznik oraz pamięć, w której przechowywane są mikroinstrukcje sterownika (Barkalov, 2005).

3.2. Skończony automat stanów FSM

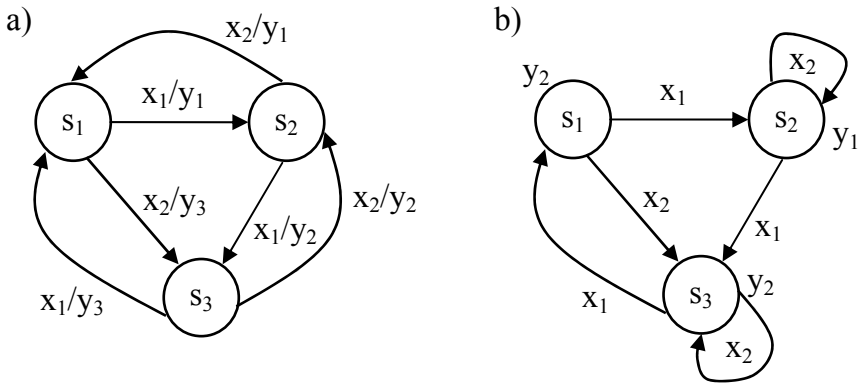
Automat skończony jest najbardziej popularnym modelem opisującym zachowanie układów sterowania (Traczyk, 1986; Adamski, 1990; Belhadj, 1993; Gajski i in., 1994; Baranov, 1994). *FSM* składa się ze zbioru stanów, zbioru przejść miedzy stanami oraz zbioru akcji przypisanych stanom lub przejściom (Definicja 3.1).

Definicja 3.1. Automat skończony

Automatem skończonym nazywana jest uporządkowana szóstka:
$FSM = \langle S, X, Y, \delta, \lambda, s_0 \rangle$, w której:

- $S = \{s_0, s_1, \ldots, s_j\}$ jest skończonym zbiorem stanów,
- $X = \{x_0, x_1, \ldots, x_k\}$ jest skończonym zbiorem wejść,
- $Y = \{y_0, y_1, \ldots, y_l\}$ jest skończonym zbiorem wyjść,
- $\delta : S \times X \rightarrow S$ jest funkcja stanu następnego,
- $\lambda : S \times X \rightarrow Y$ jest funkcją wyjścia automatu Mealy'go,
- $\lambda : S \rightarrow Y$ jest funkcją wyjścia automatu Moore'a,
- $s_0 \in S$ jest pewnym wyróżnionym stanem początkowym, od którego automat zaczyna działanie.

Ze względu na powiązanie akcji wyjściowych z przejściami lub stanami wyróżnia się dwa rodzaje automatów: Mealy'ego i Moore'a (Rys. 3.1).



Rys. 3.1. Przykład automatu skończonego: a) Mealy'ego, b) Moore'a

Realizacja sprzętowa tak zdefiniowanej maszyny *FSM*, w najprostszej postaci, polega na skojarzeniu z każdym stanem jednego przerzutnika, np. typu *D*. W takiej realizacji w danej

chwili aktywnym jest tylko jeden przerzutnik, co jest równoważne aktywności skojarzonego z nim stanu (Łabiak, 2005).

3.3. Realizacja układu sterowania jako jednopoziomowy automat stanów

Układ logiczny skończonego automatu stanu z wyjściami typu Mealy'ego jest definiowany przez system funkcji boolowskich:

$$Y = Y(X, T), \tag{3.1}$$

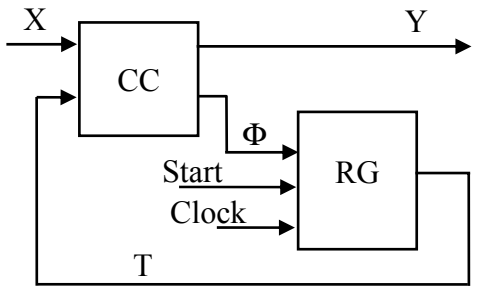
$$\Phi = \Phi(X, T). \tag{3.2}$$

Układ logiczny skończonego automatu stanu z wyjściami typu Moore'a jest definiowany przez system funkcji boolowskich:

$$Y = Y(T), \tag{3.3}$$

$$\Phi = \Phi(X, T). \tag{3.4}$$

W sytuacji takiej $X = \{x_1, \ldots, x_L\}$ jest zbiorem warunków logicznych, $Y = \{y_1, \ldots, y_N\}$ jest zbiorem mikrooperacji, $T = \{T_1, \ldots, T_R\}$ jest zbiorem zmiennych wewnętrznych, wykorzystywanych do kodowania stanów automatu $A = \{a_1, \ldots, a_M\}$, gdzie $R = \lceil log_2 M \rceil$, $\Phi = \{\Phi_1, \ldots, \Phi_R\}$ jest zbiorem funkcji wzbudzeń przerzutników, stanowiących pamięć automatu. Automat opisany w taki sposób może zostać zrealizowany przez jednopoziomowy układ cyfrowy automatu skończonego (Rys. 3.2) (Baranov, 1994; Barkalov, 2005).



Rys. 3.2. Struktura jednopoziomowego *FSM*

W układzie takim układ *CC* stanowi kombinacyjną część automatu i realizuje systemy (3.1) i (3.2) dla automatu z wyjściami typu Mealy'ego oraz (3.3) i (3.4) dla automatu Moore'a. Układ *RG* stanowi pamięć automatu i jest zbudowanych z przerzutników typu *D* (Solovjev, 2001). Systemy (3.1), (3.2) oraz (3.3) i (3.4) tworzone są na podstawie tablicy przejść i wyjść automatu (Baranov,1994).

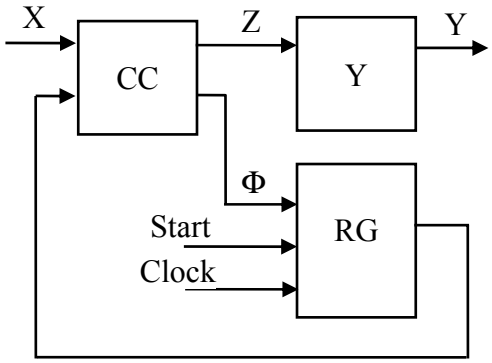
Dla automatu z wyjściami typu Mealy'ego tablica zawiera następujące kolumny: a_m, $K(a_m)$, a_s, $K(a_s)$, X_h, Y_h, Φ_h, h, gdzie $a_m \in A$ jest bieżącym stanem automatu; $K(a_m)$ jest binarnym kodem stanu a_m, $a_s \in A$ jest następnym stanem automatu; $K(a_s)$ jest binarnym kodem stanu a_s; X_h stanowi koniunkcję pewnych elementów ze zbioru X, które powodują przejście $\langle a_m, a_s \rangle$; $Y_h \subseteq Y$ jest zbiorem mikrooperacji formowanych podczas przejścia $\langle a_m, a_s \rangle$; $\Phi_h \subseteq \Phi$ jest zbiorem funkcji wzbudzeń przerzutników, które są równe *1* i powodują przełączenie pamięci automatu z $K(a_m)$ na $K(a_s)$; h jest numerem przejścia $(h = 1, \ldots, H)$.

Dla automatu z wyjściami typu Moore'a tablica przejść i wyjść zawiera kolumny: a_m, $K(a_m)$, a_s, $K(a_s)$, X_h, Y_h, Φ_h, h. Kolumna a_m zawiera zbiór $Y(a_m) \subseteq Y$.

Tak zaprojektowany układ automatu cyfrowego posiada możliwie największą wydajność w przypadku implementacji z wykorzystaniem układów *PLD*. Jednak podstawową wadą jednopoziomowych układów jest stosunkowo duża liczba funkcji, zależnych od warunków logicznych i zmiennych wewnętrznych, realizowanych przez układ kombinacyjny *CC*. Zmniejszenie liczby funkcji realizowanych przez układ *CC* doprowadzi do redukcji zasobów sprzętowych wymaganych do implementacji układu automatu cyfrowego. Jednym ze sposobów redukcji liczby funkcji jest zastosowanie dwupoziomowej struktury automatu (Baranov, 1994).

3.4. Dwupoziomowy automat stanów

Strukturę dwupoziomową skończonego automatu stanów można uzyskać poprzez zastosowanie metody maksymalnego kodowania zbiorów mikrooperacji (Baranov, 1994; Barkalov, 1994). Wyróżnijmy w tablicy przejść-wyjść automatu *Q* różnych zbiorów mikrooperacji $Y_q \subseteq Y$. Zakodujmy każdy taki zbiór $Y_q \subseteq Y$ binarnym kodem $K(Y_q)$. Do tego celu potrzebne będzie $R_1 =]log_2 Q[$ zmiennych boolowskich. Zmienne te będzie reprezentował zbiór $Z = \{z_1, \ldots z_{R_1}\}$. W takim przypadku skończony automat stanów może zostać przedstawiony jako dwupoziomowa struktura (Rys. 3.3).



Rys. 3.3. Struktura dwupoziomowego *FSM*

Struktura dwupoziomowego automatu stanów zostanie omówiona na przykładzie automatu Meale'go. W strukturze tej występują dwa układy kombinacyjne *CC* i *Y*. Układ *CC* realizuje system (3.2) oraz system:

$$Z = Z(T, X). \tag{3.5}$$

Układ *Y* dekoduje mikrooperacje, realizując system:

$$Y = Y(Z). \tag{3.6}$$

W celu realizacji systemu (3.5) należy przekształcić początkową tablicę przejść-wyjść poprzez usunięcie kolumny Y_h i wstawienie kolumny Z_h. Kolumna Z_h zawiera zmienne $z_r \in Z$, które są równe 1 w kodzie $K(Y_q)$, dla zbioru mikrooperacji Y_q znajdującego się w tym samym wierszu początkowej tablicy przejść-wyjść.

System (3.5) jest formowany na podstawie przekształconej tablicy przejść-wyjść w następujący sposób:

$$z_r = \bigvee_{h=1}^{H} C_{rh} A_m^h X_h \ (r = 1,..., R_1), \tag{3.7}$$

gdzie C_{rh} jest zmienną Boolowską, która jest równa 1, tylko jeżeli funkcja z_r jest zapisana w h-tym wierszu przekształconej tablicy przejść-wyjść automatu.

W celu realizacji systemu (3.6) należy utworzyć tablice dekodera zbiorów mikrooperacji. Tablica ta zawiera kolumny $K(Y_q)$, Y_q i q. Na podstawie tej tablicy można uformować system (3.6) w następujący sposób:

$$y_n = \bigvee_{q=1}^{Q} C_{nq} Z_q \ (n = 1,..., N), \tag{3.8}$$

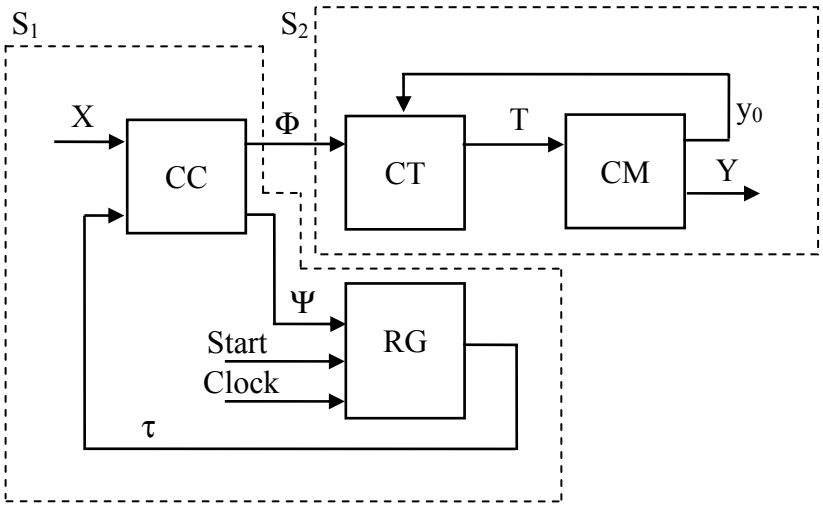
gdzie C_{nq} jest zmienną Boolowską, która jest równa 1, tylko jeżeli funkcja y_n jest zapisana w q - tym wierszu tablicy dekodera, Z_q jest koniunkcją zmiennych ze zbioru Z, odpowiadających kodowi $K(Y_q)$, $(q = 1, \ldots, Q)$.

Podejście takie zapewnia zmniejszenie liczby funkcji realizowanych przez układ kombinacyjny *CC*, co prowadzi do zmniejszenia wymaganych zasobów sprzętowych do implementacji tego układu. Ponieważ układ *Y* posiada regularną strukturę może zostać zaimplementowany z wykorzystaniem pamięci *ROM*. Kody $K(Y_q)$ były by adresem, natomiast zbiory mikrooperacji były by przechowywane jako dane. Pamięć taka musiałaby przechowywać Q N - bitowych słów.

3.5. Układy mikroprogramowane

Układy mikroprogramowane są specyficzną techniką realizacji układów sterowania, polegającą na bezpośredniej transformacji sieci działań na mikroinstrukcje tzw. mikroprogramu sterowania (Misiurewicz, 1982; Molski, 1986; Barkalov, 1997; Łuba, 2003; Kamionka-Mikuła i in., 2006).

Mikroprogramowany układ sterujący jest rozwinięciem dwupoziomowej syntezy skończonego automatu stanów (Barkalov, 2005). W tym przypadku jednostka sterująca zostaje poddana dekompozycji na dwa podstawowe bloki: układ zarządzający oraz układ pamięci, w którym przechowywane są mikroinstrukcje kontrolera (Rys. 3.4).



Rys. 3.4. Schemat blokowy mikroprogramowanego układu sterującego

W mikroprogramowanym układzie sterującym wyróżnić można cztery podstawowe bloki funkcjonalne: układ kombinacyjny *CC*, rejestr *RG*, licznik *CT* oraz pamięć *CM*. Układ kombinacyjny oraz rejestr tworzą skończony automat stanów S_1, którego zadaniem jest wyznaczenie adresu mikroinstrukcji. Licznik oraz pamięć wchodzą w skład układu mikroprogramowanego S_2, w którym adresowane są mikroinstrukcje. Rejestr *RG* przechowuje kod stanu, w jakim znalazł się automat S_1. Ilość przerzutników niezbędnych do zbudowania rejestru można wyznaczyć ze wzoru $R =]log_2 M[$, gdzie M oznacza ilość stanów automatu S_1. Zmienne τ określają kod stanu automatu S_1. Licznik *CT* jest odpowiedzialny za generowanie mikroinstrukcji z pamięci *CM*. Zmienna T reprezentuje adres mikroinstrukcji. Zmienna y_0 wykorzystywana jest do poprawnego określenia trybu adresowania (Barkalov, 1998, 2005).

Największą zaletą mikroprogramowanych układów sterujących jest zmniejszenie liczby funkcji logicznych realizowanych przez sterownik. Jest to spowodowane wykorzystaniem bloku pamięci, w którym przechowywane są mikroinstrukcje kontrolera. Dodatkową zaletą jest wykonywanie przejścia przez ścieżkę zawierająca wiele bloków warunkowych, w jednym cyklu.

3.6. Podsumowanie

W rozdziale zaprezentowano trzy sposoby realizacji układu sterującego. Pierwsza metoda bazowała na wykorzystaniu jednopoziomowej syntezy skończonego automatu stanów. System funkcji wyjść automatu został poddany dekompozycji. W efekcie uzyskano układ o strukturze dwupoziomowej, w której układ kombinacyjny automatu generuje kody mikroinstrukcji, które z kolei są dekodowane na podstawie kodu w układzie drugiego poziomu. Trzecia z przedstawionych metod to mikroprogramowany układ sterujący. W tym przypadku jednostka sterująca także została poddana dekompozycji, jednakże mikroinstrukcje przechowywane są w pamięci kontrolera. Taki sposób pozwala znacznie zmniejszyć ilość funkcji realizowanych przez układ kombinacyjny.

4. Mikroprogramowany układ sterujący

Teorie o układach mikroprogramowanych (ang. *Compositional Microprogram Control Unit, CMCU*) powstały już na początku lat 80. Ich głównym założeniem było wykorzystanie prostych układów cyfrowych, oraz pamięci *ROM* do tworzenia złożonych układów sterujących. Rozwój w dziedzinie techniki cyfrowej spowodował pojawienie się zintegrowanych układów takich jak *System-on-a-Chip* (*SoC*) czy *System-on-Programmable-Chip* (*SoPC*), w których bloki funkcjonalne projektowanego układu implementowane są z wykorzystaniem matryc programowalnych *FPGA* (Grushnitsky i in., 2002; Maxfield, 2004).

Główną cechą matryc *FPGA* jest wykorzystanie elementów *LUT* do realizacji funkcji logicznych. Ilość wejść elementu *LUT* jest ściśle ograniczona zazwyczaj do 4-6 wejść (Jenkins, 1994; Maxfield, 2004). W związku z tym konieczne jest zastosowanie dekompozycji funkcji logicznych opisujących zachowanie jednostki sterującej (Sasao, 1999; Łuba, 2003). Negatywnym następstwem funkcjonalnej dekompozycji jest powstawanie układów wielopoziomowych, czego konsekwencją będzie zmniejszenie wydajności układu w porównaniu do struktury jednopoziomowej, jak również wykorzystanie dużej ilości elementów *LUT* (Barkalov i in., 2006, 2006c; Wiśniewski, 2005).

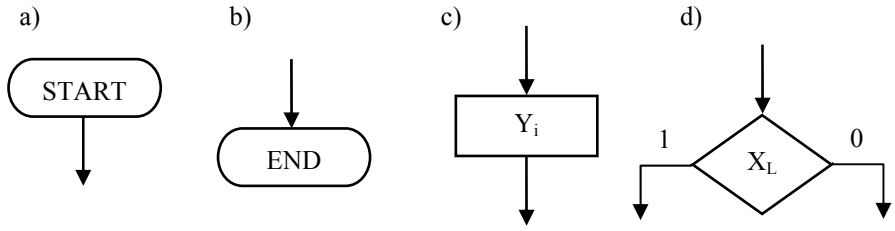
Rozwiązaniem tego problemu może być mikroprogramowany układ sterujący, w którym część sterująca realizowana jest z wykorzystaniem elementów logicznych, natomiast pamięć z wykorzystaniem dedykowanych bloków pamięci (ang. *Dedicated Memory Block, DMB*) układu *FPGA*. Wykorzystanie mikroprogramowanych układów sterujących do projektowania jednostek sterujących pozwala na implementację dużo mniejszej ilości funkcji logicznych, które są dużo prostsze, przy jednoczesnym wykorzystaniu zasobów pamięci wewnętrznej systemu *SoPC*.

W rozdziale czwartym zostaną przedstawione zagadnienia dotyczące sposobu modelowania i syntezy mikroprogramowanych układów sterujących ze szczególnym uwzględnieniem metody współdzielenia kodów. W sposób szczegółowy zostanie opisana zaproponowana metoda modelowania i syntezy mikroprogramowanego układu sterującego z konwerterem adresów. Metoda ta umożliwia zmniejszenie rozmiaru pamięci projektowanego układu. Ponadto zostaną omówione zaproponowane modyfikacje tej metody umożliwiające dalsze zmniejszenie rozmiaru pamięci układów mikroprogramowanych.

4.1. Sieć działań

Opis układów sterujących przy pomocy sieci działań (ang. *Flow Chart*) jest pierwszym krokiem w projektowaniu układów mikroprogramowanych. Specyficzną właściwością sieci działań jest wyeksponowanie stanów wejść i wyjść układów sterujących, co znacznie ułatwia realizację układu sterującego jako mikroprogramowanego (Barkalov, Węgrzyn, 2006).

Składnikami sieci działań są bloki: początkowy, końcowy, operacyjny i warunkowy (Rys. 4.1).



Rys. 4.1. Elementy sieci działań - bloki: a) początkowy, b) końcowy, c) operacyjny, d) warunkowy

W sieci działań przepływ sterowania odbywa się po drodze od bloku początkowego przez bloki operacyjne i warunkowe. Bloki operacyjne opisują stan wyjść układu sterującego, zaś bloki warunkowe analizują stan jego wejść. Symbole sygnałów wyjściowych Y_i wpisane w blokach operacyjnych nazywane są mikroinstrukcjami.

W rozprawie, do opisu algorytmów sterowania, wykorzystano liniową sieć działań (Barkalov i in., 2006d).

Definicja 4.1. Liniowa sieć działań

Liniową siecią działań nazywamy sieć zawierającą nie mniej niż 75% bloków operacyjnych.

Przy syntezie mikroprogramowanego układu sterującego na podstawie danej sieci działań istotne jest właściwe przyporządkowanie stanów wewnętrznych układu sterowania poszczególnym segmentom sieci działań. Przyporządkowanie to implikuje jednocześnie podział sieci działań na mikroinstrukcje, tj. takie jej segmenty, które są wykonywane w jednym, elementarnym takcie pracy układu sterującego (Łuba, 2003).

4.2. Wprowadzenie teoretyczne i podstawowe definicje

Niech dany będzie algorytm sterowania układu cyfrowego przedstawiony jako sieć działań Γ (ang. *Flow-Chart of Algorithm, FCA*) (Baranov, 1994) składający się ze zbioru bloków $B = \{b_0, b_E\} \cup B_1 \cup B_2$ i zbioru połączeń $E = \{ (b_i, b_j) \mid b_i, b_j \in B \}$, gdzie: b_0 jest blokiem początkowym, b_E jest blokiem końcowym, B_1 jest zbiorem bloków operacyjnych zawierających kolekcję mikrooperacji (mikroinstrukcji) $Y_q \subseteq Y$, gdzie $Y = \{y_1, \ \ldots, \ y_N\}$, B_2 jest zbiorem bloków warunkowych z elementami zbioru warunków logicznych $X = \{x_1, \ \ldots, \ x_L\}$. Wprowadzamy następujące definicje (Barkalov, 2002).

Definicja 4.2. Łańcuch bloków operacyjnych

Łańcuch bloków operacyjnych (ang. *Operational Linear Chain, OLC*) w sieci działań Γ to skończona sekwencja bloków operacyjnych $\alpha_g = \langle b_{g1}, \ \ldots, \ b_{gF_g} \rangle$ takich, że dla każdej pary sąsiadujących bloków wektora α_g istnieje połączenie $\langle b_{gi}, b_{gi+1} \rangle \in E \ \ (i = 1, \ldots, F_g - 1)$.

Definicja 4.3. Wejście łańcucha

Wejściem łańcucha α_g jest blok $b_q \in B_1$, dla którego istnieje połączenie $\langle b_t, \ b_q \rangle \in E$, gdzie $b_t = b_0$ lub $b_t \in B_2$ lub $b_t \notin D^g$, gdzie $D^g \subseteq B_1$ jest zbiorem bloków z łańcucha α_g.

Definicja 4.4. Wyjście łańcucha

Wyjściem łańcucha α_g jest blok $b_q \in B_1$, dla którego istnieje połączenie $\langle b_q, \ b_t \rangle \in E$, gdzie $b_t = b_E$ lub $b_t \in B_2$ lub $b_t \notin D^g$.

Definicja 4.5. Wejście główne łańcucha

Wejściem głównym łańcucha (ang. *Main Input, MI*) nazywamy wejście $b_q \in B_1$, jeżeli jego wejście nie jest połączone z żadnym wyjściem innego bloku operacyjnego.

Definicja 4.6. Łańcuchy pseudoekwiwalentne

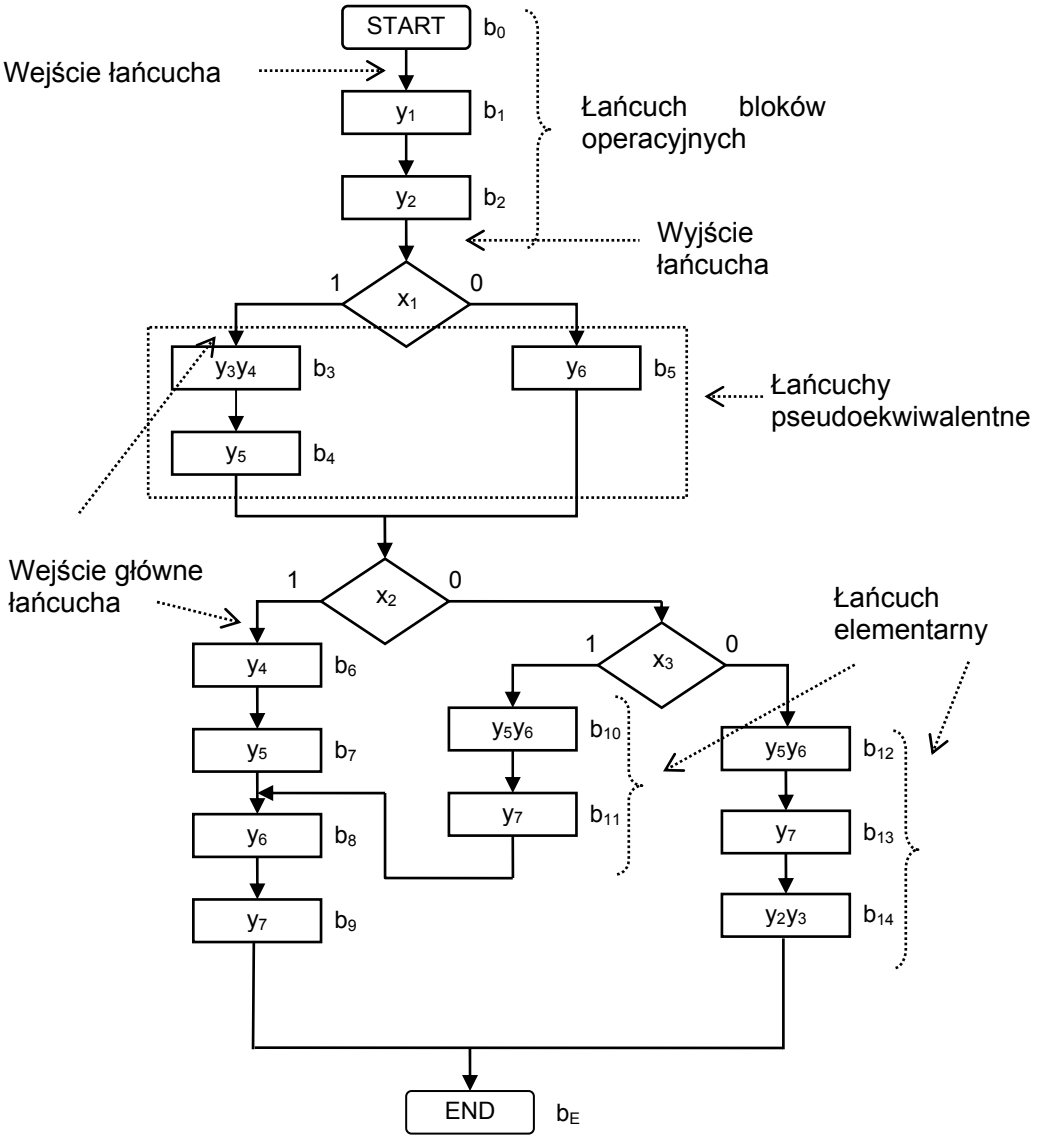
Łańcuchy bloków operacyjnych α_i, α_j nazywane są pseudoekwiwalentnymi, jeżeli ich wyjścia są połączone z wejściem tego samego bloku $b_q \in B$ sieci działań Γ.

Definicja 4.7. Łańcuch elementarny

Łańcuchem elementarnym (ang. *Elementary Operational Linear Chain, EOLC*) nazywamy łańcuch mający tylko jedno wejście.

Dowolny łańcuch α_g, za wyjątkiem łańcuchów elementarnych, może mieć więcej niż jedno wejście. Poprzez I_g^j będziemy oznaczać j-te wejście łańcucha α_g. Dowolny łańcuch ma tylko jedno wyjście O_g, które jest elementem zbioru wyjść $O(\Gamma)$.

Na rysunku 4.2 zaprezentowano elementy sieci działań z definicji 4.2, 4.3, 4.4, 4.5, 4.6 oraz 4.7.



Rys. 4.2. Elementy sieci działań

4.3. Mikroprogramowany układ sterujący ze współdzieleniem kodów

Niech zbiór $C = \{\alpha_1, \ \ldots, \ \alpha_G\}$ będzie zbiorem wszystkich łańcuchów sieci działań Γ spełniających warunek:

$$\begin{aligned} &\left|D^i \cap D^j\right| = 0 \quad (i, j \in \{1, \ \ldots, \ G\}, i \neq j); \\ &B_1 = D^1 \cup D^2 \cup \ldots \cup D^G; \\ &G \rightarrow min. \end{aligned} \tag{4.1}$$

Niech F_g będzie ilością elementów łańcucha $\alpha_g \in C$ i niech $F_{max} = max(F_1, \ \ldots, \ F_G)$. Zakodujmy łańcuch $\alpha_g \in C$ kodem $K(\alpha_g)$ na R_1 bitach, gdzie

$$R_1 =]log_2 G[. \tag{4.2}$$

Zakodujmy elementy łańcucha $\alpha_g \in C$ kodem $K(b_t)$ na R_2 bitach, gdzie

$$R_2 =]log_2 F_{max}[. \tag{4.3}$$

Niech kodowanie zostanie wykonane w taki sposób, że warunek:

$$K(b_{gi+1}) = K(b_{gi}) + 1 \ (g = 1, \ \ldots, \ G) \tag{4.4}$$

będzie spełniony dla każdej pary sąsiadujących elementów $b_{gi}, b_{gi+1} \ (i = 1, \ \ldots, \ F_g - 1)$ łańcucha $\alpha_g \in C$.

W takim przypadku adres $A(b_t)$ mikroinstrukcji $Y(b_t)$ od bloku $b_t \in B_1$, gdzie $b_t \in D^g$, jest reprezentowany jako:

$$A(b_t) = K(\alpha_g) * K(b_t), \tag{4.5}$$

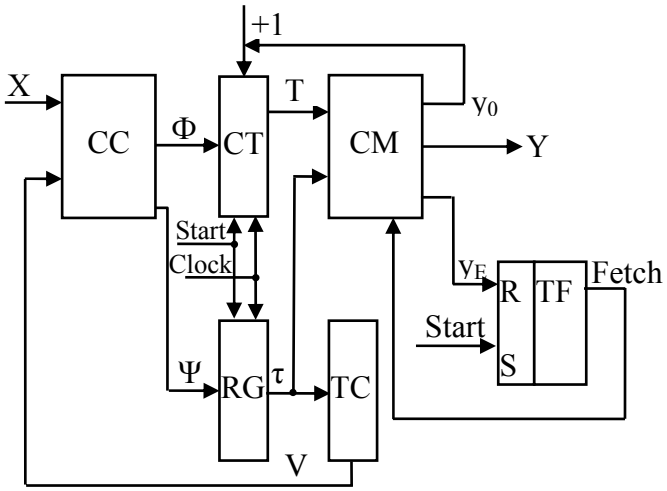
gdzie $*$ jest znakiem łączenia. Reprezentacja (4.5) adresu mikroinstrukcji nazywana jest współdzieleniem kodów (Barkalov, 2002).

Dokonajmy podziału zbioru łańcuchów C na klasy łańcuchów pseudoekwiwalentnych $\Pi_C = \{B_1, \ldots B_I\}$. Zakodujmy każdą klasę $B_i \in \Pi_C$ kodem $K(B_i)$ na R_0 bitach, gdzie

$$R_0 =]log_2 I[. \tag{4.6}$$

Do kodowania zastosujmy zmienne ze zbioru $V = \{v_1, \ldots v_{R_0}\}$.

W takim przypadku sieć działań opisująca algorytm sterowania układu cyfrowego może zostać zrealizowana za pomocą mikroprogramowanego układu sterującego ze współdzieleniem kodów (ang. *Compositional Microprogram Control Unit with Codes Sharing*, CMCU_CS). Na rysunku 4.3 przedstawiono strukturę mikroprogramowanego układu *CMCU_CS,* która bazuje na reprezentacji adresu (4.5) oraz przekształceniu kodu łańcucha $\alpha_g \in C$ w kody klas łańcuchów pseudoekwiwalentnych.



Rys. 4.3. Struktura mikroprogramowanego układu *CMCU_CS*

Układ kombinacyjny *CC* implementuje funkcje wzbudzeń przerzutników licznika *CT*

$$\Phi = \Phi(V, X) \tag{4.7}$$

oraz rejestru *RG*

$$\psi = \psi(V, X). \tag{4.8}$$

Oznacza to, że układ kombinacyjny *CC* reprezentuje cześć mikroprogramowanego układu sterującego odpowiedzialną za wyznaczanie adresu mikroinstrukcji. Licznik *CT* przechowuje kod $K(b_t)$ elementu łańcucha, który jest reprezentowany poprzez wewnętrzne zmienne $T_r \in T$, gdzie $|T| = R_2$. Rejestr *RG* przechowuje kod $K(\alpha_g)$ łańcucha $\alpha_g \in C$, który

jest reprezentowany przez wewnętrzne zmienne $\tau_r \in \tau$, gdzie $|\tau| = R_1$. Konwerter kodu *TC* implementuje funkcje

$$V = V(\tau) \tag{4.9}$$

oraz generuje kody klas $B_i \in \prod_c$. Pamięć *CM* przechowuje mikrooperację $y_n \in Y$, zmienną synchronizującą y_0 oraz zmienną y_E służącą do sterowania pobieraniem mikroinstrukcji z pamięci. Przerzutnik *TF* wyznacza sygnał *Fetch*, co umożliwia pobranie mikroinstrukcji z pamięci *CM*.

Mikroprogramowany układ *CMCU_CS* działa w następujący sposób: w chwili aktywacji układu (aktywny impuls *Start*), do licznika *CT* oraz rejestru *RG* wczytywany jest kod zero, który odpowiada adresowi pierwszej mikroinstrukcji realizowanego algorytmu. W tym samym czasie ustawiany jest przerzutnik *TF* i sygnał *Fetch=1* umożliwia pobranie mikroinstrukcji z pamięci *CM*. Bieżąca mikroinstrukcja $Y(b_q)$ zostaje odczytana z pamięci. Jeżeli kolejny blok w sieci działań należy do tego samego łańcucha $\alpha_g \in C$, to zmienna $y_0 = 1$. Oznacza to zwiększenie wartości licznika ($CT = CT + 1$), natomiast zawartość rejestru *RG* nie ulega zmianie. Taka sytuacja odpowiada adresowaniu kolejnego elementu bieżącego łańcucha $\alpha_g \in C$. Jeżeli zostanie osiągnięte wyjście łańcucha $b_q = O_g$, to $y_0 = 0$. Wówczas do rejestru *RG* ładowana jest wartość funkcji Ψ określająca kod następnego łańcucha, a do licznika *CT* wartość funkcji Φ określająca kod jego wejścia.

Zawartość licznika *CT* oraz rejestru *RG* zostaje zmieniona za pomocą impulsu synchronizującego *Clock*. Jeżeli osiągnięty został adres $A(b_t)$ taki, że $\langle b_t, b_E \rangle \in E$, wówczas wewnętrzna zmienna $y_E = 1$, co powoduje, zresetowanie przerzutnika *TF* (*Fetch=0*) a tym samym zakończenie działania mikroprogramowanego układu sterującego.

Taka organizacja jednostki sterującej umożliwia wykorzystanie minimalnej ilości sygnałów sprzężenia zwrotnego, w porównaniu z wszystkimi znanymi strukturami mikroprogramowanych układów sterujących (Barkalov, 2002; Barkalov, Węgrzyn, 2006).

Lecz jeśli zachodzi warunek

$$R_1 + R_2 > R, \tag{4.10}$$

w którym parametr *R* określony jest poprzez ilości bloków operacyjnych $M = |B_1|$ w taki sposób, że

$$R =]log_2 M[, \tag{4.11}$$

wówczas rozmiar pamięci *CM* zwiększy się w porównaniu z wartością

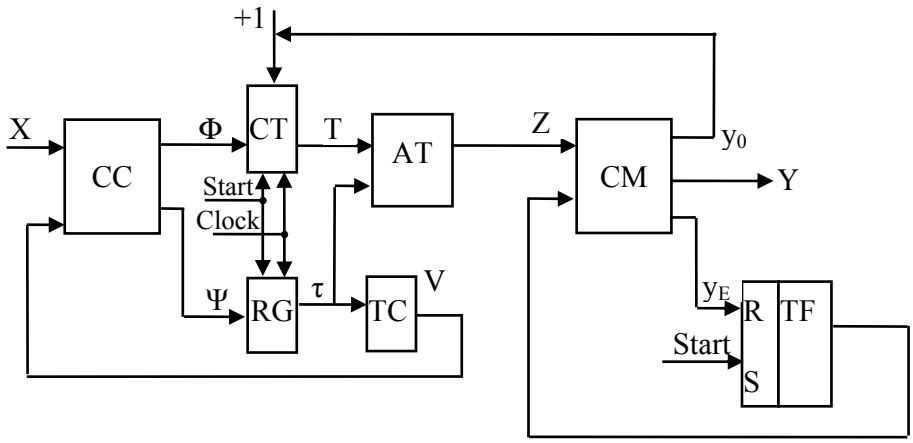
$$V_0 = 2^R * N_0, \tag{4.12}$$

gdzie N_0 jest ilością bitów reprezentujących mikrooperację $y_n \in Y$ oraz zmienne wewnętrzne y_0 i y_E. Taki rozmiar pamięci *CM* charakteryzuje dobrze znane struktury mikroprogramowanych układów sterujących (Barkalov, 2002; Barkalov, Wiśniewski, 2005a). Jeżeli warunek (4.10) zostanie spełniony, ilość dedykowanych bloków pamięci potrzebnych do zaimplementowania pamięci *CM* zwiększy się.

W rozprawie zaproponowano metody organizacji mikroprogramowanego układu sterującego ze współdzieleniem kodów, które umożliwią zmniejszenie rozmiaru pamięci *CM* w porównaniu z wartością (4.12) dla algorytmów sterowania opisanych liniową siecią działań.

4.4. Mikroprogramowany układ sterujący z konwerterem adresów

Zaproponowana metoda bazuje na wprowadzeniu bloku konwertera adresów do mikroprogramowanego układu o strukturze *CMCU_CS*. Konwerter adresów przekształca adres (4.5) na adres o minimalnej pojemności bitowej *R*. Wprowadzenie bloku konwertera do układu sterującego ze współdzieleniem kodów prowadzi do utworzenia mikroprogramowanego układu sterującego o strukturze *CMCU_AT* (ang. *Compositional Microprogram Control Unit with Address Transformer, CMCU_AT*) (Rys. 4.4).



Rys. 4.4. Struktura mikroprogramowanego układu *CMCU_AT*

Konwerter adresów generuje funkcje

$$Z = Z(T, \tau), \tag{4.13}$$

które są używane do adresowania mikroinstrukcji, $Z = \{z_1, \ \ldots, \ z_R\}$. Zasada działania mikroprogramowanych układów sterujących *CMCU_CS* oraz *CMCU_AT* jest taka sama.

Metoda projektowania układu o strukturze *CMCU_AT* składa się z następujących kroków:

1. *Przekształcenie początkowej sieci działań.*

   Przekształcenie to ogranicza się do dodania dodatkowych bloków operacyjnych do początkowej sieci działań oraz dodatkowych zmiennych wewnętrznych do bloków tej sieci, w taki sposób, że:

   - jeżeli dane jest połączenie $\langle b_0, b_q \rangle \in E$, gdzie $b_q \in B_2$, wówczas do sieci działań Γ dodawany jest dodatkowy pusty blok $b_t \in B_1$, a połączenie $\langle b_0, b_q \rangle$ jest zastępowane przez połączenia $\langle b_0, b_t \rangle$ i $\langle b_t, b_q \rangle$;
   - jeżeli dane jest połączenie $\langle b_q, b_E \rangle \in E$, gdzie $b_q \in B_1$, wówczas zmienna wewnętrzna y_E jest dodawana do bloku b_q;
   - jeżeli dane jest połączenie $\langle b_q, b_E \rangle \in E$, gdzie $b_q \in B_2$, to blok $b_t \in B_1$ z wewnętrzną zmienną y_E, jest dodawany do sieci działań Γ, a połączenie $\langle \mathrm{b_q}, \mathrm{b_E} \rangle$ jest zastępowane przez połączenia $\langle \mathrm{b_q}, \mathrm{b_t} \rangle$ i $\langle \mathrm{b_t}, \mathrm{b_E} \rangle$.

2. *Utworzenie zbioru łańcuchów bloków operacyjnych dla przekształconej sieci działań.*

   Pierwszym etapem tworzenia zbioru łańcuchów jest wyznaczenie zbioru wejść głównych *MI(Γ)* sieci działań zgodnie z definicją 4.5. Szczegółowy opis metody wyznaczania zbioru łańcuchów operacyjnych znajduje się w pracy (Barkalov, 2002).

3. *Zakodowanie łańcuchów bloków operacyjnych.*

   Poszczególne łańcuchy bloków operacyjnych $\alpha_g \in C$ kodowane są z wykorzystaniem naturalnego kodu binarnego (*NKB*) kodem $K(\alpha_g)$ na R_1 bitach, gdzie R_1 wyznaczane jest zgodnie ze wzorem (4.2).

4. *Zakodowanie elementów łańcuchów.*

Niech pierwszy element każdego łańcucha $\alpha_g \in C$ zawiera kod zero. Kody następnych elementów łańcucha wyznaczane są zgodnie z definicją 4.2 i są kodowane kodem $K(b_t)$ na R_2 bitach, gdzie R_2 obliczane jest na podstawie wzoru (4.3).

5. *Adresowanie mikroinstrukcji.*

Adresy mikroinstrukcji kodowane są z wykorzystaniem naturalnego kodu binarnego (*NKB)* kodem $C(b_t)$ na R bitach. Wartość R wyznaczana jest zgonie ze wzorem (4.11).

6. *Wyznaczenie zawartości pamięci.*

Wyznaczenie zawartości pamięci polega na utworzeniu wektora mikroinstrukcji $\alpha = \alpha_1 * \ldots * \alpha_G$ gdzie * jest znakiem łączenia. Następnie należy ponumerować elementy wektora α liczbami od *0* do $|B_1| - 1$.

Adresem każdej mikroinstrukcji jest binarnie zgodny z (4.11) R-bitowy odpowiednik nadanego numeru. Do wszystkich bloków, które nie są blokami wyjściowymi należy dodać zmienną wewnętrzną y_0, a do bloków połączonych z elementem wyjściowym sieci działań dodaje się zmienną wewnętrzną y_E. Zawartość pamięci reprezentowana jest poprzez tabelę o następujących kolumnach $C(b_t)$, $Y(b_t)$, b_t, gdzie $C(b_t)$ – jest adresem mikroinstrukcji, $Y(b_t)$ – jest mikroinstrukcją, b_t – jest blokiem operacyjnym zawierającym mikroinstrukcję.

7. *Zakodowanie klas łańcuchów pseudoekwiwalentnych.*

Pierwszym etapem kodowania łańcuchów pseudoekwiwalentnych jest wyznaczenie ich zbioru zgodnie z definicją 4.6. Klasy łańcuchów pseudoekwiwalentnych kodowane są z wykorzystaniem naturalnego kodu binarnego (*NKB*) kodem $K(B_i)$ na R_0 bitach, gdzie R_0 wyznaczane jest zgodnie ze wzorem (4.6).

8. *Utworzenie tabeli przejść mikroprogramowanego układu sterującego.*

Utworzenie tabeli przejść mikroprogramowanego układu sterującego polega na wyznaczeniu formuł przejść dla klas łańcuchów pseudoekwiwalentnych.

Niech $\Pi_C' \subset \Pi_C$ zawiera klasy $B_i \in \Pi_C$ takie, że wyjścia łańcuchów $\alpha_g \in B_i$ nie zawierają wewnętrznej zmiennej y_E. Wyrażenie

$$B_i \to \bigvee X_i(I_g^j) I_g^j \tag{4.14}$$

nazywane jest uogólnioną formułą przejść (ang. G*eneralized Formula of Transition, GFT*), w której $X_i(I_g^j)$ jest koniunkcją warunków logicznych $x_l \in X$, która określa przejście z wyjścia każdego łańcucha $\alpha_g \in B_i$ do wejścia I_g^j (Baranov, 1994). Tabela przejść jest podstawą wyznaczenia funkcji (4.7)-(4.8).

Tabela przejść zawiera następujące kolumny: B_i, $K(B_i)$, I_g^j, $A(I_g^j)$, X_h, Φ_h, Ψ_h, h, gdzie adres $A(I_g^j)$ wejścia I_g^j $(g = 1, \ldots G;\ j = 1, \ldots, F_g)$ jest reprezentowany zgodnie z (4.5); Φ_h jest zbiorem funkcji wzbudzeń przerzutników licznika *CT*, które gdy są równe *1* ładują do licznika *CT* kod elementu $K(b_t)$ gdzie $b_t \in B_1$ odpowiada wejściu I_g^j; Ψ_h jest zbiorem funkcji wzbudzeń przerzutników rejestru *RG*, które gdy są równe *1* ładują do rejestru *RG* kod łańcucha $\alpha_g \in C$; $h = 1, \ldots, H$ jest numerem przejścia, gdzie *H* równe jest ilości członów równania systemu (4.14).

9. *Utworzenie tabeli konwertera adresów.*

Tabela konwertera adresów jest podstawą utworzenia funkcji (4.13) i składa się z następujących kolumn: b_m, $A(b_m)$, $Y(b_m)$, $C(b_m)$, Z_m, m, gdzie Z_m jest zbiorem zmiennych $z_r \in Z$, które są równe 1 w adresie $C(b_m)$ mikroinstrukcji $Y(b_m)$ odpowiadającej blokowi $b_m \in B_1$.

10. *Utworzenie tabeli konwertera kodu.*

Tabela konwertera kodu jest podstawą utworzenia funkcji (4.9) i składa się z następujących kolumn: α_g, $K(\alpha_g)$, B_i, $K(B_i)$, V_g, g, gdzie V_g jest zbiorem zmiennych $v_r \in V$, które są równe 1 w kodzie $K(B_i)$ gdzie $\alpha_g \in B_i$ i $B_i \in \Pi_C'$.

11. *Wyznaczenie funkcji Φ, Ψ, Z, V.*

Funkcje $D_r \in \Phi \cup \Psi$ wyznaczane są na podstawie tabeli przejść, według wzoru:

$$P_h = \left(\bigwedge_{r=1}^{R_0} v_r^{l_{rh}} \right) * X_h \ (h = 1, \ldots, H\ ). \tag{4.15}$$

Pierwsza koniunkcja w formule (4.15) odpowiada kodowi $K(B_i)$ klasy $B_i \in \Pi_C'$ z h- tego wiersza tabeli przejść, $l_{rh} \in \{0, 1\}$ jest wartością r–tego bitu tego kodu, $v_r^0 = \overline{v_r}$, $v_r^1 = v_r$, $(r = 1, \ldots, R_0)$. Funkcje $D_r \in \Phi \cup \Psi$ ustalane są według wzoru:

$$D_r = \bigvee_{h=1}^{H} C_{rh} P_h \ (r = 1, \ldots R_1 + R_2), \tag{4.16}$$

gdzie C_{rh} jest zmienną Boolowską, która jest równa *1* wtedy i tylko wtedy gdy zmienna D_r $(r = 1, \ldots R_1 + R_2)$ jest zapisana w h–tym wierszu tabeli przejść *(h=1,...,H)*.

Funkcje *Z* wyznaczane są na podstawie tabeli konwertera adresów, według wzoru:

$$z_r = \bigvee_{m=1}^{M} C_{rm} \left(\bigwedge_{r=1}^{R_1} \tau_r^{l_{rm}} \right) * \left(\bigwedge_{r=1}^{R_2} T_r^{E_{rm}} \right) (r = 1, \ldots, R), \tag{4.17}$$

gdzie C_{rm} jest zmienną Boolowską, która jest równa *1* wtedy i tylko wtedy gdy zmienna z_r $(r = 1, \ldots, R)$ jest zapisana w m-tym wierszu tabeli konwertera adresów $(m = 1, \ldots, M)$; $l_{rm}, E_{rm} \in \{0, 1\}$ są wartościami r–tego bitu w kodach $K(\alpha_g)$, $K(b_t)$, $\tau_r^0 = \overline{\tau_r}$, $\tau_r^1 = \tau_r$ $(r = 1, \ldots, R_1)$, $T_r^0 = \overline{T_r}$, $T_r^1 = T_r$ $(r = 1, \ldots, R_2)$.

Funkcja (4.17) ma

$$M_0 = 2^{R_1 + R_2} - M \tag{4.18}$$

niewykorzystanych stanów na wejściu, które mogą być wykorzystane do optymalizacji tych funkcji (De Micheli, 1994).

Funkcje *V* tworzone są na podstawie tabeli konwertera kodu zgodnie ze wzorem:

$$v_r = \bigvee_{g=1}^{G_0} C_{rg} \left(\bigwedge_{r=1}^{R_1} \tau_r^{l_{rg}} \right) (r = 1, \ldots, R_0), \tag{4.19}$$

gdzie C_{rg} jest zmienną Boolowską, która jest równa *1* wtedy i tylko wtedy gdy funkcja $v_r \in V$ jest zapisana w g–tym wierszu tabeli $(g = 1, \ldots, G_0)$; $l_{rm} \in \{0, 1\}$ jest wartością r-tego bitu kodu $K(\alpha_g)$ z g–tego wiersza tej tabeli, $\tau_r^0 = \overline{\tau_r}$, $\tau_r^1 = \tau_r$ $(r = 1, \ldots, R_1)$.

Funkcje $y_n \in Y$ oraz wewnętrzne zmienne y_0, y_E są tworzone na podstawie tabeli wyznaczającej zawartość pamięci. Zauważmy, że najbardziej naturalnym sposobem implementacji pamięci *CM* i konwertera kodu *TC* jest zastosowanie dedykowanych bloków pamięci.

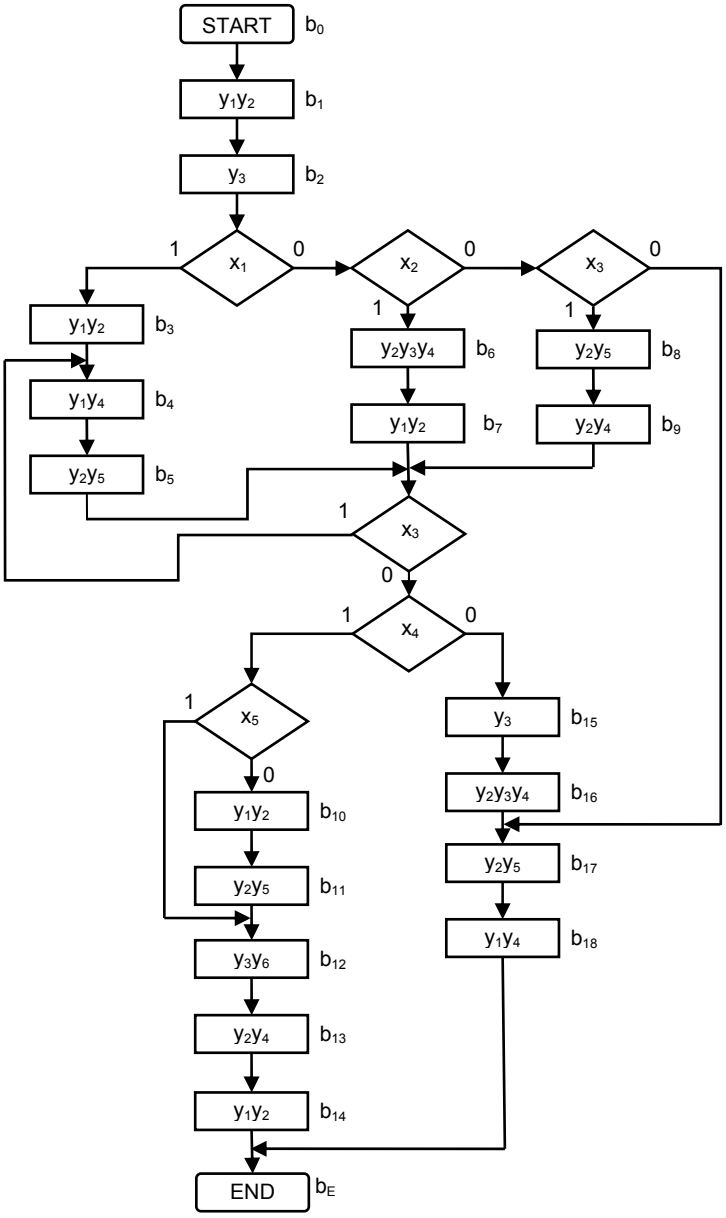
12. Implementacja mikroprogramowanego układu sterującego.

Ostatni etap to implementacja układu. Polega ona na zakodowaniu otrzymanych w procesie projektowania funkcji wzbudzeń opisujących układ w matrycach programowalnych. W chwili obecnej na rynku dostępne są układy producentów takich jak *Xilinx*, *Altera*, *Lattice*, *Actel*. Niektóre z nich zawierają w swojej strukturze dedykowane bloki pamięci (*Xilinx – Block RAM*, *Altera – M9K, M144K*). W przypadku zastosowania matryc *FPGA* część logiczna układu może zostać zaimplementowana z wykorzystaniem elementów *LUT*. Natomiast pamięć mikroprogramowanego układu sterującego może być zaimplementowana przy użyciu dedykowanych bloków pamięci. Partycjonowanie bloków układu mikroprogramowanego pomiędzy logikę rozproszoną (elementy *LUT*) a dedykowane bloki pamięci dokonywane jest automatycznie przez oprogramowanie do implementacji dostarczane przez producentów układów scalonych, na podstawie charakterystycznych konstrukcji języka *VHDL* (Chang, 1999; http1; http2; http3).

Metody implementacji z wykorzystaniem *CPLD*, *FPGA*, *PROM* opisano w pracach (Jenkins, 1994; Salcic, Samailagic, 1997; Salcic, 1998; Kania, 2004; Wiśniewski, 2005).

4.4.1. Synteza mikroprogramowanego układu sterującego CMCU_AT

Metoda syntezy mikroprogramowanego układu sterującego *CMCU_AT* zostanie omówiona na przykładzie sieci działań Γ_1 przedstawionej na rysunku 4.5.



Rys. 4.5. Początkowa sieć działań Γ_1

1. Przekształcenie początkowej sieci działań.

Dla rozpatrywanej sieci działań Γ_1 (Rys. 4.5) przekształcenie ogranicza się do dodania wewnętrznej zmiennej y_E do bloków b_{14} oraz b_{18}. Natomiast struktura sieci działań pozostaje niezmieniona.

2. *Utworzenie zbioru łańcuchów.*

Dla sieci działań Γ_1 utworzono następujący zbiór wejść głównych łańcuchów $MI = \{b_1, b_3, b_6, b_8, b_{10}, b_{15}\}$ oraz zbiór łańcuchów $C = \{\alpha_1, \ \ldots, \ \alpha_6\}$, gdzie $\alpha_1 = \langle b_1, b_2 \rangle$, $I_1^1 = b_1$, $O_1 = b_2$; $\alpha_2 = \langle b_3, b_4, b_5 \rangle$, $I_2^1 = b_3$, $I_2^2 = b_4$, $O_2 = b_5$; $\alpha_3 = \langle b_6, b_7 \rangle$, $I_3^1 = b_6$, $O_3 = b_7$; $\alpha_4 = \langle b_8, b_9 \rangle$, $I_4^1 = b_8$, $O_4 = b_9$; $\alpha_5 = \langle b_{10}, \ \ldots, \ b_{14} \rangle$, $I_5^1 = b_{10}$, $I_5^2 = b_{12}$, $O_5 = b_{14}$; $\alpha_6 = \langle b_{15}, \ \ldots, \ b_{18} \rangle$, $I_6^1 = b_{15}$, $I_6^2 = b_{17}$, $O_6 = b_{18}$.

W wyniku analizy zbioru łańcuchów otrzymano: $G = 6$, $F_{max} = 5$, $O(\Gamma_1) = \{b_2, b_5, b_7, b_9, b_{14}, b_{18}\}$. Do bloków operacyjnych $b_q \notin O(\Gamma)$ została dodana wewnętrzna zmienna y_0.

W omawianym przykładzie $R_1 = 3$, $R_2 = 3$, $M = 18$, $R = 5$ tak, więc warunek (4.10) został spełniony, przez co uzasadnione jest zastosowanie proponowanej metody.

3. *Zakodowanie łańcuchów.*

Łańcuchy *CMCU_AT* zakodowano w następujący trywialny sposób: $K(\alpha_1) = 000$, $K(\alpha_2) = 001, \ldots, K(\alpha_6) = 101$.

4. *Zakodowanie elementów łańcuchów.*

Dla *CMCU_AT* otrzymano następujące kody elementów łańcuchów: $K(b_1) = K(b_3) = K(b_6) = K(b_8) = K(b_{10}) = K(b_{15}) = 000$, $K(b_2) = K(b_4) = K(b_7) = K(b_9) = K(b_{11}) = K(b_{16}) = 001$, $K(b_5) = K(b_{12}) = K(b_{17}) = 010$, $K(b_{13}) = K(b_{18}) = 011$, $K(b_{14}) = 100$.

Kody łańcuchów $\alpha_g \in C$ oraz ich elementów umożliwiają znalezienie adresu (4.5). W tabeli 4.1 zaprezentowano adresy $A(b_t)$ mikroinstrukcji dla mikroprogramowanego układu sterującego o strukturze *CMCU_CS*.

Tabela 4.1. Adresy mikroinstrukcji dla układu *CMCU_CS*

b_t	$A(b_t)$	b_t	$A(b_t)$	b_t	$A(b_t)$
b_1	000000	b_7	010001	b_{13}	100011
b_2	000001	b_8	011000	b_{14}	100100
b_3	001000	b_9	011001	b_{15}	101000
b_4	001001	b_{10}	100000	b_{16}	101001
b_5	001010	b_{11}	100001	b_{17}	101010
b_6	010000	b_{12}	100010	b_{18}	101011

Pierwsze trzy pozycje adresu $A(b_t)$ odpowiadają kodowi łańcucha $K(\alpha_g)$ kolejne trzy bity – kodowi jego elementu.

5. *Adresowanie mikroinstrukcji.*

Adresy mikroinstrukcji dla sieci działań Γ_1 zakodowano na $R=5$ bitach w następujący sposób: $C(b_1) = 00000$, $C(b_2) = 00001$, …, $C(b_{18}) = 10001$.

6. *Wyznaczenie zawartości pamięci.*

Dla mikroprogramowanego układu sterującego z konwerterem adresów tabela reprezentująca zawartość pamięci ma $M = 18$ wierszy (Tabela 4.2).

Tabela 4.2. Adresy mikroinstrukcji *CMCU_AT*

C(b_t)	**Y(b_t)**	**b_t**	**C(b_t)**	**Y(b_t)**	**b_t**	**C(b_t)**	**Y(b_t)**	**b_t**
00000	$y_0y_1y_2$	b_1	00110	y_1y_2	b_7	01100	$y_0y_2y_4$	b_{13}
00001	y_3	b_2	00111	$y_0y_2y_5$	b_8	01101	$y_1y_2y_E$	b_{14}
00010	$y_0y_1y_2$	b_3	01000	y_2y_4	b_9	01110	y_0y_3	b_{15}
00011	$y_0y_1y_4$	b_4	01001	$y_0y_1y_2$	b_{10}	01111	$y_0y_2y_3y_4$	b_{16}
00100	y_2y_5	b_5	01010	$y_0y_2y_5$	b_{11}	10000	$y_0y_2y_5$	b_{17}
00101	$y_0y_2y_3y_4$	b_6	01011	$y_0y_3y_6$	b_{12}	10001	$y_1y_4y_E$	b_{18}

7. *Zakodowanie klas łańcuchów pseudoekwiwalentnych.*

W mikroprogramowanym układzie sterującym *CMCU_AT* zbiór łańcuchów pseudoekwiwalentnych zawiera $\Pi_C = \{B_1, B_2, B_3\}$, gdzie $B_1 = \{\alpha_1\}$, $B_2 = \{\alpha_2, \alpha_3, \alpha_4\}$, $B_3 = \{\alpha_5, \alpha_6\}$. A zatem, $R_0=2$, $V = \{v_1, v_2\}$. Klasy $B_i \in \Pi_C$ zakodowano w następujący sposób: $K(B_1) = 00$, $K(B_2) = 01$, $K(B_3) = 10$.

8. *Utworzenie tabeli przejść mikroprogramowanego układu sterującego.*

Dla mikroprogramowanego układu *CMCU_AT* podzbiór łańcuchów pseudoekwiwalentnych zawiera $\Pi'_C = \{B_1, B_2\}$. Zgodnie z wyrażeniem (4.14) utworzono następujące formuły przejść:

$$B_1 \to x_1 I_2^1 \vee \overline{x_1} x_2 I_3^1 \vee \overline{x_1}\,\overline{x_2} x_3 I_4^1 \vee \overline{x_1}\,\overline{x_2}\,\overline{x_3} I_6^2;$$

$$B_2 \to x_3 I_2^2 \vee \overline{x_3} x_4 x_5 I_5^2 \vee \overline{x_3} x_4 \overline{x_5} I_5^1 \vee \overline{x_3}\,\overline{x_4} I_6^1.$$

W tabeli 4.3 przedstawiono fragment tabeli przejść mikroprogramowanego układu sterującego *CMCU_AT* odpowiadający formule przejść B_1.

Tabela 4.3. Fragment tabeli przejść układu *CMCU_AT*

$\mathbf{B_i}$	$\mathbf{K(B_i)}$	$\mathbf{I_g^j}$	$\mathbf{A(I_g^j)}$	$\mathbf{X_h}$	$\mathbf{\Psi_h}$	$\mathbf{\Phi_h}$	**H**
B_1	00	I_2^1	001000	x_1	D_3	-	1
		I_3^1	010000	$\overline{x_1}x_2$	D_2	-	2
		I_4^1	011000	$\overline{x_1}\overline{x_2}x_3$	D_2D_3	-	3
		I_6^2	101010	$\overline{x_1}\overline{x_2}\overline{x_3}$	D_1D_3	D_5	4

Adresy wejść pobierane są z tabeli 4.1. Kolumna X_h wypełniana jest na podstawie koniunkcji $X_i(I_g^j)$ z wyrażenia (4.14). W tabeli 4.3 zarówno licznik *CT* jak i rejestr *RG* ma wejścia informacyjne typu *D*.

9. Utworzenie tabeli konwertera adresów.

W przypadku mikroprogramowanego układu sterującego *CMCU_AT* tabela ta ma $M = 18$ wierszy. W tabeli 4.4 zaprezentowano fragment tabeli konwertera adresów.

Tabela 4.4. Fragment tabeli konwertera adresów *CMCU_AT*

$\mathbf{b_m}$	$\mathbf{A(b_m)}$	$\mathbf{Y(b_m)}$	$\mathbf{C(b_m)}$	$\mathbf{Z_m}$	**m**
b_1	000000	$Y(b_1)$	00000	-	1
b_2	000001	$Y(b_2)$	00001	z_5	2
b_3	001000	$Y(b_3)$	00010	z_4	3
b_4	001001	$Y(b_4)$	00011	z_4z_5	4

10. Utworzenie tabeli konwertera kodu.

W przypadku *CMCU_AT* tabela konwertera kodu ma 4 wiersze (Tabela 4.5).

Tabela 4.5. Tabela konwertera kodu *CMCU_AT*

$\mathbf{\alpha_g}$	$\mathbf{K(\alpha_g)}$	$\mathbf{B_i}$	$\mathbf{K(B_i)}$	$\mathbf{V_g}$	**g**
α_1	000	B_1	00	-	1
α_2	001	B_2	01	v_2	2
α_3	010	B_2	01	v_2	3
α_4	011	B_2	01	v_2	4

11. Wyznaczenie funkcji Φ, Ψ, Z, V .

Podstawą do utworzenia funkcji Φ, Ψ jest tabela przejść (Tabela 4.3). Zgodnie ze wzorem (4.16) dla omawianego przykładu otrzymano następujące funkcje: dla licznika CT $D_2 = P_2 \vee P_3 = \overline{v_1} v_2 x_1 x_2 \vee \overline{v_1} v_2 x_1 x_2 x_3$; dla rejestru RG $D_5 = P_4 = \overline{v_1} v_2 x_1 x_2 x_3$.

Podstawą do utworzenia funkcji Z jest tabela konwertera adresów (Tabela 4.4). Zgodnie ze wzorem (4.17) dla omawianego przykładu otrzymano: $z_5 = \overline{\tau_1} \tau_2 \tau_3 T_1 T_2 T_3 \vee \overline{\tau_1} \tau_2 \tau_3 \overline{T_1} T_2 T_3$.

Podstawą do utworzenia funkcji V jest tabela konwertera kodu (Tabela 4.5). Zgodnie ze wzorem (4.19) dla omawianego przykładu otrzymano: $v_2 = \overline{\tau_1} \tau_2 \tau_3 \vee \overline{\tau_1} \tau_2 \overline{\tau_3} \vee \overline{\tau_1} \tau_2 \tau_3$.

12. Implementacja mikroprogramowanego układu sterującego.

W przypadku matryc programowalnych *FPGA* blok konwertera adresów oraz pamięć sterownika mogą zostać zaimplementowane na dwa sposoby. Pierwsza możliwość to realizacja z wykorzystaniem dedykowanych bloków pamięci matryc *FPGA*. Takie rozwiązanie pozwala znacznie zmniejszyć ilość wykorzystanych bloków logicznych. Drugi sposób to implementacja układu konwertera oraz pamięci z wykorzystaniem bloków logicznych *FPGA*. Opcja ta jest stosowana w przypadku, gdy rozmiar pamięci sterownika przekracza rozmiar dostępnego miejsca w pamięciach dedykowanych (Wiśniewski i in., 2006).

Zaproponowana metoda syntezy mikroprogramowanego układu sterującego z konwerterem adresów umożliwia zachowanie rozmiaru pamięci (4.12) wówczas, gdy spełniony jest warunek (4.10). Jednakże zachodzą pewne warunki, podczas których rozmiar pamięci *CM* może zostać zmniejszony w porównaniu z (4.12). W rozprawie zaproponowano metody umożliwiające zmniejszenie rozmiaru pamięci mikroprogramowanego układu sterującego ze współdzieleniem kodów.

4.5. Mikroprogramowany układ sterujący z rozszerzonymi mikroinstrukcjami

Pamięć mikroprogramowanego układ sterującego *CMCU_AT* przechowuje elementy ze zboru $Y_0 = Y \cup \{y_0, y_E\}$ (Tabela 4.2). Każdy blok operacyjny b_q, ma *R*-bitowy adres $A(b_q)$. Niech pamięć *CM* przechowuje M_1 różnych mikroinstrukcji $Y_m \subseteq Y_0$, które będziemy nazywać rozszerzonymi mikroinstrukcjami (ang. *Expanded Microinstructions, EMI*). Zakodujmy każdą *EMI* $Y_m \subseteq Y_0$ kodem $K(Y_m)$ na

$$R_3 =]log_2 M_1[\tag{4.20}$$

bitach. Do kodowania wykorzystajmy zmienne $z_r \in Z$. Kod *K(Ym)* może być wykorzystany jako adres *EMI* Y_m $(m = 1, \ldots, M_1))$ w pamięci. Jeżeli zostanie spełniony warunek

$$R_3 < R, \tag{4.21}$$

wówczas rozmiar pamięci *CM* mikroprogramowanego układu sterującego z rozszerzonymi mikroinstrukcjami (ang. *Compositional Microprogram Control Unit with Expanded Microinstructions, CMCU_XM*) zmniejszy się w porównaniu z wartością (4.12).

Struktura mikroprogramowanego układu sterującego z rozszerzonymi mikroinstrukcjami jest taka sama jak układu z konwerterem adresów.

Metoda projektowania mikroprogramowanych układów sterujących *CMCU_XM* i *CMCU_AT* składa się z takich samych etapów. Różnice istnieją w sposobie wykonania kroku 5, 6 oraz 9:

1. *Przekształcenie początkowej sieci działań.*
2. *Utworzenie zbioru łańcuchów.*
3. *Zakodowanie łańcuchów.*
4. *Zakodowanie elementów łańcuchów.*
5. *Adresowanie rozszerzonych mikroinstrukcji.*

Adresy rozszerzonych mikroinstrukcji kodowane są z wykorzystaniem naturalnego kodu binarnego (*NKB*) kodem $K(Y_m)$ na R_3 bitach. Wartość R_3 wyznaczana jest zgonie ze wzorem (4.20).

6. *Wyznaczenie zawartości pamięci.*

   Pamięć *CM* mikroprogramowanego układu sterującego z rozszerzonymi mikroinstrukcjami reprezentowana jest poprzez tabelę o następujących kolumnach: $K(Y_m)$, Y_m, b_t, gdzie $K(Y_m)$ jest kodem mikroinstrukcji Y_m w bloku operacyjnym b_t.

7. *Zakodowanie klas łańcuchów pseudoekwiwalentnych.*

8. *Utworzenie tabeli przejść mikroprogramowanego układu sterującego.*

9. *Utworzenie tabeli konwertera adresów.*

   Tabela konwertera adresów jest podstawą utworzenia funkcji (4.13) i składa się z następujących kolumn: b_m, $A(b_m)$, Y_m, $K(Y_m)$, Z_m, m. Tutaj Z_m jest zbiorem zmiennych $z_r \in Z$, które są równe *1* w adresie $K(Y_m)$ mikroinstrukcji Y_m odpowiadającej blokowi $b_m \in B_1$.

10. *Utworzenie tabeli konwertera kodu.*

11. *Wyznaczenie funkcji Φ, Ψ, V, Z.*

12. *Implementacja mikroprogramowanego układu sterującego.*

4.5.1. Synteza mikroprogramowanego układu sterującego CMCU_XM

Cechy charakterystyczne metody syntezy mikroprogramowanego układu sterującego *CMCU_XM* zostaną omówione na przykładzie sieci działań z rysunku 4.5.

Etapy 1-4, 7-8, 10-12 syntezy mikroprogramowanego układu sterującego *CMCU_XM* wykonywane są w identyczny sposób jak układu o strukturze *CMCU_AT*. Różnice występują w krokach 5,6 oraz 9:

5. *Adresowanie rozszerzonych mikroinstrukcji.*

   Pamięć *CMCU_AT* zawiera $M_1 = 13$ rozszerzonych mikroinstrukcji (Tabela 4.2), gdzie $Y_1 = \{y_0, y_1, y_2\}$, $Y_2 = \{y_3\}$, $Y_3 = \{y_0, y_1, y_4\}$, $Y_4 = \{y_2, y_5\}$, $Y_5 = \{y_0, y_2, y_3, y_4\}$, $Y_6 = \{y_1, y_2\}$, $Y_7 = \{y_0, y_2, y_5\}$, $Y_8 = \{y_2, y_4\}$, $Y_9 = \{y_0, y_3, y_6\}$, $Y_{10} = \{y_0, y_2, y_4\}$,

$Y_{11} = \{y_1, y_2, y_E\}$, $Y_{12} = \{y_0, y_3\}$, $Y_{13} = \{y_1, y_4, y_E\}$. Zatem $R_3 = 4$, $Z = \{z_1, \ldots, z_4\}$, warunek (4.21) został spełniony a więc uzasadnione jest zastosowanie proponowanej metody.

Zakodujmy rozszerzone mikroinstrukcje Y_m $(m = 1, \ldots, 13)$ mikroprogramowanego układu sterującego *CMCU_XM* w następujący sposób: $K(Y_1) = 0000$, $K(Y_2) = 0001, \ldots,$, $K(Y_{13}) = 1100$.

6. *Wyznaczanie zawartości pamięci.*

Pamięć *CM* mikroprogramowanego układu sterującego z rozszerzonymi mikroinstrukcjami reprezentowana jest poprzez tabelę o następujących kolumnach: $K(Y_m)$, Y_m, b_t, gdzie $K(Y_m)$ - jest kodem mikroinstrukcji Y_m w bloku operacyjnym b_t. Dla *CMCU_XM* tabela ta zawiera $M_1 = 13$ wierszy (Tabela 4.6).

Tabela 4.6. Zawartość pamięci mikroprogramowanego układu *CMCU_XM*

$K(Y_m)$	**Y_m**	**b_t**	**$K(Y_m)$**	**Y_m**	**b_t**
0000	$y_0y_1y_2$	b_1,b_3,b_{10}	0111	y_2y_4	b_9
0001	y_3	b_2	1000	$y_0y_3y_6$	b_{12}
0010	$y_0y_1y_4$	b_4	1001	$y_0y_2y_4$	b_{13}
0011	y_2y_5	b_5	1010	$y_1y_2y_E$	b_{14}
0100	$y_0y_2y_3y_4$	b_6,b_{16}	1011	y_0y_3	b_{15}
0101	y_1y_2	b_7	1100	y_1,y_4y_E	b_{18}
0110	$y_0y_2y_5$	b_8,b_{11},b_{17}	1101	-	-

Optymalizacja rozmiaru pamięci *CM* jest możliwa, gdy różne bloki $b_t \in B_1$ zawierają takie same rozszerzone mikroinstrukcje (Tabela 4.6).

9. *Utworzenie tabeli konwertera adresów.*

Tabela konwertera adresów zawiera $M = 18$ wierszy. Fragment tabeli konwertera adresów dla układu *CMCU_XM* pokazano w tabeli 4.7.

Tabela 4.7. Fragment tabeli konwertera adresów dla układu *CMCU_XM*

b_m	**$A(b_m)$**	**Y_m**	**$K(Y_m)$**	**Z_m**	**m**
b_1	000000	Y_1	0000	-	1
b_2	000001	Y_2	0001	z_4	2
b_3	001000	Y_1	0000	-	3
b_4	001001	Y_3	0010	z_3	4

Zauważmy, że teraz system (4.13) ma 64-13=51 niewykorzystanych stanów na wejściu, które mogą być użyte do optymalizacji konwertera adresów *AT*.

Dalsze zmniejszenie pamięci mikroprogramowanego układu sterującego jest możliwe, jeżeli zostanie spełniony warunek:

$$R_4 < R_3 , \tag{4.22}$$

gdzie R_4 jest pojemnością bitową kodu $K(Y_q)$ mikroinstrukcji (kolekcji mikrooperacji) $Y_q \subseteq Y$. Parametr R_4 obliczany jest na podstawie wzoru:

$$R_4 =]log_2 Q[, \tag{4.23}$$

gdzie Q jest ilością różnych kolekcji mikrooperacji (mikroinstrukcji) w blokach $b_q \in B_1$ początkowej sieci działań.

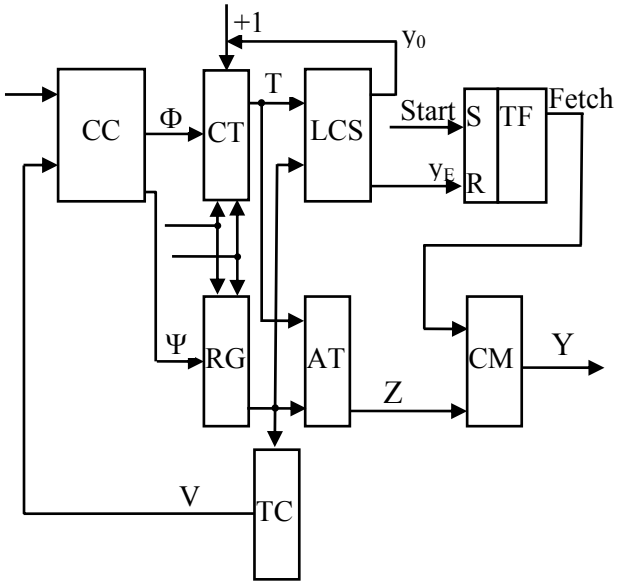
4.6. Mikroprogramowany układ sterujący z kodowaniem kolekcji mikroopracji

Niech początkowa sieć działań Γ zawiera Q mikroinstrukcji $Y_q \subseteq Y$. Zakodujmy każdą mikroinstrukcje Y_q kodem $K(Y_q)$ na R_4 bitach $(q = 1, \ldots, Q)$. Do kodowania mikroinstrukcji wykorzystajmy zmienne $z_r \in Z$. Wprowadźmy konwerter adresów *AT* do układu o strukturze *CMCU_CS*, który będzie przekształcał adresy (4.5) w kody $K(Y_q)$ $(q = 1, \ldots, Q)$. Wykorzystajmy układ *LCS* (ang. *Local Control Signal, LCS*) do generowania wewnętrznych zmiennych

$$y_0 = y_0(\tau, T) , \tag{4.24}$$

$$y_E = y_E(\tau, T) . \tag{4.25}$$

Wprowadzenie układu *LCS* do układu sterującego ze współdzieleniem kodów prowadzi do utworzenia mikroprogramowanego układu sterującego o strukturze *CMCU_EM* (ang. *Compositional Microprogram Control Unit with Encoding of Collections of Microoperations, CMCU_EM*) (Rys. 4.6).



Rys. 4.6. Struktura mikroprogramowanego układu sterującego *CMCU_EM*

Konwerter adresów *AT* generuje adresy mikroinstrukcji odpowiadające kodom $K(Y_q)$ $(q = 1, \ldots, Q)$, układ *LCS* steruje synchronizacją rejestru *RG* oraz licznika *CT* (używając wewnętrznej zmiennej y_0) oraz pobieraniem mikroinstrukcji (używając wewnętrznej zmiennej y_E). Pozostałe bloki mikroprogramowanego układu sterującego *CMCU_EM* realizują takie same funkcje jak bloki mikroprogramowanego układu sterującego *CMCU_CS*.

Metoda projektowania układu *CMCU_EM* składa się z trzynastu etapów. Pierwsze cztery etapy oraz 7, 8, 11 i 13 projektowania mikroprogramowanego układu *CMCU_EM* i *CMCU_AT* są takie same:

1. *Przekształcenie początkowej sieci działań.*
2. *Utworzenie zbioru łańcuchów dla przekształconej sieci działań.*
3. *Zakodowanie łańcuchów bloków operacyjnych.*
4. *Zakodowanie elementów łańcuchów.*
5. *Zakodowanie kolekcji mikrooperacji.*

   Kolekcje mikrooperacji kodowane są z wykorzystaniem naturalnego kodu binarnego (NKB) kodem $K(Y_q)$ na R_4 bitach, gdzie R_4 wyznaczane jest zgodnie ze wzorem (4.23).

6. *Wyznaczenie zawartości pamięci.*

   Pamięć *CM* mikroprogramowanego układu sterującego z kodowaniem mikrooperacji reprezentowana jest poprzez tabelę o następujących kolumnach: $K(Y_q)$, Y_q, b_t, gdzie $K(Y_q)$ - jest kodem zbioru mikrooperacji Y_q w bloku operacyjnym b_t.

7. *Zakodowanie klas łańcuchów pseudoekwiwalentnych.*

8. *Utworzenie tabeli przejść mikroprogramowanego układu sterującego.*

9. *Utworzenie tabeli konwertera adresów.*

   Tabela konwertera adresów jest podstawą utworzenia funkcji (4.13) i składa się z następujących kolumn: b_m, $A(b_m)$, Y_q, $K(Y_q)$, Z_m, m, gdzie Y_q jest zbiorem mikrooperacji z bloku $b_m \in B_1$; Z_m jest zbiorem zmiennych $z_r \in Z$, które są równe *1* w kodzie $K(Y_q)$ z m-tego wiersza tabeli konwertera adresów $(m = 1, \ldots, M)$.

10. *Utworzenie tabeli układu LCS.*

    Tabela układu *LCS* jest podstawą do utworzenia funkcji (4.24)-(4.25) i składa się z następujących kolumn: b_m, $A(b_m)$, y_0, y_E, m, gdzie $A(b_m)$ jest adresem mikroinstrukcji z bloku b_m; y_0 jest zmienną synchronizującą; y_E – jest zmienną służącą do pobierania mikroinstrukcji z pamięci.

11. *Utworzenie tabeli konwertera kodu.*

12. *Wyznaczenie funkcji* $\Phi, \Psi, V, Z, y_0, y_E$.

    Funkcje Φ i Ψ wyznaczane są na podstawie tabeli przejść mikroprogramowanego układu sterującego *CMCU_EM* według wzoru (4.16). Funkcje Z wyznaczane są na podstawie tabeli konwertera adresów zgodnie ze wzorem (4.17). Tabela konwertera adresów ma M_2 niewykorzystanych stanów na wejściu:

$$M_2 = 2^{R_1+R_2} - Q. \tag{4.26}$$

Funkcje y_0, y_E wyznaczane są na podstawie tabeli układu *LCS*. Funkcje y_0, y_E mają M_0 niewykorzystanych stanów na wejściu i stany te mogą być wykorzystane do optymalizacji tych funkcji.

Funkcje V wyznaczane są na podstawie tabeli konwertera kodu zgodnie ze wzorem (4.19).

13. *Implementacja mikroprogramowanego układu sterującego.*

4.6.1. Synteza mikroprogramowanego układu sterującego CMCU_EM

Metoda syntezy mikroprogramowanego układu sterującego *CMCU_EM* zostanie omówiona na przykładzie sieci działań z rysunku 4.5. Etapy wspólne dla omawianych metod zostały ominięte.

5. Zakodowanie kolekcji mikrooperacji.

Początkowa sieć działań Γ_1 (Rys. 4.5) zawiera $Q = 7$ mikroinstrukcji, gdzie $Y_1 = \{y_1, y_2\}$, $Y_2 = \{y_3\}$, $Y_3 = \{y_1, y_4\}$, $Y_4 = \{y_2, y_5\}$, $Y_5 = \{y_2, y_3, y_4\}$, $Y_6 = \{y_2, y_4\}$, $Y_7 = \{y_3, y_6\}$. $R_4 = 3$, zatem do zakodowania mikroinstrukcji wystarczą trzy zmienne ze zboru $Z = \{z_1, z_2, z_3\}$. Ponieważ warunek (4.22) został spełniony projektowanie mikroprogramowanego układu sterującego *CMCU_EM* jest uzasadnione. Zakodujmy mikroinstrukcje $Y_q \subseteq Y$ zgodnie z naturalnym kodem binarnym: $K(Y_1) = 000$, $K(Y_2) = 001, \ldots, K(Y_7) = 110$.

6. Wyznaczenie zawartości pamięci.

Dla mikroprogramowanego układu sterującego z kodowaniem mikroinstrukcji tabela ta zawiera $Q = 7$ wierszy (Tabela 4.8).

Tabela 4.8. Zawartość pamięci mikroprogramowanego układu sterującego *CMCU_EM*

$K(Y_q)$	Y_q	b_t	$K(Y_q)$	Y_q	b_t
000	y_1y_2	$b_1,b_3,b_7,b_{10},b_{14}$	100	$y_2y_3y_4$	b_9,b_{16}
001	y_3	b_2, b_{15}	101	y_2y_4	b_9,b_{13}
010	y_1y_4	b_4,b_{18}	110	y_3y_6	b_{12}
011	y_2y_5	b_5,b_8,b_{11},b_{17}	111	-	-

9. Utworzenie tabeli konwertera adresów.

Tabela konwertera adresów mikroprogramowanego układu sterującego *CMCU_EM* ma $M = 18$ wierszy (Tabela 4.9).

Tabela 4.9. Tabela konwertera adresów układu *CMCU_EM*

$\mathbf{b_m}$	$\mathbf{A(b_m)}$	$\mathbf{Y_q}$	$\mathbf{K(Y_q)}$	$\mathbf{Z_m}$	**m**	$\mathbf{b_m}$	$\mathbf{A(b_m)}$	$\mathbf{Y_q}$	$\mathbf{K(Y_q)}$	$\mathbf{Z_m}$	**M**
b_1	000000	Y_1	000	-	1	b_{10}	100000	Y_1	000	-	10
b_2	000001	Y_2	001	z_3	2	b_{11}	100001	Y_4	011	z_2z_3	11
b_3	001000	Y_1	000	-	3	b_{12}	100010	Y_7	110	z_1z_2	12
b_4	001001	Y_3	010	z_2	4	b_{13}	100011	Y_6	101	z_1z_3	13
b_5	001010	Y_4	011	z_2z_3	4	b_{14}	100100	Y_1	000	-	14
b_6	010000	Y_5	100	z_1	6	b_{15}	101000	Y_2	001	z_3	15
b_7	010001	Y_1	000	-	7	b_{16}	101001	Y_5	100	z_1	16
b_8	011000	Y_4	011	z_2z_3	8	b_{17}	101010	Y_4	011	z_2z_3	17
b_9	011001	Y_6	101	z_1z_3	9	b_{18}	101011	Y_5	010	-	18

10. Utworzenie tabeli układu LCS.

Tabela układu *LCS* ma $M = 18$ wierszy. W tabeli 4.10 zaprezentowano trzy pierwsze oraz trzy ostatnie wiersze tabeli układu *LCS* dla mikroprogramowanego układu sterującego *CMCU_EM*.

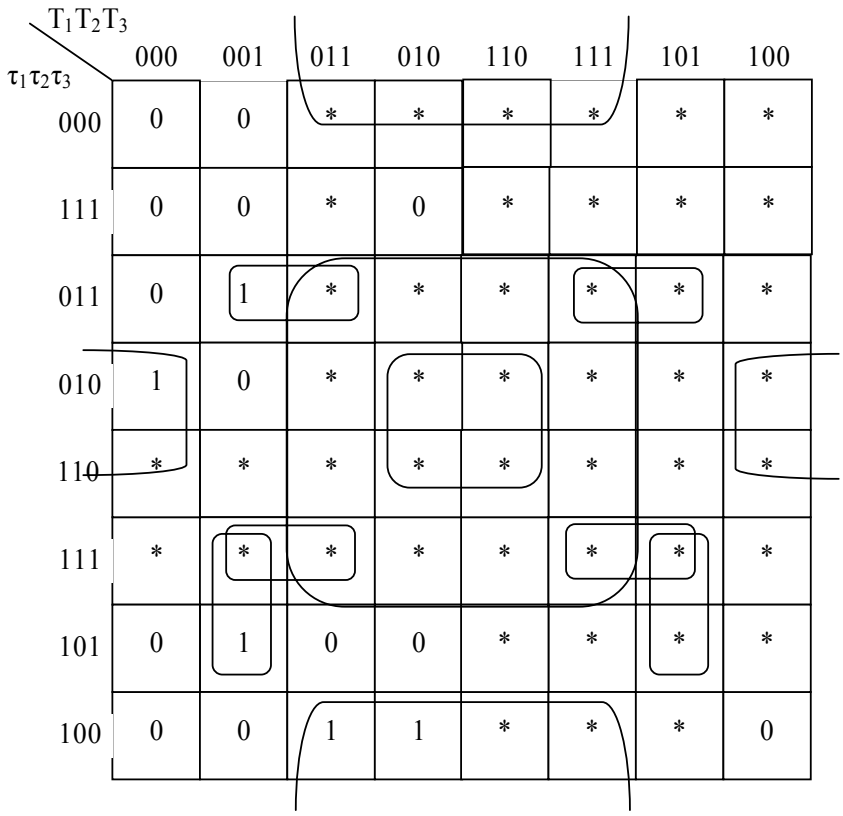
Tabela 4.10. Fragment tabeli bloku *LCS* dla *CMCU_EM*

$\mathbf{b_m}$	$\mathbf{A(b_m)}$	$\mathbf{y_0}$	$\mathbf{y_E}$	**m**	$\mathbf{b_m}$	$\mathbf{A(b_m)}$	$\mathbf{y_0}$	$\mathbf{y_E}$	**m**
b_1	000000	1	0	1	b_{16}	101001	1	0	16
b_2	000001	0	0	2	b_{17}	101010	1	0	17
b_3	001000	1	0	3	b_{18}	101011	0	1	18

12. Wyznaczenie funkcji $\Phi, \Psi, V, Z, y_0, y_E$.

Funkcje Φ i Ψ wyznaczane są na podstawie tabeli przejść mikroprogramowanego układu sterującego *CMCU_EM* według wzoru (4.16).

Funkcje Z wyznaczane są na podstawie tabeli konwertera adresów zgodnie ze wzorem (4.17). Tabela konwertera adresów ma $M_2 = 57$ niewykorzystanych stanów na wejściu. Na rysunku 4.7 przedstawiono tablicę Karnaugh'a dla funkcji $z_1 \in Z$.



Rys. 4.7. Tablica Karnaugh'a dla funkcji z_1

Na podstawie tablicy Karnaugh'a otrzymamy $z_1 = \tau_2 \overline{\tau_3}\,\overline{T_3} \vee \tau_2\tau_3 T_3 \vee \tau_1\tau_3\overline{T_2}T_3$. Równanie to jest dużo prostsze niż formuła, którą byśmy otrzymali z tabeli 4.9. Formuła utworzona na podstawie tabeli konwertera adresów miałaby *30* literałów.

Funkcje y_0, y_E wyznaczane są na podstawie tabeli układu *LCS*. Funkcje y_0, y_E mają M_0 niewykorzystanych stanów na wejściu i stany te mogą być wykorzystane do optymalizacji tych funkcji. Początkowa formuła dla funkcji y_0, utworzona na podstawie tabeli 4.10 miała *72* literałów. Po optymalizacji otrzymamy równanie składające się tylko z *11* literałów:

$$y_0 = \overline{T_1}\,\overline{T_2}\,\overline{T_3} \vee \overline{\tau_2}\tau_3\overline{T_2} \vee \tau_1\overline{T_1} \vee \tau_1\tau_2\overline{T_3},$$

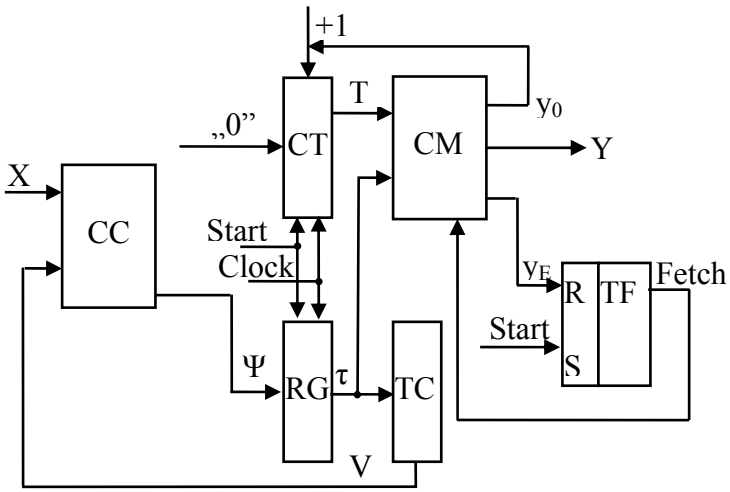
Natomiast dla formuły y_E otrzymamy równanie

$$y_E = T_3 \vee \tau_3 T_2 \overline{T_3}.$$

Funkcje V wyznaczane są na podstawie tabeli konwertera kodu zgodnie ze wzorem (4.19).

4.7. Mikroprogramowany układ sterujący z elementarnymi łańcuchami

Strukturę mikroprogramowanego układu sterującego z elementarnymi łańcuchami (ang. *Compositional Microprogram Control Unit with Elementary Operational Linear Chains, CMCU_EOLC)* przedstawiono na rysunku 4.8.

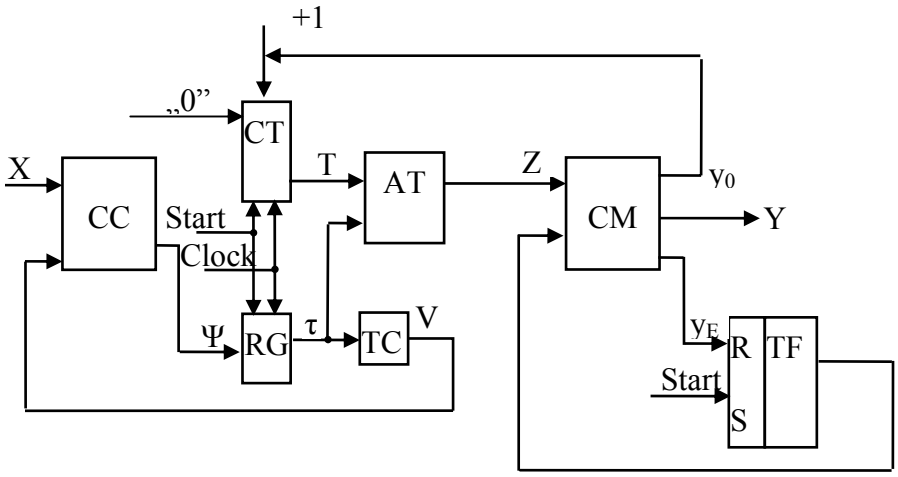


Rys. 4.8. Struktura mikroprogramowanego układu *CMCU_EOLC*

W mikroprogramowanym układzie sterującym *CMCU_EOLC* układ kombinacyjny *CC* implementuje tylko system funkcji wzbudzeń (4.7) rejestru *RG*. Różnicą w stosunku do poprzednio opisywanych układów jest to, że w układach z elementarnymi łańcuchami na wejście licznika *CT* podawany jest bezpośrednio kod *0* (Barkalov, Wiśniewski, 2004). Poza wymienioną różnicą zasada działania układów z elementarnymi łańcuchami jest taka sama jak układu *CMCU_CS* .

4.8. Mikroprogramowany układ sterujący CMCU_EOLC_AT

Mikroprogramowany układ sterujący (Rys.4.9) o strukturze *CMCU_EOLC_AT* (ang. *Compositional Microprogram Control Unit with Elementary Operational Linear Chains and Address Transformer, CMCU_EOLC_AT*) powstał poprzez wprowadzenie bloku konwertera adresów do układu o strukturze *CMCU_EOLC*.



Rys. 4.9. Struktura mikroprogramowanego układu *CMCU_EOLC_AT*

Metoda projektowania *CMCU_EOLC_AT* jest taka sama jak układu o strukturze *CMC_CS* za wyjątkiem kroków 2, 8 oraz 11. Metoda ta składa się z następujących etapów:

1. *Przekształcenie początkowej sieci działań.*
2. *Utworzenie zbioru łańcuchów elementarnych dla przekształconej sieci działań.*

   Zbiór łańcuchów elementarnych wyznaczany jest zgodnie z definicją 4.7.
3. *Zakodowanie elementarnych łańcuchów.*
4. *Zakodowanie elementów łańcuchów.*
5. *Adresowanie mikroinstrukcji.*
6. *Wyznaczenie zawartości pamięci.*
7. *Zakodowanie klas pseudoekwiwalentnych łańcuchów.*
8. *Utworzenie tabeli przejść mikroprogramowanego układu sterującego.*

   Tabela przejść mikroprogramowanego układu o strukturze *CMCU_EOLC_AT* nie zawiera kolumny Φ_h z funkcjami wzbudzeń dla licznika *CT*. Formuł przejść wyznaczane są tak jak w poprzednich metodach według wzoru (4.14).
9. *Utworzenie tabeli konwertera adresów.*
10. *Utworzenie tabeli konwertera kodu.*
11. *Wyznaczenie funkcji* Ψ, V, Z .

    Funkcje $D_r \in \Psi$ są wyznaczane na podstawie tabeli przejść według wzoru:

$$D_r = \bigvee_{h=1}^{H} C_{rh} P_h \ (r = 1, \ldots, R_1), \tag{4.27}$$

gdzie C_{rh} jest zmienną Boolowską, która jest równa *1* wtedy i tylko wtedy gdy zmienna D_r $(r = 1, \ldots, R_1)$ jest zapisana w h-tym wierszu tabeli przejść $(h = 1, \ldots, H)$.

W porównaniu do wszystkich poprzednio opisanych metod w kroku tym nie wyznacza się funkcji wzbudzeń dla licznika *CT*. Pozostałe funkcje wyznaczane są tak, jak w metodzie syntezy układu o strukturze *CMCU_CS*.

12. *Implementacja mikroprogramowanego układu sterującego.*

4.8.1. Synteza mikroprogramowanego układu sterującego CMCU_EOLC_AT

Metoda syntezy mikroprogramowanego układu sterującego *CMCU_EOLC_AT* zostanie omówiona na przykładzie sieci działań Γ_1 przedstawionej na rysunku 4.5.

1. *Przekształcenie początkowej sieci działań.*

   Dla rozpatrywanej sieci działań Γ_1 (Rys. 4.5) przekształcenie ogranicza się do dodania wewnętrznej zmiennej y_E do bloków b_{14} oraz b_{18}. Natomiast struktura sieci działań pozostaje niezmieniona.

2. *Utworzenie zbioru łańcuchów elementarnych.*

   Dla sieci działań Γ_1 utworzono następujący zbiór łańcuchów elementarnych $C = \{\alpha_1, \ \ldots, \ \alpha_9\}$, gdzie $\alpha_1 = \langle b_1, b_2 \rangle$, $I_1^1 = b_1$, $O_1 = b_2$; $\alpha_2 = \langle b_3 \rangle$, $I_2^1 = O_2 = b_3$; $\alpha_3 = \langle b_4, b_5 \rangle$, $I_3^1 = b_4$, $O_3 = b_5$; $\alpha_4 = \langle b_6, b_7 \rangle$, $I_4^1 = b_6$, $O_4 = b_7$; $\alpha_5 = \langle b_8, b_9 \rangle$, $I_5^1 = b_8$, $O_5 = b_9$; $\alpha_6 = \langle b_{10}, b_{11} \rangle$, $I_6^1 = b_{10}$, $O_6 = b_{11}$; $\alpha_7 = \langle b_{12}, b_{13}, b_{14} \rangle$, $I_7^1 = b_{12}$ $O_7 = b_{14}$; $\alpha_8 = \langle b_{15}, b_{16} \rangle$, $I_8^1 = b_{15}$, $O_8 = b_{16}$; $\alpha_9 = \langle b_{17}, b_{18} \rangle$, $I_9^1 = b_{17}$, $O_9 = b_{18}$. W wyniku analizy zbioru łańcuchów otrzymamy: $G = 9$, $F_{max} = 3$, $O(\Gamma_1) = \{b_2, b_3, b_5, b_7, b_9, b_{11}, b_{14}, b_{16}, b_{18}\}$. Do bloków operacyjnych $b_q \notin O(\Gamma)$ została dodana wewnętrzna zmienna y_0.

   W omawianym przykładzie mamy $R_1 = 4$, $R_2 = 2$, $M = 18$, $R = 5$ tak, więc warunek (4.10) został spełniony, przez co uzasadnione jest zastosowanie proponowanej metody.

3. *Zakodowanie łańcuchów elementarnych.*

Łańcuchy *CMCU_EOLC_AT* zakodowano w następujący sposób: $K(\alpha_1) = 0000$, $K(\alpha_2) = 0001, \ldots, K(\alpha_9) = 1000$.

4. *Zakodowanie elementów łańcuchów.*

Dla *CMCU_EOLC_AT* otrzymano następujące kody elementów łańcuchów: $K(b_1) = K(b_3) = K(b_4) = K(b_6) = K(b_8) = K(b_{10}) = K(b_{12}) = K(b_{15}) = K(b_{17}) = 00$, $K(b_2) = K(b_5) = K(b_7) = K(b_9) = K(b_{11}) = K(b_{13}) = K(b_{16}) = K(b_{18}) = 01$, $K(b_{14}) = 10$.

Kody łańcuchów $\alpha_g \in C$ oraz ich elementów umożliwiają znalezienie adresu (4.5). W tabeli 4.11 zaprezentowano adresy $A(b_t)$ mikroinstrukcji dla mikroprogramowanego układu sterującego o strukturze *CMCU_EOLC*.

Tabela 4.11. Adresy mikroinstrukcji *CMCU_EOLC*

$\mathbf{b_t}$	$\mathbf{A(b_t)}$	$\mathbf{b_t}$	$\mathbf{A(b_t)}$	$\mathbf{b_t}$	$\mathbf{A(b_t)}$
b_1	0000 00	b_7	0011 01	b_{13}	0110 01
b_2	0000 01	b_8	0100 00	b_{14}	0110 10
b_3	0001 00	b_9	0100 01	b_{15}	0111 00
b_4	0010 00	b_{10}	0101 00	b_{16}	0111 01
b_5	0010 01	b_{11}	0101 01	b_{17}	1000 00
b_6	0011 00	b_{12}	0110 00	b_{18}	1000 01

Pierwsze trzy pozycje adresu $A(b_t)$ odpowiadają kodowi łańcucha $K(\alpha_g)$, kolejne trzy bity – kodowi jego elementu.

5. *Adresowanie mikroinstrukcji.*

Adresy mikroinstrukcji dla sieci działań Γ_1 zakodowano na R=5 bitach w następujący sposób: $C(b_1) = 00000$, $C(b_2) = 00001$, …, $C(b_{18}) = 10001$.

6. *Wyznaczenie zawartości pamięci.*

Dla mikroprogramowanego układu sterującego *CMCU_EOLC_AT* tabela reprezentująca zawartość pamięci ma $M = 18$ wierszy (Tabela 4.12).

Tabela 4.12. Adresy mikroinstrukcji *CMCU_EOLC_AT*

$C(b_t)$	$Y(b_t)$	b_t	$C(b_t)$	$Y(b_t)$	b_t	$C(b_t)$	$Y(b_t)$	b_t
00000	$y_0y_1y_2$	b_1	00110	y_1y_2	b_7	01100	$y_0y_2y_4$	b_{13}
00001	y_3	b_2	00111	$y_0y_2y_5$	b_8	01101	$y_1y_2y_E$	b_{14}
00010	$y_0y_1y_2$	b_3	01000	y_2y_4	b_9	01110	y_0y_3	b_{15}
00011	$y_0y_1y_4$	b_4	01001	$y_0y_1y_2$	b_{10}	01111	$y_0y_2y_3y_4$	b_{16}
00100	y_2y_5	b_5	01010	$y_0y_2y_5$	b_{11}	10000	$y_0y_2y_5$	b_{17}
00101	$y_0y_2y_3y_4$	b_6	01011	$y_0y_3y_6$	b_{12}	10001	$y_1y_4y_E$	b_{18}

7. *Zakodowanie klas łańcuchów pseudoekwiwalentnych.*

W mikroprogramowanym układzie sterującym *CMCU_EOLC_AT* zbiór łańcuchów pseudoekwiwalentnych zawiera $\Pi_C = \{B_1, B_2, B_3, B_4, B_5, B_6\}$, gdzie $B_1 = \{\alpha_1\}$, $B_2 = \{\alpha_2\}$, $B_3 = \{\alpha_3, \alpha_4, \alpha_5\}$, $B_4 = \{\alpha_6\}$, $B_5 = \{\alpha_8\}$, $B_6 = \{\alpha_7, \alpha_9\}$. A zatem, $R_0 = 3$, $V = \{v_1, v_2, v_3\}$. Klasy $B_i \in \Pi_C$ zakodowano w następujący sposób: $K(B_1) = 000$, $K(B_2) = 001$, $K(B_3) = 010$, $K(B_4) = 011$, $K(B_5) = 100$, $K(B_6) = 101$.

8. *Utworzenie tabeli przejść mikroprogramowanego układu sterującego.*

Dla mikroprogramowanego układu *CMCU_EOLC_AT* podzbiór łańcuchów pseudoekwiwalentnych zawiera $\Pi'_C = \{B_1, B_2, B_3, B_4, B_5\}$. Zgodnie z wyrażeniem (4.14) utworzono następujące formuły przejść:

$$B_1 \to x_1 I_2^1 \vee \overline{x_1} x_2 I_4^1 \vee \overline{x_1}\,\overline{x_2} x_3 I_5^1 \vee \overline{x_1}\,\overline{x_2}\,\overline{x_3} I_9^1;$$

$$B_2 \to I_3^1;$$

$$B_3 \to x_3 I_3^1 \vee \overline{x_3} x_4 x_5 I_7^1 \vee \overline{x_3} x_4 \overline{x_5} I_6^1 \vee \overline{x_3}\,\overline{x_4} I_8^1;$$

$$B_4 \to I_7^1;$$

$$B_5 \to I_9^1;$$

W tabeli 4.13 przedstawiono fragment tabeli przejść mikroprogramowanego układu sterującego *CMCU_EOLC_AT* odpowiadający formułom przejść B_1, B_2.

Tabela 4.13. Fragment tabeli przejść *CMCU_EOLC_AT*

$\mathbf{B_i}$	$\mathbf{K(B_i)}$	$\mathbf{I_g^j}$	$\mathbf{A(I_g^j)}$	$\mathbf{X_h}$	$\mathbf{\Psi_h}$	**h**
B_1	000	I_2^1	000100	x_1	D_4	1
		I_4^1	001100	$\overline{x}_1 x_2$	D_3D_4	2
		I_5^1	010000	$\overline{x}_1\overline{x}_2 x_3$	D_2	3
		I_9^1	100000	$\overline{x}_1\overline{x}_2\overline{x}_3$	D_1	4
B_2	001	I_3^1	001000	1	D_3	5

Adresy wejść pobierane są z tabeli 4.11. Kolumna X_h wypełniana jest na podstawie koniunkcji $X_i(I_g^j)$ z wyrażenia (4.14). W tabeli 4.13 rejestr *RG* ma wejścia informacyjne typu *D*.

9. *Utworzenie tabeli konwertera adresów.*

W przypadku mikroprogramowanego układu sterującego *CMCU_EOLC_AT* tabela *AT* ma *M=18* wierszy. W tabeli 4.14 zaprezentowano fragment tabeli konwertera adresów.

Tabela 4.14. Fragment tabeli konwertera adresów *CMCU_EOLC_AT*

$\mathbf{b_m}$	$\mathbf{A(b_m)}$	$\mathbf{Y(b_m)}$	$\mathbf{C(b_m)}$	$\mathbf{Z_m}$	**M**
b_1	000000	$Y(b_1)$	00000	-	1
b_2	000001	$Y(b_2)$	00001	z_5	2
b_3	000100	$Y(b_3)$	00010	z_4	3
b_4	001000	$Y(b_4)$	00011	z_4z_5	4

10. *Utworzenie tabeli konwertera kodu.*

W przypadku układu *CMCU_EOLC_AT* tabela konwertera kodu ma *7* wierszy (Tabela 4.15).

Tabela 4.15. Tabela konwertera kodu *CMCU_EOLC_AT*

$\mathbf{\alpha_g}$	$\mathbf{K(\alpha_g)}$	$\mathbf{B_i}$	$\mathbf{K(B_i)}$	$\mathbf{V_g}$	**g**
α_1	0000	B_1	000	-	1
α_2	0001	B_2	001	v_3	2
α_3	0010	B_3	010	v_2	3
α_4	0011	B_3	010	v_2	4
α_5	0100	B_3	010	v_2	5
α_6	0101	B_4	011	v_2v_3	6
α_8	0111	B_5	100	v_1	7

11. *Wyznaczenie funkcji* Ψ, Z, V.

Podstawą do utworzenia funkcji Ψ jest tabela przejść (Tabela 4.13). Zgodnie ze wzorem (4.16) dla rejestru RG otrzymano następującą funkcję: $D_3 = P_2 \vee P_5 = \overline{v_1 v_2 v_3 x_1 x_2} \vee \overline{v_1 v_2 v_3}$.

Podstawą do utworzenia funkcji Z jest tabela konwertera adresów (Tabela 4.14). Zgodnie ze wzorem (4.17) otrzymano: $z_5 = \overline{\tau_1 \tau_2 \tau_3 \tau_4 T_1 T_2} \vee \overline{\tau_1 \tau_2 \tau_3 \tau_4 T_1 T_2}$.

Podstawą do utworzenia funkcji V jest tabela konwertera kodu (Tabela 4.15). Zgodnie ze wzorem (4.19) otrzymano: $v_1 = \overline{\tau_1} \tau_2 \tau_3 \overline{\tau_4} \vee \overline{\tau_1} \tau_2 \tau_3 \tau_4 \vee \tau_1 \overline{\tau_2 \tau_3 \tau_4}$.

12. *Implementacja mikroprogramowanego układu sterującego.*

4.9. Mikroprogramowany układ sterujący CMCU_EOLC_XM

Budowa mikroprogramowanego układu sterującego *CMCU_EOLC_XM* (ang. *Compositional Microprogram Control Unit with Elementary Operational Linear Chains and Expanded Microinstructions, CMCU_EOLC_XM*) jest taka sama jak układu *CMCU_EOLC_AT* (Rys. 4.9).

Metoda projektowania mikroprogramowanych układów sterujących *CMCU_EOLC_XM* i *CMCU_EOLC_AT* składa się z takich samych etapów. Różnice istnieją w sposobie wykonania kroków 5, 6 oraz 9:

1. Przekształcenie początkowej sieci działań.

2. Utworzenie zbioru łańcuchów elementarnych.

3. Zakodowanie łańcuchów elementarnych.

4. Zakodowanie elementów łańcuchów.

5. Adresowanie mikroinstrukcji.

Adresy mikroinstrukcji kodowane są z wykorzystaniem naturalnego kodu binarnego (*NKB*) kodem $K(Y_m)$ na R_3 bitach. Wartość R_3 wyznaczana jest zgonie ze wzorem (4.20).

6. Wyznaczenie zawartości pamięci.

Pamięć *CM* mikroprogramowanego układu sterującego *CMCU_EOLC_XM* reprezentowana jest poprzez tabelę o następujących kolumnach: $K(Y_m)$, Y_m, b_t, gdzie $K(Y_m)$ - jest kodem mikroinstrukcji Y_m w bloku operacyjnym b_t.

7. *Zakodowanie klas łańcuchów pseudoekwiwalentnych.*

8. *Utworzenie tabeli przejść mikroprogramowanego układu sterującego.*

9. *Utworzenie tabeli konwertera adresów.*

   Tabela konwertera adresów jest podstawą utworzenia funkcji (4.13) i składa się z następujących kolumn: b_m, $A(b_m)$, Y_m, $K(Y_m)$, Z_m, m. Tutaj Z_m jest zbiorem zmiennych $z_r \in Z$, które są równe *1* w adresie $K(Y_m)$ mikroinstrukcji Y_m odpowiadającej blokowi $b_m \in B_1$.

10. *Utworzenie tabeli konwertera kodu.*

11. *Wyznaczenie funkcji* Ψ, Z, V.

12. *Implementacja mikroprogramowanego układu sterującego.*

4.9.1. Synteza mikroprogramowanego układu sterującego CMCU_EOLC_XM

Cechy charakterystyczne syntezy mikroprogramowanego układu sterującego *CMCU_EOLC_XM* zostaną omówione na przykładzie sieci działań z rysunku 4.5.

Etapy 1-4, 7-8, 10-12 wykonywane są w identyczny sposób jak układu o strukturze *CMCU_EOLC_AT*. Różnice występują w krokach 5, 6 oraz 9:

5. *Adresowanie mikroinstrukcji.*

   Pamięć układu *CMCU_EOLC_AT* zawiera $M_1 = 13$ rozszerzonych mikroinstrukcji (Tabela 4.12), gdzie $Y_1 = \{y_0, y_1, y_2\}$, $Y_2 = \{y_3\}$, $Y_3 = \{y_0, y_1, y_4\}$, $Y_4 = \{y_2, y_5\}$, $Y_5 = \{y_0, y_2, y_3, y_4\}$, $Y_6 = \{y_1, y_2\}$, $Y_7 = \{y_0, y_2, y_5\}$, $Y_8 = \{y_2, y_4\}$, $Y_9 = \{y_0, y_3, y_6\}$, $Y_{10} = \{y_0, y_2, y_4\}$, $Y_{11} = \{y_1, y_2, y_E\}$, $Y_{12} = \{y_0, y_3\}$, $Y_{13} = \{y_2, y_3, y_4\}$, $Y_{14} = \{y_1, y_4, y_E\}$.

   Zatem $R_3 = 4$, $Z = \{z_1, \ldots, z_4\}$, warunek (4.21) został spełniony a więc uzasadnione jest zastosowanie proponowanej metody.

   Zakodujmy rozszerzone mikroinstrukcje $Y_m\,(m = 1, \ldots, 14)$ mikroprogramowanego układu sterującego *CMCU_EOLC_XM* w następujący sposób: $K(Y_1) = 0000$, $K(Y_2) = 0001, \ldots,$, $K(Y_{14}) = 1101$.

6. *Wyznaczanie zawartości pamięci.*

Pamięć *CM* mikroprogramowanego układu sterującego *CMCU_EOLC_XM* reprezentowana jest poprzez tabelę o następujących kolumnach: $K(Y_m)$, Y_m, b_t, gdzie $K(Y_m)$ - jest kodem mikroinstrukcji Y_m w bloku operacyjnym b_t. Tabela ta zawiera $M_1 = 14$ wierszy (Tabela 4.16).

Tabela 4.16. Zawartość pamięci *CMCU_EOLC_XM*

$K(Y_m)$	Y_m	b_t	$K(Y_m)$	Y_m	b_t
0000	$y_0y_1y_2$	b_1,b_{10}	0111	y_2y_4	b_9
0001	y_3	b_2	1000	$y_0y_3y_6$	b_{12}
0010	$y_0y_1y_4$	b_4	1001	$y_0y_2y_4$	b_{13}
0011	y_2y_5	b_5,b_{11}	1010	$y_1y_2y_E$	b_{14}
0100	$y_0y_2y_3y_4$	b_6,	1011	y_0y_3	b_{15}
0101	y_1y_2	b_3,b_7	1100	$y_2y_3y_4$	b_{16}
0110	$y_0y_2y_5$	b_8, b_{17}	1101	y_1,y_4y_E	b_{18}

Optymalizacja rozmiaru pamięci *CM* dla *CMCU_EOLC_XM* jest możliwa, gdy różne bloki $b_t \in B_1$ zawierają takie same rozszerzone mikroinstrukcje.

9. *Utworzenie tabeli konwertera adresów.*

Tabela konwertera adresów zawiera M=18 wierszy. Fragment tabeli konwertera adresów dla układu *CMCU_EOLC_XM* zaprezentowano w tabeli 4.17.

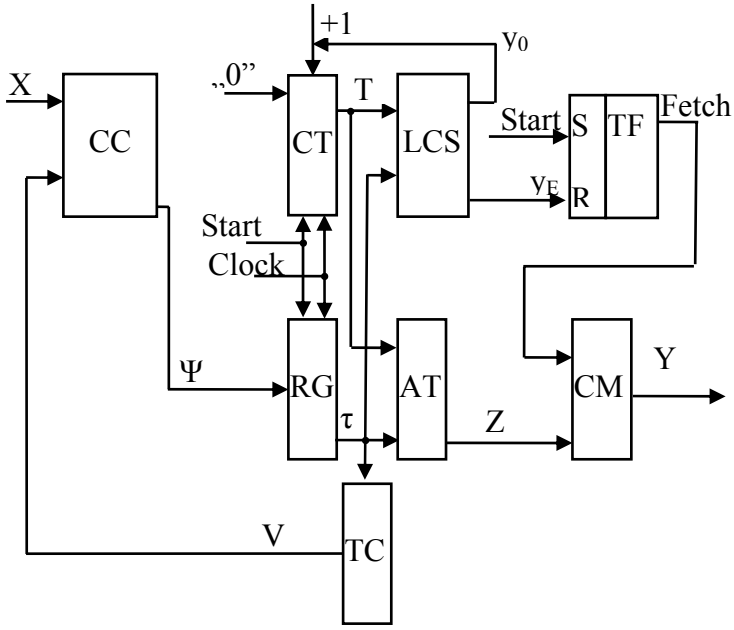
Tabela 4.17. Fragment tabeli konwertera adresów układu *CMCU_EOLC_XM*

b_m	$A(b_m)$	Y_m	$K(Y_m)$	Z_m	m
b_1	000000	Y_1	0000	-	1
b_2	000001	Y_2	0001	z_4	2
b_3	000100	Y_1	0000	-	3
b_4	001000	Y_3	0010	z_3	4

4.10. Mikroprogramowany układ sterujący CMCU_EOLC_EM

Mikroprogramowany układ sterujący (Rys.4.10) o strukturze *CMCU_EOLC_EM* (ang. *Compositional Microprogram Control Unit with Elementary Operational Linear Chains*

and Encoding of Collections of Microoperations, *CMCU_EOLC_EM*) powstał poprzez wprowadzenie bloku *LCS* do układu o strukturze *CMCU_EOLC_AT*.



Rys. 4.10. Struktura mikroprogramowanego układu *CMCU_EOLC_EM*

Metoda projektowania układu *CMCU_EOLC_EM* składa się z trzynastu etapów. Pierwsze cztery etapy oraz etapy 7, 8, 11 i 13 projektowania układów *CMCU_EOLC_EM* i *CMCU_EOLC_AT* są takie same.

1. *Przekształcenie początkowej sieci działań.*

2. *Utworzenie zbioru łańcuchów elementarnych.*

3. *Zakodowanie łańcuchów elementarnych.*

4. *Zakodowanie elementów łańcuchów.*

5. *Zakodowanie kolekcji mikrooperacji.*

   Kolekcje mikrooperacji kodowane są z wykorzystaniem naturalnego kodu binarnego (*NKB*) kodem $K(Y_q)$ na R_4 bitach, gdzie R_4 wyznaczane jest zgodnie ze wzorem (4.23).

6. *Wyznaczenie zawartości pamięci.*

   Pamięć *CM* mikroprogramowanego układu sterującego z kodowaniem mikrooperacji reprezentowana jest poprzez tabelę o następujących kolumnach: $K(Y_q)$, Y_q, b_t, gdzie $K(Y_q)$ - jest kodem zbioru mikrooperacji Y_q w bloku operacyjnym b_t.

7. *Zakodowanie klas łańcuchów pseudoekwiwalentnych.*

8. *Utworzenie tabeli przejść mikroprogramowanego układu sterującego.*

9. *Utworzenie tabeli konwertera adresów.*

   Tabela konwertera adresów jest podstawą utworzenia funkcji (4.13) i składa się z następujących kolumn: b_m, $A(b_m)$, Y_q, $K(Y_q)$, Z_m, m, gdzie Y_q jest zbiorem mikrooperacji z węzła $b_m \in B_1$; Z_m jest zbiorem zmiennych $z_r \in Z$, które są równe 1 w kodzie $K(Y_q)$ z m-tego wiersza tabeli konwertera adresów $(m = 1, \ldots, M)$.

10. *Utworzenie tabeli układu LCS.*

    Tabela układu *LCS* jest podstawą do utworzenia funkcji (4.24)-(4.25) i składa się z następujących kolumn: b_m, $A(b_m)$, y_0, y_E, m, gdzie $A(b_m)$ jest adresem mikroinstrukcji z węzła b_m; y_0 jest zmienną synchronizującą; y_E – jest zmienną służącą do pobierania mikroinstrukcji z pamięci.

11. *Utworzenie tabeli konwertera kodu.*

12. *Wyznaczenie funkcji* Ψ, V, Z, y_0, y_E.

    Funkcje Ψ wyznaczane są na podstawie tabeli przejść mikroprogramowanego układu sterującego *CMCU_EOLC_EM* według wzoru (4.27). Funkcje Z wyznaczane są na podstawie tabeli konwertera adresów zgodnie ze wzorem (4.17).

    Funkcje y_0, y_E wyznaczane są na podstawie tabeli układu *LCS*. Funkcje y_0, y_E mają M_0 niewykorzystanych stanów na wejściu i stany te mogą być wykorzystane do optymalizacji tych funkcji.

    Funkcje V wyznaczane są na podstawie tabeli konwertera kodu zgodnie ze wzorem (4.19).

13. *Implementacja mikroprogramowanego układu sterującego.*

4.10.1. Synteza mikroprogramowanego układu sterującego CMCU_EOLC_EM

Metoda syntezy mikroprogramowanego układu sterującego *CMCU_EOLC_EM* zostanie omówiona na przykładzie sieci działań z rysunku 4.5. Etapy wspólne dla omawianych metod zostały ominięte.

5. *Zakodowanie kolekcji mikrooperacji.*

Początkowa sieć działań Γ_1 (Rys.4.5) zawiera $Q = 7$ mikroinstrukcji, gdzie $Y_1 = \{y_1, y_2\}$, $Y_2 = \{y_3\}$, $Y_3 = \{y_1, y_4\}$, $Y_4 = \{y_2, y_5\}$, $Y_5 = \{y_2, y_3, y_4\}$, $Y_6 = \{y_2, y_4\}$, $Y_7 = \{y_3, y_6\}$. $R_4 = 3$, zatem do zakodowania mikroinstrukcji wystarczą trzy zmienne ze zboru $Z = \{z_1, z_2, z_3\}$. Ponieważ warunek (4.22) został spełniony projektowanie mikroprogramowanego układu sterującego *CMCU_EOLC_EM* jest uzasadnione. Zakodujmy mikroinstrukcje $Y_q \subseteq Y$ zgodnie z naturalnym kodem binarnym: $K(Y_1) = 000$, $K(Y_2) = 001, \ldots, K(Y_7) = 110$.

6. *Wyznaczenie zawartości pamięci.*

Dla układu *CMCU_EOLC_EM* tabela ta zawiera $Q = 7$ wierszy (Tabela 4.18).

Tabela 4.18. Zawartość pamięci układu sterującego *CMCU_EOLC_EM*

$K(Y_q)$	**Y_q**	**b_t**	**$K(Y_q)$**	**Y_q**	**b_t**
000	y_1y_2	$b_1,b_3,b_7,b_{10},b_{14}$	100	$y_2y_3y_4$	b_9,b_{16}
001	y_3	b_2, b_{15}	101	y_2y_4	b_9,b_{13}
010	y_1y_4	b_4,b_{18}	110	y_3y_6	b_{12}
011	y_2y_5	b_5,b_8,b_{11},b_{17}	111	-	-

9. *Utworzenie tabeli konwertera adresów.*

Tabela konwertera adresów mikroprogramowanego układu sterującego *CMCU_EOLC_EM* ma $M = 18$ wierszy (Tabela 4.19).

Tabela 4.19. Tabela konwertera adresów układu *CMCU_EOLC_EM*

b_m	**$A(b_m)$**	**Y_q**	**$K(Y_q)$**	**Z_m**	**m**	**b_m**	**$A(b_m)$**	**Y_q**	**$K(Y_q)$**	**Z_m**	**m**
B_1	000000	Y_1	000	-	1	b_{10}	010100	Y_1	000	-	10
B_2	000001	Y_2	001	z_3	2	b_{11}	010101	Y_4	011	z_2z_3	11
B_3	000100	Y_1	000	-	3	b_{12}	011000	Y_7	110	z_1z_2	12
B_4	001000	Y_3	010	z_2	4	b_{13}	011001	Y_6	101	z_1z_3	13
B_5	001001	Y_4	011	z_2z_3	4	b_{14}	011010	Y_1	000	-	14
B_6	001100	Y_5	100	z_1	6	b_{15}	011100	Y_2	001	z_3	15
B_7	001101	Y_1	000	-	7	b_{16}	011101	Y_5	100	z_1	16
B_8	010000	Y_4	011	z_2z_3	8	b_{17}	100000	Y_4	011	z_2z_3	17
B_9	010001	Y_6	101	z_1z_3	9	b_{18}	100001	Y_3	010	z_2	18

10. *Utworzenie tabeli układu LCS.*

Tabela układu *LCS* ma $M = 18$ wierszy. W tabeli 4.20 zaprezentowano trzy pierwsze oraz trzy ostatnie wiersze tabeli układu *LCS* dla mikroprogramowanego układu sterującego *CMCU_EOLC_EM*.

Tabela 4.20. Fragment tabeli układu *LCS CMCU_EOLC_EM*

$\mathbf{b_m}$	$\mathbf{A(b_m)}$	$\mathbf{y_0}$	$\mathbf{y_E}$	**m**	$\mathbf{b_m}$	$\mathbf{A(b_m)}$	$\mathbf{y_0}$	$\mathbf{y_E}$	**m**
b_1	000000	1	0	1	b_{16}	011101	1	0	16
b_2	000001	0	0	2	b_{17}	100000	1	0	17
b_3	000100	1	0	3	b_{18}	100001	0	1	18

4.11. Podsumowanie

W rozdziale czwartym zaprezentowano metodę syntezy mikroprogramowanych układów sterujących ze współdzieleniem kodów z wykorzystaniem konwertera adresów jak również omówiono modyfikacje tej metody. Proponowana metoda umożliwia wykorzystanie współdzielenia kodów, gdy rozmiar adresu generowanych przez licznik oraz rejestr jest większy niż minimalny rozmiar adresu mikroinstrukcji, który jest obliczany zgodnie ze wzorem (4.12).

Celem zaproponowanych w rozprawie metod jest zmniejszenie rozmiaru pamięci mikroprogramowanego układu sterującego ze współdzieleniem kodów. Uzyskanie minimalnego rozmiaru pamięci powinno być jednym z kryteriów przy wyborze metody projektowania układów sterujących.

Metoda współdzielenia kodów umożliwia zmniejszenie zużycia zasobów sprzętowych (elementów *LUT*) w układzie adresującym *CC* poprzez użycie pseudoekwiwalentnych łańcuchów początkowej sieci działań. Zastosowanie konwertera adresów *AT* umożliwia użycie metody współdzielenia kodów, niezależnie od charakterystyki sieci działań. Zastosowanie konwertera adresów prowadzi do zwiększenia czasu trwania cyklu pracy systemu cyfrowego wykorzystującego mikroprogramowany układ sterujący. W związku z tym metoda może być zastosowana tylko wówczas, gdy projektowany układ spełnia wymogi czasowe. Stwierdzenie to odnosi się do wszystkich zaproponowanych w rozprawie struktur mikroprogramowanych układów sterujących.

Zaproponowana metoda może być wykorzystana dla wszystkich znanych modyfikacji mikroprogramowanego układu sterującego ze współdzieleniem kodów (Barkalov, 2002; Barkalov, Wisniewski 2004; Barkalov i in., 2005) np.:

- *CMCU* z optymalnym kodowaniem pseudoekwiwalentnych łańcuchów;
- *CMCU* z optymalnym kodowaniem pseudoekwiwalentnych elementarnych łańcuchów;
- *CMCU* z przekształceniem kodu elementarnego łańcucha na kod klas pseudoekwiwalentnych elementarnych łańcuchów.

Zastosowanie mikroprogramowanego układu sterującego jest uzasadnione tylko wówczas, jeżeli początkowa sieć działań zawiera nie mniej niż 75% węzłów operacyjnych (Barkalov, 2002). Oznacza to, że zaproponowane metody są efektywne tylko w przypadku implementacji algorytmów sterowania zgodnych z taką charakterystyką.

5. Program konwertujący sieci działań do CMCU - *fca2cmcu*

Rozdział piąty obok rozdziału czwartego zawiera dalszą prezentację uzyskanych rezultatów. Opisywany w rozdziale program *fca2cmcu* jest praktyczną implementacją opracowanych struktur mikroprogramowanych układów sterujących. Kolejno są omawiane: format danych wejściowych, budowa systemu, format pliku wyjściowego.

5.1. Format danych wejściowych

Do opisu danych wejściowych wykorzystano formę tekstową sieci działań (Barkalov, Salomatin, Starodubow, 1991). Reprezentacja tekstowa sieci działań musi zawierać wszystkie informacje o strukturze sieci, jak również o wszystkich sygnałach wejściowych i wyjściowych. Zastosowany w pracy format spełnia te wymagania.

W pliku można wyróżnić dwie części: pierwszą opisującą strukturę sieci, w której elementy zaczynają się od numeru bloku oraz drugą opisującą sygnały wyjściowe poszczególnych bloków, w której elementy zaczynają się od znaku Y. Opis poszczególnych symboli używanych przy tworzeniu formy tekstowej sieci działań zamieszczono w tabeli 5.1.

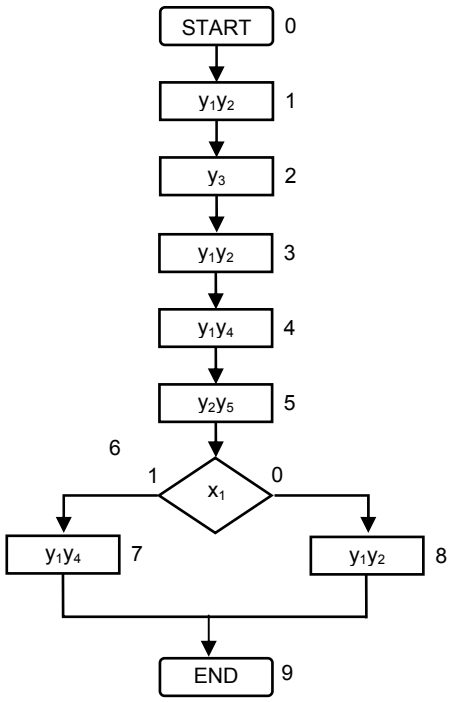
Tabela 5.1. Znaczenie symboli wykorzystywanych do opisu sieci działań (# - numer)

Symbol	**Znaczenie**
#S: #.	Blok początkowy
#E.	Blok końcowy
#O: Y#, #.	Blok operacyjny, Y - zbiór wyjść bloku
#X: x#, #, #.	Blok warunkowy, x - sygnał wejściowy
Y#: y#, ..., y#.	Kolekcja mikrooperacji (sygnałów wyjściowych)

W części opisującej strukturę sieci każda cyfra na początku linii oznacza numer bloku, następnie podawany jest symbol oznaczający rodzaj bloku. Ostatnia cyfra w linii oznacza numer bloku następnego. W przypadku bloku warunkowego X przedostatnia cyfra oznacza numer bloku następnego dla ścieżki prawdy, a ostatnia numer bloku następnego dla ścieżki w kierunku fałszu. Plik ma strukturę listy połączeń. Na rysunku 5.1 przedstawiono przykładową sieć działań z odpowiadającym jej opisem tekstowym.

a) b)

```
0S: 1.
1O: Y1, 2.
2O: Y2, 3.
3O: Y1, 4.
4O: Y3, 5.
5O: Y4, 6.
6X:  x1,  7,
8.
7O: Y3, 9.
8O: Y1, 9.
9E.

Y1: y1, y2.
Y2: y3.
Y3: y1, y4.
Y4: y2, y5.
```

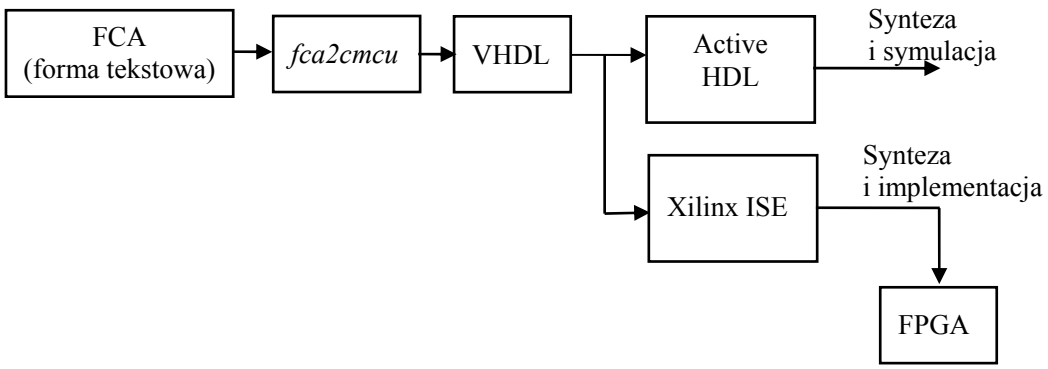


Rys. 5.1. Sieć działań: a) reprezentacja tekstowa; b) reprezentacja graficzna

5.2. Budowa systemu

Program *fca2cmcu* został napisany w *Języku ANSI C* i może być uruchamiany na dowolnej platformie, dla której istnieje kompilator *Języka ANSI C*. Wejściem dla programu jest plik z opisem tekstowym sieci działań.

Podstawowym zadaniem programu jest translacja tekstowej postaci sieci działań do struktur mikroprogramowanych układów sterujących opisanych w języku *VHDL*. Na rysunku 5.2 przedstawiono typową ścieżkę projektową z wykorzystaniem oprogramowania *fca2cmcu*.



Rys. 5.2. Typowa ścieżka projektowa z wykorzystaniem oprogramowania *fca2cmcu*

W wyniku działania programu tworzonych jest osiem plików wynikowych zawierających przedstawione w pracy struktury dla wejściowej sieci działań. Kolejnym krokiem może być synteza i symulacja działania struktur mikroprogramowanych w środowisku *Active-HDL* oraz implementacja na przykład z wykorzystaniem pakietu *Xilinx ISE*.

5.3. Plik wyjściowy

Wynikiem działania programu *fca2cmcu* jest plik z opisem w języku *VHDL* wszystkich opracowanych w rozprawie struktur mikroprogramowanych układów sterujących realizujących algorytm sterowania zapisany w postaci sieci działań. Dla każdej sieci wejściowej generowane jest osiem struktur w języku *VHDL:*

- *CMCU_CS,*
- *CMCU_AT,*
- *CMCU_XM,*
- *CMCU_EM,*
- *CMCU_EOLC_CS,*
- *CMCU_EOLC_AT,*
- *CMCU_EOLC_XM,*
- *CMCU_EOLC_EM*.

Każdy plik składa się z części opisującej poszczególne moduły (*CC*, *CT*, *RG*, *AT*, *TC*, *TF*, *CM*, *LCS*) oraz z części głównej opisującej strukturę danego układu mikroprogramowanego. Przykładowy plik wynikowy zamieszczono w Dodatku B.

6. Wyniki eksperymentów

W rozdziale szóstym zebrano i opisano wyniki syntezy oraz symulacji mikroprogramowanych układów sterujących znajdujących się w niniejszej pracy. Przedstawiono porównanie z wynikami syntezy mikroprogramowanego układu sterującego bez konwertera adresów (*CMCU_CS, CMCU_EOLC_CS*).

6.1. Rozmiar pamięci

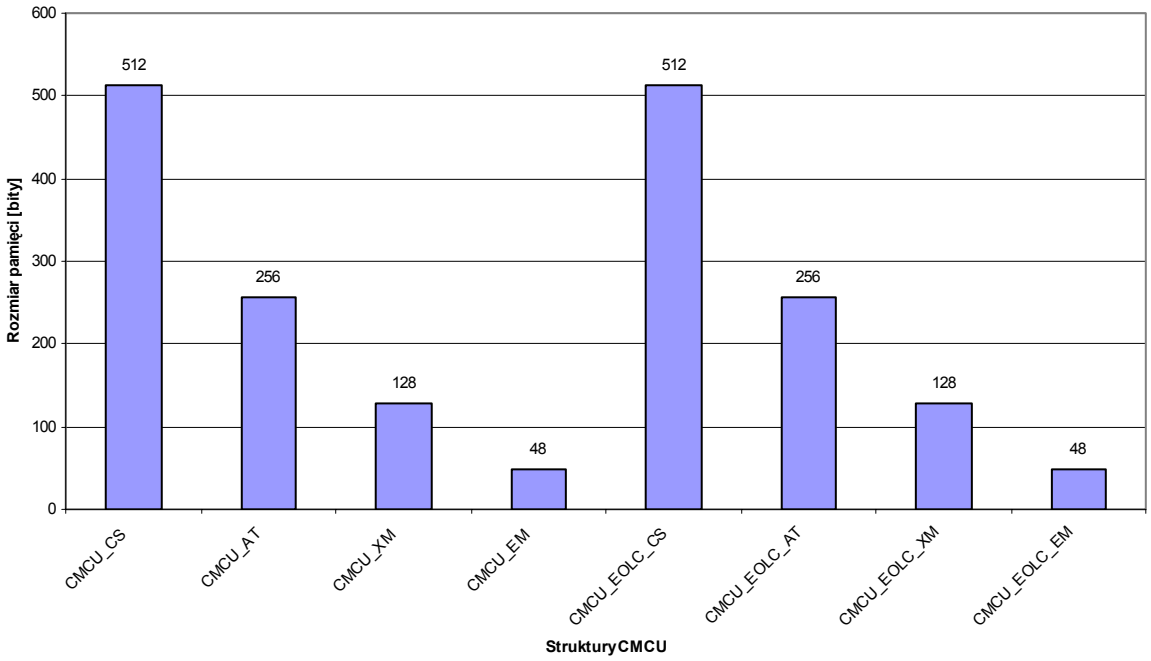
Zestaw testowy zawierał 20 sieci działań, co daje 160 różnych struktur *CMCU*. Poniżej przedstawione zostały rozmiary pamięci mikroprogramowanych układów sterujących wygenerowanych przez program *fca2cmcu* na podstawie zestawu sieci testowych.

Tabela 6.1. Rozmiary pamięci jednostki sterującej [bity]

Sieć testowa	**CMCU**				**CMCU_EOLC**			
	CS	AT	XM	EM	CS	AT	XM	EM
we_00	512	256	128	48	512	256	128	48
we_01	2304	1152	288	112	4608	1152	288	112
we_02	2304	1152	288	112	4608	1152	288	112
we_03	1280	640	320	128	1280	640	320	128
we_04	2560	640	320	128	5120	640	320	128
we_05	2816	704	352	144	2816	704	352	144
we_06	1664	832	416	352	3328	832	416	352
we_07	2560	1280	320	128	2560	1280	320	128
we_08	1152	576	288	112	1152	576	288	112
we_09	2304	576	288	112	2304	576	288	112
we_10	3328	1664	416	352	3328	1664	832	352
we_11	1280	640	320	128	1280	640	320	128
we_12	2560	1280	320	128	2560	1280	320	128
we_13	1152	576	288	112	1152	576	288	112
we_14	1152	576	288	112	2304	576	288	112
we_15	1664	1664	416	176	3328	1664	416	176
we_16	3328	1664	416	176	1664	1664	416	176
we_17	1152	1152	288	112	2304	1152	288	112
we_18	1664	832	416	176	3328	832	416	176
we_19	1152	576	288	112	1152	576	288	112

Na podstawie tabeli 6.1 można stwierdzić, że zastosowanie opisanych w pracy metod skutkuje zmniejszeniem rozmiaru pamięci jednostki sterującej implementowanej jako mikroprogramowany układ sterujący. Należy zauważyć, że dla niektórych przypadków sięga ono 90% w stosunku do układu bez konwertera adresów (*CMCU_CS*, *CMCU_EOLC_CS*).

Na rysunku 6.1 przedstawiono rozmiary pamięci *CMCU* dla algorytmu sterowania opisanego siecią działań z rysunku 4.5.



Rys. 6.1. Rozmiar pamięci dla sieci testowej *we_00*

Wykresy obrazujące rozmiary pamięci *CMCU* dla pozostałych sieci testowych zamieszczono w Dodatku C.

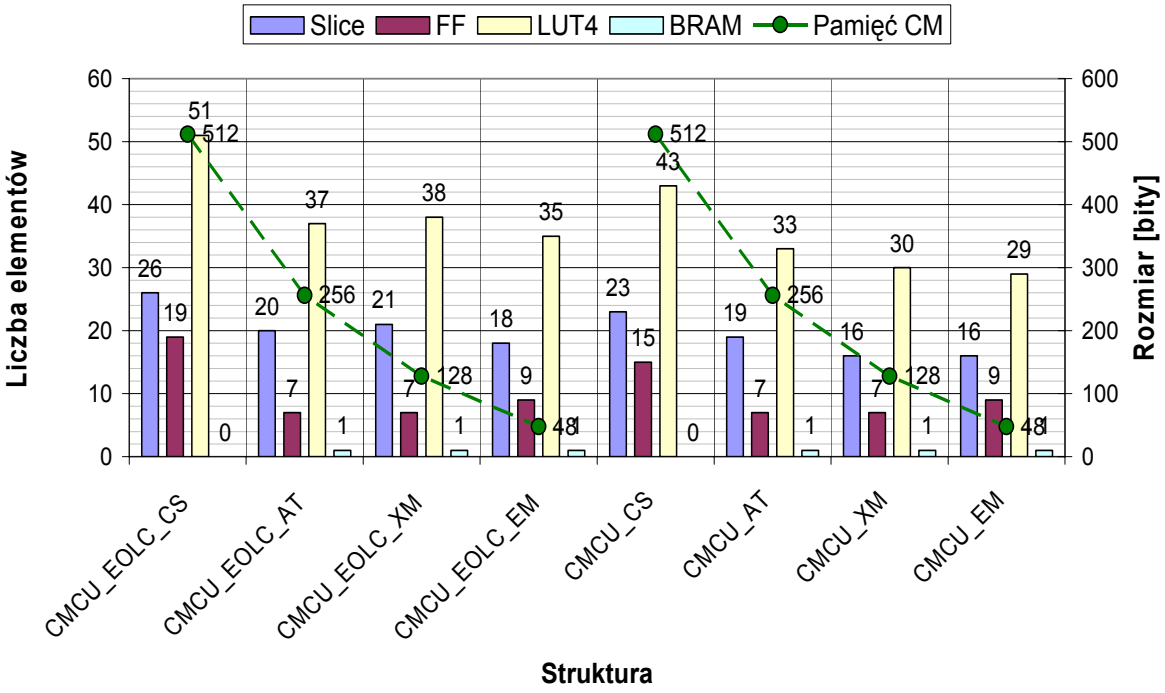
6.2. Analiza zużycia zasobów sprzętowych

Do syntezy i implementacji struktur mikroprogramowanych realizujących algorytm sterowania opisany siecią działań z rysunku 4.5 wykorzystano oprogramowanie *Xilinx ISE 8.2i*. Zastosowano cztery dostępne w pakiecie *ISE* strategie optymalizacji. Dwie z nich jako podstawowe kryterium wykorzystują ilość zużytych zasobów (area level 1, area level 2), dwie kolejne – maksymalną częstotliwość pracy układu (speed level 1, speed level 2). Platformę docelową stanowił układ *FPGA Xilinx Virtex–II Pro* (*xc2vp30-7ff896c*).

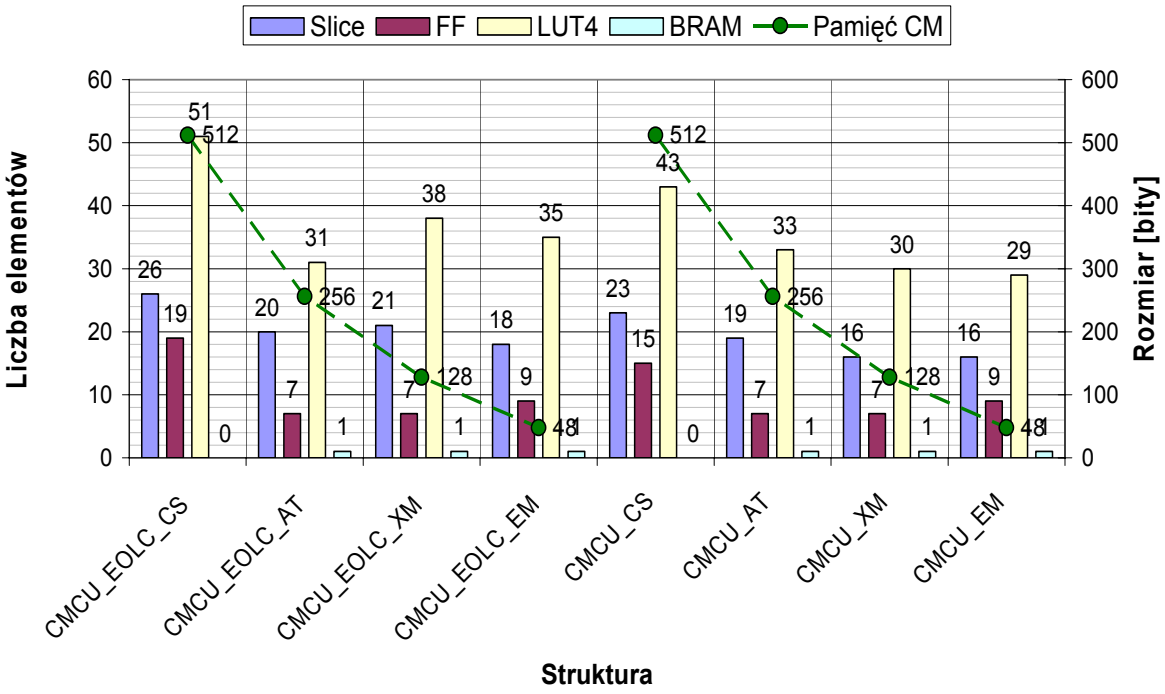
Układ *xc2vp30* zawiera w swojej strukturze następujące zasoby sprzętowe:

- 13696 „plastrów” (S*lice*),
- 27392 przerzutników,
- 27392 czterowejściowych elementów LUT,
- 136 dedykowanych bloków pamięci (*Block RAM*).

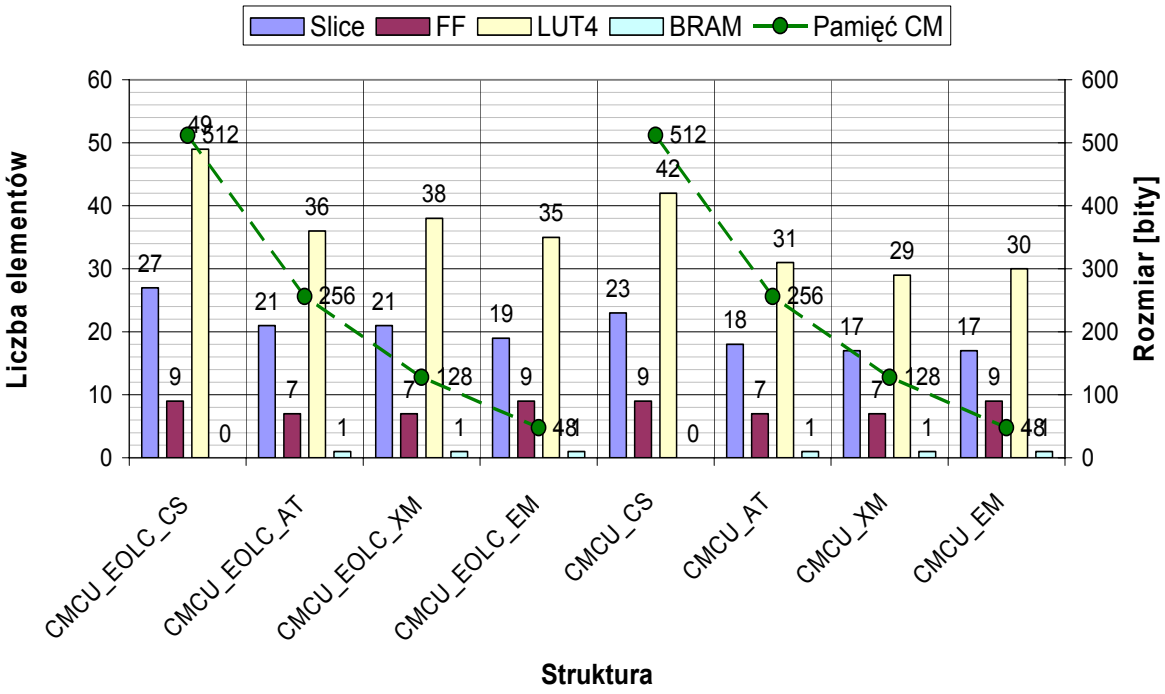
Na rysunkach 6.2, 6.3, 6.4, 6.4 przedstawiono zużycie zasobów sprzętowych dla sieci działań z rysunku 4.5 dla różnych strategii optymalizacji.



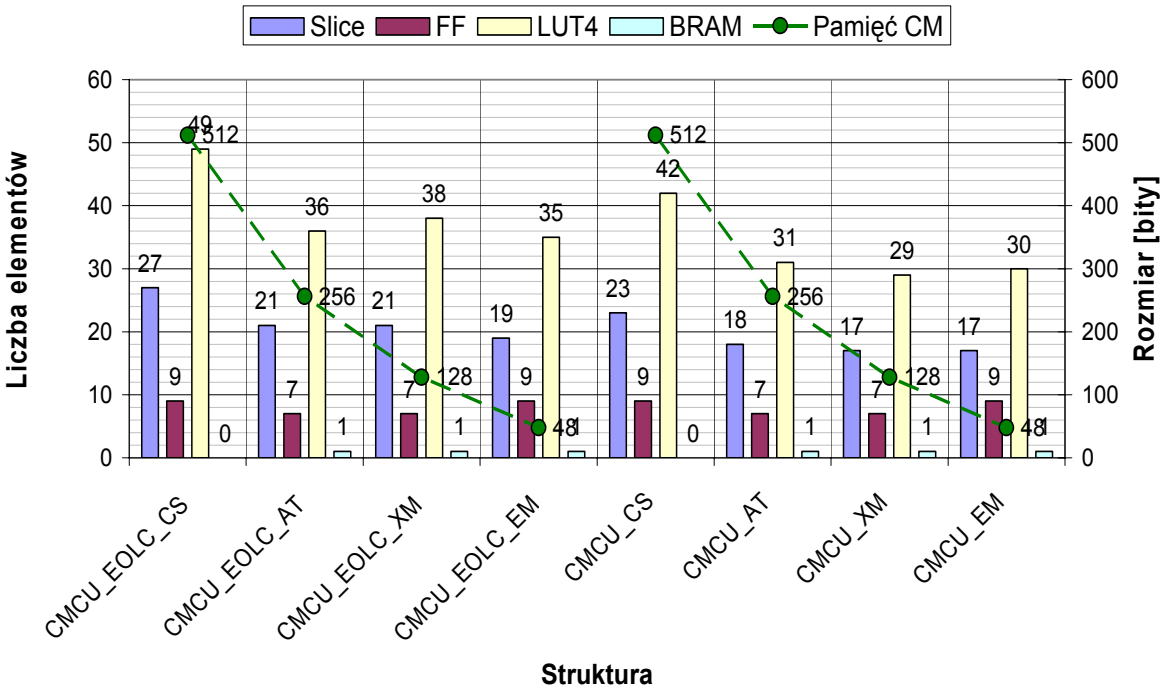
Rys. 6.2. Zużycie zasobów sprzętowych dla sieci testowej *we_00 (speed level 1*)



Rys. 6.3. Zużycie zasobów sprzętowych dla sieci testowej *we_00 (speed level 2*)



Rys. 6.4. Zużycie zasobów sprzętowych dla sieci testowej *we_00 (area level 1*)



Rys. 6.5. Zużycie zasobów sprzętowych dla sieci testowej *we_00 (area level 2*)

Na podstawie wykresów przedstawionych na rysunkach 6.2, 6.3, 6.4, 6.5 można stwierdzić, że przy spełnieniu warunków 4.10, 4.21, 4.22 ilość elementów *LUT* oraz *BRAM* (dedykowane bloki pamięci) w zaproponowanych układach mikroprogramowanych jest nie większa niż w układzie bez bloku konwertera adresów. Można przypuszczać, że dla dużych sieci działań, gdy ilość kolekcji mikrooperacji będzie większa niż ilość słów w bloku pamięci, rozmiar pamięci może być również zmniejszony w porównaniu z wartością 4.12.

Warto zwrócić uwagę na fakt, że dodanie kolejnych bloków (konwertera adresów, układu *LCS*) do struktury mikroprogramowanych układów sterujących nie pociąga za sobą zwiększenia zużycia zasobów sprzętowych.

Wykresy obrazujące zużycie zasobów sprzętowych dla wybranych sieci testowych zamieszczono w Dodatku D. Ze względu na brak znaczących różnic w zużyciu zasobów sprzętowych dla różnych strategii optymalizacji, w Dodatku D zamieszczono jedynie wykresy dla optymalizacji *speed level 1*.

6.3. Parametry czasowe

W tabeli 6.2 przedstawiono częstotliwości maksymalne oraz czasy trwania cyklu dla struktur zaimplementowanych w układzie *FPGA* dla sieci działań z rysunku 4.5. Parametry czasowe dla wybranych sieci testowych zamieszczono w Dodatku E.

Tabela 6.2. Parametry czasowe

Struktury	**Minimalny czas trwania cyklu [ns]**				**Częstotliwość maksymalna [MHz]**			
	speed 1	*speed 2*	*area 1*	*area 2*	*speed 1*	*speed 2*	*area 1*	*area 2*
CMCU_CS	5,443	5,443	5,965	5,965	183,736	183,736	167,636	167,636
CMCU_AT	2,528	2,528	3,685	3,685	395,515	395,515	271,37	271,370
CMCU_XM	2,488	2,488	3,647	3,647	401,905	401,905	274,198	274,198
CMCU_EM	3,687	3,687	4,945	4,945	271,201	271,201	202,22	202,220
CMCU_EOLC_CS	5,360	5,36	6,818	6,818	186,564	186,564	146,668	146,668
CMCU_EOLC_AT	2,510	2,51	3,717	3,717	398,438	398,438	269,041	269,041
CMCU_EOLC_XM	2,532	2,532	3,706	2,532	395,010	395,015	269,844	395,010
CMCU_EOLC_EM	5,277	5,277	5,986	5,986	189,516	189,516	167,054	167,054

Wprowadzenie konwertera adresów powinno skutkować wydłużeniem czasu trwania cyklu a co za tym idzie zmniejszeniem częstotliwości maksymalnej. Jednakże na podstawie wyników zaprezentowanych w tabeli 6.2 można zauważyć, że czas trwania cyklu dla układu o strukturze *CMCU_AT* został skrócony w porównaniu do struktury bez konwertera adresów (*CMCU_CS)*. Jest to spowodowane faktem, że pamięć tego układu została zaimplementowana z wykorzystaniem dedykowanych bloków pamięci, natomiast układu *CMCU_CS* z wykorzystaniem elementów *LUT*.

Korzystając z oprogramowania *Xilinx ISE 8.2i* możemy określić w jaki sposób ma zostać zaimplementowana pamięć układu. Dla omawianych struktur ustawiono atrybuty, które jako priorytetowe przyjmują wykorzystanie dedykowanych bloków pamięci. Takie postępowanie, umożliwiło implementację projektowanych układów na dowolnej platformie sprzętowej.

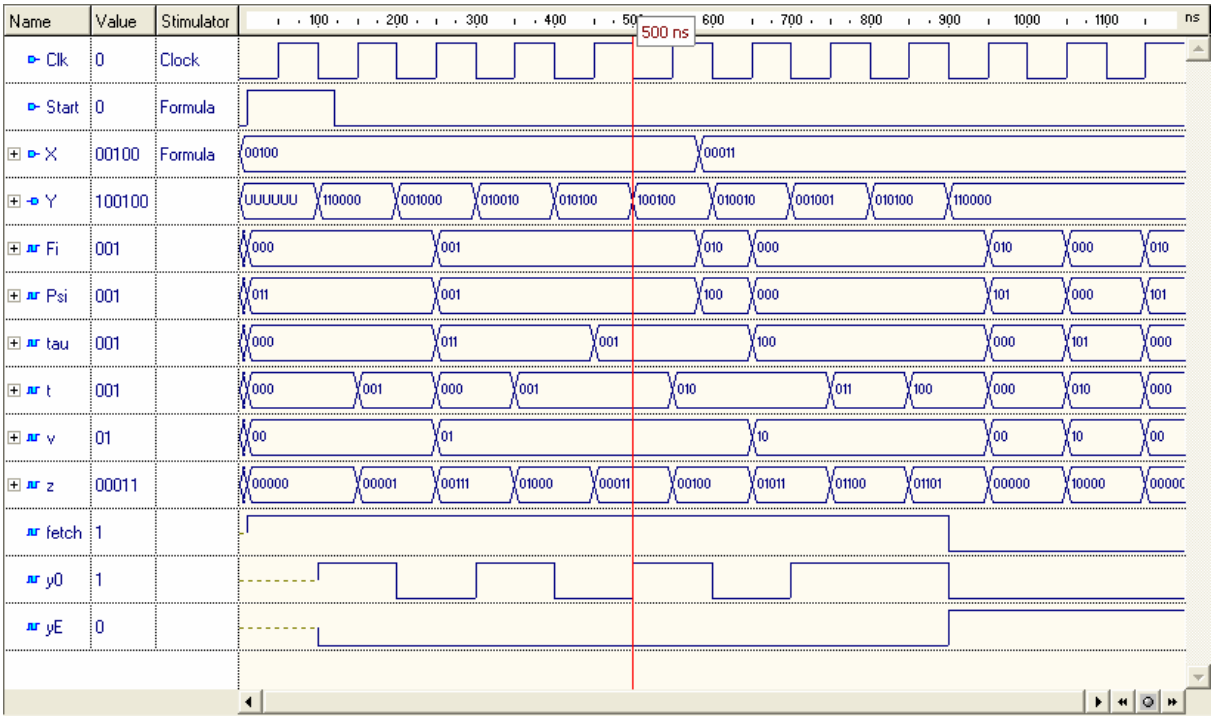
Dla układów *CMCU_EOLC_XM, CMCU_EOLC_EM* zgodnie z oczekiwaniami zastosowanie konwertera adresów *AT*, powoduje wydłużenie czasu trwania cyklu, a co za tym idzie zmniejszenie częstotliwości maksymalnej.

6.4. Symulacja działania układu

Na rysunku (6.6) zaprezentowano przebiegi czasowe symulacji działania układu *CMCU_AT* dla sieci działań z rysunku 4.5.

Sygnałem zegarowym jest sygnał *Clk*. Sygnał *Start* jest sygnałem zerującym (ustawiającym układ w stanie początkowym odpowiadającym pierwszemu blokowi operacyjnemu sieci działań). Sygnał *X* jest wektorem sygnałów wejściowych (odpowiadającym warunkom bloków warunkowych). Sygnał *Y* jest wektorem sygnałów wyjściowych (odpowiadających mikrooperacjom). Sygnał *Fi* jest wektorem wzbudzeń licznika *CT*. Sygnał *Psi* jest wektorem wzbudzeń rejestru *RG*. Sygnał *tau* jest wektorem wzbudzeń konwertera kodu *TC*. Sygnał t jest sygnałem wyjściowym z licznika *CT*. Sygnał *v* jest sygnałem sprzężenia zwrotnego z konwertera kodu *TC* do układu kombinacyjnego *CC*. Sygnał *z* jest sygnałem adresującym pamięć *AT*. Sygnał *fetch* odpowiada za zezwolenie odczytu z pamięci *CM*. Sygnał y_0 jest sygnałem zezwalającym na inkrementacje licznika *CT*. Sygnał y_E jest sygnałem zatrzymującym działanie układu.

Zaprezentowane przebiegi czasowe odpowiadają przejściu układu *CMCU_AT* poprzez następujące bloki sieci działań: b_1, b_2, b_8, b_9, b_4, b_5, b_{12}, b_{13}, b_{14}. Jak widać realizacja przejścia przez wiele kolejno następujących bloków warunkowych dokonywana jest w jednym cyklu.



Rys. 6.6. Przebiegi czasowe

7. Podsumowanie

Niniejsza praca jest wynikiem prowadzonych badań nad zmniejszeniem rozmiaru pamięci jednostki sterującej implementowanej jako mikroprogramowany układ sterujący ze współdzieleniem kodów. W celu potwierdzenia tezy zostało zaproponowanych i opracowanych siedem metod syntezy mikroprogramowanego układu sterującego. Wszystkie metody bazują na wprowadzeniu do mikroprogramowanego układu sterującego bloku konwertera adresów, co umożliwia zastosowanie metody współdzielenia kodów dla dowolnej sieci działań. Takie podejście umożliwia minimalizację ilości elementów *LUT* w układzie generującym funkcje wzbudzeń automatu (dzięki współdzieleniu kodów) oraz zmniejszenie rozmiaru pamięci (dzięki konwerterowi adresów).

Zaproponowane w rozprawie metody mogą być zastosowane dla wszystkich znanych modyfikacji mikroprogramowanego układu sterującego ze współdzieleniem kodów (Barkalov, 2002; Barkalov, Wiśniewski 2004; Barkalov i in., 2005) np.:

- *CMCU* z optymalnym kodowaniem pseudoekwiwalentnych łańcuchów;
- *CMCU* z optymalnym kodowaniem pseudoekwiwalentnych elementarnych łańcuchów;
- *CMCU* z przekształceniem kodu elementarnego łańcucha na kod klas pseudoekwiwalentnych elementarnych łańcuchów.

Praktycznym potwierdzeniem tezy jest realizacja autorskiego oprogramowania *fca2cmcu*, umożliwiającego implementację jednostki sterującej za pomocą wybranych metod. Ponadto przeprowadzono implementację przykładowych modeli w strukturach programowalnych – jako docelowe wybrano popularne i dostępne autorce układy programowalne *FPGA* firmy *Xilinx*.

W ramach pracy doktorskiej przeprowadzone zostały testy z wykorzystaniem zestawu sieci testowych, które potwierdziły tezę pracy.

W zamierzeniu autorki jest dalsze rozwijanie opracowanego systemu. Kolejnym krokiem będzie opracowanie systemu ekspertowego, który umożliwi automatyczny wybór metody projektowania, która zapewni optymalne wykorzystanie elementów *LUT* oraz bloków pamięci dla projektowanego układu. Ponadto prowadzone są badania nad przekształceniem początkowej sieci działań w sposób umożliwiający optymalne kodowanie *OLC* oraz *EOLC*.

Najważniejsze częściowe wyniki pracy doktorskiej zostały przedstawione w trzech artykułach oraz trzech referatach na konferencjach o zasięgu międzynarodowym.

Bibliografia

Adamski M., Stryjski R. (1982): *Dekompozycja sieci Petriego metodą kolorowania*. Wyższa Szkoła Inżynierska w Zielonej Górze, Raporty, Zielona Góra.

Adamski M. (1990): *Projektowanie układów cyfrowych systematyczną metodą strukturalną*. – Seria Monografie, nr 49, Wydawnictwo Wyższej Szkoły Inżynierskiej w Zielonej Górze.

Adamski M. (1998): *Metodologia projektowania reprogramowalnych sterowników logicznych z wykorzystaniem elementów CPLD i FPGA*. Reprogramowalne Układy Cyfrowe - RUC: Materiały I Krajowej Konferencji Naukowej, Szczecin, Polska, s. 15 – 22.

Adamski M. (1998a): *SFC, Petri Nets and Application Specific Logic Controllers*, IEEE International Conference on Systems, Man and Cybernetics - SMC '98, San Diego, USA, Vol.1, s. 728 – 733.

Adamski M., Chodań M. (2000): *Modelowanie układów sterowania dyskretnego z wykorzystaniem sieci SFC*, Wydawnictwo Politechniki Zielonogórskiej, Zielona Góra.

Adamiski M., Barkalov A. (2006): *Architectural and sequential synthesis of digital devices*. University of Zielona Góra, 199 s.

Albicki A. (1980): *Konstruktywne metody realizacji automatów*. – Prace Naukowe, Elektronika, Wydawnictwa Politechniki Warszawskiej, z. 44.

Banaszak Z., Kuś J., Adamski M. (1993): *Sieci Petriego. Modelowanie, Sterowanie i Synteza Systemów Dyskretnych*, Wydawnictwo Wyższej Szkoły Inżynierskiej, Zielona Góra.

Baranov S. (1994): *Logic Synthesis for Control Automata*. Boston: Kluwer Academic Publishers.

Barkalov A.A. (1994): *Structures of the multilevel circuits of microprogram automata on PLA*, Cybernetics and System Analysis, No. 4, s. 22-29.

Barkalov A.A., Palagin A. V. (1997): *Synthesis of microprogram control units*, IC NAS of Ukraine, Kiev.

Barkalov A.A. (1998): *Principles of optimization of logical circuit of Moore finite-state machine*. Cybernetics and system analysis, No. 1, s. 65-72.

Barkalov A. A. (2002): *Synthesis of Control Units on PLDs*, Donetsk: DonNTU.

Barkalov A. A. (2003): *Synthesis of Operational Units*, Donetsk: DonNTU, s. 306.

Barkalov A. A. (2005): *Synteza jednostek sterujących w strukturach programowalnych*, KNWS'05: Materiały II konferencji naukowej, Zielona Góra.

Barkalov A.A., Salomatin V.A., Starodubow K.E (1991): *System of CAD for design of control units with PALs and ROMs*. Control systems and machines, No. 5, s.22-26

Barkalov A., Wisniewski R. (2004): *Optimization of compositional microprogram control unit with elementary operational linear chains*. Control Systems and Computers, No 4.

Barkalov A.A., Wisniewski R., Titarenko L. (2005): *Synthesis of compositional microprogram control unit on FPGA*. Proceedings of the 12th International Conference MIXDES 2005, Vol. 1, s. 205-208, Krakow, Poland.

Barkalov A., Titarenko L., Kołopieńczyk M. (2006): *Optymalizacja jednostki sterującej poprzez zastosowanie metody współdzielenia kodów*. Pomiary Automatyka Kontrola, nr 7, wyd. spec., s. 29-31.

Barkalov A., Titarenko L., Kołopieńczyk M. (2006a): *Optymalizacja rozmiaru pamięci jednostki sterującej poprzez zastosowanie metody współdzielenia kodów*. Informatyka Teoretyczna i Stosowana, R. 5, nr 9, s. 105-11.

Barkalov A., Titarenko L., Kołopieńczyk M. (2006b): *Optimization of circuit of control unit with code sharing*. Proceedings of IEEE East-West Design & Test Workshop - EWDTW '06. Sochi, Rosja- Kharkov, s. 171-174.

Barkalov A., Titarenko L., Kołopieńczyk M. (2006c): *Optimization of control memory size of control unit with codes sharing*. Mixed Design of Integrated Circuits and Systems - MIXDES 2006: proceedings of the international conference. Gdynia, Polska, Łódź, s. 354-358.

Barkalov A., Titarenko L., Kołopieńczyk M. (2006d): *Optimization of control unit with code sparing*. Discrete-Event System Design - DESDes '06 : a proceedings volume from the 3rd IFAC Workshop. Rydzyna, Polska, 2006 .- Zielona Góra : International Federation of Automatic Control by University of Zielona Góra Press, s. 195-200.

Barkalov A., Węgrzyn M. (2006): *Design of control units with programmable logic*. University of Zielona Góra Press.

Belhadj H., Gerbaux L., Bertrand M.C., Saucier G. (1993): *Specification and Synthesis of Communicating Finite State Machines*, w: Saucier G., Trilhe J., (red.), Synthesis for Control Dominated Circuits, Elsevier Science Publishers B.V., North-Holland, IFIP, s. 91-102.

Benso A., Chiusano S., Prinetto P. (2001): *A self-repairing execution unit for microprogrammed processors*, IEEE Micro, Vol. 21, Issue 5, s. 16-22.

Blanchard F. (1979): *Comprende, maitriser et appliquer le Grafcet*, Capadeus, Tolouse.

Blinski K., Dagless E. L., Saul J. M., Adamski M. (1994): *Parallel controller synthesis from a Petri net specification*, in Proceedings of the European design automation conference EURO-DAC'94 (IEEE Computer Society Press, 1994), s. 96-101.

Bolton M. (1991): *Digital Systems Design with Programmable Logic*. Addison-Wesley, Wokingham.

Bomar B.W. (2002): *Implementation of microprogrammed control in FPGAs*, IEEE Transactions on Industrial Electronics,Vol. 49, Issue 2, s. 415-422.

Brown S., Vranesic Z. (2000): *Fundamentals of Digital Logic*. New York: McGraw Hill.

Chang K. C. (1999): *Digital Systems Design with VHDL and Synthesis: An Integrated Approach.* IEEE Computer Society, Los Alamitos.

DeMicheli G. (1994): *Synthesis and Optimization of Digital Circuits*. McGraw Hill, NY.

Gajski D. D., Vahid F., Narayan S. i Gong J. (1994): *Specification and Design of Embedded Systems*. - Prentice Hall, Englewood Cliffs, New Jersey.

Gorzałczany M. B. (2000): *Układy cyfrowe metody syntezy* Tom II *Układy sekwencyjne układy mikroprogramowane*, Wydawnictwo Politechniki Świętokrzyskiej, Kielce.

Grushnitsky R., Mursaev A., Ugrjumov E. (2002): *Development of systems on chips with programmable logic*, SPb: BHV – Petersburg, s. 608.

Halang W.A. (1989): *Languages and tools for the graphical and textual system independent programming of programmable logic controllers*, Microprocessing and microprogramming, North Holland, Vol.27, s. 583-590.

http1: http://www.altera.com

http2: http://www.latticesemi.com

http3: http://www.xilinx.com

In-Cheol Park, Se-Kyoung Hong, Chong-Min Kyung (1994): *Two Complementary Approaches for Microcode Bit Optimization*, IEEE Transactions on Computers, Vol. 43, Issue 2, s. 234-239.

International Electrotechnical Commission (1992): *International standard IEC 1131-3*, Programmable Controllers, Part 3: Programming Languages, Geneva.

Iwai H. (2004): Future CMOS Scaling. Szczecin: Proceeding of the 11th International Conference "Mixed Design of Integrated Circuits and Systems".

Jiang J., Holding D.J. (1996): *The Formalisation and Analysis of Sequential Function Chart using a Petri Net Approach*, proceedings of the 13th World Congress of IFAC, San Francisco, CA, USA, Vol. J – discrete Event Systems, s. 513-518.

Jenkins J. (1994): *Design with FPGAs and CPLDs*. New York: Prentice Hall.

Kalinowski J. (1984): *Wykorzystanie sieci Petriego do projektowania systemów cyfrowych*, Rozprawa doktorska, Politechnika Warszawska, Warszawa.

Kania D. (2004): *Synteza logiczna przeznaczona dla matrycowych struktur programowalnych typu PAL*. Gliwice: Zeszyty naukowe Politechniki Śląskiej.

Kamionka-Mikuła H., Małysiak H., Pochopień B. (2004): *Układy cyfrowe. Teoria i przykłady*. Wydawnictwo Pracowni Komputerowej Jacka Skalmierskiego, Gliwice.

Kamionka-Mikuła H., Małysiak H., Pochopień B. (2006): *Synteza i analiza układów cyfrowych*. Wydawnictwo Pracowni Komputerowej Jacka Skalmierskiego, Gliwice.

Kozłowski T., Dagless E.L., Saul J.M., Adamski M., Szajna J. (1995): *Parallel controller synthesis using Petri nets*, IEE Proceedings-E, Computers and Digital Techniques, Vol.142, No.4, s.263-271.

Lewis R.W. (1995): *Programming industrial control systems using IEC 1131-3*, The Institution of Electrical Engineers, London.

Łabiak G. (2005): *Wykorzystanie hierarchicznego modelu współbieżnego automatu w projektowaniu sterowników cyfrowych*. Oficyna Wydaw. Uniwersytetu Zielonogórskiego, s.156.

Łuba T. (2003): *Synteza układów cyfrowych*, Praca zbiorowa pod redakcją prof. Tadeusza Łuby Warszawa: WKŁ, s. 296.

Łuba T. (2005): *Synteza układów logicznych*. Oficyna wydawnicza Politechniki Warszawskiej.

Łuba T., Rawski M., Jachna Z. (2002): *Functional Decomposition as a Universal Method of Logic Synthesis for Digital Circuits*. Proc. Of the 9th Int. Conf. Mixed Design Integrated Circuits and Systems, s 285-290, Wrocław.

Łuba T., Zbierzchowski B. (2002): *Układy logiczne*, Wyższa Szkoła Informatyki Stosowanej i Zarządzania, Warszawa 2002.

Małysiak H., Pochopień B. (red.) (2002): Układy cyfrowe. Zadania. Wydawnictwo Politechniki Śląskiej, Gliwice.

Maxfield C. (2004): *The Design Warriors Guide to FPGAs*. Amsterdam: Elseveir.

Misiurewicz P. (1982): *Podstawy techniki cyfrowej*. WNT, Warszawa.

Molski M. (1986): Modułowe i mikroprogramowalne układy cyfrowe. WKŁ, Warszawa.

Murata T. (1989): *Petri Nets: Properties, Analysis and Applications*, Proceedings of the IEEE, Vol.77, No.4, s. 541-580.

Peterson J.L. (1981): *Petri Net Theory and The Modeling of Systems*, Prentice-Hall, Inc., Englewood Cliffs, N.J.

Pochopień B. (2004): *Podstawy techniki cyfrowej*. Wyższa Szkoła Biznesu, Dąbrowa Górnicza.

Pochopień B. (2005): *Podstawy techniki cyfrowej w zadaniach.* Wyższa Szkoła Biznesu, Dąbrowa Górnicza.

Reisig W. (1988): *Sieci Petriego. Wprowadzenie*, Wydawnictwa Naukowo - Techniczne, Warszawa.

Salcic Z. (1998): *VHDL and FPGAs in digital systems design, prototyping and customization*. Boston: Kluwer Academic Publishers.

Salcic Z., Samailagic A. (1997): *Digital systems design and prototyping using field programmable logic*. Kluwer Academic Publishers, Boston.

Sasao T. (1999): *Switching theory for logic synthesis*, Kluwer Academic Publishers - s. 362.

Solovjev V. (2001): *Design of digital systems using the programmable logic integrate circuits*. Moscow: Hot line – Telecom.

Titarenko L. Kołopieńczyk M. (2006): *Optymalizacja mikroprogramowanego układu sterującego poprzez zastosowanie współdzielenia kodów*. Pomiary Automatyka Kontrola, nr 6, wyd. spec., s. 53-.55.

Traczyk W. (1986): *Układy cyfrowe. Podstawy teoretyczne i metody syntezy*. Wydawnictwo Naukowo-Techniczne, Warszawa.

Varadharajan V., Baker K.D. (1987): *Directed graph based representation for software system design.* – Software Engineering J., s. 21-28.

Wang C.Y., Roy K. (1999): *An activity-driven encoding scheme for power optimization in microprogrammed control unit*. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 7, Issue 1, s. 130-134.

Węgrzyn A. (2003): *Symboliczna analiza układów sterowania binarnego z wykorzystaniem wybranych metod analizy sieci Petriego.* Oficyna Wydaw. Uniwersytetu Zielonogórskiego, 125 s.

Wegrzyn A., Chodań M. (2000): *Formal analysis of SFC nets, ACS 2000*: Proceedings of the Seventh International Conference. Szczecin, Polska, 2000.- Szczecin: Wydaw. i Drukarnia Wydziału Informatyki Politechniki Szczecińskiej, s. 102-107.

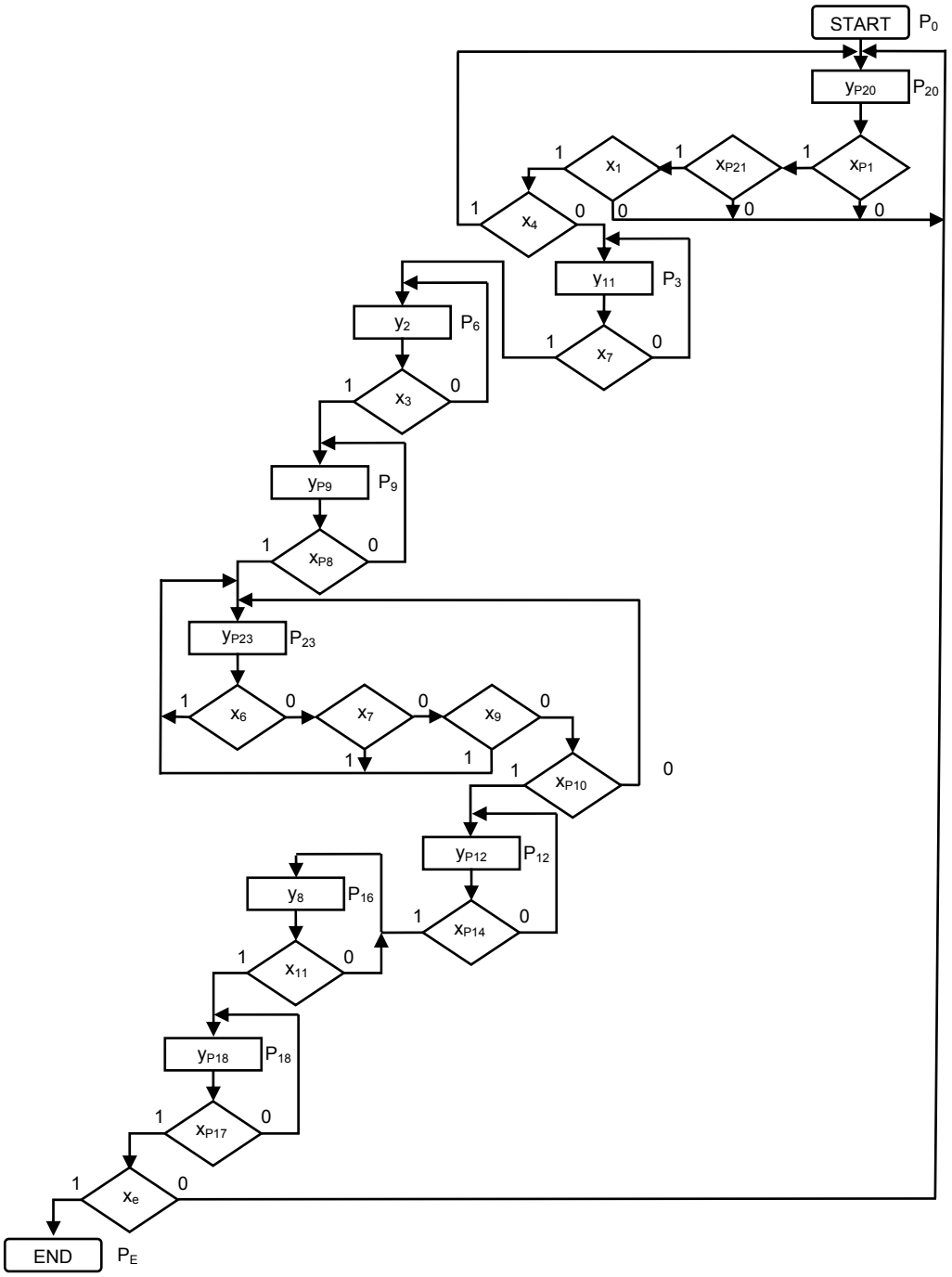
Wisniewski R. (2005): *Częściowa rekonfiguracja mikroprogramowanych układów sterujących implementowanych z wykorzystaniem struktur FPGA*, OWD 2005, Archiwum Konferencji PTETiS, Wisła.

Wiśniewski R. (2005a): *Projektowanie układów mikroprogramowanych z wykorzystaniem wbudowanych bloków pamięci w matrycach programowalnych*, KNWS' 05: materiały II konferencji naukowej, Zielona Góra.

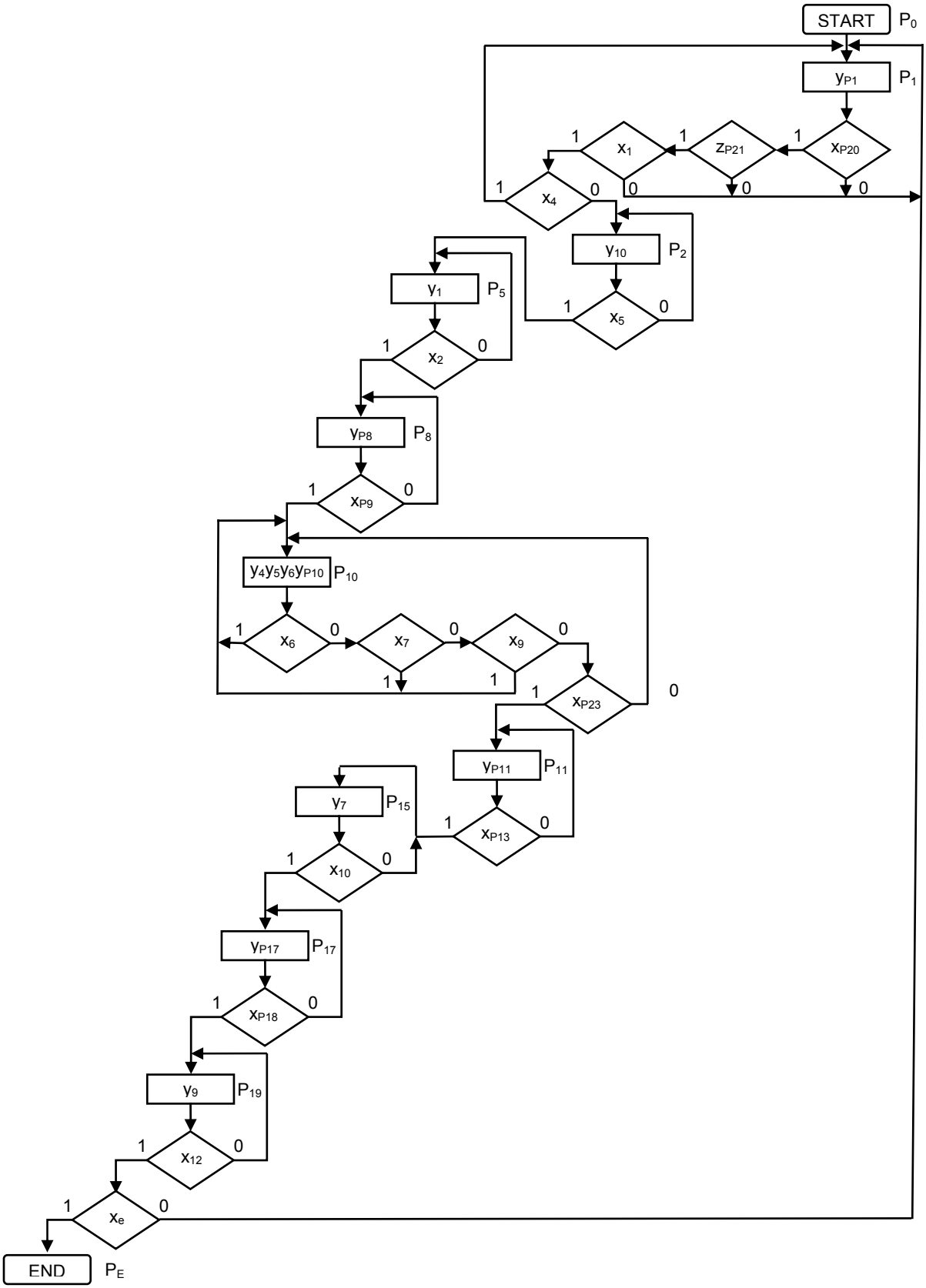
Zieliński C. (2003): *Podstawy projektowania układów cyfrowych*. Wydawnictwo Naukowe PWN, Warszawa.

Dodatek A – Implementacja automatu współbieżnego w CMCU

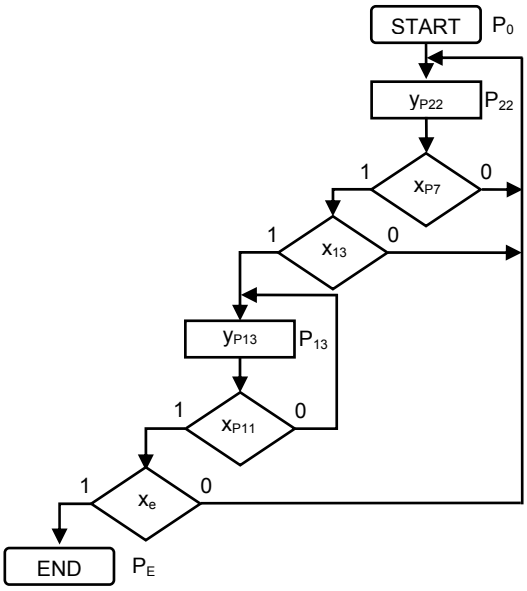
W wyniku dekompozycji automatu współbieżnego opisanego siecią Petriego (Rys. 2.6) otrzymano cztery sekwencyjne automaty składowe (Rys. 2.9a, 2.9b, Rys. 2.10a, 2.10b), które następnie przekształcono na odpowiadające im sieci działań (Rys. 2.12, Rys. A1, A2, A.3).



Rys. A.1. Sieć działań dla automatu z rysunku 2.9b



Rys. A.2. Sieć działań dla automatu z rysunku 2.9a



Rys. A.3. Sieć działań dla automatu z rysunku 2.10b

Każdy z tych automatów implementowany jest jako mikroprogramowany układ sterujący: CMCU I – automat realizujący napełniane zbiornika 1; CMCU II - automat realizujący napełniane zbiornika 2; CMCU III – automat realizujący działanie wózka; CMCU IV – automat komunikacyjny.

W tabeli A.1. przedstawione zostały rozmiary pamięci mikroprogramowanych układów sterujących wygenerowanych przez program *fca2cmcu* dla poszczególnych sieci działań uzyskanych w wyniku dekompozycji automatu współbieżnego opisanego siecią Petriego.

Tabela A.1. Rozmiary pamięci [bity]

Układ	**Sieć testowa**	**CMCU**				**CMCU_EOLC**			
		CS	AT	XM	EM	CS	AT	XM	EM
CMCU I	fca_1	448	224	224	192	448	224	224	192
CMCU II	fca_2	320	160	160	128	320	160	160	128
CMCU III	fca_2	96	48	48	32	96	48	48	32
CMCU IV	fca_3	32	16	16	8	32	16	16	8

Dodatek B – Pliki wyjściowe

Przykładowy plik wynikowy programu *fca2cmcu*.

```
-- CMCU_OLC_Ux_CC -------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity CC is
  port(
    x : in  STD_LOGIC_VECTOR(1 to 5);
    v : in  STD_LOGIC_VECTOR(1 to 2);
    o : out STD_LOGIC_VECTOR(1 to 6)
  );
end CC;

architecture CC_arch of CC is
begin
  o(1) <= (not v(1) and not v(2) and not x(1) and not x(2) and not x(3)) or
          (not v(1) and     v(2) and not x(3) and     x(4) and     x(5)) or
          (not v(1) and     v(2) and not x(3) and     x(4) and not x(5)) or
          (not v(1) and     v(2) and not x(3) and not x(4));
  o(2) <= (not v(1) and not v(2) and not x(1) and     x(2)) or
          (not v(1) and not v(2) and not x(1) and not x(2) and     x(3));
  o(3) <= (not v(1) and not v(2) and     x(1)) or
          (not v(1) and not v(2) and not x(1) and not x(2) and     x(3)) or
          (not v(1) and not v(2) and not x(1) and not x(2) and not x(3)) or
          (not v(1) and     v(2) and     x(3)) or
          (not v(1) and     v(2) and not x(3) and not x(4));
  o(4) <= '0';
  o(5) <= (not v(1) and not v(2) and not x(1) and not x(2) and not x(3)) or
          (not v(1) and     v(2) and not x(3) and     x(4) and     x(5));
  o(6) <= (not v(1) and     v(2) and     x(3));
end CC_arch;
-- end of CMCU_OLC_Ux_CC ------------------
-- CMCU_OLC_Ux_CT -------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

entity CT is
  port(
    clk   : in  STD_LOGIC;
    reset : in  STD_LOGIC;
    count_nload : in  STD_LOGIC;
    d     : in  STD_LOGIC_VECTOR(1 to 3);
    q     : out STD_LOGIC_VECTOR(1 to 3)
  );
end CT;

architecture CT_arch of CT is
signal state : STD_LOGIC_VECTOR(1 to 3);
begin
  process(clk, reset, count_nload) is
  begin
    if reset = '1' then
      state <= (others => '0');
    elsif clk'Event and clk = '1' then
      if count_nload = '0' then
        state <= d;
      else
        state <= state + '1';
      end if;
    end if;
  end process;
  q <= state;
end CT_arch;
-- end of CMCU_OLC_Ux_CT ------------------
-- CMCU_[E]OLC_Ux_RG ----------------------
library IEEE;
use IEEE.STD_LOGIC_1164.all;
```

```
entity RG is
  port(
    clk   : in  STD_LOGIC;
    reset : in  STD_LOGIC;
    nload : in  STD_LOGIC;
    d     : in  STD_LOGIC_VECTOR(1 to 3);
    q     : out STD_LOGIC_VECTOR(1 to 3)
  );
end RG;

architecture RG_arch of RG is
begin
  q <= (others => '0') when reset = '1' else
       d when (clk'Event) and (clk = '1') and (nload = '0');
end RG_arch;
-- end of CMCU_[E]OLC_Ux_RG ---------------
-- CMCU_OLC_Ux_TC -------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity TC is
  port(
    di : in  STD_LOGIC_VECTOR(1 to 3);
    do : out STD_LOGIC_VECTOR(1 to 2)
  );
end TC;

architecture TC_arch of TC is
begin
  do(1) <= (    di(1) and not di(2) and not di(3)) or
           (    di(1) and not di(2) and     di(3));
  do(2) <= (not di(1) and not di(2) and     di(3)) or
           (not di(1) and     di(2) and not di(3)) or
           (not di(1) and     di(2) and     di(3));
end TC_arch;
-- end of CMCU_OLC_Ux_TC ------------------
-- CMCU_[E]OLC_Ux_TF ----------------------
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity TF is
  port(
    set : in  STD_LOGIC;
    rst : in  STD_LOGIC;
    q   : out STD_LOGIC
  );
end TF;

architecture TF_arch of TF is
begin
  q <= '1' when set = '1' else
       '0' when rst = '1';
end TF_arch;
-- end of CMCU_[E]OLC_Ux_TF ---------------
-- CMCU_OLC_U2_AT -------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity AT is
  port(
    tau : in  STD_LOGIC_VECTOR(1 to 3);
    t   : in  STD_LOGIC_VECTOR(1 to 3);
    z   : out STD_LOGIC_VECTOR(1 to 5)
  );
end AT;

architecture AT_arch of AT is
begin
  z(1) <= (    tau(1) and not tau(2) and     tau(3) and not t(1) and     t(2) and not t(3)) or
          (    tau(1) and not tau(2) and     tau(3) and not t(1) and     t(2) and     t(3));
  z(2) <= (not tau(1) and     tau(2) and     tau(3) and not t(1) and not t(2) and     t(3)) or
          (    tau(1) and not tau(2) and not tau(3) and not t(1) and not t(2) and not t(3)) or
          (    tau(1) and not tau(2) and not tau(3) and not t(1) and not t(2) and     t(3)) or
          (    tau(1) and not tau(2) and not tau(3) and not t(1) and     t(2) and not t(3)) or
          (    tau(1) and not tau(2) and not tau(3) and not t(1) and     t(2) and     t(3)) or
          (    tau(1) and not tau(2) and not tau(3) and     t(1) and not t(2) and not t(3)) or
          (    tau(1) and not tau(2) and     tau(3) and not t(1) and not t(2) and not t(3)) or
```

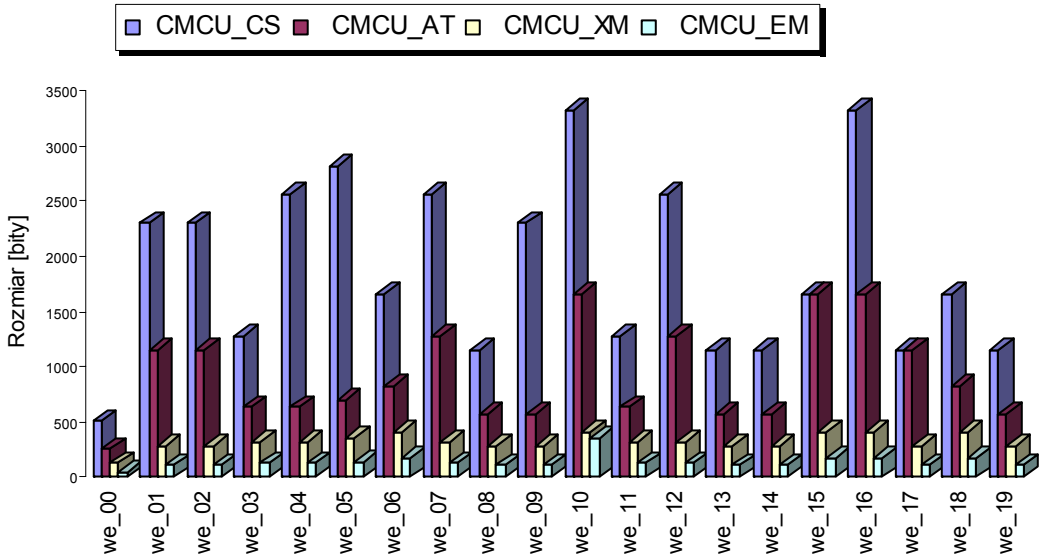
```
          (    tau(1) and not tau(2) and     tau(3) and not t(1) and not t(2) and     t(3));
  z(3) <= (not tau(1) and not tau(2) and     tau(3) and not t(1) and     t(2) and not t(3)) or
          (not tau(1) and     tau(2) and not tau(3) and not t(1) and not t(2) and not t(3)) or
          (not tau(1) and     tau(2) and not tau(3) and not t(1) and not t(2) and     t(3)) or
          (not tau(1) and     tau(2) and     tau(3) and not t(1) and not t(2) and not t(3)) or
          (    tau(1) and not tau(2) and not tau(3) and not t(1) and     t(2) and     t(3)) or
          (    tau(1) and not tau(2) and not tau(3) and     t(1) and not t(2) and not t(3)) or
          (    tau(1) and not tau(2) and     tau(3) and not t(1) and not t(2) and not t(3)) or
          (    tau(1) and not tau(2) and     tau(3) and not t(1) and not t(2) and     t(3));
  z(4) <= (not tau(1) and not tau(2) and     tau(3) and not t(1) and not t(2) and not t(3)) or
          (not tau(1) and not tau(2) and     tau(3) and not t(1) and not t(2) and     t(3)) or
          (not tau(1) and     tau(2) and not tau(3) and not t(1) and not t(2) and     t(3)) or
          (not tau(1) and     tau(2) and     tau(3) and not t(1) and not t(2) and not t(3)) or
          (    tau(1) and not tau(2) and not tau(3) and not t(1) and not t(2) and     t(3)) or
          (    tau(1) and not tau(2) and not tau(3) and not t(1) and     t(2) and not t(3)) or
          (    tau(1) and not tau(2) and     tau(3) and not t(1) and not t(2) and not t(3)) or
          (    tau(1) and not tau(2) and     tau(3) and not t(1) and not t(2) and     t(3));
  z(5) <= (not tau(1) and not tau(2) and not tau(3) and not t(1) and not t(2) and     t(3)) or
          (not tau(1) and not tau(2) and     tau(3) and not t(1) and not t(2) and     t(3)) or
          (not tau(1) and     tau(2) and not tau(3) and not t(1) and not t(2) and not t(3)) or
          (not tau(1) and     tau(2) and     tau(3) and not t(1) and not t(2) and not t(3)) or
          (    tau(1) and not tau(2) and not tau(3) and not t(1) and not t(2) and not t(3)) or
          (    tau(1) and not tau(2) and not tau(3) and not t(1) and     t(2) and not t(3)) or
          (    tau(1) and not tau(2) and not tau(3) and     t(1) and not t(2) and not t(3)) or
          (    tau(1) and not tau(2) and     tau(3) and not t(1) and not t(2) and     t(3)) or
          (    tau(1) and not tau(2) and     tau(3) and not t(1) and     t(2) and     t(3));
end AT_arch;
-- end of CMCU_OLC_U2_AT ------------------
-- CMCU_OLC_U2_CM -------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity CM is
  port(
    clk   : in  STD_LOGIC;
    fetch : in  STD_LOGIC;
    addr  : in  STD_LOGIC_VECTOR (1 to 5);
    data  : out STD_LOGIC_VECTOR (0 to 7)
  );
  attribute rom_extract : string;
  attribute rom_extract of CM: entity is "yes";
  attribute rom_style : string;
  attribute rom_style of CM: entity is "block";
end CM;

architecture CM_arch of CM is
begin
  process(clk, fetch, addr)
  begin
    if clk'event and clk = '1' and fetch = '1' then
      case addr is
        when "00000" => data <= "11100000";
        when "00001" => data <= "00010000";
        when "00010" => data <= "11100000";
        when "00011" => data <= "11001000";
        when "00100" => data <= "00100100";
        when "00101" => data <= "10111000";
        when "00110" => data <= "01100000";
        when "00111" => data <= "10100100";
        when "01000" => data <= "00101000";
        when "01001" => data <= "11100000";
        when "01010" => data <= "10100100";
        when "01011" => data <= "10010010";
        when "01100" => data <= "10101000";
        when "01101" => data <= "01100001";
        when "01110" => data <= "10010000";
        when "01111" => data <= "10111000";
        when "10000" => data <= "10100100";
        when "10001" => data <= "01001001";
        when others => data <= (others => '0');
      end case;
    end if;
  end process;
end CM_arch;
-- end of CMCU_OLC_U2_CM ------------------
-- CMCU_OLC_U2 ----------------------------
library IEEE;
```

```
use IEEE.STD_LOGIC_1164.all;

entity CMCU_OLC_U2 is
  port(
    Clk   : in  STD_LOGIC;
    Start : in  STD_LOGIC;
    X     : in  STD_LOGIC_VECTOR(1 to 5);
    Y     : out STD_LOGIC_VECTOR(1 to 6)
  );
end CMCU_OLC_U2;

architecture CMCU_OLC_U2_arch of CMCU_OLC_U2 is
component CC is
  port(
    x : in  STD_LOGIC_VECTOR(1 to 5);
    v : in  STD_LOGIC_VECTOR(1 to 2);
    o : out STD_LOGIC_VECTOR(1 to 6)
  );
end component;
component CT is
  port(
    clk   : in  STD_LOGIC;
    reset : in  STD_LOGIC;
    count_nload : in  STD_LOGIC;
    d     : in  STD_LOGIC_VECTOR(1 to 3);
    q     : out STD_LOGIC_VECTOR(1 to 3)
  );
end component;
component RG is
  port(
    clk   : in  STD_LOGIC;
    reset : in  STD_LOGIC;
    nload : in  STD_LOGIC;
    d     : in  STD_LOGIC_VECTOR(1 to 3);
    q     : out STD_LOGIC_VECTOR(1 to 3)
  );
end component;
component TC is
  port(
    di : in  STD_LOGIC_VECTOR(1 to 3);
    do : out STD_LOGIC_VECTOR(1 to 2)
  );
end component;
component TF is
  port(
    set : in  STD_LOGIC;
    rst : in  STD_LOGIC;
    q   : out STD_LOGIC
  );
end component;
component AT is
  port(
    tau : in  STD_LOGIC_VECTOR(1 to 3);
    t   : in  STD_LOGIC_VECTOR(1 to 3);
    z   : out STD_LOGIC_VECTOR(1 to 5)
  );
end component;
component CM is
  port(
    clk   : in  STD_LOGIC;
    fetch : in  STD_LOGIC;
    addr  : in  STD_LOGIC_VECTOR (1 to 5);
    data  : out STD_LOGIC_VECTOR (0 to 7)
  );
end component;

signal V     : STD_LOGIC_VECTOR(1 to 2);
signal Psi   : STD_LOGIC_VECTOR(1 to 3);
signal Tau   : STD_LOGIC_VECTOR(1 to 3);
signal Fi    : STD_LOGIC_VECTOR(1 to 3);
signal T     : STD_LOGIC_VECTOR(1 to 3);
signal Z     : STD_LOGIC_VECTOR(1 to 5);
signal Fetch : STD_LOGIC;
signal y0    : STD_LOGIC;
signal yE    : STD_LOGIC;
signal nClk  : STD_LOGIC;
begin
```

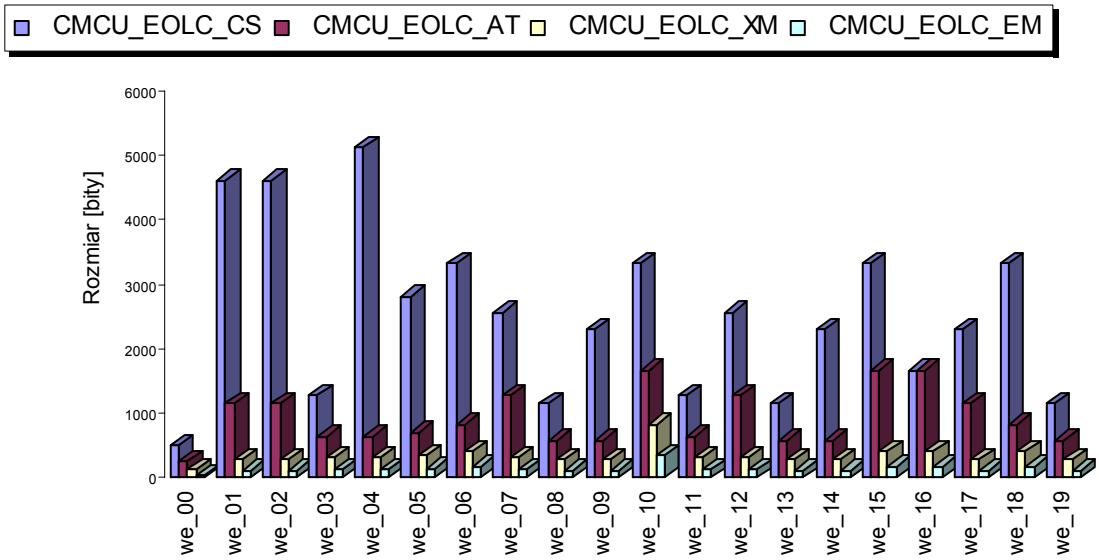
```
  nClk <= not Clk;
  CC_Block: CC port map(x => X, v => V, o(1 to 3) => Psi, o(4 to 6) => Fi);
  CT_Block: CT port map(clk => Clk, reset => Start, count_nload => y0, d => Fi, q => T);
  RG_Block: RG port map(clk => Clk, reset => Start, nload => y0, d => Psi, q => Tau);
  TC_Block: TC port map(di => Tau, do => V);
  TF_Block: TF port map(set => Start, rst => yE, q => Fetch);
  AT_Block: AT port map(tau => Tau, t => T, z => Z);
  CM_Block: CM port map(clk => nClk, fetch => Fetch, addr => Z, data(0) => y0, data(1 to 6) =>
Y, data(7) => yE);
end CMCU_OLC_U2_arch;
-- end of CMCU_OLC_U2 ----------------------
```

Dodatek C – Rozmiar pamięci CMCU

Wprowadzenie bloku konwertera adresów do struktury mikroprogramowanego układu sterującego skutkuje zmniejszeniem rozmiaru pamięci jednostki sterującej. Zdarzają się wyjątki, dla których rozmiar pamięci jest taki sam jak dla układów bez konwertera adresów. Na wykresach z rysunków C.1 oraz C.2 zamieszczono rozmiary pamięci dla struktur bez elementarnych łańcuchów oraz z elementarnymi łańcuchami.



Rys. C.1. Rozmiar pamięci dla sieci testowych dla struktur OLC



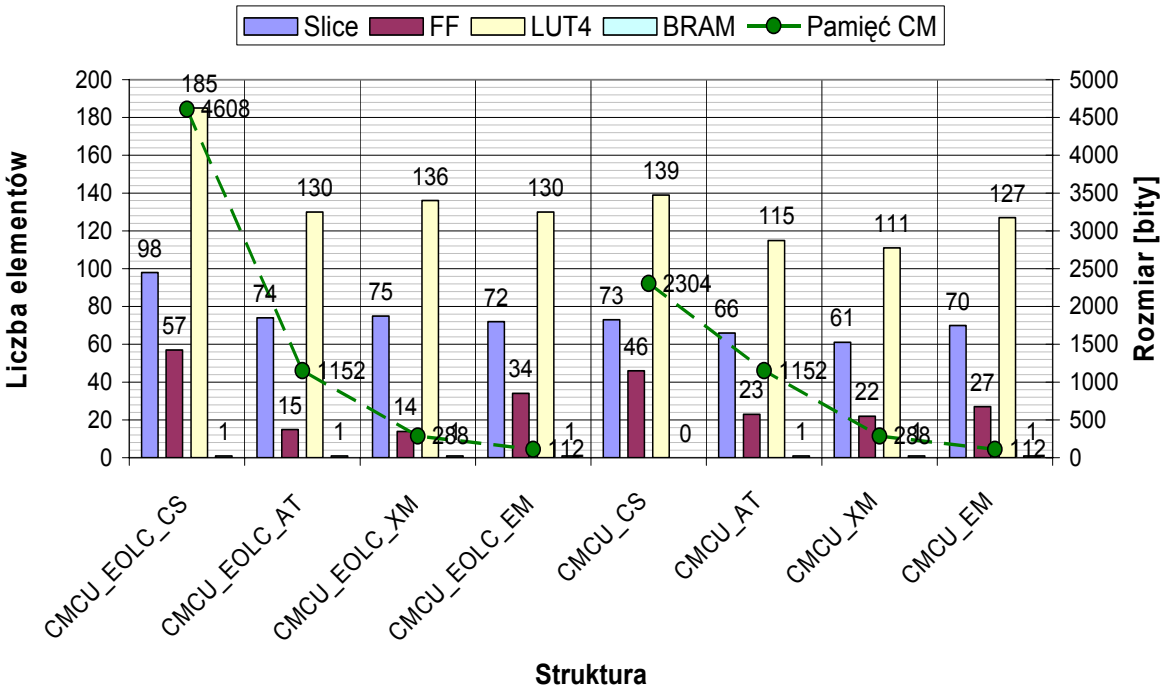
Rys. C.2. Rozmiar pamięci dla sieci testowych dla struktur z *EOLC*

Dodatek D – Zużycie zasobów sprzętowych

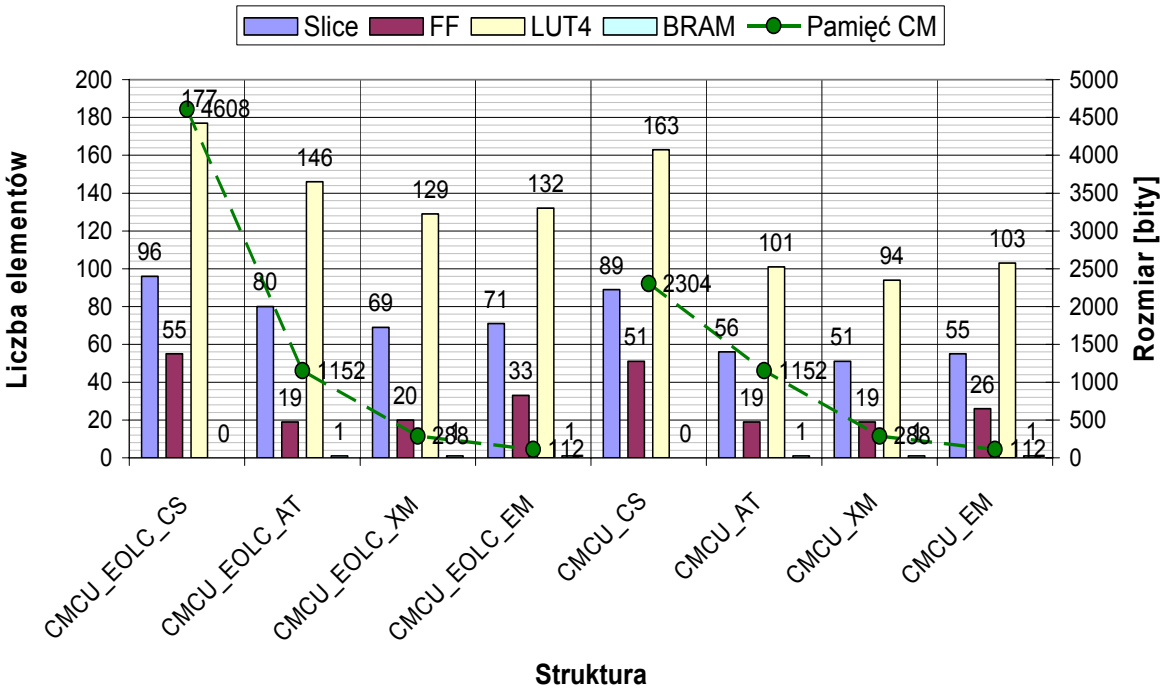
Na podstawie wykresów wygenerowanych dla zestawu sieci testowych można stwierdzić, że przy spełnieniu warunków 4.10, 4.21, 4.22 ilość elementów *LUT* oraz *BRAM* (dedykowane bloki pamięci) w zaproponowanych układach mikroprogramowanych jest nie większa niż w układzie bez bloku konwertera adresów.

Dodanie kolejnych bloków (konwertera adresów, układu *LCS*) do struktury mikroprogramowanych układów sterujących zazwyczaj nie pociąga za sobą zwiększenia zużycia zasobów sprzętowych. Również zastosowanie różnych strategii optymalizacji nie wpływa znacząco na zużycie zasobów sprzętowych.

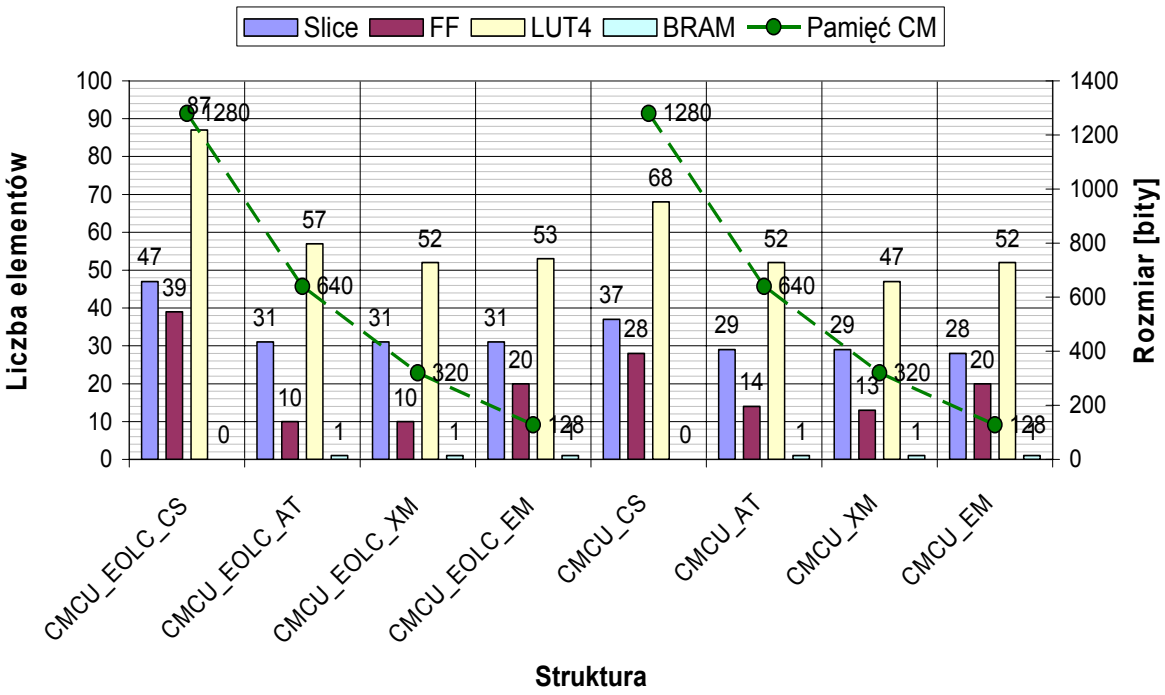
Optymalizacja *speed level 1*



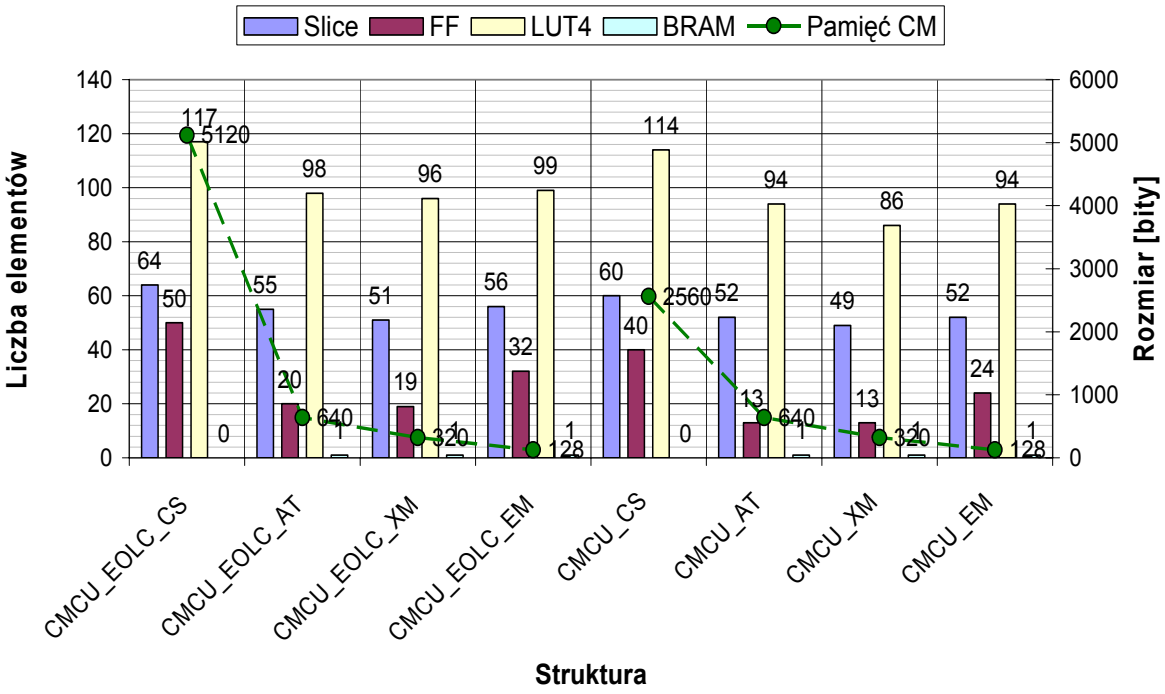
Rys. D.1. Zużycie zasobów sprzętowych dla sieci testowej *we_01 (speed level 1)*



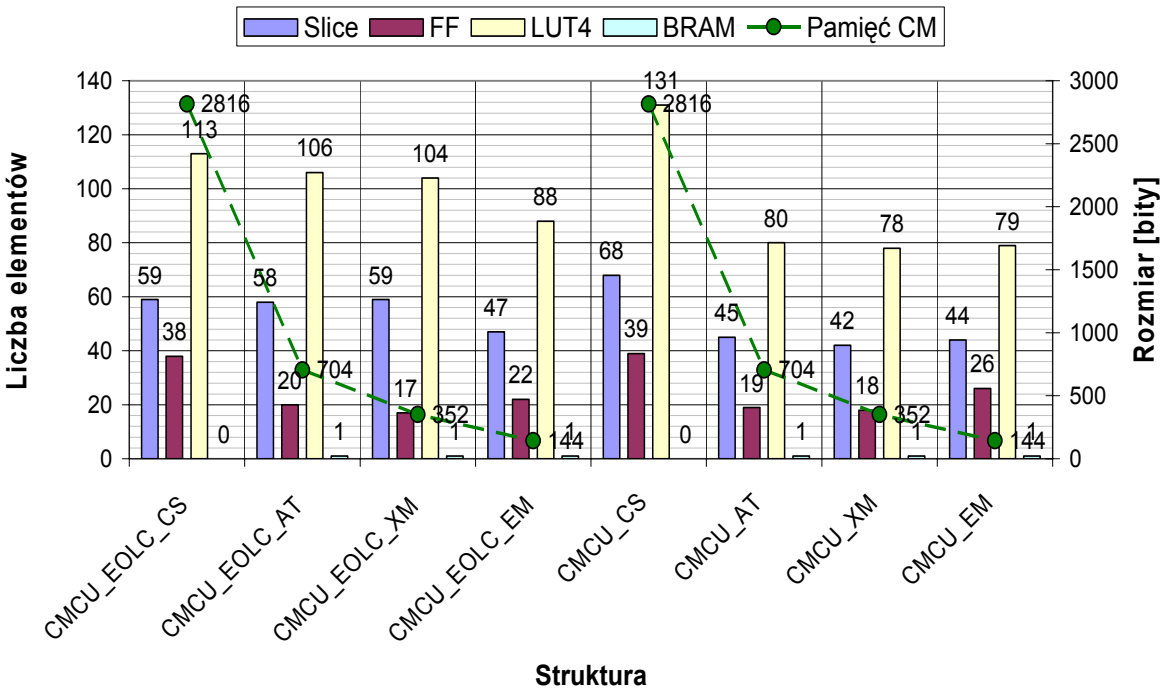
Rys. D.2. Zużycie zasobów sprzętowych dla sieci testowej *we_02 (speed level 1)*



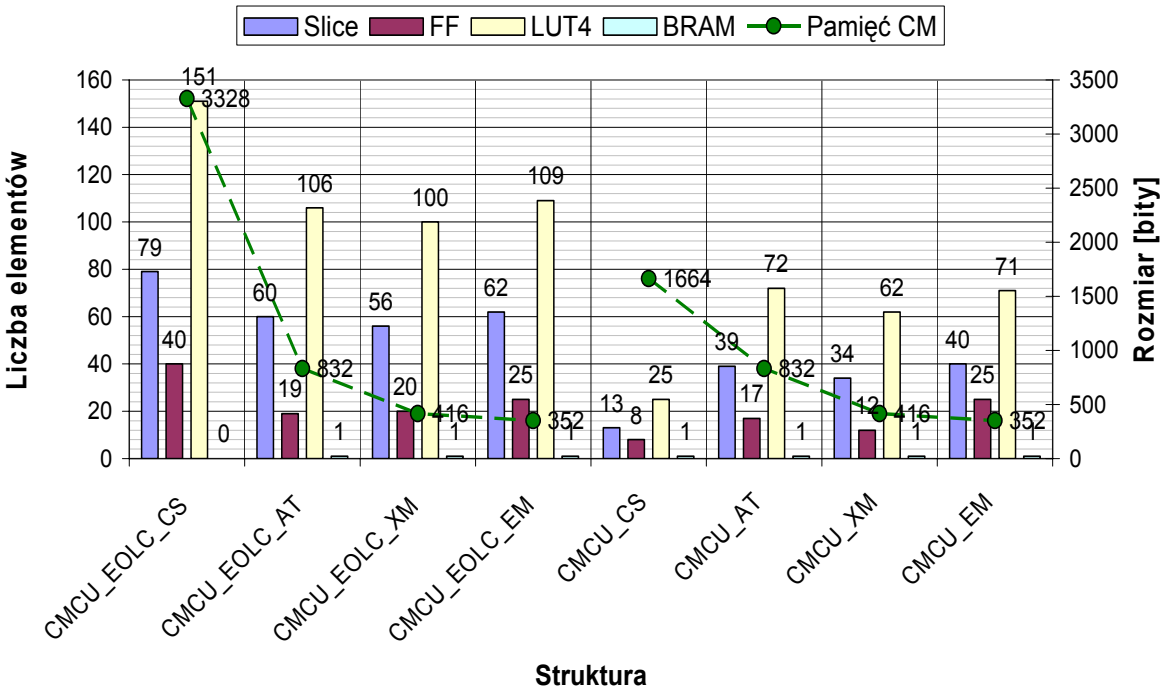
Rys. D.3. Zużycie zasobów sprzętowych dla sieci testowej *we_03 (speed level 1)*



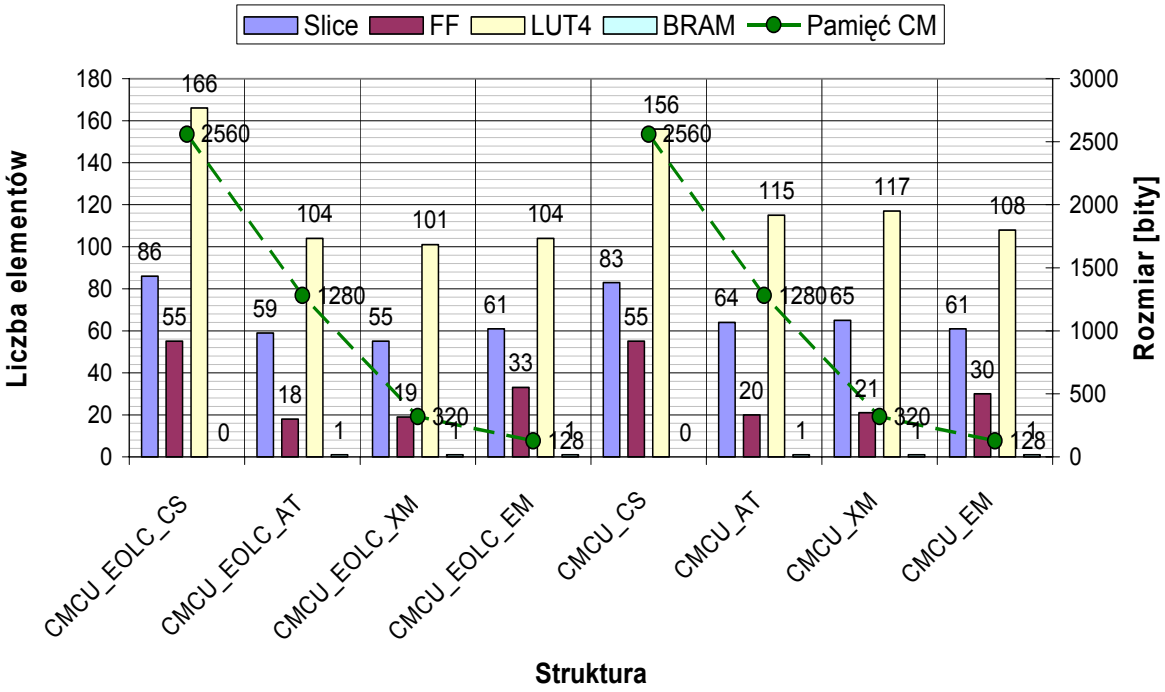
Rys. D.4. Zużycie zasobów sprzętowych dla sieci testowej j *we_04 (speed level 1)*



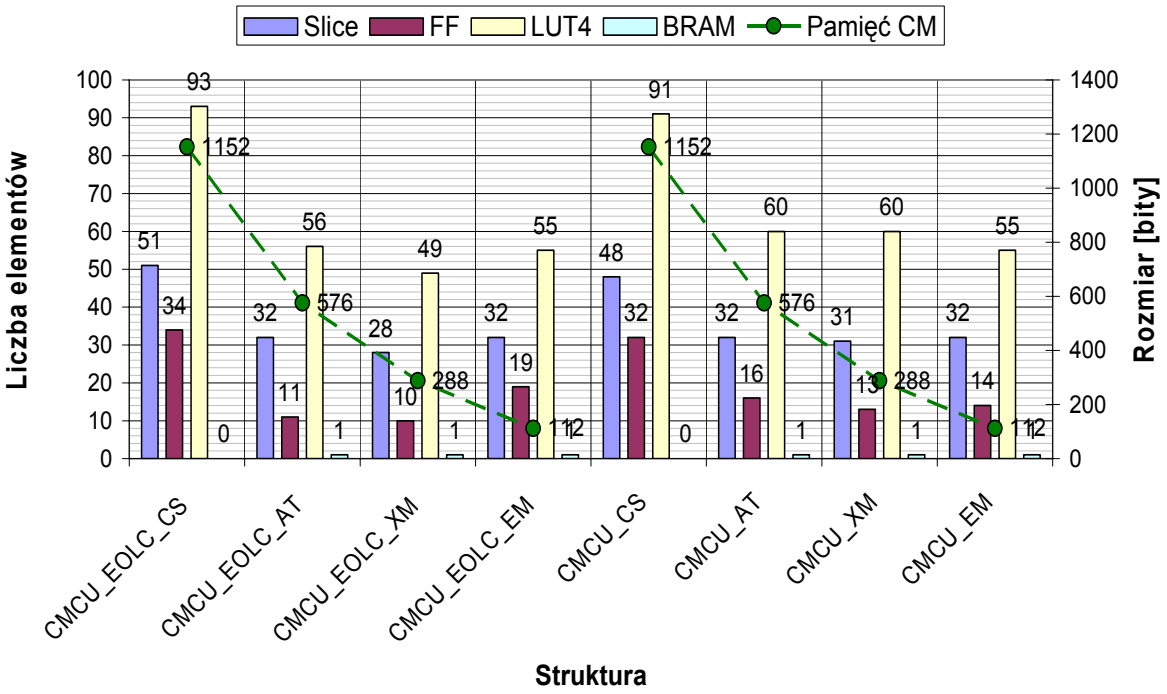
Rys. D.5. Zużycie zasobów sprzętowych dla sieci testowej *we_05 (speed level 1)*



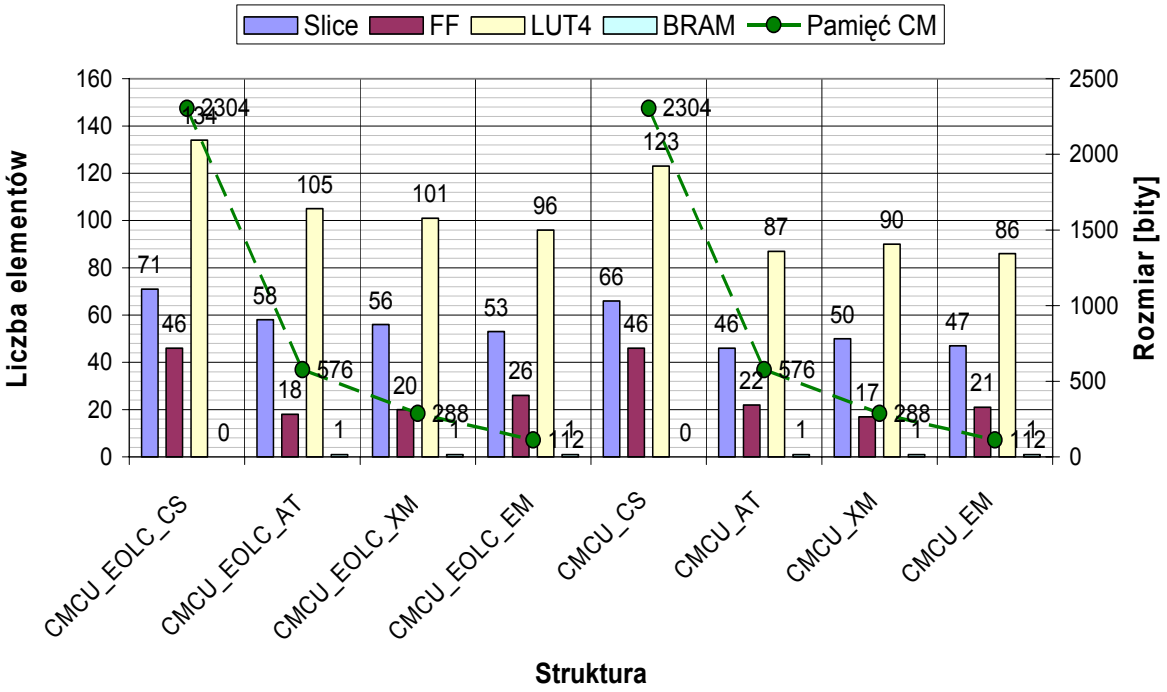
Rys. D.6. Zużycie zasobów sprzętowych dla sieci testowej *we_06 (speed level 1)*



Rys. D.7. Zużycie zasobów sprzętowych dla sieci testowej *we_07 (speed level 1)*



Rys. D.8. Zużycie zasobów sprzętowych dla sieci testowej *we_08 (speed level 1)*



Rys. D.9. Zużycie zasobów sprzętowych dla sieci testowej *we_09 (speed level 1)*

Dodatek E – Parametry czasowe dla sieci testowych

Tabela E.1. Parametry czasowe dla wybranych sieci testowych

Sieć testowa	Struktura CMCU	Minimalny czas trwania cyklu [ns]				Częstotliwość maksymalna [MHz]			
		speed 1	speed 2	area 1	area 2	speed 1	speed 2	area 1	area 2
we_01	EOLC_CS	7,194	7,194	14,373	14,373	139,003	139,003	69,573	69,573
	EOLC_AT	4,095	4,095	5,340	5,340	244,200	244,200	187,264	187,264
	EOLC_XM	4,047	4,047	5,404	5,404	247,078	247,078	185,037	185,037
	EOLC_EM	5,340	5,340	8,079	8,079	187,269	187,269	123,775	123,775
	CS	6,967	6,967	9,073	9,073	143,532	143,532	110,222	110,222
	AT	3,269	3,269	4,205	4,205	305,941	305,941	237,821	237,821
	XM	3,269	3,269	4,180	4,180	305,941	305,941	239,239	239,239
	EM	5,610	5,610	9,514	9,514	178,263	178,263	105,113	105,113
we_02	EOLC_CS	8,604	8,515	14,810	14,810	116,230	117,437	67,523	67,523
	EOLC_AT	3,778	3,778	5,728	5,728	264,725	264,725	174,580	174,580
	EOLC_XM	3,755	3,755	5,693	5,693	266,283	266,283	175,658	175,658
	EOLC_EM	6,359	6,359	10,357	10,357	157,262	157,262	96,549	96,549
	CS	7,293	7,293	9,466	9,466	137,120	137,120	105,641	105,641
	AT	3,248	3,248	4,562	4,562	307,926	307,926	219,194	219,194
	XM	3,355	3,355	4,571	4,571	298,031	298,031	218,786	218,786
	EM	5,715	5,715	7,147	7,147	174,981	174,981	139,910	139,910
we_03	EOLC_CS	6,461	6,461	6,907	6,907	154,768	154,768	144,788	144,788
	EOLC_AT	2,834	2,834	3,814	3,814	352,802	352,802	262,220	262,220
	EOLC_XM	2,867	2,867	3,814	3,814	348,772	348,772	262,220	262,220
	EOLC_EM	5,392	5,392	6,730	6,730	185,457	185,457	148,578	148,578
	CS	5,473	5,473	10,076	10,076	182,725	182,725	99,248	99,248
	AT	2,462	2,462	3,498	3,498	406,248	406,248	285,874	285,874
	XM	2,482	2,482	3,602	3,602	402,933	402,933	277,640	277,640
	EM	5,022	5,022	7,516	7,516	199,136	199,136	133,042	133,042
we_04	EOLC_CS	6,650	6,583	9,855	9,855	150,378	151,904	101,475	101,475
	EOLC_AT	3,447	3,447	5,015	5,015	290,099	290,099	199,400	199,400
	EOLC_XM	3,386	3,310	4,929	4,929	295,360	302,115	202,879	202,879
	EOLC_EM	5,988	5,988	8,832	8,832	166,995	166,995	113,228	113,228
	CS	7,897	7,846	13,158	13,158	126,624	127,455	75,998	75,998
	AT	2,802	2,802	3,758	3,758	356,837	356,837	266,098	266,098
	XM	2,802	2,802	3,748	3,748	356,837	356,837	266,804	266,804
	EM	5,965	5,965	9,231	9,231	167,645	167,645	108,327	108,327
we_05	EOLC_CS	6,248	6,248	11,649	11,649	160,049	160,049	85,845	85,845
	EOLC_AT	3,915	3,915	5,038	5,038	255,447	255,447	198,484	198,484
	EOLC_XM	3,757	3,709	5,037	5,037	266,157	269,647	198,540	198,540
	EOLC_EM	5,392	5,392	7,514	7,514	185,443	185,443	133,079	133,079
	CS	8,260	8,260	12,122	12,122	121,060	121,060	82,497	82,497
	AT	2,679	2,679	4,220	4,220	373,336	373,336	236,965	236,965
	XM	2,679	2,679	4,337	4,337	373,336	373,336	230,595	230,595
	EM	4,729	4,699	6,094	6,094	211,452	212,802	164,083	164,083

Spis rysunków

Spis tabel