2018

Synteza logiczna
Układy sekwencyjne

TADEUSZ ŁUBA

FUNKCJA BOOLOWSKA , UKŁADY KOMBINACYJNE, MIMIMALIZACJA, REDUKCJA ARGUMENTÓW, DEKOMPOZYCJA FUNKCJONALNA I LINIOWA, ANALIZA DANYCH, REDUKCJA ATRYBUTÓW

Rewolucyjny rozwój technologii mikroelektronicznych spowodował, że projektowanie systemów cyfrowych może być realizowane wyłącznie za pomocą komputerowych narzędzi projektowania przystosowanych do przetwarzania dużych ilości różnorodnych danych. Celem materiałów dydaktycznych jest omówienie zaawansowanych metod syntezy logicznej niezbędnych zarówno w projektowaniu systemów cyfrowych jak też w analizie i eksploracji danych. Dlatego głównymi zagadnieniami omawianymi w materiałach są metody minimalizacji i dekompozycji funkcji boolowskich, jak też redukcja atrybutów i indukcja reguł decyzyjnych. Takie ujęcie tych zagadnień jest zgodne z pilną potrzebą ważnych zastosowań takich jak: dystrybucja adresów IP, skanowanie wirusów, wykrywanie niepożądanych danych, itp. Nie mniejsze potrzeby stosowania zaawansowanych metod syntezy logicznej dotyczą analizy i eksploracji danych. Inaczej mówiąc celem tego podręcznika jest przygotowanie przyszłych inżynierów do umiejętnego wykorzystania ogromnego potencjału syntezy logicznej o czym świadczą wyniki prezentowanych metod i algorytmów.

Spis treści

1 Pojęcia podstawowe

W omawianych do tej pory układach kombinacyjnych wektor wyjściowy Y_t w chwili t jest bezpośrednio zależny od wektora wejściowego X_t w chwili t. Zajmiemy się teraz układami sekwencyjnymi, dla których wektor wyjściowy Y_t w chwili t zależy od sekwencji wektorów wejściowych X_t, X_{t-1}, X_{t-2},..., w chwilach t, $t-1$, $t-2$,... .

Modelem matematycznym układu sekwencyjnego jest automat. Automat jest definiowany przez określenie:

a) zbioru liter wejściowych ***X*** (lub ***V***) i wyjściowych ***Y***,

b) zbioru stanów wewnętrznych ***S***,

c) funkcji przejść (oznaczanej δ),

d) funkcji wyjść (oznaczanej λ).

W formalnym zapisie automat A określa się jako piątkę $\langle \boldsymbol{S}, \boldsymbol{X}, \boldsymbol{Y}, \delta, \lambda \rangle$, gdzie funkcja przejść jest definiowana jako odwzorowanie δ: $\boldsymbol{S} \times \boldsymbol{X} \rightarrow \boldsymbol{S}$, natomiast funkcja wyjść λ jest odwzorowaniem λ: $\boldsymbol{S} \times \boldsymbol{X} \rightarrow \boldsymbol{Y}$ (tzw. automat Mealy'ego) lub jako λ: $\boldsymbol{S} \rightarrow \boldsymbol{Y}$ (tzw. automat Moore'a).

Ważnym pojęciem w układach cyfrowych jest pojęcie automatu zupełnego i niezupełnego. Automat, określony funkcjami przejść δ i wyjść λ nazywamy automatem zupełnym, jeśli dziedziny tych funkcji są równe zbiorom $\boldsymbol{S} \times \boldsymbol{X}$ oraz $\boldsymbol{S}$ (w przypadku funkcji wyjść automatu Moore'a). Jeśli natomiast dziedziną którejkolwiek funkcji jest podzbiór wymienionych zbiorów, to jest: $D_\delta \subset \boldsymbol{S} \times \boldsymbol{X}$, $D_\lambda \subset \boldsymbol{S} \times \boldsymbol{X}$ lub $D_\lambda \subset \boldsymbol{S}$, to taki automat nazywamy niezupełnym.

Funkcje przejść-wyjść jest najwygodniej przedstawiać za pomocą tablicy przejść-wyjść. Tablice przejść-wyjść przykładowych automatów podane w tabeli 4.1 reprezentują automat Moore'a (a) oraz automat Mealy'ego (b). Tablicom tym odpowiadają grafy automatów pokazane na rys. 4.1.

Tablica 4.1

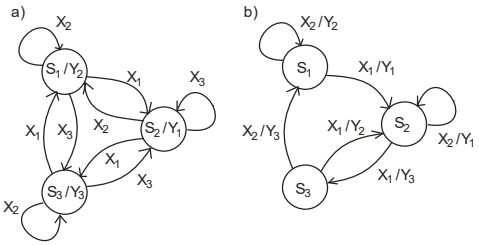
a)

	v_1	v_2	v_3	y
S_1	S_2	S_1	S_3	y_2
S_2	S_3	S_1	S_2	y_1
S_3	S_1	S_3	S_2	y_3

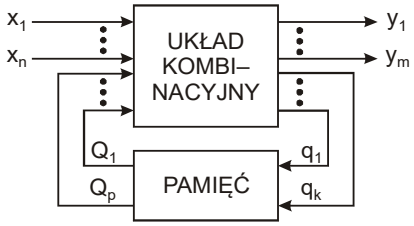
b)

	v_1	v_2	v_1	v_2
S_1	S_2	S_1	y_1	y_2
S_2	S_3	S_2	y_3	y_1
S_3	S_2	S_1	y_2	y_3

Techniczną realizacją automatu jest układ sekwencyjny zbudowany z układu kombinacyjnego i pamięci (rys. 4.2). Układ kombinacyjny automatu jest układem wielowyjściowym, wytwarzającym sygnały wyjściowe automatu y_1, …, y_m oraz sygnały wzbudzające układ pamięciowy q_1, …, q_k. Wejściami układu kombinacyjnego są sygnały wejściowe automatu x_1, …, x_n oraz sygnały wyjściowe Q_1, …, Q_p układu



Rys. 4.1. Graf automatu a) Moore'a, b) Mealy'ego.



Rys. 4.2. Układ sekwencyjny

pamięciowego. Układ kombinacyjny jest opisany tablicami wyjść i wzbudzeń (sposób tworzenia tablic wzbudzeń poznamy w dalszej części rozdziału). W zależności od realizacji bloku pamięci wyróżniamy sekwencyjne układy synchroniczne oraz sekwencyjne układy asynchroniczne.

W układach synchronicznych pamięć automatu jest zbudowana z tzw. przerzutników – automatów elementarnych synchronizowanych specjalnym sygnałem zegarowym oznaczonym *clk*. W układach asynchronicznych pamięć jest realizowana bezpośrednio przez sprzężenia zwrotne z wyjść na wejścia układu kombinacyjnego.

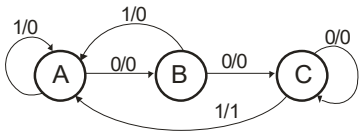
Synteza układów sekwencyjnych jest złożonym i obszernym zadaniem składającym się z kilku etapów. Zwyczajowo wyróżnia się następujące etapy:

- synteza abstrakcyjna (utworzenie tablicy przejść-wyjść),
- redukcja (minimalizacja) liczby stanów,
- kodowanie stanów, liter wejściowych i wyjściowych,
- synteza kombinacyjna (obliczanie funkcji wzbudzeń przerzutników i funkcji wyjściowych).

Omawianie procesu syntezy układów sekwencyjnych rozpoczynamy od etapu syntezy abstrakcyjnej, której celem jest utworzenie tablicy przejść-wyjść. Tablicę taką najczęściej tworzy się za pośrednictwem grafu automatu. Jest to najważniejszy a zarazem najobszerniejszy etap całego procesu syntezy.

Tworzenie grafu automatu, a tym samym odpowiedniej tablicy przejść-wyjść, jest istotną czynnością w syntezie układów sekwencyjnych. Rozważmy tablicę przejść-wyjść układu sekwencyjnego spełniającego funkcję tzw. detektora sekwencji

Układ ma badać kolejne „trójki" symboli wejściowych. Sygnał wyjściowy pojawiający się podczas trzeciego skoku układu ma wynosić 1, gdy „trójka" ma postać 001, a 0, gdy „trójka" jest innej postaci – stąd nazwa „detektor sekwencji". Graf tego automatu jest pokazany na rys. 4.3. Odpowiadająca mu tablica przejść-wyjść jest pokazana w tablicy 4.2.



Rys. 4.3. Graf detektora sekwencji

Tablica 4.2

S \ X	0	1	0	1
A	*B*	*A*	0	0
B	*C*	*A*	0	0
C	*C*	*A*	0	1

Łatwo sprawdzić, że każda sekwencja 001 na wejściu układu spowoduje powstanie na jego wyjściu sygnału $y = 1$.

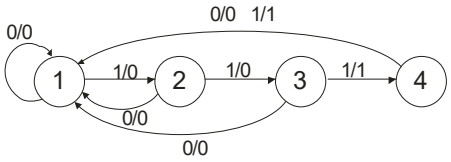
Równie prosta jest konstrukcja grafu automatu wykrywającego ciąg dokładnie trzech albo dokładnie czterech następujących bezpośrednio po sobie jedynek w dowolnie długim słowie wejściowym.

PRZYKŁAD 4.1

Graf stanów tego automatu pokazano na rys. 4.4. W stanie S1 (początkowym) sygnał 0 na wejściu powoduje pozostanie w tym stanie (nie może to być początek wejściowej sekwencji złożonej z trzech lub czterech jedynek). Pojawiające się na wejściu kolejne trzy jedynki powodują przejście przez stany S2, S3 do stanu S4, przy czym przy przejściu ze stanu S3 do S4 sygnalizowane jest wystąpienie trzech kolejnych jedynek sygnałem 1 na wyjściu. Pojawienie się w stanie S4 czwartej kolejnej jedynki powoduje przejście do stanu początkowego S1 i wygenerowanie na wyjściu sygnału 1. Pojawienie się na wejściu sygnału 0 powoduje przejście z każdego stanu do stanu początkowego S1. Na podstawie grafu stanów z rys. 4.4 utworzono tablicę przejść-wyjść automatu (tab. 4.3).

Tablica 4.3

S \ x	0	1	0	1
1	1	2	0	0
2	1	3	0	0
3	1	4	0	1
4	1	1	0	1



Rys. 4.4. Graf stanów automatu wykrywającego trzy albo cztery jedynki

Syntezę abstrakcyjną bardziej złożonego automatu omawiamy w przykładzie 4.2.

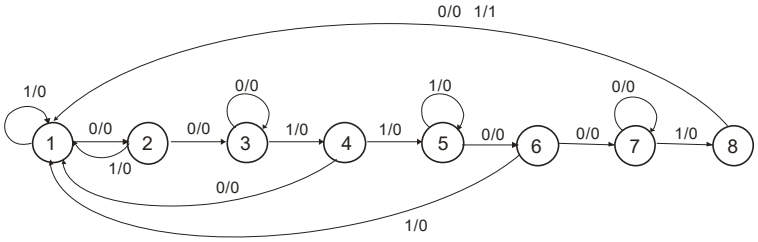
PRZYKŁAD 4.2

Zaprojektować układ wykrywający parzystą liczbę sekwencji 0011.

Konstrukcję grafu stanów automatu pokazano na rys. 4.5. W stanie S1 (początkowym) sygnał 1 na wejściu powoduje pozostanie w tym stanie (nie może to być początek wejściowej sekwencji 0011). Załóżmy dalej, że w stanie automat jest w stanie S1, a na wejściu pojawią się kolejno sygnały 0, 0, 1, 1, 0, 0, 1, 1, czyli dwie sekwencje 0011. Spowoduje to kolejne przejścia automatu przez stany: S2, S3, S4, S5, S6, S7, S8. Ze stanu S8 automat przejdzie do stan S1, przy czym dla sygnału wejściowego 1 na wyjściu zostanie wygenerowany sygnał 1 oznaczający wykrycie parzystej liczby sekwencji 0011. W stanie S3 pod wpływem sygnału wejściowego 0 układ pozostaje, oczekując na przyjście sygnału 1 podobna sytuacja ma miejsce dla stanu S7. W stanie S5 pod wpływem sygnału wejściowego 1 układ pozostaje, oczekując na przyjście sygnału 0. Ze stanów S2 i S6 pod wpływem sygnału wejściowego 1 sygnał wraca do stanu początkowego S1, analogicznie dla stanów S4 i S8 dla sygnału wejściowego. Na podstawie grafu stanów z rys. 4.5 utworzono tablicę przejść-wyjść automatu (tab. 4.4).

Tablica 4.4

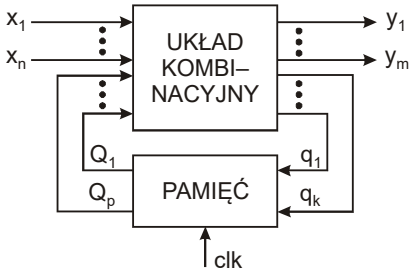
S \ x	0	1	0	1
1	2	1	0	0
2	3	1	0	0
3	3	4	0	0
4	1	5	0	0
5	6	5	0	0
6	7	1	0	0
7	7	8	0	0
8	1	1	0	1



Rys. 4.5. Graf stanów automatu z przykładu 4.1

2 Synchroniczne układy sekwencyjne

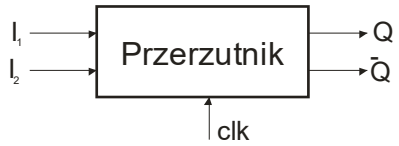
W synchronicznym układzie sekwencyjnym układ kombinacyjny (rys. 4.6) automatu jest układem wielowyjściowym, wytwarzającym sygnały wyjściowe automatu y_1, …, y_m oraz sygnały wzbudzające układ pamięciowy q_1, …, q_k. Wejściami układu kombinacyjnego są sygnały wejściowe automatu x_1, …, x_n oraz sygnały wyjściowe Q_1, …, Q_p układu pamięciowego. Układ kombinacyjny jest opisany tablicami wyjść i wzbudzeń (sposób tworzenia tablic wzbudzeń poznamy w dalszej części rozdziału).



Rys. 4.1. Synchroniczny układ sekwencyjny

Pamięć automatu jest zbudowana z tzw. przerzutników – automatów elementarnych synchronizowanych specjalnym sygnałem zegarowym oznaczonym *clk*.

Przerzutnik – to automat typu Moore'a o dwóch stanach wewnętrznych, jednym lub dwóch wejściach informacyjnych, dwóch wyjściach (prostym i zanegowanym) oraz wejściu synchronizującym (zegarowym). Symbol graficzny przerzutnika pokazano na rys. 4.7.



Rys. 4.2. Symbol graficzny przerzutnika

W zależności od rodzaju wejść informacyjnych wyróżnia się przerzutniki typu: D, T, SR i JK. Ich tablice przejść są przedstawione kolejno w tabl. 4.5a, b, c, d.

Tablica 4.1

a)

Q \ *D*	0	1
0	0	1
1	0	1

Q'

b)

Q \ *T*	0	1
0	0	1
1	1	0

Q'

c)

Q \ *SR*	00	01	11	10
0	0	0	–	1
1	1	0	–	1

d)

Q \ *JK*	00	01	11	10
0	0	0	1	1
1	1	0	0	1

Bezpośrednio z tablicy przejść można wyznaczyć równanie charakterystyczne przerzutnika określające zależność stanu następnego *Q*' od sygnałów wejściowych i stanu bieżącego *Q*. Na przykład dla przerzutników D i T odpowiednie równania będą:

$$Q' = D$$
$$Q' = \bar{T}Q + T\bar{Q}$$

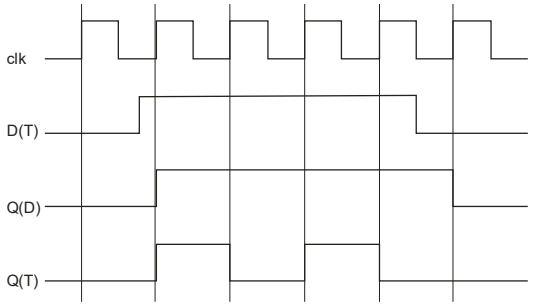
Podane w tabl. 4.6 tablice przejść przerzutników interpretujemy następująco. Przerzutnik typu D jest to element opóźniający sygnał wejściowy o takt. Przerzutnik typu T przy podaniu jedynki na wejście *T* zmienia w kolejnym takcie swój stan na przeciwny. Przerzutnik typu SR jest to przerzutnik z dwoma wejściami: ustawiającym (*Set*) i zerującym (*Reset*), czyli przy *S* = 1, *R* = 0 przerzutnik przechodzi do stanu *Q* = 1, przy *S* = 0, *R* = 1 przerzutnik przechodzi do stanu *Q* = 0, przy czym podawanie jedynek na oba wejścia jednocześnie jest zabronione. Przerzutnik JK działa jak SR, gdy jedynka jest podana na co najwyżej jedno wejście i jak typu T, gdy jedynka jest podana na oba wejścia. Zatem na wejścia przerzutnika JK może być podana dowolna kombinacja wartości sygnałów *J* i *K*, przy czym zawsze przy sygnałach *J* = 1, *K* = 0 przerzutnik ten „zapala się", to znaczy przechodzi do stanu 1, zaś przy sygnałach *J* = 0, *K* = 1 przerzutnik „gaśnie", to znaczy przechodzi do stanu 0. Przy sygnałach *J* = *K* = 1 przerzutnik zmienia stan na przeciwny.

Przy projektowaniu układów synchronicznych posługujemy się również tzw. tablicami wzbudzeń przerzutników. Określają one jakie muszą być sygnały wejściowe przerzutnika, aby uzyskać określoną zmianę stanu (przejście ze stanu bieżącego *Q* do następnego *Q*'). Tablice wzbudzeń omawianych przerzutników podane są w tabl. 4.6.

Tablica 4.2

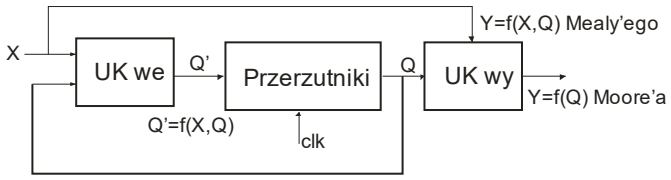
Q Q'	D	T	S R	J K
0 0	0	0	0 –	0 –
0 1	1	1	1 0	1 –
1 0	0	1	0 1	– 1
1 1	1	0	– 0	– 0

Tablice przejść i tablice wzbudzeń przerzutników nie określają bezpośrednio roli sygnału zegarowego. Znaczenie sygnału zegarowego w pracy przerzutnika jest istotne o tyle, że wyznacza on momenty zmiany stanu bieżącego *Q*. Zmiany te mogą nastąpić tylko w chwilach sygnalizowanych przez ustalone zbocze sygnału zegarowego (np. narastające). Na rys. 4.8 są pokazane przebiegi sygnałów wejściowych przerzutnika typu D i T oraz sygnały na ich wyjściach *Q*.



Rys. 4.3. Wykresy czasowe sygnałów dla przerzutników D i T

Omawianie procesu syntezy układów sekwencyjnych rozpoczynamy od etapu syntezy kombinacyjnej. Jest to najważniejszy a zarazem najobszerniejszy etap całego procesu syntezy. Polega on (jak to pokazano na schemacie blokowym układu sekwencyjnego – rys. 4.9) na obliczaniu funkcji sterujących wejściami przerzutników (funkcje wzbudzeń) oraz na obliczaniu funkcji wyjściowych. Warto podkreślić różnicę w obliczaniu funkcji wyjściowych dla automatów typu Moore'a a i typu Mealy'ego. Otóż w pierwszym przypadku funkcje te są zależne tylko od wyjść *Q* przerzutników, natomiast w drugim, są zależne zarówno od wyjść *Q* jak i od wejść zewnętrznych *X*.



Rys. 4.4. Schemat blokowy układu sekwencyjnego

Syntezę kombinacyjną omówimy na przykładzie automatu, który dla ustalenia uwagi nazywać będziemy detektorem sekwencji. Jest to automat typu Mealy'ego, o trzech stanach wewnętrznych oznaczonych *A*, *B*, *C* (tabl. 4.7), jednym sygnale wejściowym *x* oraz jednym sygnale wyjściowym. Pierwszą czynnością jaką należy wykonać jest zatem kodowanie stanów wewnętrznych. Kodowanie stanów polega na przyporządkowaniu abstrakcyjnym symbolom stanów *A*, *B*, *C* ciągów binarnych o możliwie najmniejszej liczbie bitów. Dla trzech stanów do jednoznacznego zakodowania wystarczą dwa bity. Dlatego przyjmiemy, że *A* = 00, *B* = 01, *C* = 11. Przyjęcie kodowania determinuje jednocześnie liczbę przerzutników układu sekwencyjnego. Oznaczając przerzutniki Q_1, Q_0 uzyskujemy zakodowaną tablicę

Tablica 4.3

S \ *X*	0	1	0	1
A	*B*	*A*	0	0
B	*C*	*A*	0	0
C	*C*	*A*	0	1
			Y	

przejść wyjść (tabl. 4.8). Tablicę tę podajemy łącznie z nadmiarowym stanem wewnętrznym. Tak uzyskana tablica posłuży do obliczania funkcji wzbudzeń dla różnych przerzutników. Naszym zadaniem będzie realizacja tego automatu w trzech różnych strukturach: z zastosowaniem przerzutników typu D, T oraz JK.

Tablica 4.4

Q_1Q_0 \ X	0	1	0	1
A 00	01	00	0	0
B 01	11	00	0	0
C 11	11	00	0	1
(–) 10	– –	– –	–	–
			Y	

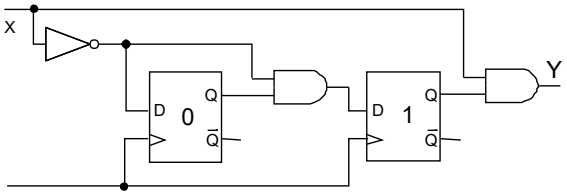
Tablica 4.5

Q_1Q_0 \ X	0	1
00	0	0
01	1	0
11	1	0
10	–	–
	D_1	

Q_1Q_0 \ X	0	1
00	1	0
01	1	0
11	1	0
10	–	–
	D_2	

Zgodnie z tym co powiedzieliśmy zakodowana tablica reprezentuje dwa przerzutniki. Zatem w celu obliczenia funkcji wzbudzeń tablicę tę należy „rozpisać" na dwie tabelki odpowiadające poszczególnym przerzutnikom Q_1 i Q_0. Uzyskuje się w ten sposób oddzielne tablice wzbudzeń dla każdego przerzutnika (tabl. 4.9). Są one zapisane w formie tablic Karnaugha, a więc można na nich zakreślać pętelki w celu uzyskania minimalnych wyrażeń boolowskich dla funkcji wzbudzeń D_1 i D_0. W przypadku przerzutników typu D funkcje wzbudzeń są tożsame z funkcjami stanu następnego oznaczonym Q_1' oraz Q_0'. Również bezpośrednio z tabelki 4.8 obliczamy funkcję wyjściową *Y*:

$$Q_1' = D_1 = \bar{x}Q_1$$
$$Q_0' = D_0 = \bar{x}$$
$$Y = xQ_1$$



Rys. 4.5. Schemat logiczny detektora sekwencji z przerzutnikami typu D

Schemat logiczny tak zaprojektowanego układu (rys. 4.10) jest zbudowany z dwóch przerzutników, do których wejść sterujących D_1 i D_0 dołączone są wyjścia układów kombinacyjnych realizujących funkcje wzbudzeń oraz funkcję wyjściową.

Dla realizacji z przerzutnikami T uzyskane poprzednio tabelki dla funkcji stanu następnego Q_1' i Q_0' należy dodatkowo przekształcić w celu uzyskania tablic wzbudzeń dla przerzutników T_1 i T_0. Transformacji takiej dokonujemy na podstawie tablicy wzbudzeń przerzutnika T. W rezultacie uzyskujemy tablice dla funkcji wzbudzeń T_1 i T_0 (tabl. 4.10). Są one podane w formie tablic Karnaugha, zatem po zakreśleniu pętelek bezpośrednio obliczamy wyrażenia boolowskie dla T_1 i T_0.

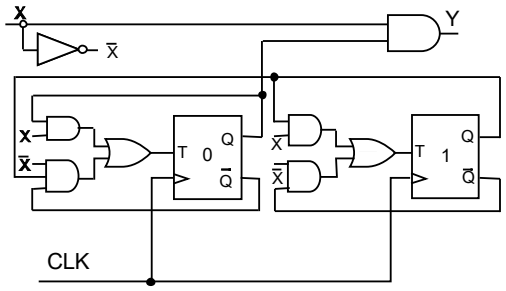
Tablica 4.6

Q_1Q_0 \ X	0	1
00	0	0
01	1	0
11	0	1
10	–	–
	T_1	

Q_1Q_0 \ X	0	1
00	1	0
01	0	1
11	0	1
10	–	–
	T_2	

$$T_1 = \bar{x}\bar{Q}_1 Q_0 + xQ_1$$
$$T_0 = \bar{x}\bar{Q}_0 + xQ_0$$

Podobnie jak poprzednio schemat logiczny tak zaprojektowanego układu jest zbudowany z dwóch przerzutników, do których wejść sterujących T_1 i T_0 dołączone są wyjścia układów kombinacyjnych realizujących funkcje wzbudzeń oraz funkcję wyjściową (rys. 4.11). Funkcja wyjściowa nie uległa zmianie.



Rys. 4.6. Schemat logiczny detektora sekwencji z przerzutnikami typu T

Nieco bardziej skomplikowane jest obliczanie funkcji wzbudzeń dla przerzutników JK. W tym przypadku każdą tabelkę funkcji stanu następnego tj,. Q_1' oraz Q_0' należy przekształcić na dwie tabelki: jedną dla wejścia *J*, a drugą dla wejścia *K*. Oczywiście podobnie jak poprzednio odpowiednie wartości sygnałów binarnych jakie należy wpisywać do poszczególnych kratek tabelek wzbudzeń uzyskujemy z tablicy wzbudzeń dla JK. W rezultacie powstają cztery tabelki wzbudzeń: dla J_1, K_1 oraz dla J_0, K_0. Pokazane są one w tabl. 4.11.

Tablica 4.7

Q_1Q_0 \ X	0	1
00	0	0
01	1	0
11	–	–
10	–	–

J_1

Q_1Q_0 \ X	0	1
00	–	–
01	–	–
11	0	1
10	–	–

K_1

Q_1Q_0 \ X	0	1
00	1	0
01	–	–
11	–	–
10	–	–

J_0

Q_1Q_0 \ X	0	1
00	–	–
01	0	1
11	0	1
10	–	–

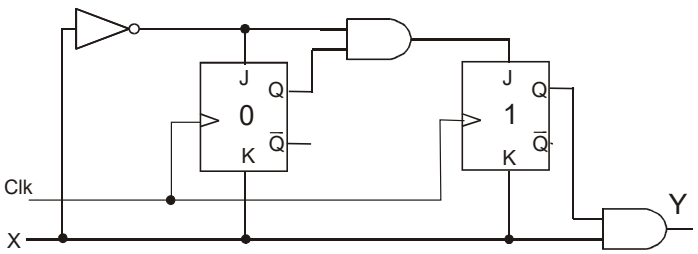
K_0

Obliczone na ich podstawie funkcje wzbudzeń są następujące :

$J_1 = \bar{x}Q_0$ $\quad K_1$=x

$J_0 = \bar{x}$ $\quad K_0$=x

Schemat logiczny układu pokazano na rys. 4.12.



Rys. 4.7. Schemat logiczny detektora sekwencji z przerzutnikami typu JK

Omówiony przykład syntezy układu dla detektora sekwencji jest zbyt prosty. Dla bardziej skomplikowanego treningu w projektowaniu sekwencyjnych układów cyfrowych omówimy syntezę licznika mod 5 ze sterowaniem. Licznik to układ cyfrowy (blok funkcjonalny), w którym zliczane są impulsy zegarowe. Pojawienie się impulsu zwiększa lub zmniejsza zawartość licznika o 1. Czyli jest to prosty układ sekwencyjny, który musi pamiętać poprzednią zawartość reprezentowaną stanem wewnętrznym.

PRZYKŁAD 4.3

Należy zaprojektować licznik pracujący w trzech różnych trybach, a mianowicie: liczenie do przodu, liczenie do tyłu oraz zerowanie. Wybór tych trybów będzie dokonywany sygnałami sterującymi x_1 i x_2.

Pierwszą czynnością jest formalne zapisanie działania układu w postaci tablicy przejść wyjść. Jest to czynność, którą zgodnie z klasyfikacją etapów syntezy automatów zaliczamy do etapu syntezy abstrakcyjnej. Oznaczając dla licznika mod 5 (czyli zliczającego do 5) stany wewnętrzne wg naturalnego kodu binarnego: S_0, S_1, …, S_4 wymienione trzy tryby pracy możemy zapisać w formie tablicy podanej w tabl. 4.12a. Kolejny etap syntezy to kodowanie. Do zakodowania mamy zarówno stany wewnętrzne S_0 do S_4, jak też litery wejściowe *a*, *b*, *c*. Sposób kodowania podany jest w tabl. 4.12b.

Tablica 4.8

a)

X / S	a	b	c	Y
S_0	S_1	S_4	S_0	0
S_1	S_2	S_0	S_0	0
S_2	S_3	S_1	S_0	0
S_3	S_4	S_2	S_0	0
S_4	S_0	S_3	S_0	1

b)

x_1x_2 / $Q_2Q_1Q_0$	00	01	11	10	Y
000	001	100	000	000	0
001	010	000	000	000	0
010	011	001	000	000	0
011	100	010	000	000	0
100	000	011	000	000	1

Uzyskaną tablicę należy zapisać w formie dogodnej do dalszych obliczeń. W tym celu dla oznaczenia poszczególnych wierszy i kolumn tej tablicy przyjmujemy kod Gray'a. Oczywiście chcąc, aby tablica taka jak najwierniej odpowiadała tablicy Karnaugha, niektóre wiersze będą miały nieokreślone stany następne. Uzyskana tablica podana jest w tabl. 4.13. Reprezentuje ona (łącznie) trzy funkcje stanu następnego Q_2',Q_1',Q_0'.

W celu obliczenia funkcji wzbudzeń należy uzyskaną tablicę rozpisać na trzy tablice, podane w tabl. 4.14. Reprezentują one poszczególne przerzutniki tj. ich stany następne, a w przypadku przerzutników typu D są to jednocześnie funkcje wzbudzeń.

PRZYKŁAD 4.3 c.d

Tablica 4.9

$Q_2Q_1Q_0$ \ x_1x_2	00	01	11	10
000	001	100	000	000
001	010	000	000	000
011	100	010	000	000
010	011	001	000	000
110	– – –	– – –	– – –	– – –
111	– – –	– – –	– – –	– – –
101	– – –	– – –	– – –	– – –
100	000	011	000	000

$Q'_2Q'_1Q'_0$

Tablica 4.10

a)

$Q_2Q_1Q_0$ \ x_1x_2	00	01	11	10
000	0	1	0	0
001	0	0	0	0
011	1	0	0	0
010	0	0	0	0
110	–	–	–	–
111	–	–	–	–
101	–	–	–	–
100	0	0	0	0

b)

$Q_2Q_1Q_0$ \ x_1x_2	00	01	11	10
000	0	0	0	0
001	1	0	0	0
011	0	1	0	0
010	1	0	0	0
110	–	–	–	–
111	–	–	–	–
101	–	–	–	–
100	0	1	0	0

c)

$Q_2Q_1Q_0$ \ x_1x_2	00	01	11	10
000	1	0	0	0
001	0	0	0	0
011	0	0	0	0
010	1	1	0	0
110	–	–	–	–
111	–	–	–	–
101	–	–	–	–
100	0	1	0	0

Mając zatem tablice wzbudzeń poszczególnych przerzutników podane w formie tablic Karnaugha możemy bezpośrednio z tych tabelek wyznaczyć odpowiednie wyrażenia boolowskie na funkcje wzbudzeń D_2, D_1, D_0.

$$D_2 = \bar{x}_1\bar{x}_2Q_1Q_0 + \bar{x}_1x_2\bar{Q}_2\bar{Q}_1\bar{Q}_0$$
$$D_1 = \bar{x}_1\bar{x}_2\bar{Q}_1Q_0 + \bar{x}_1\bar{x}_2Q_1\bar{Q}_0 + \bar{x}_1x_2Q_1Q_0 + \bar{x}_1x_2Q_2$$
$$D_0 = \bar{x}_1\bar{x}_2\bar{Q}_0 + \bar{x}_1Q_1\bar{Q}_0 + \bar{x}_1x_2Q_2$$

Proces projektowania licznika jest jeszcze bardziej skomplikowany dla realizacji na przerzutnikach typu JK. W tym przypadku każdą uzyskaną poprzednio tablicę stanu następnego należy rozpisać na dwie oddzielne dla sterowania J oraz K.

Oczywiście $Y = Q_2$.

PRZYKŁAD 4.3 c.d.

Trzy kolejne tablice reprezentują funkcje wzbudzeń wejścia *J* oraz *K* poszczególnych przerzutników kolejno: J_2, K_2 (tabl. 4.15), J_1, K_1 (tabl. 4.16) oraz J_0, K_0 (tabl. 4.17). Z tablic tych bezpośrednio obliczamy funkcje wzbudzeń:

$$J_2 = \bar{x}_1\bar{x}_2Q_1Q_0 + \bar{x}_1x_2\bar{Q}_1\bar{Q}_0$$

$$K_2 = 1$$

$$J_1 = \bar{x}_1\bar{x}_2Q_0 + \bar{x}_1x_2Q_2$$

$$K_1 = \bar{x}_2Q_0 + x_2\bar{Q}_0 + x_1$$

$$J_0 = \bar{x}_1\bar{x}_2\bar{Q}_2 + \bar{x}_1x_2Q_2 + \bar{x}_1Q_1$$

$$K_0 = 1$$

Oczywiście $Y = Q_2$.

Tablica 4.11

a)

$Q_2Q_1Q_0$ \ x_1x_2	00	01	11	10
000	0	1	0	0
001	0	0	0	0
011	1	0	0	0
010	0	0	0	0
110	–	–	–	–
111	–	–	–	–
101	–	–	–	–
100	–	–	–	–

J_2

b)

$Q_2Q_1Q_0$ \ x_1x_2	00	01	11	10
000	–	–	–	–
001	–	–	–	–
011	–	–	–	–
010	–	–	–	–
110	–	–	–	–
111	–	–	–	–
101	–	–	–	–
100	1	1	1	1

K_2

Tablica 4.12

a)

$Q_2Q_1Q_0$ \ x_1x_2	00	01	11	10
000	0	0	0	0
001	1	0	0	0
011	–	–	–	–
010	–	–	–	–
110	–	–	–	–
111	–	–	–	–
101	–	–	–	–
100	0	1	0	0

J_1

b)

$Q_2Q_1Q_0$ \ x_1x_2	00	01	11	10
000	–	–	–	–
001	–	–	–	–
011	1	0	1	1
010	0	1	1	1
110	–	–	–	–
111	–	–	–	–
101	–	–	–	–
100	–	–	–	–

K_1

PRZYKŁAD 4.3 c.d.

Tablica 4.13

a)

$Q_2Q_1Q_0$ \ x_1x_2	00	01	11	10
000	1	0	0	0
001	–	–	–	–
011	–	–	–	–
010	1	1	0	0
110	–	–	–	–
111	–	–	–	–
101	–	–	–	–
100	0	1	0	0

J_0

b)

$Q_2Q_1Q_0$ \ x_1x_2	00	01	11	10
000	–	–	–	–
001	1	1	1	1
011	1	1	1	1
010	–	–	–	–
110	–	–	–	–
111	–	–	–	–
101	–	–	–	–
100	–	–	–	–

K_0

3 Minimalizacja stanów automatu

Jak wiadomo automat $A = \langle S,V,Y,\delta,\lambda\rangle$, w którym zbiór S stanów wewnętrznych S ma liczność $|S|$ może być zrealizowany na co najmniej $\pi = \lceil \log_2 |S| \rceil$ przerzutnikach. Zatem każde zmniejszenie liczby stanów, ale takie, aby automat z punktu widzenia obserwatora zewnętrznego wykonywał taką samą pracę jest bardzo korzystne ze względu na złożoność (a tym samym koszt) jego realizacji. Proces redukowania liczby stanów automatu, ale w taki specjalny sposób, aby nie zmniejszyło to możliwości funkcjonalnych automatu nazywa się minimalizacją liczby stanów automatu. Przykład minimalizacji automatu podany jest w tabl. 4.18. Automat opisany tablicą przejść wyjść (tabl. 4.18a) ma 6 stanów wewnętrznych, zatem wymaga do swojej realizacji 3 przerzutników. Automat ten można jednak zminimalizować (na razie nie wiadomo w jaki sposób) do postaci podanej w tabl. 4.18b. Zminimalizowany automat ma 3 stany, a więc do jego realizacji wystarczą tylko 2 przerzutniki.

Proces minimalizacji automatu polega na wyznaczaniu relacji zgodności na zbiorze stanów wewnętrznych S. Następnie dla tak wyznaczonych par zgodnych obliczane są Maksymalne Klasy Zgodności. W ostatnim etapie minimalizacji dokonuje się selekcji minimalnej liczby zbiorów zgodnych spełniających tzw. *warunek pokrycia* i *zamknięcia*.

Dwa stany wewnętrzne S_i, S_j są *zgodne*, jeżeli dla każdego wejścia v mają one niesprzeczne stany wyjść, a ich stany następne są takie same lub niesprzeczne.

Dwa stany wewnętrzne S_i, S_j są *zgodne warunkowo*, jeżeli ich stany wyjść są niesprzeczne oraz dla pewnego $v \in V$ para stanów następnych do S_i, S_j (ozn. S_k, S_l):

$(S_i, S_j) \neq (S_k, S_l)$

Stany S_i, S_j są *sprzeczne*, jeżeli dla pewnego $v \in V$ ich stany wyjść są sprzeczne.

Dla automatu z tabl. 4.18a, którego stany wewnętrzne są oznaczone liczbami naturalnymi 1 do 6, para 2, 4 jest zgodna, para 1, 3 jest zgodna warunkowo, a para 5, 6 jest sprzeczna.

Tablica 4.1

a)

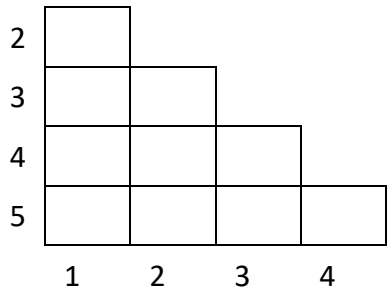
S \ x	a	b	c	d	a	b	c	d
S_1	–	S_3	S_4	S_2	–	1	1	1
S_2	S_4	–	–	–	0	–	–	–
S_3	S_6	S_6	–	–	0	1	–	–
S_4	–	S_6	S_1	S_5	–	0	0	1
S_5	–	–	S_2	–	–	–	1	–
S_6	S_3	–	S_2	S_3	0	–	0	1

b)

S \ x	a	b	c	d	a	b	c	d
A	C	B	C	A	0	1	1	1
B	C	C	A	–	0	1	1	–
C	B	C	A	B	0	0	0	1

Ze względu na to, że para zgodna warunkowo może – po dalszej analizie – okazać się parą zgodną albo sprzeczną, w obliczaniu wszystkich par zgodnych posługujemy się tzw. tablicą trójkątną. Przykładowa tablica trójkątna (tabl. 4.19) dla automatu o 5 stanach ma 4 kolumny oznaczone 1 do 4 oraz 4 wiersze oznaczone (od góry) 2 do 5. W rezultacie uzyskujemy tablicę, której kratki wypełniamy symbolami v, gdy analizowana para jest zgodna, x – gdy dana para jest sprzeczna lub w kratce zapisujemy parę (lub pary) stanów następnych w przypadku zgodności warunkowej.

Tablica 4.2



Sposób wypełnienia tablicy trójkątnej dla przykładowego automatu (podanego w tabl. 4.20) wyjaśniamy w tabl. 4.21. Jak widać para 1, 2 jest zgodna, kratkę o współrzędnych 1, 2 wypełniamy znaczkiem v; para 1, 3 jest zgodna pod warunkiem zgodności pary 3, 6 i dlatego w odpowiedniej kratce wpisujemy 3, 6; z kolei para 1, 4 ma sprzeczne wyjścia, zatem wypełniamy ją symbolem $\times$ itd.

Tablica 4.3

S \ x	a	b	c	d	a	b	c	d
1	–	3	4	2	–	1	1	1
2	4	–	–	–	0	–	–	–
3	6	6	–	–	0	1	–	–
4	–	6	1	5	–	0	0	1
5	–	–	2	–	–	–	1	–
6	3	–	2	3	0	–	0	1

Ogólnie, w tablicy trójkątnej należy wpisać wszystkie warunki oraz wykreślić kratki odpowiadające stanom o sprzecznych wyjściach. Następnie należy

wykreślić wszystkie kratki, w których jako warunek (lub jeden z warunków) wpisana jest para *k*,*l* odpowiadająca klatce (*k*,*l*) wykreślonej na poprzednim etapie. Dla wszystkich nowo wykreślonych klatek należy sprawdzić, czy odpowiadające im pary stanów występują w niewykreślonych kratkach jako warunki. Czynność wykreślania klatek prowadzi się aż do uzyskania sytuacji, gdy wszystkie pary określające warunki odpowiadają kratkom niewykreślonym.

Tablica 4.4

2	v				
3	3,6	4,6			
4	×	v	×		
5	2,4	v	v	×	
6	×	3,4	v	1,2; 3,5	×
	1	2	3	4	5

W tak uzyskanej tablicy wszystkie kratki niewykreślone, bez względu na ich zawartość, odpowiadają parom stanów zgodnych. Z tabl. 4.21 odczytujemy, że wszystkie pary zgodne dla automatu z tabl. 4.20 są następujące:

(1,2) (1,3) (1,5) (2,3) (2,4) (2,5) (3,5) (3,6) (4,6)

Dysponując zbiorem wszystkich par zgodnych przystępujemy do wyznaczenia rodziny Maksymalnych Klas Zgodności. Dla potrzeb obliczania rodziny MKZ możemy stosować jedną z dwóch metod omówionych w rozdz. 2.

W przypadku tak prostego automatu wystarczy metoda bezpośrednia. Stosowne obliczenia przebiegają następująco. Najpierw z par tworzymy trójki: 1,2,3; 1,2,5; 1,3,5; 2,3,5. Z uzyskanych czterech trójek powstaje zbiór czteroelementowy: 1, 2, 3, 5. W celu uzyskania wszystkich zbiorów MKZ uzupełniamy tę „czwórkę" tymi parami zgodnymi, które nie zawierają się w dotychczas obliczonych zbiorach zgodnych (czyli w 1, 2, 3, 5). W rezultacie uzyskujemy rodzinę MKZ = {{1,2,3,5}, {2,4}, {3,6}, {4,6}}.

Zgodnie z dotychczasowymi informacjami kolejnym etapem minimalizacji jest selekcja zbiorów zgodnych spełniających odpowiedni warunek pokrycia i zamknięcia. Skoncentrujmy się na wyjaśnieniu warunków pokrycia i zamknięcia.

Otóż pokrycie wymaga, aby każdy stan realizowanego automatu był elementem co najmniej jednej wybranej klasy. Natomiast zamknięcie wymaga, aby dla każdej litery wejściowej wszystkie następniki (stany następne) danej klasy były zawarte w jednej z wybranych klas.

Tablica 4.5

	a	*b*	*c*	*d*
1,2,3,5	4,6	**3,6**	**2,4**	2
4,6	3	6	1,2	3,5

Minimalnym zbiorem klas stanów zgodnych, spełniającym warunek pokrycia dla automatu z tabl. 4.20 jest zbiór {{1,2,3,5}, {4,6}}. Zbiór ten nie jest jednak zamknięty, gdyż warunkami dla klasy {1,2,3,5} są {3,6} i {2,4}, które nie są spełnione (tabl. 4.22), gdyż żadna z tych klas nie jest zawarta w żadnej z obu klas zbioru minimalnego. Ponieważ następnikami klasy {4,6} są klasy {1,2} i {3,5}, więc spróbujmy rozbić klasę {1,2,3,5} na klasy {1,2} i {3,5}. Na podstawie tablicy trójkątnej stwierdzamy, że klasy te nie mają warunków zamknięcia. Jeśli zatem utworzymy zbiór {{1,2}, {3,5}, {4,6}}, to wszystkie klasy będące warunkami zamknięcia dla klas tego zbioru są zawarte w pewnych jego klasach. Zatem optymalny zbiór MKZ_{opt} = {{1,2}, {3,5}, {4,6}} jest zamknięty i

pełny, a więc jest minimalnym zbiorem klas zgodnych dla tego automatu. Konstrukcja automatu minimalnego pokazana jest w tabl. 4.23.

W kolejnym przykładzie dokonamy redukcji stanów dla automatu podanego tablicy 4.24.

W tab. 4.25 zaznaczono pary sprzeczne (×), pary zgodne (v) oraz pary zgodne warunkowo (np. 1,4). Po sprawdzeniu zgodności warunkowej wykreślono pary 1,7; 3,7 oraz 4,7.

Tablica 4.6

a)

		a	b	c	d	a	b	c	d
A	1,2	4	3	4	2	0	1	1	1
B	3,5	6	6	2	–	0	1	1	–
C	4,6	3	6	1,2	3,5	0	0	0	1

b)

	a	b	c	d	a	b	c	d
A	*C*	*B*	*C*	*A*	0	1	1	1
B	*C*	*C*	*A*	–	0	1	1	–
C	*B*	*C*	*A*	*B*	0	0	0	1

Tablica 4.7

S \ *x*	0	1	0	1
1	5	4	0	0
2	2	7	1	1
3	–	1	–	0
4	–	3	–	0
5	2	–	1	–
6	6	–	1	–
7	–	8	–	0
8	–	–	–	1

Tablica 4.8

2	×						
3	1,4	×					
4	3,4	×	1,3				
5	×	v	v	v			
6	×	v	v	v	2,6		
7	~~4,8~~	×	~~1,8~~	~~3,8~~	v	v	
8	×	v	×	×	v	v	×
	1	2	3	4	5	6	7

Następnie stosując algorytm wyznaczania MKZ wg par zgodnych uzyskuje się:

$S_1 = \varnothing$	{1}
$S_2 = \varnothing$	{2}{1}
$S_3 = \{1\}$	{1,3}{2}
$S_4 = \{1,3\}$	{1,3,4}{2}
$S_5 = \{2,3,4\}$	{3,4,5}{2,5}{1,3,4}
$S_6 = \{2,3,4,5\}$	{3,4,5,6}{2,5,6}~~{3,4,6}~~{1,3,4}
$S_7 = \{5,6\}$	{5,6,7}~~{5,6,7}~~{1,3,4}{2,5,6}{3,4,5,6}
$S_8 = \{2,5,6\}$	~~{5,6,8}~~{2,5,6,8}{5,6,7}{1,3,4}{3,4,5,6}

MKZ = {1,3,4},{2,5,6,8},{3,4,5,6},{5,6,7}

Warunek pokrycia spełniają klasy {1,3,4}, {2,5,6,8}, {5,6,7}. Warunek zamknięcia jest sprawdzany w tab. 4.26.

Przyporządkowując odpowiednio nazwy stanów A, B, C otrzymujemy tablicę przejść-wyjść (tab. 4.27) automatu minimalnego.

Tablica 4.9

S \ *x*	0	1	0	1
1,3,4	5---	413	0	0
2,5,6,8	226-	7---	1	1
5,6,7	--26	--8	1	0

Tablica 4.10

	S \ *x*	0	1	0	1
A	1,3,4	B/C	A	0	0
B	2,5,6,8	C	C	1	1
C	5,6,7	B	B	1	0
		S		*y*	

Treścią kolejnego przykładu będzie zadanie nieco obszerniejsze – obejmujące dwa etapy syntezy tj. syntezę abstrakcyjna oraz minimalizacje liczby stanów.

PRZYKŁAD 4.4

Zaprojektować automat do kontroli czterobitowych słów podawanych szeregowo na jego wejście. Automat ma sprawdzać, czy słowa należą do kodu 2 z 4.

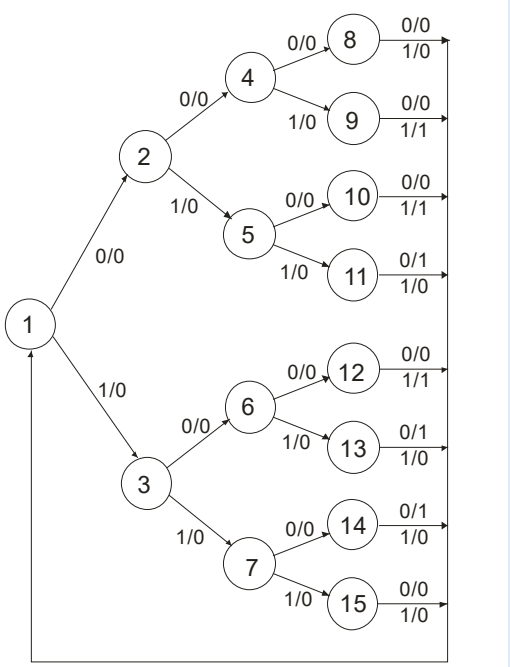
Synteza abstrakcyjna

Konstrukcję grafu stanów automatu pokazano na rys. 4.13. Automat pracuje w czterech taktach, co odpowiada przejściom ze stanu S1 np. przez stany S2, S4, S8 i powrót do stanu S1. Takich możliwych przejść jest 16 – każde odpowiada jednemu z szesnastu czterobitowych słów wejściowych. W czwartym takcie pracy automatu (przy powrocie do stanu S1) na wyjściu pojawia się sygnał 1 – jeśli w słowie wejściowym były dokładnie dwie jedynki (np. przejście S1, S2, S5, S11, S1, lub sygnał 0 – jeśli w słowie wejściowym była inna liczba jedynek niż dwie.

Na podstawie grafu stanów z rys. 4.13 utworzono tablicę przejść-wyjść automatu (tab. 4.28).

Tablica 4.11

S \ x	0	1	0	1
1	2	3	0	0
2	4	5	0	0
3	6	7	0	0
4	8	9	0	0
5	10	11	0	0
6	12	13	0	0
7	14	15	0	0
8	1	1	0	0
9	1	1	0	1
10	1	1	0	0
11	1	1	1	0
12	1	1	0	1
13	1	1	1	0
14	1	1	1	0
15	1	1	0	0



Rys. 4.1. Graf stanów automatu z przykładu 4.4

PRZYKŁAD 4.4 c.d.

Minimalizacja liczby stanów

W tablicy trójkątnej 4.29 znakiem v zaznaczono pary zgodne, krzyżykami × zaznaczono pary sprzeczne, a polach odpowiadających parom zgodnym warunkowo podano warunki. Następnie eliminujemy pary zgodne warunkowo na podstawie par sprzecznych (1,9; 2,9 itd.). Powstają nowe pary sprzeczne, które dalej eliminują pary zgodne warunkowo, itd. Okazuje się, że w wyniku tego postępowania uzyskano pary zgodne: (5,6), (8,15), (9,10), (9,12), (10,12), (11,13), (11,14), (131,14). Na tej podstawie wyznaczamy maksymalne klasy zgodności: {1}, {2},{3},{4},{5,6},{7}, {8,15}, {9,10,12}, (11,13,14}. Ponieważ dla spełnienia warunku pokrycia trzeba wziąć wszystkie klasy, zatem warunek zamkniętości też będzie spełniony.

Tablica 4.12

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
2	2,3 4,5													
3	2,6 3,7	4,6 5,7												
4	2,8 3,9	4,8 5,9	6,8 7,9											
5	2,10 3,11	4,10 5,11	6,10 7,11	8,10 9,11										
6	2,12 3,13	4,14 5,15	6,12 7,13	8,12 9,13	v									
7	2,14 3,15	4,14 5,15	6,14 7,15	8,14 9,15	10,14 11,15	12,14 13,15								
8	1,2 1,3	1,4 1,5	1,6 1,7	1,8 1,9	1,10 1,11	1,12 1,13	1,14 1,15							
9	×	×	×	×	×	×	×	×						
10	1,2 1,3	1,4 1,5	1,6 1,7	1,8 1,9	1,10 1,11	1,12 1,13	1,14 1,15	×	v					
11	×	×	×	×	×	×	×	×	×	×				
12	×	×	×	×	×	×	×	×	v	v				
13	×	×	×	×	×	×	×	×	×	×	v	×		
14	×	×	×	×	×	×	×	×	×	×	v	×	v	
15	1,2 1,3	1,4 1,5	1,6 1,7	1,8 1,9	1,10 1,11	1,12 1,13	1,14 1,15	v	×	v	×	×	×	×

PRZYKŁAD 4.4.c.d.

Jeśli wymienionym wcześniej klasom zgodności przypiszemy nazwy kolejno A, B, … , H, I, to uzyskamy tablicę przejść automatu minimalnego (tab. 4.30)

Tablica 4.13

S \ x	0	1	0	1
A	B	C	0	0
B	D	E	0	0
C	F	G	0	0
D	G	H	0	0
E	H	I	0	0
F	I	G	0	0
G	A	A	0	0
H	A	A	0	1
I	A	A	1	1

Na zakończenie wrócimy do syntezy automatów, dla których w przykładach 4.1 i 4.2 już wykonano syntezę abstrakcyjną. W wyniku tej syntezy otrzymaliśmy następujące tablice przejść-wyjść tych automatów.

Pokazane są one odpowiednio w tablicach 4.31 oraz 4.32. Odpowiadające im tablice trójkątne, pokazane w tablicach 4.33 i 4.34, wykazują, że automatów tych nie można zminimalizować.

Tablica 4.14

S \ x	0	1	0	1
1	1	2	0	0
2	1	3	0	0
3	1	4	0	1
4	1	1	0	1

Tablica 4.15

S \ x	0	1	0	1
1	2	1	0	0
2	3	1	0	0
3	3	4	0	0
4	1	5	0	0
5	6	5	0	0
6	7	1	0	0
7	7	8	0	0
8	1	1	0	1

Tablica 4.16

2	2,3		
3	2,4	×	
4	1,2	×	1,4
	1	2	3

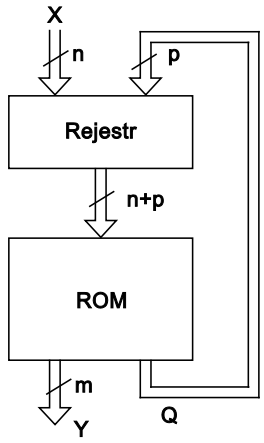
Tablica 4.1

2	2,3						
3	1,4 2,3	1,4					
4	1,2 1,5	1,3 1,5	1,3 4,5				
5	2,6	1,5 3,6	3,6 4,5	1,6			
6	2,7	3,7	1,3 3,7	1,5 1,7	1,5 6,7		
7	1,8 2,7	1,8 3,7	3,7 4,8	3,7 4,8	5,8 6,7	1,8	
8	×	×	×	×	×	×	×
	1	2	3	4	5	6	7

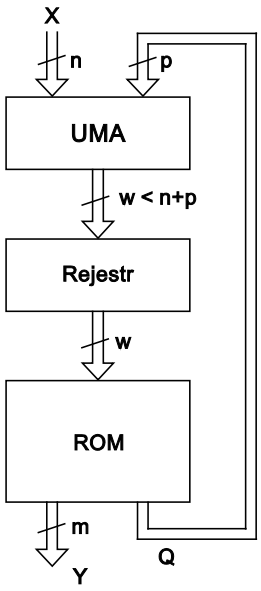
4 UKŁADY Z PAMIĘCIAMI

4.1 Informacje ogólne

Omawiane w rozdz. 3 metody syntezy układów kombinacyjnych jak najbardziej uwzględniały aktualne tendencje realizacji układów cyfrowych w strukturach z pamięciami ROM. W szczególności redukcja argumentów oraz dekompozycja funkcjonalna mogą być uznane za skuteczne metody syntezy umożliwiające realizacje funkcji boolowskich na pamięciach ROM z ograniczoną liczba wejść adresowych. Nietrudno przewidzieć, iż w przypadku układów sekwencyjnych liczba zmiennych wejściowych *n* (rys. 4.14)



Rys. 4.1. Realizacja układu sekwencyjnego z użyciem pamięci ROM



Rys. 4.2. Realizacja układu sekwencyjnego z wykorzystaniem układu modyfikacji adresu

oraz zmiennych wewnętrznych p układu sekwencyjnego niejednokrotnie przekroczy liczbę zmiennych adresowych pamięci ROM przeznaczonej do realizacji tego układu. Stąd wynika potrzeba opracowania metody umożliwiającej zmniejszenie wymiaru pamięci ROM za pośrednictwem Układu Modyfikacji Adresu (rys. 4.15).

Taka realizacja może być traktowana jako dekompozycja bloku pamięciowego z rys. 4.14 na dwa bloki: kombinacyjny układ modyfikacji adresu oraz blok pamięci o mniejszym rozmiarze, niż wymagany przy realizacji w strukturze z rys. 4.15. Dzięki tej metodzie układy sekwencyjne, które wymagałyby zbyt dużej pojemności pamięci ROM, a tym samym nie mogłyby być realizowane przy użyciu architektury FPGA z pamięciami o ograniczonej pojemności, mogą zostać zaimplementowane przy użyciu mniejszej pamięci.

Z punktu widzenia układów FPGA jest to metoda korzystna o tyle, że blok modyfikacji adresu może być realizowany przez komórki typu LUT lub matrycę PAL, a blok pamięci – w matrycach wbudowanych typu EAB.

4.2 Modyfikacja adresu

Niech $A = \langle V, S, \delta, \lambda \rangle$ będzie automatem (zupełnym lub niezupełnym), gdzie: V – zbiór liter wejściowych, S – zbiór stanów wewnętrznych, δ – funkcję przejść, a λ – funkcja wyjść. Przyporządkujmy (w sposób wzajemnie jednoznaczny) elementowi (v, s) z dziedziny funkcji przejść automatu liczbę naturalną ze zbioru $K = \{1, \ldots, t = |D_\delta|\}$. Oznaczmy odwzorowanie $D_\delta \rightarrow K$ przez K i określmy na zbiorze K podział charakterystyczny P_c w następujący sposób: do jednego bloku B podziału P_c zaliczamy te elementy z K, będące obrazami (przy odwzorowaniu K) takich par (v, s) z D_δ, którym funkcja przejść δ przyporządkowuje taki sam stan s' z S.

Na przykład dla automatu o tablicy przejść podanej w tablicy 4.35a i odwzorowania K podanego w tablicy 4.35b podział P_c ma następującą postać:

$$P_C = \left(\overline{1,8,12,14}; \overline{2,7,10,16}; \overline{6,9,13}; \overline{3,5,11,15}; \overline{4}\right)$$

Tablica 4.2

a)

	v_1	v_2	v_3	v_4
s_1	s_1	s_2	s_4	–
s_2	–	–	s_5	s_4
s_3	s_3	s_2	s_1	s_3
s_4	s_2	–	s_4	s_1
s_5	s_3	s_1	s_4	s_2

b)

	v_1	v_2	v_3	v_4
s_1	1	2	3	–
s_2	–	–	4	5
s_3	6	7	8	9
s_4	10	–	11	12
s_5	13	14	15	16

Niech $\Pi = \{P_1, \ldots, P_w\}$ będzie zbiorem dwublokowych podziałów na K, gdzie w spełnia nierówność:

$$\lceil \log_2 |S| \rceil \le w < \lceil \log_2 |S| \rceil + \lceil \log_2 |V| \rceil$$

Oznaczmy przez $(B_1, \ldots, B_i, \ldots, B_t)$ bloki iloczynu podziałów $P_1 \cdot P_2 \cdot \ldots \cdot P_w$). Ustalonej komórce pamięci (rys. 4.15) odpowiada zawsze jej adres w UMA – wektor $(a_1, \ldots, a_w)$, który tu będzie wyznaczany w sposób następujący: *i*-ta składowa wektora $(a_1, \ldots, a_w)$ odpowiadająca elementom *k* ze zbioru B_i jest 0, jeśli B_i należy do pierwszego bloku podziału P_i, natomiast 1 – jeśli B_i należy do drugiego bloku podziału P_i.

Zgodnie ze schematem z rys. 4.15 ustalonej komórce pamięci są przyporządkowane tylko takie elementy z *K*, którym odpowiadają pary (*v*, *s*) o tej samej wartości funkcji przejść δ automatu. W związku z tym adresy komórek spełniających powyższy warunek wyznaczymy za pomocą podziałów ze zbioru Π wtedy i tylko wtedy, gdy:

$$P = P_1 \cdot P_2 \cdot \ldots \cdot P_i \cdot \ldots \cdot P_w \leq P_c$$

przy czym liczba zajętych komórek (liczba słów) pamięci ROM będzie $2^w \times p$, gdzie $p = \lceil \log_2 |S| \rceil$.

Poszczególnym podziałom P_i na *K*, spełniającym powyższy warunek, odpowiadają tym samym zmienne adresowe $a_1, \ldots, a_w$, wytwarzane w układzie adresowania UMA. Przy ustalonym kodowaniu stanów wewnętrznych i liter wejściowych automatu, zmienne te są określonymi funkcjami zmiennych wewnętrznych *q* i zewnętrznych *x* tego automatu, a ustalenie podziałów P_i na *K* zapewniających najprostszy – przy danym *w* – układ adresowania nie sprawia trudności. Istotną rolę w syntezie układu UMA będzie spełniać pojęcie zgodności podziałów na zbiorze *K* z podziałami na zbiorze stanów *S* lub zbiorze wejść automatu $A = \langle V, S, \delta, \lambda \rangle$.

Podział *P* na zbiorze *K* nazywamy zgodnym z podziałem π na zbiorze *S* wtedy i tylko wtedy, gdy dla dowolnych wejść v_a, v_b warunek, że s_i i s_j należą do jednego bloku podziału π implikuje warunek, że elementy z *K* przyporządkowane odpowiednio parom (v_a, s_i) oraz (v_b, s_j) należą do jednego bloku podziału *P*.

Podział *P* na *K* nazywamy zgodnym z podziałem θ na *V* wtedy i tylko wtedy, gdy dla dowolnych stanów s_a, s_b warunek, że v_i, v_j należą do jednego bloku podziału θ implikuje warunek, że elementy z *K* przyporządkowane parom (v_i, s_a) oraz (v_j, s_b) należą do jednego bloku podziału *P*. W szczególności podział *P* na zbiorze *K* może być zgodny ze zbiorem $\{\pi, \theta\}$, jeśli jest zgodny zarówno z podziałem π, jak też θ. Natomiast podział *P* będziemy nazywać zgodnym ze zbiorem $\{\pi_1, \ldots, \pi_\alpha\}$ podziałów na *S* ($\{\theta_1, \ldots, \theta_\beta\}$ podziałów na *V*) wtedy i tylko wtedy, gdy *P* jest zgodny z podziałem π: $\pi = \pi_1 \cdot \pi_2 \cdot \ldots \cdot \pi_\alpha$ (gdy *P* jest zgodny z podziałem θ: $\theta = \theta_1 \cdot \theta_2 \cdot \ldots \cdot \theta_\beta$).

Na przykład dla automatu o tablicy przejść podanej w tablicy 4.35a i funkcji *K* podanej w tablicy 4.35b podział:

$$P = \left(\overline{1,2,3,6,8,9}; \overline{4,5,10,11,12}; \overline{13,14,15,16}\right)$$

jest zgodny z podziałem $\pi = \left(\overline{s_1, s_3}; \overline{s_2, s_4}; \overline{s_5}\right)$ natomiast:

$$P = \left(\overline{1,2,6,7}; \overline{3,4,5,8,9}; \overline{10,13,14}; \overline{11,12,15,16}\right)$$

jest zgodny ze zbiorem podziałów $\{\pi, \theta\}$: $\pi = (\overline{s_1, s_2, s_3}; \overline{s_4, s_5})$, $\theta = (\overline{v_1, v_2}; \overline{v_3, v_4})$

Obecnie przedstawimy metodę selekcji podziałów na *K* umożliwiającą uproszczenie układu adresowania pamięci dla układu o schemacie blokowym pokazanym na rys. 4.15, w którym *w* (liczba zmiennych adresowych) spełnia warunek:

$$\lceil \log_2 |S| \rceil \leq w < \lceil \log_2 |S| \rceil + \lceil \log_2 |V| \rceil$$

Punktem wyjścia rozważań będzie zakodowana tablica przejść automatu $A = \langle V, S, \delta, \lambda \rangle$, gdzie $\lceil \log_2 |S| \rceil = p$, $\lceil \log_2 |V| \rceil = n$. Załóżmy, że podziały kodujące stany wewnętrzne są $\pi_1, \ldots, \pi_p$, natomiast podziały kodujące wejścia są $\theta_1, \ldots, \theta_n$. Dla każdego podziału π i każdego podziału θ znajdujemy zgodny z nimi podział *P*. W celu zapewnienia jednoznaczności kodowania zmiennych adresowych $a_1, \ldots, a_w$ i jednocześnie odpowiedniego przyporządkowania elementów ze zbioru *K* – przy ustalonej funkcji *K* – poszczególnym komórkom pamięci, należy utworzyć *w* podziałów *P* na zbiorze *K* takich, że:

$$P_1 \cdot P_2 \cdot \ldots \cdot P_w \leq P_c \qquad (4\text{-}1)$$

gdzie P_c jest podziałem charakterystycznym automatu *A*.

Proces generacji podziałów *P* ułatwia pojęcie *r*-przydatności podziału (zbioru podziałów) *P* względem podziału P_c (porównaj rozdz. 3). Zauważmy bowiem, że warunek 4-1 mogą spełniać wyłącznie takie podziały *P*, które są *r*-przydatne względem P_c, gdzie $r \leq w$.

Wszystkich podziałów *P*, zgodnych z podziałami π oraz θ ustalonymi przez kodowanie automatu *A*, jest $n + p$. Oznaczmy je przez $P_z = \{P_1, \ldots, P_{n+p}\}$. Sytuacja optymalna będzie więc wtedy, gdy w zbiorze P_z istnieje *w*-przydatny (względem P_c) podzbiór. W tym przypadku każda zmienna adresowa *a* będzie funkcją jednej zmiennej – wewnętrznej *q* albo zewnętrznej *x* – automatu *A*. Jeśli ze zbioru P_z nie można wyselekcjonować *w*-elementowego podzbioru *w*-przydatnego względem P_c, to do dalszych obliczeń należy wybrać *u*-przydatny (względem P_c) podzbiór $\{P_{i_1}, \ldots, P_{i_u}\}$, gdzie *u* = max. Pozostałe $w - u$ podziały należy dobrać tak, aby:

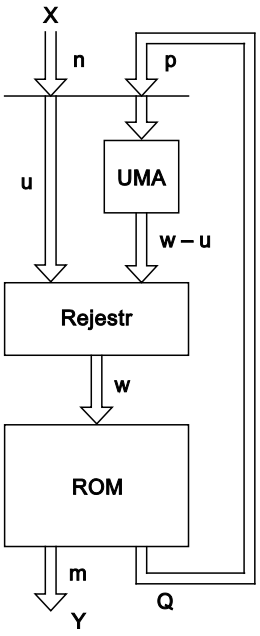
$$P_{i_1} \cdot P_{i_2} \cdot \ldots P_{i_u} \cdot P_{i_{u+1}} \cdot \ldots \cdot P_{i_w} \leq P_c$$

Oczywiście podziały $\{P_{i_{u+1}} \cdot \ldots \cdot P_{i_w}\}$ nie należą do zbioru P_z. Odpowiednia realizacja automatu jest przedstawiona na rys. 4.16. W układzie z tego rysunku wejścia adresowe $a_1, \ldots, a_u$ są funkcjami jednej

zmiennej wewnętrznej *q* albo zewnętrznej *x* automatu, natomiast wejścia a_{u+1}, …, a_w są funkcjami, których argumenty należą do zbioru $\{q_1, \ldots, q_p, x_1, \ldots, x_n\}$.

Przyjmijmy, że strukturę układu UMA będziemy określać układem liczb (b_1, …, b_i, …, b_w), gdzie b_i jest liczbą argumentów zmiennej adresowej a_i. Dla schematu automatu z rys. 4.16 mamy więc (b_1, …, b_u, b_{u+1}, …, b_w), przy czym $b_j = 1$ dla $1 \le j \le u$ oraz $1 < b_j \le m+p$ dla $u < j \le w$. Przy ustalonym kodowaniu stanów wewnętrznych i liter wejściowych automatu, najprostszą strukturę UMA (dla danego *w* oraz *u* = max) uzyskamy wtedy, gdy zmienne adresowe będą dobrane tak, że:

$$\sum_{i=u+1}^{w} b_i = \min.$$



Rys. 4.3. Realizacja automatu wg warunku 4-1

Z powyższych rozważań wynika, że największy wpływ na stopień skomplikowania układu adresowania ma ta jego część, w której zmienne adresowe są zależne od więcej niż jednego argumentu. Dlatego też najistotniejszy jest taki dobór kodowań stanów wewnętrznych i liter wejściowych automatu, przy którym można uzyskać maksymalny zbiór podziałów *P* (zgodnych z π albo θ) *w*-przydatny względem P_c Do kodowania należy więc wybierać takie podziały, których *r*-przydatność względem P_c jest równa co najwyżej *w*. Kodowania te będziemy generowali korzystając z faktu, że *r*-przydatność podziału *P* zgodnego z podziałem $\pi = (B_1, \ldots, B_i, \ldots, B_\alpha)$ jest równa:

$$r = \lceil \log_2 \alpha \rceil + \left\lceil \log_2 \max \left| \bar{\delta}(B_i) \right| \right\rceil$$

gdzie: B_i – blok podziału π, $\bar{\delta}$ – globalna funkcja przejść, a symbol $\max\left|\bar{\delta}(B_i)\right|$ oznacza liczbę elementów w najliczniejszym zbiorze $\bar{\delta}(B_i)$. Analogiczne zależności można wyprowadzić dla podziałów *P* zgodnych z θ, jak też zgodnych z $\{\pi, \theta\}$.

Z powyższego wynika, że na zbiorze *K* ponumerowanych komórek pamięci sposób postępowania jest analogiczny do dekompozycji funkcji boolowskiej i polega na wykonaniu następujących czynności:

1. Wybór zbioru *U* (wstępne kodowanie)
2. Określenie podziałów:

   $P(U)$, $P_g = P(V)$
3. Wyznaczenie podziału $\Pi_g \ge P_g$

   $P(U) \bullet \Pi_g \le P_F$

   (czasami istnieje potrzeba wprowadzenia zbioru W)
4. Obliczenie funkcji *G* reprezentującej układ modyfikacji adresu oraz funkcji *H* reprezentującej adresowanie pamięci ROM.

5 Przykłady syntezy

PRZYKŁAD 4.5

Metoda syntezy układu sekwencyjnego w strukturze z rys. 4.16 zostanie omówiona na przykładzie automatu z tab. 4.35, którą dla dalszych obliczeń zapisujemy po odpowiedniej permutacji wierszy, tak jak to pokazano w tabl. 4.36 a i b.

Tablica 4.1

a)

$x_1 x_2$	00	01	11	10
s_1	s_1	s_2	s_4	–
s_2	–	–	s_5	s_4
s_4	s_2	–	s_4	s_1
s_3	s_3	s_2	s_1	s_3
s_5	s_3	s_1	s_4	s_2

b)

$x_1 x_2$ / $q_1 q_2 q_3$	00	01	11	10
s_1 000	1	2	3	–
s_2 001	–	–	4	5
s_4 010	6	–	7	8
s_3 111	9	10	11	12
s_5 101	13	14	15	16

W automacie tym niezbędna liczba wejść adresowych pamięci jest 5. Aby zrealizować ten automat z wykorzystaniem pamięci o 4 wejściach adresowych należy zaprojektować odpowiedni układ modyfikacji adresu. W tym celu dzieli się tablicę przejść-wyjść (stany następne ponumerowano liczbami naturalnymi – tablica 4.36b) na 8 części w taki sposób, aby każda z nich zawierała co najwyżej dwa różne stany następne. Oczywiście podział tablicy przejść na podtablice jest określony przez kodowanie stanów wewnętrznych i liter wejściowych automatu. Dla automatu z tab. 4.36a są to podziały $\pi = (\overline{s_1, s_2, s_4}; \overline{s_3, s_5})$ na zbiorze stanów automatu oraz θ na zbiorze liter wejściowych, ale w tym przypadku θ jest podziałem zerowym.

Dla $U = \{x_1, x_2, q_1\}$, $V = \{q_2, q_3\}$ podziały te wyznaczają zgodne z nimi podziały na zbiorze ponumerowanych komórek pamięci (wg tab. 4.36b).

$$P(U)|P_c = \left(\overline{(1)(6)}; \overline{(2)}; \overline{(3,7)(4)}; \overline{(5)(8)}; \overline{(9)(13)}; \overline{(10)(14)}; \overline{(11)(15)}; \overline{(12)(16)};\right)$$

$$P(V) = \left(\overline{1,2,3}; \overline{4,5,13,14,15,16}; \overline{6,7,8}; \overline{9,10,11,12}\right)$$

Zgodnie z metodą dekompozycji szeregowej należy teraz wyznaczyć odpowiedni podział Π_G, który dla zapewnienia warunku: $P(U) \cdot \Pi_G \leq P_C$ musi oddzielać elementy 1 od 6; 3,7 od 4 itp. – jak to wynika z podziału ilorazowego $P(U)|P_C$. Łatwo sprawdzić, że jest to niemożliwe, gdyż odpowiedni podział Π_G miałby w swoim pierwszym bloku B_1 dwa pierwsze bloki $P(V)$, a w drugim – B_2 – dwa ostatnie bloki $P(V)$, czyli

$$B_1 = \left\{\overline{1,2,3}; \overline{4,5,13,14,15,16}\right\}, B_2 = \{\overline{6,7,8}; \overline{9,10,11,12}\}.$$

PRZYKŁAD 4.5 c.d.

Tym samym po oddzieleniu 1 od 6 nie można spełnić warunku na oddzielenie elementów 3,7 od 4. Dlatego powiększamy zbiór *B* dodając do niego argument x_1. Zmienna x_1 oddziela elementy 1,2 od 3 (tab. 4.36b), tym samym element 3 będzie można przenieść do drugiego bloku podziału Π_G.

Zatem:

$V' = \{x_1, q_2, q_3\}$

$P(V') = \left(\overline{1,2}; \overline{3}; \overline{4,5,15,16}; \overline{13,14}; \overline{6},; \overline{7,8}; \overline{9,10}; \overline{11,12}\right)$

$\Pi'_G = \left(\overline{1,2,4,13,14,15,16}; \overline{3,6,7,8,9,10,11,12}\right)$

W tej sytuacji

$P(U) \cdot \Pi'_G = \left(\overline{1}; \overline{2}; \overline{6}; \overline{3,7}; \overline{4}; \overline{5}; \overline{8}; \overline{9}; \overline{10}; \overline{11}; \overline{12}; \overline{13}; \overline{14}; \overline{15}; \overline{16}\right)$

czyli $P(U) \cdot \Pi'_G \leq P_C$.

Tablica prawdy funkcji *g* reprezentującej UMA jest podana w tablicy 4.37, a tablica określająca zawartość pamięci – w tablicy 4.38.

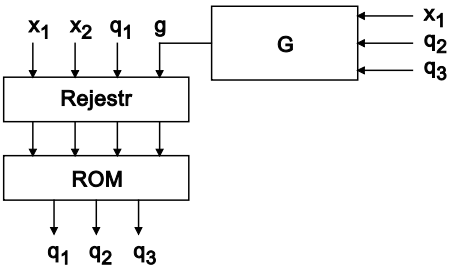
Tablica 4.2

	q_2	q_3	x_1	g
$\overline{1,2}$	0	0	0	0
$\overline{3}$	0	0	1	1
$\overline{4,5,15,16}$	0	1	1	0
$\overline{13,14}$	0	1	0	0
$\overline{6}$	1	0	0	1
$\overline{7,8}$	1	0	1	1
$\overline{9,10}$	1	1	0	1
$\overline{11,12}$	1	1	1	1

Z tablicy 4.37 wyznaczamy $g = q_2 + \bar{q}_3 x_1$. Schemat realizacji automatu pokazano na rysunku 4.17.

Tablica 4.3

	q_2	q_3	x_1	g
$\overline{1,2}$	0	0	0	0
$\overline{3}$	0	0	1	1
$\overline{4,5,15,16}$	0	1	1	0
$\overline{13,14}$	0	1	0	0
$\overline{6}$	1	0	0	1
$\overline{7,8}$	1	0	1	1
$\overline{9,10}$	1	1	0	1
$\overline{11,12}$	1	1	1	1



Rys. 4.1. Schemat logiczny realizacji automatu z przykładu 4.5

PRZYKŁAD 4.5 c.d.

Ten sam automat można również zrealizować w strukturze pokazanej na rys. 4.18. W strukturze tej wejściami bezpośrednimi są tylko zmienne wewnętrzne zbioru $U = \{q_1, q_2\}$, czyli $V = \{x_1, x_2, q_3\}$. Taką realizację uzyskamy przy podziale tablicy przejść (tab. 4.35) automatu na 4 części, z których każda zawierać będzie nie więcej niż 4 stany następne. Jedną z możliwości takiego podziału pokazano w tablicy 4.39a.

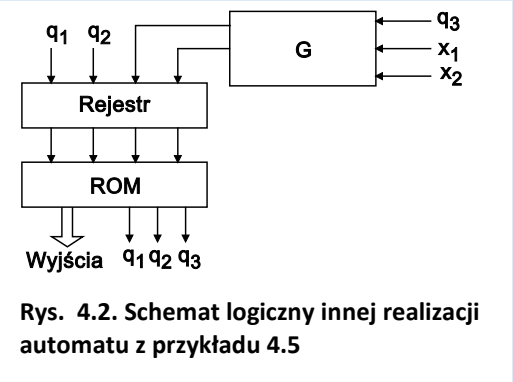
Tablica 4.4

a)

s \ x_1x_2	00	01	11	10
s_1	s_1	s_2	s_4	–
s_2	–	–	s_5	s_4
s_3	s_3	s_2	s_1	s_3
s_4	s_2	–	s_4	s_1
s_5	s_3	s_1	s_4	s_2

b)

$q_1q_2q_3$ \ x_1x_2	00	01	11	10
s_1 000	1	2	3	–
s_2 001	–	–	4	5
s_3 010	6	7	8	9
s_4 111	10	–	11	12
s_5 101	13	14	15	16



Rys. 4.2. Schemat logiczny innej realizacji automatu z przykładu 4.5

Z kolei w tablicy 4.39b dokonano numeracji zajętych kratek tablicy przejść (są to jednocześnie abstrakcyjne adresy komórek pamięci ROM) oraz wprowadzono kodowanie: wstępne dla zmiennych q_1, q_2 i wtórne dla zmiennej q_3. Na tej podstawie wyznaczamy podziały $P(U)$, $P(U)|P_C$ oraz $P(V)$:

$$P(U) = \left(\overline{1,2,3,4,5}; \overline{6,7,8,9}; \overline{10,11,12}; \overline{13,14,15,16}\right)$$

$$P(U)|P_c = \left(\overline{(1)(3,5)(2)(4)}; \overline{(6,9)(7)(8)}; \overline{(10)(11)(12)}; \overline{(13)(14)(15)(16)}\right)$$

$$P(V) = \left(\overline{1,6}; \overline{10,13}; \overline{2,7}; \overline{14}; \overline{3,8}; \overline{4,11,15}; \overline{9}; \overline{5,12,16}\right)$$

Bloki $P(V)$ są elementami kratek należących do wierszy S_1, S_3 albo S_2, S_4, S_5 i kolumn x_1x_2 kodowanych 00, 01, 11, 10. Na przykład blok z elementami 1,6 należy do wierszy S_1, S_3 i kolumny $x_1x_2 = 00$, natomiast blok 10,13 należy do wierszy S_2, S_4, S_5 i tej samej kolumny $x_1x_2 = 00$. Z bloków podziału $P(V)$ konstruujemy podział Π_G:

$$\Pi_G = \left(\overline{1,6,9,14}; \overline{3,5,8,12,16}; \overline{2,7,10,13}; \overline{4,11,15}\right)$$

Korzystamy przy tym z warunków „rozdziału” zapisanych w podziale ilorazowym $P(U)|P_C$. Sposób tej konstrukcji jest pokazany w tablicy 4.40.

Tablica 4.5

(1)	(3,5)	(2)	(4)
$\overline{1,6}$	$\overline{3,8}$	$\overline{2,7}$	$\overline{4,11,15}$
$\overline{9}$	$\overline{5,12,16}$	$\overline{10,13}$	
$\overline{14}$			

PRZYKŁAD 4.5 c.d.

Oczywiście, w celu obliczenia tablicy prawdy (co pomijamy) dla funkcji g_1, g_2 modyfikujących adres należy zakodować bloki podziału Π_G. Wystarczy przyjąć dla pierwszego bloku kod 00, a dla następnych kolejno 01, 10, 11. Wtedy poszczególne wiersze tej tablicy będą wyznaczane przez bloki podziałów $P(U)$ i Π_G. Na przykład blok $\overline{1,6}$ determinuje wektor x_1, x_2, q = 000, któremu odpowiadają wyjścia g_1g_2 = 00. Dalej analogicznie. Natomiast do obliczenia tablicy adresowania pamięci ROM trzeba obliczyć iloczyn $P(U) \cdot \Pi_G$:

$$P(U) \cdot \Pi'_G = \left(\overline{1}; \overline{2}; \overline{3,5}; \overline{4}; \overline{6,9}; \overline{7}; \overline{8}; \overline{10}; \overline{11}; \overline{12}; \overline{13}; \overline{14}; \overline{15}; \overline{16}\right)$$

Bloki tego iloczynu determinują rzeczywiste adresy i zawartość komórek pamięci ROM. Na przykład, abstrakcyjnemu adresowi, reprezentowanemu przez blok $\overline{1}$ odpowiada adres rzeczywisty $q_1q_2g_1g_2$ = 0000. W komórce ROM o tym adresie musi być zapisany kod stanu S_1, czyli $q_1q_2q_3$ = 000. Podobnie, dla trzeciego bloku $\overline{3,5}$ mamy adres $q_1q_2g_1g_2$ = 0001 oraz zawartość jako kod stanu S_4, czyli 111.

Omówione przykłady pozwalają stwierdzić, że proces kodowania stanów wewnętrznych automatu dla realizacji na pamięciach znakomicie się upraszcza i głównie polega na wyznaczaniu kodowania wstępnego. Wyznaczanie kodowania wstępnego z reguły sprowadza się do odpowiedniego podziału tablicy przejść automatu. N przykład dla automatu z tablicy 4.41a kodowanie wstępne uzyskamy dzieląc tę tablica na 4 podtablice w sposób pokazany w tab. 4.41b.

Tablica 4.6

a)

x_1x_2	00	01	11	10
a	*e*	–	*c*	–
b	*e*	*a*	*f*	*f*
c	–	*e*	*c*	*f*
d	*a*	*a*	–	*c*
e	*a*	*d*	*b*	*b*
f	*d*	*a*	*c*	*c*

b)

x_1x_2	00	01	11	10
a	*e*	–	*c*	–
b	*e*	*a*	*f*	*f*
c	–	*e*	*c*	*f*
d	*a*	*a*	–	*c*
e	*a*	*d*	*b*	*b*
f	*d*	*a*	*c*	*c*

Zapewni to realizację tego automatu na pamięci o trzech wejściach adresowych, ze zbiorem wejść bezpośrednich $U = \{q_1, x_1\}$ i zbiorem wejść do UMA, $V = \{q_2, q_3, x_2\}$. Oczywiście nie można być pewnym, że możliwa będzie dekompozycja rozłączna i być może dla spełnienia warunku dekompozycji trzeba będzie powiększyć liczbę wejść do UMA.

Jeszcze inna sytuacja może się zdarzyć, gdy dla uzyskania właściwego podziału tablicy przejść na podtablice trzeba dokonać odpowiedniej permutacji wierszy tej tablicy. Z sytuacją taką mamy do czynienia w automacie o tablicy przejść podanej w tab. 4.42a. W tym przypadku odpowiednia permutacja oraz podział na podtablice powinny być takie jak w tab. 4.42b.

Tablica 4.7

a)

	000 v_1	001 v_2	110 v_3	111 v_4	010 v_5
a	*a*	*b*	*e*	*e*	*c*
b	*c*	*b*	*b*	*a*	*b*
c	*a*	*a*	*c*	–	*b*
d	*c*	*e*	*e*	*d*	*e*
e	*b*	*c*	*b*	–	–

b)

	000 v_1	001 v_2	110 v_3	111 v_4	010 v_5
b	*c*	*b*	*b*	*a*	*b*
e	*b*	*c*	*b*	–	–
a	*a*	*b*	*e*	*e*	*c*
c	*a*	*a*	*c*	–	*b*
d	*c*	*e*	*e*	*d*	*e*

Mogą być również takie automaty, w których struktura układu sekwencyjnego jest niezależna od kodowania stanów wewnętrznych. Sytuację taką omawiamy w poniższym przykładzie.

PRZYKŁAD 4.6

Automat z tablicy 4.43 zrealizować w strukturze UMA/ROM, stosując ROM o możliwie najmniejszej pojemności. W rozwiązaniu podać wyrażenie boolowskie funkcji modyfikującej adres oraz zawartość ROM.

Tablica 4.8

	v_1	v_2	v_3	v_4	v_5
a	*d*	–	*c*	*c*	*d*
b	*b*	*b*	–	*a*	–
c	*a*	*d*	*d*	*d*	–
d	–	*c*	*b*	–	*b*

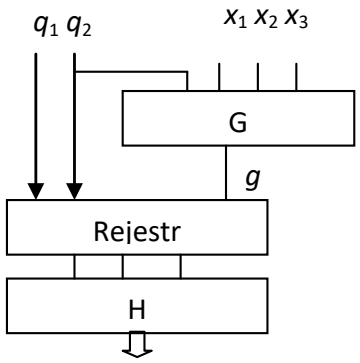
Dla danego automatu należy wstępnie przyjąć

$U = \{q_1, q_2\}$ $V = \{x_1, x_2, x_3\}$

co spowoduje ponumerowanie komórek pamięci, jak w tab. 4.44 i schemat dekompozycji, jak na rys. 4.19.

Tablica 4.9

$x_1x_2x_3$ / q_1q_2	v_1 000	v_2 001	v_3 010	v_4 011	v_5 100
a 00	1	–	2	3	4
b 01	5	6	–	7	–
c 10	8	9	10	11	–
d 11	–	12	13	–	14



Rys. 4.19. Schemat dekompozycji do przykładu 4.6

$$P_F = \left(\overline{1,4,9,10,11}; \overline{2,3,12}; \overline{5,6,13,14}; \overline{7,8}\right)$$

$$P(U)|P_F = \left(\overline{(1,4)(2,3)}; \overline{(5,6)(7)}; \overline{(8)(9,10,11)}; \overline{(12)(13,14)}\right)$$

$$P(V) = \left(\overline{1,5,8}; \overline{6,9,12}; \overline{2,10,13}; \overline{3,7,11}; \overline{4,14,}\right)$$

Podział Π_G tworzymy z bloków $P(V)$:

1. blok $\overline{1,5,8}$; $\overline{4,14}$; $\overline{6,9,12}$
2. blok $\overline{2,10,13}$; $\overline{3,7,11}$

PRZYKŁAD 4.6 c.d.

Konflikty wymagają rozdzielenia elementów 6, 12 od 9 oraz 4 od 14, co można uzyskać za pośrednictwem zmiennej q_2. Dlatego dla dekompozycji nierozłącznej wybieramy q_2 i liczymy podział $P(q_2)$:

$$P(q_2) = \left(\overline{1{,}2{,}3{,}4{,}8{,}9{,}10{,}11};\ \overline{5{,}6{,}7{,}12{,}13{,}14}\right)$$

co ułatwia obliczenie $P(V')$:

$$P(V') = P(V) \cdot P(q_2) = \left(\overline{1{,}8};\ \overline{5};\ \overline{6{,}12};\ \overline{9};\ \overline{2{,}10};\ \overline{13};\ \overline{3{,}11};\ \overline{7};\ \overline{4};\ \overline{14}\right)$$

Z bloków $P(V')$ łatwo można utworzyć Π'_G:

$$\Pi'_G = \left(\overline{1{,}4{,}5{,}6{,}8{,}12};\ \overline{2{,}3{,}7{,}9{,}10{,}11{,}13{,}14}\right)$$

$$P(U) \cdot \Pi'_G = P(q_1, q_1) \cdot \Pi'_G =$$

$$= \left(\overline{1{,}2{,}3{,}4};\ \overline{5{,}6{,}7};\ \overline{8{,}9{,}10{,}11};\ \overline{12{,}13{,}14}\right) \cdot \left(\overline{1{,}4{,}5{,}6{,}8{,}12};\ \overline{2{,}3{,}7{,}9{,}10{,}11{,}13{,}14}\right) =$$

$$= \left(\overline{1{,}4};\ \overline{2{,}3};\ \overline{5{,}6};\ \overline{7};\ \overline{8};\ \overline{9{,}10{,}11};\ \overline{12};\ \overline{13{,}14}\right)$$

Na tej podstawie możemy wyznaczyć tablice prawdy bloków układu modyfikacji (tab. 4.45 dla G) oraz układu adresowania pamięci ROM (tab. 4.46 dla H), zgodnie z rys. 4.19.

Tablica 4.10.

	$q_2x_1x_2x_3$	
1,8	0000	
4	0100	
5	1000	
6,12	1001	
2,10	0010	
3,11	0011	
7	1011	
9	0001	
13	1010	
14	1100	

Tablica 4.11.

	$q_1\ q_2 g$	*Stan*
1,4	000	d
2,3	001	c
5,6	010	b
7	011	a
8	100	a
9,10,11	101	a
12	110	c
13,14	111	b

6 Zadania

ZADANIE 4.1

Zrealizować na przerzutnikach typu D oraz JK automaty z przykładów 4.1 oraz 4.2.

ZADANIE 4.2

Zminimalizować i zrealizować na przerzutnikach typu D oraz JK automat podany w tablicy 4.47.

Tablica 4.2

S\x	0	1	0	1
1	1	7	0	0
2	4	3	1	1
3	–	5	–	0
4	–	2	–	0
5	4	–	1	–
6	8	–	1	–
7	–	6	–	0
8	–	–	–	1

ZADANIE 4.3

Automat o tablicy przejść podanej w tabl. 4.48 zrealizować na pamięci ROM o 4 wejściach adresowych i najprostszym bloku UMA. W rozwiązaniu podać wyrażenie boolowskie dla UMA. Przyjąć oznaczenia wejść jako: x_1, x_2.

Tablica 4.1

	v_1	v_2	v_3	v_4
a	–	*c*	*d*	*a*
b	*b*	*c*	*a*	–
c	*c*	–	*a*	*b*
d	–	*d*	*a*	*b*
e	*c*	–	–	*a*

4.7. Bibliografia

[4.1] Astola J. T. Stanković R. S.: *Fundamentals of Switching Theory and Logic Design*, Dordrecht: Springer, 2006.

[4.2] Borowik G.: *Improved State Encoding for FSM Implementation in FPGA Structures with Embedded Memory Blocks,* Electronics and Telecommunications Quarterly, vol. 54, no 1, 9-28, March 2008.

[4.3] Borowik G., Falkowski B., Łuba T.: Cost-Efficient Synthesis for Sequential Circuits Implemented Using Embedded Memory Blocks of FPGA's, IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems, pp. 99-104, Kraków, Poland, April 11-13, 2007.

[4.4] Brown S, Vranesic Z.: *Fundamentals of Digital Logic with VHDL Design.* McGraw Hill, Boston 1998.

[4.5] Czerwiński R., Kania D.: *Finite State Machine Logic Synthesis for Complex Programmable Logic Devices*. Springer, Berlin Heidelberg 2013.

[4.6] Jóźwiak L., Ślusarczyk A., Chojnacki A.: Fast and Compact Sequential Circuits through the Information-Driven Circuit Synthesis, Proc. of Euromicro Symposium on Digital Systems Design (2001), 46-53, Warsaw, Poland, 4-6 September 2001.

[4.7] Kamionka-Mikuła H., Małysiak H., Pochopień B.: *Synteza i analiza układów cyfrowych*. Wydawnictwo Pracowni Komputerowej Jacka Skalmierskiego, 2012.

[4.8] Łuba T., Górski K., Wroński L.B.: *ROM-based Finite State Machines with PLA Address Modifiers*, Proc. of EURO-DAC'92, IEEE Computer Society Press, pp. 272-277, Hamburg 1992.

[4.9] Łuba T., Borowik G.: Synteza logiczna, Oficyna Wydawnicza PW, Warszawa 2015.

[4.10] Łuba T., Borowik G., Kraśniewski A.: *Synthesis of finite state machines for implementation with programmable structures,* Electronics and Telecommunications Quarterly, vol. 55, no 2, pp. 183-200, 02/2009.

[4.11] Rawski M., Selvaraj H., Łuba T., Szotkowski P.: *Multilevel Synthesis of Finite State Machines Based on Symbolic Functional Decomposition*, International Journal of Computational Intelligence and Applications, Vol. 6, No. 2, June 2006, pp. 257-271, Imperial College Press 2007.

[4.12] Stańczyk U., Cyran K., Pochopień B.: *Theory of logic circuits – volume 1 Fundamental issues*. Publishers of the Silesian University of Technology, Gliwice 2007.

[4.13] Stańczyk U., Cyran K., Pochopień B.: *Theory of logic circuits – volume 2 Circuit design and analysis*. Publishers of the Silesian University of Technology, Gliwice 2007.

[4.14] Zieliński C.: *Podstawy projektowania układów cyfrowych*. PWN, Warszawa 2003.